주기억장치 데이터베이스를 위한 포인터 기반 모델의 구축

정회원 배 명 남*, 최 완**

A Construction of Pointer-based Model for Main Memory Database Systems

Myung-Nam Bae*, Wan Choi** *Regular Members*

요 약

주기억장치 데이터베이스 시스템(MMDBMS)은 디스크가 아닌 주기억장치를 주요 저장 매체로 사용하므로, 고성능이 요구되는 데이터베이스 응용을 효과적으로 지원한다. 최근, 인터넷을 중심으로 고속 데이터 처리와 함께, 보다 향상된 데이터 표현력과 보다 엄격한 데이터 일관성 보장에 대한 요구가 증대되고 있다. MMDBMS에서는 모든 데이터가 주기억장치에 상주하기 때문에, 성능에 대한 오버헤드 없이 그러한 요구를 만족시킬 수 있는 방법을 제공할 수 있다. 이들은 구체적으로 데이터를 처리하기 위한 연산과 참조 무결성과 같은 제약들로 이루어진다. 이들로 구성된 데이터 모델은 DBMS에서 데이터베이스의 표현력을 측정하는 중요한 컴포넌트이다.

본 논문에서는 통신 서비스의 제공에 필요한 요구들과 이를 지원하는 데이터 모델에 대해 설명한다. 다루는 주요 이슈들은 1) 포인터를 사용한 테이블간의 관계성 정의, 2) 관계성에 의한 데이터들의 항해, 3) 포인터 연산에 대한 참조 무결성 지원, 4) 조인시 균일한 처리 시간 보장, 5) 객체지향 특성 지원, 6) 다중 테이블간 인덱스의 공유 등이다. 본 논문에서는 복잡한 응용 환경에서 이러한 이슈들을 지원할 수 있도록 설계한 데이터 모델에 대해 설명한다.

ABSTRACT

The main memory database systems (MMDBMS) efficiently supports various database applications that require high performance since it employs main memory rather than disk as a primary storage. Recently, it has been increased needs that have the fast data processing as well as the efficient modeling of application requiring for a complicated structure, and conformity to applications that need the strict data consistency. In MMDBMS, because all the data is located in the main memory, it can support the usable expression methods of data satisfying their needs without performance overhead. The method has the operation to manipulate the data and the constraint such as referential integrity in more detail. The data model consists of this methods is an essential component to decide the expression power of DBMS.

In this paper, we discuss about various requests to provide the communication services and propose the data model that support it. The mainly discussed issues are 1) definition of the relationship between tables using the pointer, 2) navigation of the data using the relationship, 3) support of the referential integrity for pointer, 4) support of the uniform processing time for the join, 5) support of the object-oriented concepts, and 6) sharing of an index on multi-tables. We discuss the pointer-based data model that designed to include these issues to efficiently support complicated application environments.

* 한국전자통신연구원 컴퓨터시스템연구부 바이오정보연구팀(mnbae@etri.re.kr),
** 한국전자통신연구원 컴퓨터시스템연구부 (wchoi@etri.re.kr)
논문번호 : 020375-0830, 접수일자 : 2002년 9월 3일

Ⅰ. 서 론

미래의 통신 서비스는 데이터베이스 기술과 매우 밀접한 관계를 가지게 될 것이다. 이는 통신 네트워크와 서비스의 각종 연산과 관리에 필요한 정보들이 데이터베이스에 존재해야 하기 때문이다. 이에 따라, 1990년대 이후 각종 ITU-T의 권고 안에서 실시간 트랜잭션 개념과 객체지향 개념을 권고하고 있다. 뿐만 아니라, 각종 이동 통신 시스템에서도 데이터베이스에 대한 동일한 요구 사항들을 갖고 있는 상태이다. 따라서, 다양한 통신 응용을 지원하기 위한 가장 이상적인 데이터베이스 시스템은 고가용성과 함께 객체지향 모델을 지원하는 것이다[1].

현재, 통신 환경에서, 전통적인 교환/전송 서비스의 중요도는 감소하고 있는 반면, 차별화된 부가가치 서비스 제공에 대한 관심과 중요도는 상대적으로 매우 증대되고 있다. 이를 위해서는, 매우 안정된 구조 뿐만 아니라 새로운 시스템 변화 요구에 따라 변화를 쉽게 반영할 수 있는 구조 역시 매우 중요하다[2]. 미래의 통신 응용은 각종 서비스를 빠르고 효과적으로 생성, 배포, 관리하여야 하기 때문이다. 이러한 측면에서, 새로운 기술에 빠르게 변화하고 대처할 수 있는 능력은 CSP(Communication Service Provider)에게 비용과 서비스 측면에서 경쟁 우위의 잠재성을 부여해 줄 수 있으므로, 효율적이고 안정적인 데이터베이스 시스템은 이러한 요구를 만족시키기 위한 주요한 요소이다. 이러한 능력은 이들이 기반하고 있는 소프트웨어 기술의 진보성 때문에 가능하다고 할 수 있다.

이미, ITU-T의 각종 권고안과 또 다른 표준안의 대부분은 객체지향을 기반으로 하고 있다. 그 결과, 통신 응용을 위한 데이터베이스 시스템은 객체지향 모델의 지원으로부터 시작하는 것이 타당하다. 즉, 응용 시스템에 고가용성을 보장하면서 객체지향 모델링 개념을 지원하고 있는 데이터 관리 시스템이 적합하다. 그러나, 현재의 객체지향 데이터베이스 시스템들은 대부분의 트랜잭션이 짧고, 처리량(throughput)을 최대화하기 위해 예측 가능한 시간 내에 완료될 수 있어야 하며, 또한 제한적인 하드웨어 구조위에서 운용 가능하도록 해야 한다는 등의 통신 환경에서의 요구 사항을 만족시키지 못하고 있다. 이들은 대부분 통신 응용과는 다소 상이한 범용 응용을 위한 데이터의 구조나 처리 특성에 최적으로 맞추어져 있기 때문이다. 본 논문에서는 이러한 요구사항에 따라, 통신 응용 시스템에 고가용성을 보장하는 데이터베이스 시스템에 새로이 객체지향 모델링 개념을 확장 지원하는 방법에 대해 기술한다.

통신 응용에서, 고속 데이터 처리를 위한 방법은 데이터를 디스크 보다 빠른 주기억장치에 유지시키는 것이다. 이는 컴퓨터의 하드웨어 능력이 크게 향상됨에 따라 고속의 데이터 처리가 필요한 실시간 응용 분야에 급격히 확대되고 있다. 주기억장치 DBMS(Main-memory resident DBMS, 이하 MMDBMS)[3]는 모든 데이터를 주기억장치에 상주시킴으로써, 데이터에 대한 고속 처리를 제공하는 시스템 소프트웨어이다. 일반적으로, MMDBMS는 고속 데이터 처리와 함께, 사용자가 데이터를 표현하고 구조화하는 주요한 개념적 도구인 데이터 모델을 함께 제공한다. 현재, 대부분의 순수 MMDBMS는 관계형 데이터 모델을 기반으로 하고 있다. 관계형 모델은 모든 데이터가 일련의 정형화된 테이블 형태로 존재하는데, 각 테이블은 데이터의 특성에 따라 여러 개의 컬럼(column)이 포함된다. 컬럼은 해당 데이터의 특성에 따라 값의 범위나 그 값에 적용할 수 있는 제한 사항(constraints)을 정의할 수 있다. 또한, 잘 정의된 표준 검색 언어인 SQL(Structured Query Language)를 통해 관계형 데이터베이스에 있는 데이터를 직접 조회하고 유지하는데 사용된다. 그러나, 관계형 모델의 특성상 모든 데이터는 항상 정형화해야 하고, 여러 테이블에 나누어 저장되므로 정보를 추출하기 위해선 조인 연산이 수행되어야 한다. 특히, MIB이나 LDAP의 DIT(Directory Information Tree)와 같은 복잡한 객체들은 질의 시점에 반드시 조인을 통해 재구성되어야 한다. 이러한 재구성 단계는 매우 비싼 연산이며 시스템의 성능에도 많은 제약을 준다[4]. 관계형 모델이 갖는 이러한 한계들을 해결하기 위한 방법으로, 기존의 OODBMS를 기반으로 다량의 공유 버퍼를 갖는 형태의 OOMMDBMS 들[1, 5, 6]이 제시되었다. 이들은 풍부한 데이터 모델을 지원하고 있지만, 하위의 OODBMS없이 독자적으로 수행될 수 없고 또한 많은 시스템 자원을 요구한다는 측면에서 통신 응용 등과 같이 제한된 분야에서 활용되기에는 어려움이 많다.

본 논문에서는 통신 환경에서 실시간 응용을 지원하기 위해, ETRI에서 개발한 MMDBMS인 Tachyon의 데이터 모델에 대해 설명한다. Tachyon

데이터 모델은 기존 응용과의 호환을 위해 관계형 모델과 표준 검색 언어인 SQL을 제공하고 있으며, 추가로 MMDBMS에서 모든 데이터는 포인터로 직접 접근 가능하다는 점에 착안하여, 높은 수준의 모델링 기능들을 제공하기 위해 1) 포인터를 사용한 다양한 관계성의 표현, 2) 관계성에 의한 데이터들의 항해, 3) 포인터 연산에 대한 일관성 지원, 4) 죠인시 균일한 처리 시간 보장, 5) 상속, 사용자 정의 타입과 같은 객체지향 특성 지원, 6) 다중 테이블간 인덱스의 공유 등과 같은 데이터 표현 방식을 추가로 제공한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 Tachyon의 주요 특징과 개괄적인 구조에 대해 간략히 소개한다. 제 3장은 관련 연구로서, 기존 MMDBMS에 적용된 데이터 모델들에 대해 설명하고, 각각의 특징과 장단점에 대해 논의한다. 제 4장에서는 Tachyon 데이터 모델에 대한 세부 내용과 주요 이슈에 대해, 제 5장에서는 성능과 질의 최적화 측면의 몇몇 고려 사항에 대해 논의한다. 제 6장에서는 본 논문을 요약하고, 현재의 연구 진행 방향에 대해 기술한다.

II. Tachyon

이 장에서는 ETRI에서 개발된 차세대 MMDBMS Tachyon에 대하여 간략히 소개하며, 주요 특징은 다음과 같다.

- 효과적인 실시간 응용의 지원을 위해, 모든 데이터를 주기억장치에 상주시켜 고성능을 제공한다. 응용의 요구에 따라 디스크와의 연동없이 주기억장치 상의 데이터 만을 대상으로 독자적으로 수행될 수도 있다.
- 고장 감내(fault tolerant)를 보장하기 위한 이중화 구조를 제공한다.
- 다양한 응용을 위해, 실시간 OS인 SROS[7], VxWorks외에도 범용 OS인 Solaris, NetBSD, Windows NT/2K 등 다중 플랫폼을 지원한다.
- 클라이언트 서버 모델이며, ESQL, JDBC, ODBC 인터페이스를 통한 다양한 개발 환경을 제공한다. 특히, Tachyon과 응용 프로그램이 동일 호스트에 존재할 경우, 공유 메모리(shared memory)를 저장 매체로 사용하여, TCP/UDP없이 연동하여 수십 배 이상 성능을 향상시킬 수 있다.
- Tachyon에 할당된 LWP를 최대한 활용하기 위해, 다중 쓰레드 구조를 채택하고 있다. 서버뿐만 아니라 다중 쓰레드 기반 클라이언트 개발을 위한 인터페이스(다중 쓰레드용 ESQL)를 추가로 제공한다.

Tachyon의 개괄적인 구조는 그림 2.1과 같다. 메모리 관리자는 일정 주기억장치 영역을 가용한 메모리 블록의 풀(pool)로 관리하며, 상위 관리자들의 요구에 따라 풀에서 메모리 블록을 할당해 주고, 사용이 완료된 메모리 블록은 다시 풀에 반환하여 재사용될 수 있도록 관리한다.

인덱스 관리자는 저장된 사용자 데이터를 신속하게 검색할 수 있도록 하는 기능을 제공한다. 즉, 데이터의 특정 부분을 키로 선정하고, 이를 사용하여 해당 데이터에 직접 접근할 수 있도록 한다. 현재, Tachyon에서는 T트리[8], CSB+ 트리[9]를 지원하고 있다.

객체 관리자는 사용자의 데이터를 표현하기 위한 각종 메타 정보를 관리하는 기능을 제공한다. 모든 데이터는 값으로 표현되기 위한 타입, 크기, 제약 등을 가지게 되는데, 이러한 정보를 저장하고 사용하기 위한 제반 메타 정보들로 구성된다. 본 논문에서는 메타 정보에 대한 구조와 운용 방식을 통해, Tachyon 데이터 모델을 설명하고자 한다.

동시성 제어기[10]는 한 데이터에 여러 트랜잭션들이 동시에 접근하여 수정할 때, 데이터베이스의 일관성(consistency)을 보장하는 기능을 보장한다. 즉, 동시에 수행하는 다수 트랜잭션이 순차적으로 수행되도록 하는 직렬성(serialization)을 보장하여, 데이터베이스가 항상 올바른 상태로 유지되도록 한다.

백업/회복 관리자는 시스템에서 발생하는 각종 오류로부터 데이터베이스를 보호하기 위한 기능을 제공한다. 백업 및 회복 관리자는 정상 동작시 변경 연산에 대한 로깅[11]을 수행하고, 주기적으로 주기억장치내 데이터베이스를 디스크로 백업함으로써, 오류 발생시 백업된 데이터베이스와 로깅 정보를 이용하여 전체 데이터베이스를 일관성있는 상태로 회복시켜준다.

이중화 관리자는 네트워크로 연결된 두 DBMS를 로그 정보를 이용해 이중화하여 관리함으로써, 시스템 고장 시에도 트랜잭션 단위의 일관성을 제공하면서 데이터베이스에 대한 지속적인 접근 서비스를 제공하는 기능을 제공한다.

질의 처리기는 Tachyon에서 제공하는 SQL[12]

형태의 선언적인 언어를 최적화하고, 하위 단계 관리자에 대한 호출로 변환하는 기능을 제공한다. DBMS 사용자는 SQL을 통하여 데이터베이스에 접근하므로 응용 시스템을 보다 손쉽게 개발할 수 있다.

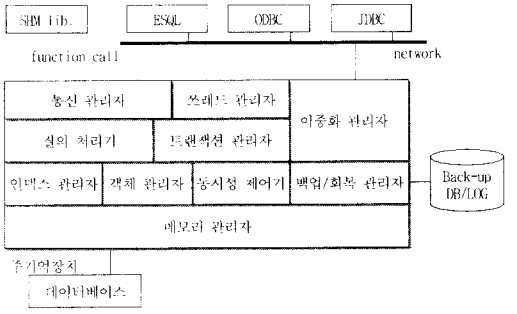

그림 2.1 Tachyon MMDBMS의 구조

트랜잭션 관리자는 우선 순위 큐를 지원하며, 트랜잭션스케쥴링에 있어 우선 순위 역행 현상을 보완하기 위해 트랜잭션에 대한 우선 순위 상속(priority inheritance)[13]을 수행함으로써 실시간 응용을 효과적으로 지원하도록 한다.

쓰레드 관리자는 서버내의 여러 쓰레드가 하나의 LWP를 공유하도록 하는 방식으로 프로세스 위주의 작업에서 발생하는 문맥 교환으로 인한 오버헤드를 피하도록 하는 기능을 제공한다. 특히, MMDBMS에서는 작은 단위의 잦은 접근을 요구한다는 점에서 적합한 방법이라고 할 수 있다

Ⅲ. MMDBMS를 위한 데이터 모델

데이터베이스 모델(이하, 데이터 모델)이란 실제 응용 환경에서 데이터베이스 관리 시스템에 저장하고자 하는 데이터들을 명확한 형태로 표현하고, 사용자나 설계자들이 필요한 데이터가 무엇인가를 정의할 수 있도록 하는 개념적인 도구이다. 따라서, 데이터 모델의 표현력은 응용의 구조를 결정하는데 매우 중요하다. 현재 대부분의 순수 MMDBMS 데이터 모델이 데이터가 디스크에 존재한다는 기존 디스크 기반 DBMS 데이터 모델을 기반으로 하고 있다는 점에서 중요하게 고려하여야 할 관점들을 살펴보고자 한다.

3.1 관계형 데이터베이스 모델

관계형 데이터베이스에서는 모든 데이터들을 테이블과 같은 형태로 나타내고, 저장된 데이터로부터 원하는 정보를 추출하는 방법을 정의하고 있다. 여기에서 테이블은 실세계에서 기본적으로 구분 가능한 대상인 엔티티(entity)를 의미하며, 테이블은 엔티티를구체화하기 위한 세부 특성들을 필드로 갖는다[14]. 예를들어, 출 중계호 라우팅 정보를 엔티티로 정의한다면, 이를 구체화하는 루트 자원 번호, 자원의 IP 등의 특성들은 출 중계호 라우팅 정보 테이블의 필드로 정의될 것이다. 관계형 데이터 모델은 데이터가 규격화되어 있고, 단순하고 정제된 처리 알고리즘으로 지원이 가능하다는 장점이 있다. 이를 위해, 응용 환경의 데이터 구조는 반드시 관계형 데이터베이스로 표현하기 위한 일련의 정규화(normalization) 과정을 거쳐야 한다. 정규화는 테이블 형태로 표현할 때 발생하는 값의 중복과 이에 따른 문제를 해결하기 위해, 테이블을 보다 작은 규모의 테이블로 재분할하는 것을 말한다. 사용자가 정보를 요구할 경우에는 이후 분할된 테이블을 다시 통합하는 죠인(join) 과정을 거쳐 추출되어 제공된다.

이와 같이 관계형 데이터베이스에서 데이터의 표현은 반드시 테이블 형태로 규격화된 정보만을 표현하고, 또한 정규화에 맞추어 테이블을 분할하여야 한다. 반면에, 응용 환경에서의 데이터는 일반적으로 관계형 데이터베이스와 같이 규격화되지 않는다. 따라서, 관계형 MMDBMS를 사용하는 응용 환경의 데이터 구조는 관계형 데이터베이스가 제공하는 모델에 맞도록 많은 제약과 한계를 갖고서 설계되어야 한다는 단점이 있다. 이러한 단점에도 불구하고, 현재 대부분의 MMDBMS[15, 16]는 그 단순성과 구현의 용이성 때문에 관계형 모델을 제공하고 있는 추세이다. 단, MMDBMS에서 관계형 모델은 모든 데이터가 주기억장치에 존재하기 때문에, 데이터의 논리적 메모리 주소를 값으로 갖는 데이터 타입을 제공하는 수준의 표현력만을 제공하고 있다.

3.2 객체지향 데이터베이스 모델

통신 응용의 한 예로, 네트워크 관리 기능은 통신 네트워크와 서비스에 대한 관리 표준으로, 데이터 네트워크에서 효율성과 생산성을 최대화하기 위해 제어하는 일련의 체계화된 과정으로, 구성/장애/보안 등의 분야를 관리하고 있다. 이때, 계속적으로 진화하는 네트워크에 따라, 새로운 장비와 서비스의 추가 등에 대한 변경을 수용하여야 한다. 이를 위해서는 데이터 추상화와 비정형화된 복잡한 구조의 효과적인 표현을 제공할 수 있는 소프트웨어 모델

링 수단이 필요하다[17]. 이러한 노력으로, 기존의 디스크 기반 OODBMS(exodus와 같은) 위에 다량의 공유 버퍼를 갖도록 확장된 OO(MM)DBMS들[1, 5, 6] 혹은 이러한 형태의 순수 OODBMS[17, 18]들이 제시되었다. 이러한 방식은 하위의 OODBMS가 제공하고 있는 객체지향 데이터 모델을 그대로 사용하면서, 모든 데이터를 주기억장치상에 위치시키는 방식이다. 현재, 이러한 형태의 확장은 기존의 응용에 대한 변화없이, 빠른 데이터 처리를 보장할 수 있기 때문에 가장 많이 사용되고 있다. 그러나, 이들은 공통적으로 디스크와의 연계가 고려되었다는 점에서, 단지 순수 디스크 기반 OODBMS에 비해 아주 빠른 성능을 제공하는 것이며, 정확한 의미에서 균일한 처리 시간을 보장한다고 할 수 없다. 또한, 하위의 OODBMS와 함께 운영되어야 한다는 점에서, 디스크가 존재하지 않거나 혹은 제한된 자원만이 사용가능한 특성화된 하드웨어 구조를 갖는 응용 분야에 적용하기 어렵다는 단점이 있다(이와 관련하여, Tachyon 모델과의 보다 근본적인 차이점은 관련 절에서 설명하도록 한다).

3.3 참조(reference)에 기반한 객체관계형 모델

응용 환경의 비정형화 된 데이터는 관계형 모델의 엄격하고 제한적인 제약 때문에 관계형 데이터베이스로 사상하기 어렵다[19, 20]. 이에 따라, 범용 데이터 구조인 객체지향 특성들을 관계형 모델에 확장 적용한 객체관계형 모델이 대두되었다. 현재, 객체관계형 모델을지원하는 MMDBMS[21]의 가장 중요한 특징인 데이터의 메모리 주소인 참조(reference)[1] 를 테이블내 필드로 허용하고, 죠인을 하지 않고도 참조가 지칭하는 다른 테이블내 데이터에 접근할 수 있다는 점이다. 이는 다른 테이블내 정보에 접근하기 위해서는 반드시 죠인이 요구되는 관계형 모델과 차별화되는 특성이다. 또한, 관계형 테이블내 모든 필드는 동일한 크기의 동일한 타입이어야 함에 비해, 객체관계형에서는 값들의 집합, 리스트, 심지어 다른 테이블 자체도 필드로 가질 수 있도록 그 표현력이 확장되었다. 테이블내에 이러한 복잡한 데이터 표현이 가능하도록 한 것은 응용 데이터 구조와 MMDBMS내 데이터베이스 구조와의 의미적 차이(semantic gap)를 줄임으로서, 구조의 차이에서 오는 변환의 복잡성과 의미 오류를 최소화할 수 있다는 장점이 있기 때문이다[22]. 그러나, [21] 역시, Tachyon 모델과는 달리, 주요한 모델링 요소인 테이블 상속과 상속에 기반한 제한없는 테이블 죠인을 지원하지는 못하고 있다.

3.4 Tachyon 데이터 모델의 설계 배경

Tachyon에 앞서 개발된 DREAM-S[23]는 지난 10여년간 전전자 교환기에서 호 처리 및 가입자 데이터 관리를 위해 사용되었으며, 모든 정보를 테이블 형태로 구조화하여 관리하였다. 관계형 모델만을 제공하는 DREAM-S는 각 테이블들을 독립하여 관리하였으며, 각 테이블간의 참조에 대해, 관련성과 제약을 명시할 관계성 표현 수단을 제공하지 않고 있었다. 그 결과, DREAM-S의 구조와 처리 단계를 단순화할 수 있었지만, 모델 표현력의 부재로 테이블간의 시맨틱 검증이 필요한 문제 해결에 많은 제약을 내포하게 되었다. 또한, 관계형 모델의 주요 연산인 죠인의 경우, 예측 가능하지 않은 완료 시간과 과도한 메모리 사용은 큰 문제점이 되었다. 따라서, Tachyon은 DREAM-S의 이러한 대표적인 문제점인 테이블간의 의미 표현력을 보다 증대시키고, 부가의 메모리 공간과 불필요한 메모리 복사 연산 없이도 죠인이 가능하도록 설계되었다. 또한, C++나 OOCHILL을 사용하는 개발자들이 만든 객체지향 구조를 Tachyon에서도 그대로 사용하도록 하기 위해, Tachyon 모델에서는 테이블 상속을 지원하고 이에 따른 질의 상속을 함께 허용하였다. 테이블 상속이라 함은 테이블들에서 공통된 구조를 추출하여 부모 테이블(parent table)로 정의하고 다른 테이블은 부모 테이블의 자식 테이블(child table)로 정의하는 것이다. 이때, 부모 테이블에 정의한 질의나 구조는 모든 자식 테이블에서도 마치 자신이 정의한 것처럼 확장된다. 질의 상속은 부모 테이블에 적용한 질의는 내부적으로 모든 자식 테이블에도 자동 적용됨을 의미한다. 이외에도 사용자 정의 타입, 다중 테이블 인덱스와 같은 여러 설계 시멘틱들을 추가하였다. 이러한 모든 확장은 SQL을 기반으로 수용되었다.

본 논문의 주안점은 고가용성과 고성능을 제공하면서, 동시에 응용 환경의 객체지향 모델이 Tachyon에서도 그대로 사용될 수 있도록 하는 데이터 모델에 대해 설명하는 것이다.

Ⅳ. Tachyon 데이터 모델

1) 주기억장치상의 데이터에 직접 접근할 수 있는 메모리 주소(혹은 포인터)

데이터 모델은 데이터베이스 설계 내용을 구체화하고 실제화시키기 위한 개념적인 도구이다. 이를 위해서는 사용자가 정의한 테이블과 테이블간의 관계성과 같은 설계 시멘틱을 유지하고, 그 무결성을 항상 보장하기 위한 메타 구조가 필요하다. 본 논문에서는 Tachyon이 제공하는 다양한 설계 시멘틱과 이를 보장하기 위한 메타 구조인 시스템 카타로그를 중심으로 기술한다. 실제로, 시스템 카타로그는 Tachyon 모델의 모든 설계 시멘틱을 가장 잘 활용하고 있는 예제이며, 또한, 여기에 적용된 각종 질의 연산 및 제약 조건은 사용자의 테이블에도 그대로 적용되므로, 개발자의 이해를 돕기 위해 이의 운용을 실제 Tachyon SQL과 함께 설명한다.

Tachyon의 시스템 카타로그는 그림 4.1과 같으며, 5개의 메타테이블과 이들간의 상호 연관성으로 이루어진다. 이들은 사용자가 정의한 테이블 정의, 제약 등과 같은 모든 데이터베이스 설계 시멘틱을 저장하고 유지하기 위한 것으로, 이에 대한 분석과 이해를 통해 Tachyon 모델을 정확하게 이해하고 활용할 수 있다.

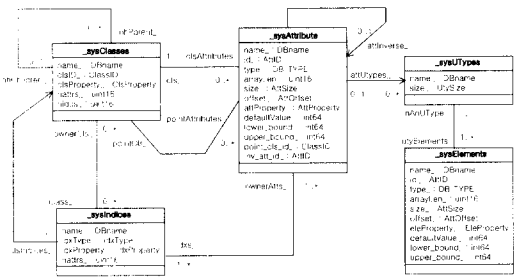

그림 4.1 Tachyon의 시스템 카타로그

여기에서 메타테이블은 데이터베이스 생성 시점에 Tachyon이 가장 먼저 초기화한 테이블들로 사용자가 정의한 설계 시멘틱을 저장하고 유지하기 위한 구조이다. 한 예로, 사용자가 정의한 모든 테이블 정보는 _sysClasses에, 모든 필드 정보는 _sysAttribute에 저장된다. 상호 연관성이란 데이터들간의 관계성을 나타내는 것으로, 한 예로, 사용자 테이블은 여러 사용자 필드들로 구성된다라는 관계성은 _sysClasses의 clsAttributes_필드가 _sysAttribute의 포인터들을 가짐으로써 표현된다. 역으로, _sysAttribute는 cls_ 필드로 자신을 갖는 테이블의 포인터를 가지게 될 것이다. 이로써, 사용자 테이블과 사용자 필드들은 상호 관계성이 있다라고 한다. 보다 구체적인 예로서, '사용자 테이블R1'의 필드가 $a1_{R1}$, $a2_{R1}$로 정의되었다면, 테이블 정보를 유지하는 메타테이블 _sysClasses에는 { (name_:"R1"),...,(nattrs_:2), ..., (clsAttributes_:{$a1_{R1}$, $a2_{R1}$}), ...} 형태의 튜플(=R1)이 생성되며, 두 필드 $a1_{R1}$, $a2_{R1}$는 사용자 필드 정보를 유지하는 메타테이블 _sysAttributes_에 { (name_:"$a1_{R1}$"), ..., (cls_:R1), ...}, { (name_:"$a2_{R1}$"), ..., (cls_:R1), }로 생성된다. 이 예에서 'R1' 테이블의 clsAttribute_가 {$a1_{R1}$,$a2_{R1}$}을 가지고 있음을 알 수 있다(이후, 설명을 단순화하기 위해, 튜플의 필드 값은 "튜플ID.필드명" 형태로 표기하도록 한다. 즉, '사용자 테이블 R1'을 표현하는 튜플이 R1이므로, 이 튜플의 이름 필드값은 R1.name_으로 표기).

또한, 시스템 카타로그는 상호연관성을 표현하는 중요한 특성 중에 하나인 다중성(Multiplicity)으로 clsAttributes_와 cls_ 필드에 대해, 1:0..n으로 정의하고 있다. 앞의 예에서 테이블 R1이 2개의 필드($a1_{R1}$와 $a2_{R1}$)를 가지는 것과 같이, 한 테이블은 여러 필드들을 갖도록 구성되기 때문이다. 특별하게, 다중성이 1:0인 경우는 아직 테이블의 구체적인 구조(필드들의 명세)가 정의되지 않았음을 의미하고 단지 테이블 틀(template)만 생성되었음을 의미한다.

이와 같이 다중성은 테이블들간의 대응관계를 나타내는데 주요한 요소이다. 본 논문에서는 다중성이 0인 경우는 NIL, 1인 경우는 해당 값, 2이상인 경우는 {값1, 값2, ...} 형태로 표기하도록 한다. 따라서, 테이블 R1만 생성된 앞의 예에서, _sysClasses = { R1 }[2], R1.clsAttribute_ = { $a1_{R1}$, $a2_{R1}$ }로 나타낼 수 있다. 만일, 아직 필드가 정의되지 않은 R2가 존재한다면, _sysClasses = { R1, R2 }가 되며, R1.clsAttribute_ = NIL이 될 것이다.

시스템 카타로그에 정의된 연관성, 다중성 등은 사용자 테이블에도 그대로 적용되며, 다음 절에서는 Tachyon 모델이 제공하는 주요 이슈들을 시스템 카타로그와 연계하여 보다 자세히 설명한다.

4.1 데이터 타입과 튜플 구성

데이터베이스에서 사용자 테이블은 각 필드에 고유 값을 명세한 튜플들의 집합으로 이루어진다. 따라서, 데이터 타입과 연산자의 제공은 데이터 모델의 가장 기본적인 요소이다. 현재, Tachyon이 제공하는 데이터 타입은 표 4.1과 같으며, 기본형과 함께 통신응용에서 주로 사용하는 이진형, 포인터형,

2) 다른 표현으로 R1 ∈ _sysClasses

IPv4/IPv6 주소형[3] 등을 추가로 지원한다. 이와 같이, 응용 위주의 데이터 타입과 관련 처리 함수의 지원은 응용 코드의 복잡도를 줄이고, 질의 실행 시점에 해당 타입의 시멘틱을 최적화 등에 활용함으로써 개발자에게 보다 융통성있는 처리 환경을 제공할 수 있다[24]. 특히, 사용자 자신만의 고유 형을 정의하여 사용할 수도 있다(4.8절에서 설명).

표 4.1 데이터 타입

(괄호 안의 숫자는 byte 크기)

기본형	수치형	int(4), smallint(2)
	범위형	range(1,2,4)
	실수형	float(4), double(8)
	문자/문자열형	char(1), varchar(n)
이진형		byte(n), bit(1,2,4)
포인터형		oid_ref(8), oid_set(8)
복합형		Array of 기본형
IPv4 주소형		cidr(5),inet(5),macaddr(6)

사용자는 이들을 사용하여 튜플의 필드들이 가질 수 있는 상태(값)의 범위와 한계를 정의한다. 이때, 정의된 필드에 대한 모든 정보는 _sysAttribute에 유지된다. 예를 들어, _sysAttribute의 type_에는 사용자가 정의한 필드의 타입 정보(int라면 DB_INT)를, size_는 필드의 물리적인 크기(int라면 4)를 갖는다. 복합형인 array는 기본형을 반복하여 정의하고, 첨자(index)를 이용하여 임의 위치의 데이터에 접근할 수 있도록 한다. 이외에도, 통신운용 환경에서 자주 사용하는 데이터 타입(data type)들인 이진형과 IPv4 주소형을 제공하고 있다. IPv4 타입을 지원함으로써, 통신 응용 프로그램은 IPv4 데이터를 별도의 타입 변환 없이 데이터베이스에 바로 저장할 수 있으며, 또한 데이터베이스에서 추출된 IPv4 데이터들 역시 별도의 변환 없이 그대로 사용할 수 있다. 만일, IPv4 타입을 지원하지 않는다면, IPv4 데이터는 데이터베이스에 표현 가능한 데이터(문자열 같은)로 변환한 후 저장되며, 다시 추출된 문자열을 IPv4 데이터로 변환한 다음에야 사용할 수 있다. 이러한 변환 과정은 처리하는 데이터량에 비례하여, 연산에 대한 오버헤드로 심각한 병목현상[25]을 발생시킬 수 있다. 추가로, IPv4와 관련하여 자주 사용되는 각종 함수들[4]의 제공으로, 응용과 MMDBMS간의 연동 횟수를 줄이고 응용 프로그램의 복잡도를 크게 줄일 수 있다는 장점이 있다.

이러한 필드의 모임으로 구성되는 튜플은 Tachyon이 자동으로 추가하는 selfOID, inverseOID 필드를 포함하여 다음과 같은 구조가 된다.

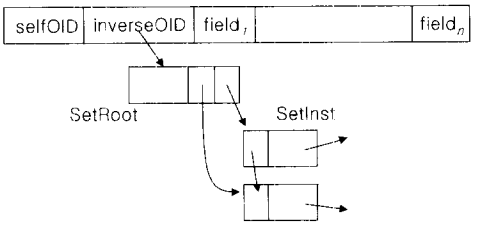

그림 4.2 튜플의 구조 및 OID_SET 필드의 구조

selfOID 필드는 튜플 자신을 유일하게 구분하기 위한 정보를 갖는다. 이 정보에는 튜플이 존재하는 메모리 주소를 함께 포함한다. 즉, 어떤 튜플 t1이 위치하고 있는 메모리 주소가 addr이라면, t1.selfOID = <addr,serial>이 된다. 이때, addr은 물리적인 메모리 주소는 아니며, 튜플이 존재하는 블록 번호(블록의 시작 주소는 재구동시 변경될 수 있으므로)와 블록의 시작에서의 상대 위치 값인 오프셋으로 이루어진 논리적인 주소이다. 따라서, 시스템이 재구동된 후에도, addr내의 블록 번호를 통해 새로운 블록 주소와 오프셋으로 계산한 결과(=물리적인 메모리 주소)를 사용해 해당 튜플에 직접 접근(direct access)이 가능하다. 또한, serial은 특정 addr에 대해 유일한 값으로 생성된다. 그 결과, <addr,serial>로 생성된 식별자는 시스템에서 유일하게 된다. 이에 따라 t1의 삭제 후에 동일한 주소 addr에 t2가 삽입된다 하여도 t2와 t1을 구분할 수 있어(t1.selfOID.serial ≠ t2.selfOID.serial), 사용자로부터 잘못된 식별자를 사용한 부적절한 접근(예를 들면, 이미 삭제된 식별자로 튜플에 접근하려는 것과 같은)을 방지할 수 있다. 이는 [21]등이 식별자를 단순히 빠른 튜플 검색과 같은 제한적인 용도로 사용하는 것과는 달리, 사용자가 식별자를 데이터로 다룰 수 있게 하기 위해 반드시 필요하다.

inverseOID는 OID_REF/SET을 사용하여 양방향이 아닌 관계성이 설정될 때, 자신을 지칭하는 다른 모든 튜플의 식별자를 가진다. 이후, 해당 튜플이 삭제된 경우 참조하는 다른 모든 튜플을 추적하여, 해당 관계성을 해제하는데 사용된다(4.4절에서 설명). 이때, inverseOID 필드는 다수의 식별자를 포함하여야 한다. 우리는 임의 개수로의 확장 가능

3) IPv6 주소형 추가는 현재 진행중

4) IPv4의 경우, network(), broadcast(), host(), masklen(), netmask() 등

한 필드를 정의하기 위한 메커니즘을 위해, 메타 클래스로 SetRoot와 SetInst를 정의하였다. SetRoot는 OID_SET 필드에 유지될 첫번째/마지막 요소와 해당 필드로의 접근에 대한 동시성 제어를 수행한다. 각 SetInst는 OID_SET 필드가 유지해야 하는 다른 튜플의 식별자를 포함한다. 이러한 구현은 OID_SET 필드의 구현에도 그대로 적용된다.

4.2 식별자를 사용한 테이블들간의 관계성 정의

테이블간의 관계성이라 함은 각 테이블에 소속된 튜플들간의 가능한 대응 관계를 의미하며, 관계는 다양성과 추적 방향성의 정의를 통해 실체화된다. 보다 구체적으로 관계는 표 4.1에서 보인 포인터형 필드의 선언으로 성립된다.

관계의 다중성은 OID_REF/SET 타입을 사용하여 1:1, 1:n, m:1, n:m과 같이 대응되는 대상 튜플군의 범주를 결정하며, 여기에서 1:1은 상호 대응되는 튜플이 반드시 한 개뿐임을 명시하는 것이다. 방향성은 양방향성과 단방향성이 있으며, 양방향성이라 함은inverse 필드를 사용하여 두 튜플이 서로 상대 튜플의 식별자를 가짐으로써, 상호 직접 접근할 수 있음을 의미한다. Tachyon은 [5, 6, 18]들과는 달리, SQL에서 직접 OID에 대한 inverse를 명세하고, 엔진에서 직접 처리되도록 확장되었다.

보다 구체적으로 설명하기 위해, 테이블 R1, R2에 대해, 1:n의 다중성과 양방향성을 갖도록 정의한 다음 예를 살펴보자.

```
create class R1 (
  a1R1 OID_SET R2
);
create class R2 (
  a1R2 OID_REF inverse R1.a1R1
);
```

위 정의에서 R1내 $a1_{R1}$과 R2의 $a1_{R2}$는 1:n의 관계성을 표현하며, 이는 곧 R1내의 어떤 튜플 $t1_{R1}$은 R2의 다수 튜플과 상호 연관성을 가질 수 있음을 의미한다. 또한, inverse 키워드는 R1에서 R2로의 접근($t1_{R1}$∈R1이라 할 때, $t1_{R1}.a1_{R1}$으로)시, R2의 $t1_{R2}$가 자신을 지칭하는 $t1_{R1}$을 가리키도록 명시한 것이다. 이를 통해 R2에서 R1으로의 참조도 가능하다.

이러한 시멘틱을 유지하여야 하는 시스템 카타로그내 상태를 살펴보자. 테이블 R1, R2는 메타테이블 _sysClasses의 튜플로, 이들의 필드들은 메타테이블 _sysAttribute에 유지된다. 즉, _sysClasses = { R1, R2 }, _sysAttribute = { $a1_{R1}$, $a1_{R2}$ }가 존재하며, 각각, R1.clsAttributes_ = { $a1_{R1}$ }, R2.clsAttributes_ = { $a1_{R2}$ }가 될 것이다. 이 단계에 대해, $a1_{R1}$.cls_ = R1, $a1_{R2}$.cls_ = R2가 된다.

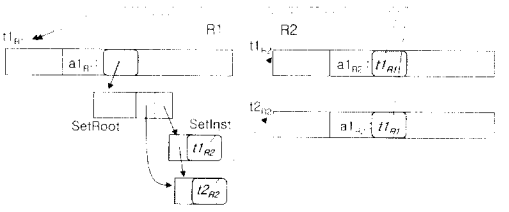

그림 4.3 양방향성 관계성(1:n) 구현

이때, 관계성 정의에서, $a1_{R1}$은 다수 R2 튜플들에 접근할 수 있도록 정의되었고, 이에 따라 $a1_{R1}$.pointCls_ = R2, $a1_{R1}$.type_ = OID_SET으로 설정될 것이다. 또한, 필드 $a1_{R1}$는 $a1_{R1}$의 inverse로 정의하였는데, $a1_{R1}$에 대해, 대응되는 inverse 필드와 테이블은 시스템 카타로그에서 $a1_{R2}$.pointCls_ = R1, $a1_{R2}$.attInverse_ = $a1_{R1}$을 통해 파악할 수 있다. 이러한 시스템 상태는 관계성의 무결성 보장을 위해서도 사용된다. 예를 들어, $t1_{R1}.a1_{R1}$ = $t1_{R1}$.selfOID와 같은 설정을 하고자 한다면, $t1_{R1}$∉ $a1_{R1}$.pointCls_ = { R2 }이므로, 이를 허용하지 않는다.

이와 같이, Tachyon에서는 포인터형 필드를 통해 다른 튜플에 직접 접근하는 방식을 사용하기 때문에, 비정형화된 데이터 처리에 보다 빠르고 안정된 시스템 운용을 보장할 수 있다. 사용자는 SQL 질의에서 포인터형 필드를 사용해, 다른 튜플에 접근하기 위해 참조 연산자(->)를 사용하여 질의에 표현하고 대상 데이터를 추출할 수 있다. 다음은 R1과 관계성이 설정되어 있는 모든 R2 튜플의 $a2_{R2}$, $a3_{R2}$ 필드값을 추출하기 위한 질의이다.[5]

질의1: select $a1_{R1}$->$a2_{R2}$, $a1_{R1}$ >$a3_{R2}$ from R1;

현재, 대부분의 관계형 MMDBMS는 식별자 개념을 제공하고 있지만, 매우 제한적인 용도로만 사용하고 있다. 일부 시스템[21]은 보다 진보된 방식으로, 식별자를 사용한 관계성 정의를 지원하고 있지만, 조건식에서 다른 튜플에 대한 식의 평가에 사용할 뿐, 실제 데이터의 추출은 허용하지 않고 있다. 즉, select 질의의 where절에서만 명시가 가능하여, 질의1과 같은 형태의 질의 명세를 제공하지

5) R2의 모든 튜플을 추출하는 질의(select $a2_{R1}$, $a3_{R1}$ from R2 ;)와는 다르다.

못하고 있다. Tachyon은 이들과 달리 추가로, 다양한 다중성과 방향성 정의, 그리고 참조 연산자를 제공하고 있어, SQL 형태의 사용자 질의를 직관적으로 표현할 수 있다.

4.3 관계성에 의한 데이터들의 항해

앞 절에서 포인터 타입을 통해 관계성 정의와 참조 연산자의 사용에 대해 설명하였다. 실제적으로 관계성을 통한 추적은, 관계된 테이블간의 죠인을 의미하는 것으로, 정의된 관계성에 따라 반복적으로 적용 가능하다. 일반적으로, MMDBMS는 고성능의 데이터 처리를 목적으로 하기 때문에, 예측 가능하지 않은 수행시간이 소요되는 다중 테이블 죠인을 지원하지 않는다. 일부 시스템은 인덱스가 정의되어 있는 경우에 한해 두 테이블간 죠인을 허용하기도 한다. 보다 진보된 형태의 [5, 6]들도 메모리 버퍼 내에 존재하지 않는 튜플의 가능성으로, 연산의 완료시간은 예측가능하지 않다는 제약이 있다. 반면에 Tachyon에서는 모든 데이터가 반드시 주기억장치에 상주할 수 있도록 독립적인 엔진과 데이터 모델을 제공하고 있다. 이에 따라, 데이터의 메모리 주소를 포함하는 식별자로 항상 대상 튜플에 직접 접근하는 방법을 사용하기 때문에, 제한 없는 다중 테이블 죠인과 예측 가능한 완료 시간을 보장할 수 있다. 보다 일반적인 예를 보이기 위해, 추가로 R2, R3간의 관계성이 다음과 같이 정의되었다고 가정하면,

```
create class R2 (
   ...
   a4R2 OID_SET R3
);
create class R3 (
   aiR3 OID_REF inverse R2.a4R2
);
```

R1과 R2는 1:n, R2와 R3는 1:n의 다중성을 갖게 될 것이다. 이때, 그림 4.4에서 $t1_{R1}.a1_{R1}$ = { $t1_{R2}$, $t2_{R2}$, $t3_{R2}$ }이므로, $t1_{R1}.a1_{R1}$을 사용한 항해를 통해, $t1_{R1}$과 연관된 $t1_{R2}$, $t2_{R2}$, $t3_{R2}$ 에로의 접근은 실제 $a1_{R1}$를 키로 한 테이블 R1, R2의 죠인 결과와 동일함을 알 수 있다. 또한, 보다 일반화된 예인 세 테이블의 죠인으로 R1에서 R2를 거쳐 R3의 데이터를 추출하는 질의를 고려해 보자.

질의2: select $a0_{R1}$, $a1_{R1}$->$a4_{R2}$->ai_{R3} from R1;

이와 같이, 참조 연산자로 기술된 경로식(path expression)을 포함한 질의가 수행되기 위해서는 질의의 파싱을 통해 각 테이블에 대한 유일한 접근 경로를 추출하고, 조건의 평가 대상 튜플의 변경에 따라 경로의 재설정, 초기화 등을 수행하는 항해 커서가 필요하다(커서의 세부 구조와 운영 방식에 대한 구체적인 설명은 [26]을 참조). 본 논문에서는 질의 파싱을 거쳐 커서가 구축된 후의 과정인 항해 과정만을 개괄적으로 설명한다.

먼저, 1) R1의 튜플 $t1_{R1}$이 R1의 current OID로 설정되고, $t1_{R1}.a1_{R1}$에 따라 R2의 current OID가 $t1_{R2}$로, 다시 $t1_{R2}.a4_{R2}$에 의해 R3의 current OID가 $t1_{R3}$로 설정된다. 2) 현재 상태에서 R1, R3의 각 current OID가 지칭하는 튜플에서 명시한 필드값(즉, $t1_{R1}.a0_{R1}$, $t1_{R3}.ai_{R3}$)을 추출한다. 다음으로, 3) R2의 $t1_{R2}.a4_{R2}$에서 R3로의 접근이 더 이상 존재하지 않으므로, $t1_{R2}.a4_{R2}$의 이전 필드인 $t1_{R1}.a1_{R1}$에 대해, 재설정이 요구된다. 그 결과, R2의 current OID는 $t2_{R2}$가 되며, 이에 따라 R3의 current OID 역시 $t2_{R3}$로 변경된다. 4) 이 상태에서 결과 필드값 ($t1_{R1}.a0_{R1}$, $t2_{R3}.ai_{R3}$)이 추출된다. 5) 다시, $t2_{R2}.a4_{R2}$에서 R3로의 또 다른 접근(=$t3_{R3}$)이 존재하므로, R3의 current OID를 $t3_{R3}$로 설정한 후 값의 추출 과정이 계속 이루어진다.

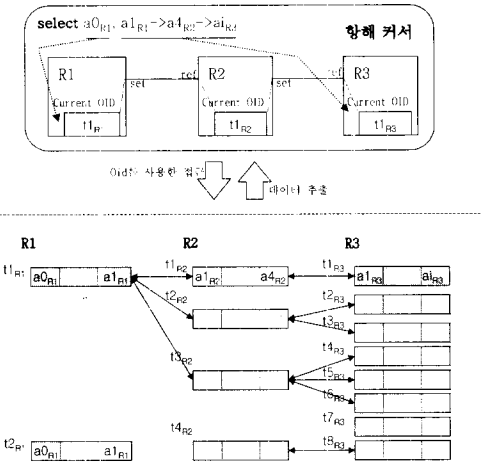

그림 4.4 OID를 사용한 다른 테이블 참조

이러한 체계는 죠인 처리에 있어 주요 병목 현상을 초래하는 메모리 복사 없이, 식별자의 참조만을 통해 죠인 결과를 얻는데 사용된다. 이러한 항해를 통해 추출된 결과는 { ($t1_{R1}.a0_{R1}$,$t1_{R3}.ai_{R3}$), ($t1_{R1}.a0_{R1}$,$t2_{R3}.ai_{R3}$), ($t1_{R1}.a0_{R1}$,$t3_{R3}.ai_{R3}$), ($t1_{R1}.a0_{R1}$, $t4_{R3}.ai_{R3}$), ($t1_{R1}.a0_{R1}$, $t5_{R3}.ai_{R3}$), ($t1_{R1}.a0_{R1}$,$t6_{R3}.ai_{R3}$) }로, 결국 세 테이블의 죠인 결과와 같다. 질의2에서 사

용된 참조연산자(->)는 범용 OODBMS와는 달리, 단순 포인터(OID_REF) 뿐만 아니라 포인터를 갖는 set(OID_SET)에도 적용될 수 있도록 확장되었다. 즉, 질의에서OID_REF에 대한 참조연산자 적용은 해당 튜플에 대한 단순 메모리 접근인 반면에, OID_SET에 대한 적용은 두 테이블간의 죠인으로 직접 확장되어 개발자가 별도의 처리 루틴 없이도 죠인 결과를 얻을 수 있다. 이는 set이 검색될 경우, 별도의 부가 루틴(부질의 혹은 관련 API)를 적용하여 다시 검색하여야 하는 시스템[17,18, 27]과 차별화 될 수 있는 특징중의 하나이다.

4.4 관계성의 방향성과 참조 무결성

예에서, R1에서 R2로의 관계성(=$a1_{R1}$을 이용)은 inverse 특성을 갖도록 정의되었다. Tachyon SQL에서 inverse의 명시는 어떤 $t1_{R1}$∈R1과 $t1_{R2}$∈R2를 볼 때, $t1_{R1}.a1_{R1}$ ∋ $t1_{R2}$.selfOID이면, 반드시 $t1_{R2}.a1_{R2}$ = $t1_{R1}$.selfOID임을 항상 보장한다. 즉, Tachyon에서는 자신 튜플에서 다른 튜플로의 관계를 설정하면, 그 역 방향으로의 관계는 자동으로 설정된다는 것이다. 따라서, 튜플 $t1_{R1}$은 $a1_{R1}$으로 $t1_{R2}$에 접근할 수 있을 뿐만 아니라, 역으로 $t1_{R2}$에서 $a1_{R2}$을 사용하여 자신을 가리키는 $t1_{R1}$의 접근도 가능하다(추가로 성능 관점에서 inverse 특성은 중요하게 고려될 수 있다. 이에 대한 설명은 5장에서 논의). 즉, 양방향성이 명시된경우, 사용자는 한 방향의 관계성만을 설정하며, 이에 대응하는 역방향 관계성은 Tachyon에 의해 자동으로 설정된다. 이와 같이, 양방향 참조는 $t1_{R1}$내 $a1_{R1}$과 $t1_{R2}$의 $a1_{R2}$와 같이, 서로 관련된 튜플의 식별자를 필드가 직접 갖도록 구현된다. 따라서, 한 튜플이 삭제된 경우 중요한 문제를 야기할 수 있다. 예를 들어, 그림 4.4에서 $t1_{R2}$가 삭제되었다면, 이를 지칭하는 $t1_{R1}.a1_{R1}$과 $t1_{R3}.a1_{R3}$는 이미 존재하지 않는 튜플의 식별자를 갖게 될 것이다. Tachyon은 이러한 상태가 발생하지 않도록 $t1_{R2}$의 삭제 후, $t1_{R1}.a1_{R1}$과 $t1_{R3}.a1_{R3}$내에 존재하는 $t1_{R2}$ 식별자를 함께 삭제함으로써, 모든 양방향 관계성에 대한 참조 무결성을 항상 보장한다. 현재, 객체지향 언어의 템플릿(template) 등을 사용한 밀결합(tightly-coupled)으로 참조 무결성을 지원하는[5,6]들과는 달리, Tachyon은 데이터 모델 수준에서 제공하고 있다. 따라서, 객체지향 언어이외에 ODBC/JDBC 등의 사용시에도 동일한 수준의 무결성을 보장할 수 있다는 장점이 있다.

보다 구체적으로, 시스템 카타로그에서 $a1_{R2}$∈_SysAttribute, $a1_{R2}$.name_ = "$a1_{R2}$"에 대해, R2의 $a1_{R2}$필드와 R1의 $a1_{R1}$ 필드에 대해, $a1_{R2}$.attInverse_ = $a1_{R1}$와 같이 설정될 것이다. 따라서, Tachyon은 $t1_{R2}$ 삭제시, 시스템 카타로그를 참조하여 $t1_{R2}$내 OID_REF타입의 필드 $a1_{R2}$의 대상 inverse 필드에서 $t1_{R2}$.selfOID를 삭제하는 연산을 자동적으로 수행한다. Tachyon은 이를 통해, $t1_{R2}$의 삭제 시점에 대응되는 $t1_{R1}.a1_{R1}$과의 일관성을 유지할 수 있다.

Tachyon에서는 문맥에 따라 관계성을 단방향으로 정의할 수 있다. 즉, 어떤 튜플은 이 관계성을 통해 참조 받기만 할 뿐, 이에 대한 역방향 참조는 필요하지 않다는 것이다. 구현시에 주의할 점은, 참조받은 튜플이 삭제될 경우, 해당 튜플을 가리키는 모든 튜플에서의 참조 무결성이 역시 보장되어야 한다는 것이다. 이를 실현하기 위해, 모든 튜플은 inverse 필드를 갖지 않는 관계성 설정의 경우, 자신을 가리키는 튜플의 식별자를 4.1절에서 설명한 inverseOID 필드에 추가한다. 예를 들어, 관계성이 create class R3 (..., $a2_{R3}$ OID_REF R4, ...);로만 정의하고, 튜플 $t8_{R3}$∈R3, $t1_{R4}$∈R4에 대해 $t8_{R3}.a2_{R3}$= $t1_{R4}$.selfOID로 관계성을 설정하였다면, 내부적으로 $t1_{R4}$.inverseOID += $t8_{R3}$.selfOID가 설정된다. 이후, $t1_{R4}$이 삭제된다면, $t1_{R4}$.inverseOID를 통해 $t8_{R3}$를 찾고, $t8_{R3}.a2_{R3}$ -= $t1_{R4}$.selfOID을 적용하여 일관성을 유지한다.

4.5 균일한 처리 시간 보장

죠인은 단일 테이블이 아닌 여러 테이블을 대상으로, 미리 정의한 관계성에 따라 각기 다른 튜플에 존재하는 데이터를 추출하는 연산이며, 예측 가능하지 않은 처리시간과 많은 메모리 공간이 소요된다[4]. 일반적인 다중 테이블 죠인 처리 과정을 살펴보자. 만일, 세 테이블 R1,R2,R5의 죠인은 먼저, R5와 R1을 통합한 새로운 테이블 R5_R1를 만들고, 다시 R5_R1과 R2를 다시 통합하는 방식이 사용되고 있다. 임시 테이블 R5_R1 뿐만 아니라 최종 결과 테이블 R5_R1_R2의 생성 과정에서 과중한 메모리 복사가 발생하고 이 작업은 전체 시스템에 병목 현상을 일으키는 주요한 요인이 된다. 이러한 오버헤드의 해결 방법을 갖지 못한 일부 MMDBMS는 죠인 자체를 허용하지 않거나, 일부에서는 인덱스를 가진 두 테이블의 경우에 한해서만 죠인을 제공하기도 한다. Tachyon에서는 관계성의 항해를 확장하여, 부가의 처리 시간과 메모리 공간 없이 가능한 죠인 방법을 제공한다.

Tachyon에서 죠인 처리 과정을 설명하기 위해, 앞 절의 예에 추가로 create class R1 (..., $a2_{R1}$ OID_SET R5, ...);를 새로이 정의하였다면, R1은 $a2_{R1}$을 통해 R5로, $a1_{R1}$을 통해 R2로의 관계성이 정의된 것이고, 죠인 질의는 참조 연산자를 통해 다음과 같은 형태로 명세 될 것이다.

질의3: select $a2_{R1}$ >$a1_{R5}$, $a2_{R1}$ >$a2_{R2}$ from R1;

이때, 질의가 적용될 테이블 관계가 그림 4.5와 같이 $t2_{R1}$∈R1에 대해 $t2_{R1}.a2_{R1}$ = { $t1_{R5}$, $t2_{R5}$ }⊂R5, $t2_{R1}.a1_{R1}$ = { $t4_{R2}$, $t5_{R2}$, $t6_{R2}$ }⊂R2라고 하자.

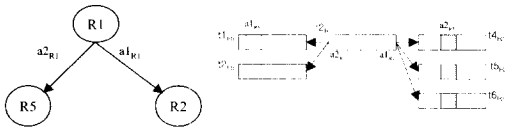

그림 4.5 참조 트리와 튜플들간의 연관

죠인의 첫번째 단계에서는 먼저, 질의에 참조 연산자를 사용한 경로를 분석하여, 기본 테이블(=R1)을 루트로 하는 참조 트리를 형성하게 된다. 다음으로, 참조 트리의 루트 테이블에서 시작하는 경로인 검색 그룹(search group)을 추출한다($a2_{R1}$과 $a1_{R1}$). 단일, 검색 그룹내에서 데이터 추출은 4.4절의 항해 방식을 그대로 사용한다. 만일, 참조 트리에 다수개의 검색 그룹이 존재한다면, 참조 그룹별로 추출된 각 데이터는 또다른 검색 그룹으로부터 추출된 모든 데이터와의 조합이 된다. 즉, 질의 3에 대한 죠인 연산은 결국 $a2_{R1}$과 $a1_{R1}$이 가진 R5,R2 튜플에 대한 카티잔(catesian product) 곱[6]으로, 먼저, $t1_{R5}$∈$t2.a2_{R1}$에 대해, 검색 그룹 $t2.a1_{R1}$내 모든 튜플 $t4_{R2}$, $t5_{R2}$, $t6_{R2}$로의 항해를 통한추출 후에, 검색 그룹 $t2.a2_{R1}$의 잔여 튜플 $t2_{R5}$에 대해 다시 $t4_{R2}$, $t5_{R2}$, $t6_{R2}$로 항해하도록 함으로써 죠인이 이루어진다. 그 결과는 { ($t1_{R5}.a1_{R5}$, $t4_{R2}.a2_{R2}$), ($t1_{R5}.a1_{R5}$, $t5_{R2}.a2_{R2}$), ($t1_{R5}.a1_{R5}$, $t6_{R2}.a2_{R2}$), ($t2_{R5}.a1_{R5}$, $t4_{R2}.a2_{R2}$), ($t1_{R5}.a1_{R5}$, $t5_{R2}.a2_{R2}$), ($t1_{R5}.a1_{R5}$, $t6_{R2}.a2_{R2}$) } 과 같은 6개의 튜플이 될 것이다. 구현의 관점에서 임의개의 검색 그룹에 대해 카티잔 곱 연산을 하기 위해서는 재귀적(recursive) 호출로써 구현된다. 재귀적 호출로 인한 일부 시스템 부담이 증가하지만, 튜플의 물리적인 복사를 사용한 죠인에 비해 매우 적고, 특히 결과 튜플의 개수가 작을수록 그 차이는 더욱 커진다[19].

Tachyon은 다중 테이블에 대해, 테이블의 개수에 제한이 없고 부가의 임시 테이블 생성이나 인덱스 선언에 대한 제약 없이도 죠인이 가능하다. 이는 해당 튜플을 물리적으로 통합하지 않고, 관계성을 표현하는 필드내 튜플 포인터 항해와 검색 그룹의 추출과 재귀적인 재설정 방식을 사용하기 때문이다. 이러한 특성은 타 MMDBMS와 비교하여 두드러진 장점중의 하나이다. 물론, OO MMDBMS인 [5, 6]들은 튜플 포인터를 사용한 죠인을 제공하지만, 이들은 디스크내의 데이터베이스에 대한 다량의 메모리 버퍼 개념으로 제공되기 때문에 기본적으로 디스크 접근을 전제하고 있다. 즉, 이들 시스템은 메모리 버퍼내에 존재하지 않는 튜플들에 대해서는 해당 페이지를 메모리 버퍼에 적재한 후 접근하여야 한다는 측면에서 일정한 죠인 완료 시간을 보장해 주지 못한다.

4.6 테이블 및 질의 상속

테이블상속은 부모 테이블의 모든 필드와 이후 적용될 연산들을 자식 테이블에 적용함을 의미한다. 이후, 자식 테이블은 자신의 고유한 필드를 추가로 정의하게 된다. 즉, create class R6 under R1 (...); 으로 명세하였다면, R6는 자신의 필드 뿐만 아니라 R1에서 정의한 필드와 R1이 다른 테이블과 관련성을 명시하기 위해 정의한 식별자 필드들도 포함된다. 한 예로, R6의 어떤 튜플 $t1_{R6}$는 R1에서 상속받은 필드 $t1_{R6}.a1_{R1}$, $t1_{R6}.a2_{R1}$을 통해, 자신이 직접 정의하지 않은 R2, R5 튜플과의 관계성을 정의할 수 있다. 시스템 카타로그에서 테이블간의 상속 관계는 _sysClasses의 필드inhChildren_과 inhParent_를 사용하여 유지한다. R6는 R1의 자식 테이블이기 때문에, R1.inhChildren_ = { R6 }가 된다. 반면에, R6는 R1이 부모 테이블이기 때문에 R6.inhParent_ = R1[7]이며, 자식 테이블을 갖지 않기 때문에 R6.inhChildren_ = NIL이 될 것이다. 이 정보는 R1에 질의 적용 시, R1 뿐만 아니라 R1의 모든 자식테이블(=R6)에도 동일하게 적용하는데 사용된다. 즉, 질의는 R1의 구조를 갖는 모든 테이블에 적용되어 결과를 추출한다. 이를 위해, R1의 필드와 R1의 자식 테이블에서 상속받은 R1 필드의 메타 정보(튜플 선두에서 상대적인 위치 등 질의 처리시 사용하는 모든 정보)는 모두 동일해야 한다. Tachyon은 테이블의 상속 계층에 따라 상속 받은

6) $a2_{R1}$ X$a1_{R1}$ = {$t1_{R5}$, $t2_{R5}$} X{$t4_{R2}$, $t5_{R2}$, $t6_{R2}$}

7) Tachyon은 단일 상속만을 지원하므로, 하나의 부모 테이블만을 가질 수 있다.

필드는 반드시 자식 테이블의 고유 필드 앞에 위치하도록 보장한다. 그 결과, 부모 테이블내 필드와 자식 테이블에서 상속받은 필드의 물리적인 상대 위치 정보는 항상 동일하게 설정되며, 이에 따라 부모 테이블에 적용된 질의는 별도의 처리 없이 자식 테이블에도 그대로 적용될 수 있다. 질의 관점에서, 질의에 명시적인 제약이 없다면 질의는 모든 자식 테이블에 자동 상속되므로, 만일 select * from R1; 이 주어졌다면, .R1.inhChildren_ = {R6}이므로, select * from only R1 ∪ select * from R6; 과 동일한 의미이다. 여기에서 only는 질의를 자식 테이블에 상속하지 않음을 의미한다. 만일, R6에 또 다른 자식 테이블들이 존재한다면, 이들에게도 같은 방식으로 적용된다.

앞의 예에서는 R6(R1의 자식 테이블)에서 R2, R5로의 관계성 정의를 살펴 보았다. 좀 더 복잡한 예로써, $t1_{R6}.a1_{R1}$에 대해 inverse 필드인 R2의 $a1_{R2}$에 대해 살펴보자. R2의 정의에서 $a1_{R2}$는 R1 튜플과의 관계성을 정의한 것이지만, 상속의 정의에 따라 R1에서 상속된 자식 테이블 R6 튜플과의 관계성도 정의 가능하다. 즉, $t5_{R2}.a1_{R2}$에 $t2_{R6}$.selfOID를 설정하는 시점에, $t2_{R6} \in$R1 뿐만 아니라 $t2_{R6} \in$R1.inhChildren_={R6}인 경우에도 허용된다. 역으로, $t2_{R6}.a1_{R1}$에 $t5_{R2}$.selfOID를 설정하는 시점에는 R6.inhParent_가 사용된다. 단, $t5_{R2}.a1_{R2}$가 R6 튜플을 지칭한다 하여도, R1에서 상속받은 필드외에 R6의 고유 필드에는 접근할 수 없다.

이러한 특성은 새로운 테이블이 기존의 테이블의 자식 테이블로 추가되어도, 이미 구축된 질의에 어떤 변화도 없이 모든 자식테이블이 질의 대상에 포함됨을 의미한다. 이러한 체계는 단순히 구조 상속만을 지원하는 [21]들과는 달리, 지속적인 스키마 진화(schema evolution)와 같은 변환에 쉽게 적응할 수 있다는 큰 장점을 제공한다.

4.7 다중 테이블간 인덱스 공유

MMDBMS는 빠른 데이터 접근을 보장하기 위해 인덱스(index) 체계를 제공한다. 일반적으로 인덱스는 하나의 테이블을 대상으로 하여, 명시된 필드의 값에 따라 테이블내 모든 튜플들에 대해 정렬된 리스트를 유지한다. 이후, 튜플 검색시, 이를 활용하여 빠른 접근 성능을 보장한다. 추가로, 필드에 UNIQUE 특성을 명시하여, 해당 필드에 대해 동일한 값을 갖는 튜플의 삽입을 제한할 수도 있다.

테이블 R1에 대해, 인덱스 I1, I2을 각각 create index I1 on R1($a0_{R1}$, $a3_{R1}$), create index I2 on R1($a0_{R1}$) UNIQUE 로 정의하였다면, 시스템 카타로그에는 _sysIndices = { I1, I2 }, I1.name_ = 'I1', I1.nattrs_ = 2, I1.ownerAtts_ = { $a0_{R1}$, $a3_{R1}$ }와 I2.name_ = 'I2', I2.nattrs_ = 1, I2.ownerAtts_ = { $a0_{R1}$ }, I2.idxProperty_ = { UNIQUE }로 유지된다. 물론, 인덱스에 사용된 필드 $a0_{R1} \in$_sysAttributes 에 대해, $a0_{R1}$.idxs_ = { I1, I2 }로 됨을 앞에서 설명하였다. 이들 정보는 최적의 질의 처리 방법을 찾는 최적화단계에서 인덱스를 선택하는데 활용되고, 이후 실행 시점에서는 튜플의 해당 필드 값이 변경되었을 경우 관련된 인덱스를 확인하고 변경사항을 인덱스에 재적용하기 위해 사용된다.

타 MMDBMS와는 달리, Tachyon은 테이블의 상속 계층에 따라, 부모 테이블의 기본 인덱스(primary index)를 자식 테이블에 상속할 수 있다. 인덱스를 상속한다 함은 하위의 테이블들이 부모테이블과 하나의 인덱스를 공유함으로써, 상속 계층(inheritance hierarchy)내 여러 테이블을 대상으로 질의할 경우, 한 질의에서 하나의 인덱스로 여러 테이블내 튜플을 추출할 수 있어 매우 효과적이다. 만일, 여러 테이블간 인덱스 공유를 허용하지 않는다면, 각각의 테이블에 별도의 질의를 적용한 후, 그 결과를 Union하여야 한다. 물론, 질의 상속을 통해 단일 질의로 결과를 추출하는 방법도 있지만, 각 테이블마다 별개의 인덱스를 선택하고 적용하는데 소요되는 부가적인 오버헤드를 추가로 갖게 된다. 또한 여러 테이블에 대한 특정 필드의 유일성 보장에도 인덱스 공유는 유용하게 사용될 수 있다.

기본 인덱스는 별도의 정의없이 모든 자식 테이블에 자동 상속된다. 따라서, R1에 I2_sysIndices를 기본 인덱스로 정의되었다면, 시스템 카타로그상에 R1.clsIndices_ += { I2 } 뿐만 아니라, R6.clsIndices_ += { I2 }가 묵시적으로 반영된다. 이는 R6 튜플이 변경될 경우, 인덱스 I2에도 그 내용을 반영하여야 하기 때문에 필요하다. 물론, I2.class_ = {R1, R6 }가 된다. 또한, 이를 통해, R1 혹은 R6 튜플에 대해, I2.ownerAtts_ 필드(=$a0_{R1}$)에 대한 모든 변경 연산에는 반드시 I2에 그 결과를 반영하도록 구성되었기 때문에, 여러 테이블에 대한 특정 필드값의 유일성 보장도 가능하다. 단, 기본 인덱스가 소속된 테이블 R1을 명시적으로 구분하기 위해, 시스템 카타로그는I2.ownerCls_ = R1을 추가로 갖는다.

4.8 사용자 정의 타입 지원

사용자정의 형(user defined data type, UDT)은 객체지향 특성의 일부로서, 사용자가Tachyon의 기본 타입들을 조합하여 Tachyon이 제공하지 않는 새로운 타입(type)을 정의하여 사용할 수 있다. 예를 들어, 인터페이스 이름과 네트워크 주소를 갖는 새로운 타입을 정의하고자 한다면, create type U1 (ifname varchar(10), ip_addr cidr);과 같이 정의할 수 있다. 이후, U1은 Tachyon의 타입으로 인정되며, 이를 위해, UDT 관련 시스템 카타로그인_sysUType과 _sysElements에 _sysUTypes={U1}, U1.utyElements={$e1_{U1}$,$e2_{U1}$}, $e1_{U1}$.name_="ifname", $e2_{U1}$.name_="ip_addr"로 등록된다.

이후, create class R1 (..., $a3_{R1}$ U1);와 같이 선언되었다면, 테이블 관련하여, R1.clsAttributes_ = { ..., $a3_{R1}$ }, $a3_{R1}$.type = DB_UTYPE, $a3_{R1}$.attUtypes_ = U1로 설정된다. 이와 같은 확장은 사용자의 응용 환경에 따라 보다 융통성있는 데이터 표현 수단을 제공하기 위해 추가하였다.

구현의 관점에서, 튜플의 삽입으로 실체화될 때, UDT의 각 요소(elements)들은 튜플의 필드로 직접 대응된다. 즉, R1의 튜플내에 U1의 요소들이 직접 추가되어, R1.clsAttributes_∪{$e1_{U1}$,$e2_{U1}$}이 된다. 실제 실행 시점에 식별자의 주소 계산에서 오는 오버헤드 없이 바로 접근할 수 있도록 하기 위해 적용하였다. 하지만, 이러한 내부 구현은 사용자에게 숨겨지며, 사용자는 질의에서 UDT 연산자(.)를 사용하여 select $a3_{R1}$.ifname, $a3_{R1}$.ip_addr from R1; 와 같이 명세한다.

V. 질의 처리를 위한 고려 사항

질의 처리란 시스템 카타로그에 따라, 사용자 요구를 파악하고, 그에 따라 데이터베이스에 반영하는 일련의 작업을 의미한다. 이 장에서는 시스템 카타로그 관점에서 질의 처리 성능과 최적화에 관련된 이슈들을 살펴보자.

5.1 중첩 질의 작성

튜플에서 OID_SET/REF 필드는 참조 무결성 유지를 위해, 관련 있는 상대 튜플의 식별자를 가져야 한다. 예를 들어, 그림 4.4와 같이 관계성을 정의하기 위해서는 먼저 $t1_{R1}$, $t1_{R2}$, $t2_{R2}$, $t3_{R2}$를 데이터베이스에각각 삽입하고, 다음으로 $t1_{R1}.a1_{R1}$= {$t1_{R2}$.selfOID}, $t1_{R1}.a1_{R1}$+={$t2_{R2}$.selfOID}, $t1_{R1}.a1_{R1}$ +={$t3_{R2}$.selfOID}를 순서대로 적용함으로써, $t1_{R1}$과 $t1_{R2}$, $t2_{R2}$, $t3_{R2}$간의 관계가 설정된다.

이 경우, 모두 네 번의 삽입(insert)과 $t1_{R1}$에 대한 세 번의 변경이 발생한다. 이러한 처리 중에는 시스템 카타로그에 대해 동일한 정보를 얻기 위해 중복 접근되고 있다. 시스템 카타로그는 모든 사용자가 공유하는 정보이므로, 이에 대한 잦은 접근은 성능 저하 요인이 된다. 또한, 실시간 처리 시스템에서 클라이언트와 서버간의 지속적인 연동으로 인한 네트워크의 잦은 사용은 성능의 병목현상을 초래한다. Tachyon에서는 시스템 카타로그에 대한 중복된 접근을 방지하고, 한번의 네트워크 연동으로 튜플들간의 관계를 설정하기 위해, 중첩된 질의를 허용하고 있다. 즉, 앞에서 설명한 관계는 insert into R1 values (... , { insert into R2 values (...),insert into R2 values (...), insert into R2 values (...) });와 같이 질의 내부에 또 다른 질의를 포함하여 하나의 질의로서 수행할 수 있도록 확장되었다.

또한, $t1_{R1}$과 R2에 존재하는 모든 튜플들과 새로운 관계성을 정의하고자 할 때에도, 사용자 클라이언트 프로그램의 호스트 변수인 buf로 모든 R2 튜플의 식별자를 읽어오고, 다시 $t1_{R1}.a1_{R1}$+=buf와 같은 변경문을 통해 설정할 수도 있지만, 중복된 시스템 카타로그 접근을 줄이고, 사용자 응용 프로그램과 Tachyon과의 대량 데이터(식별자 같은) 전송 오버헤드를 줄이기 위해, Tachyon SQL에서는 insert into R1 values (... , select oid from R2, ...);와 같이 필드에 값이 아닌 질의를 직접 제시할 수 있도록 확장되었다. 이러한 형태의 질의 사용으로 보다 빠른 성능을 달성할 수 있다.

5.2 질의 최적화 관련

실시간 처리 시스템에서는 질의 처리는 조건식을 평가하여, 이에 만족하는 아주 일부 튜플에 대한 접근으로 이루어진다는 특징이 있다. 또한, OID_REF/SET을 포함한 튜플 관련 조건식의 평가에 따라, 많은 식의 평가가 생략될 수 있다. 이러한 점을 고려하여, Tachyon에서는 다음과 같은 우선 순위에 따라 식을 평가한다.

먼저, Tachyon에서는 사용자 질의에 대해 인덱스를 적용할 수 있는지 여부를 검증한다. 일반적으로, MMDBMS를 사용하는 응용은 빠른 처리가 요구되며, 응용의 특성상 접근하는 튜플의 수가 매우

적다. 이와 같이 전체 튜플에서 일부 튜플에 제한적으로 접근하는 가장 효율적인 방법은 인덱스이다. 특히, 질의 트리에서 루트 테이블에 대한 인덱스의 적용이 가능하다면 가장 먼저 이를 선택하고, 나머지 다른 식은 인덱스 검색 결과에 대한 필터로 사용한다.

두번째로, OID_SET/REF로 연결된 여러 테이블의 조인 연산에서, 인덱스의 적용이 가능하지 않다면, 관계성의 다중성을 고려한 최적화를 수행한다. 설계 의미상, OID_SET 필드를 포함한 튜플은 다수의 다른 튜플들과 관련성을 갖게 된다. 이때, OID_SET 필드를 포함한 튜플에 대한 식의 평가가 성공하지 않는 한, OID_SET을 통해 지칭한 다른 튜플에 대한 식의 평가는 불필요하다. 따라서, 이와 같이 OID_SET 필드를 포함한 튜플내 필드식이 우선되도록 평가 순서를 조정한다.

세번째로, 자식 테이블을 갖는 테이블에 대한 질의는 자동으로 상속되며, 동일한 질의가 하위의 여러 테이블에 적용될 것이다. 각 테이블에 따라 고유의 최적화 전략을 정의할 수 있지만, 상속 관계가 커짐에 따라 이에 비례해 최적화 오버헤드도 증가하게 된다. 상속 구조에서 기본 인덱스(primary index)는 하위의 모든 테이블을 포함한 인덱스이다. 따라서, 식의 평가에 기본 인덱스를 사용할 수 있다면, 테이블 각각에 접근하기 위한 초기화 등의 부가적인 작업을 생략할 수 있다. 특히, 검색 대상이 되는 튜플이 많지 않다는 점에서 각 테이블에서의 최적화 오버헤드 감소는 매우 큰 효과가 있다. 단, 질의 상속이 아닌 단일 테이블 검색의 경우엔 해당 테이블 자신만의 고유 인덱스가 기본 인덱스에 비해 우선한다. 이는 하나의 테이블에 대해 구성된 인덱스가 여러 테이블에 대해 구성된 인덱스에 비해 빠른 검색 효율을 보장하기 때문이다. 또한, OID_REF 필드이지만 양방향성으로, OID_SET inverse로 선언되었다면 OID_SET필드를 포함한 튜플에 대한 식의 평가를 먼저 수행하도록 조정한다.

네번째로, 이러한 최적화 수단(인덱스 선정, 조건식의 순서 변경 등)은 모두 실행 시점에 이루어지기 때문에, 실행 시스템의 일부 자원을 사용한다. 통신 환경에서의 MMDBMS 응용은 초기 단계에 스키마와 질의 패턴이 고정되므로, 이러한 최적화 수단은 실행 시점이 아닌 컴파일 시점에서 준비되고 결정되어질 수 있다. 이를 위해서, 컴파일 단계에서 Tachyon의 시스템 카타로그에 직접 접근할 수 있는 API를 제공하고 있다. 또한, 일련의 최적화 수단을 사용자가 직접 지정할 수 있는 SQL 체계를 확장하여 제공하고 있다. 이러한 정책은 표준은 아니지만, 사용자가 자신의 데이터를 가장 잘 최적화할 수 있다는 관점에서 제공하고 있다.

VI. 결 론

주기억장치 데이터베이스 시스템은 디스크가 아닌 주기억장치를 주요 저장 매체로 사용하므로, 고속의 데이터 처리가 요구되는 다양한 응용을 효과적으로 지원한다. 특히, 주기억장치는 특정 위치에 직접 접근이 가능하다는 특성이 있어, 데이터들간의 상호 참조를 데이터의 주소를 포함하는 식별자를 사용하여 정의하고, 이 들의 항해를 통해 고성능의 질의 처리가 가능하다.

본 논문에서는 고속의 데이터 처리가 필요한 다양한 응용 환경에 적용할 있는 데이터 모델로써, 식별자를 사용한 다양한 관계성 정의와 이들에 대해 보장하는 일관성을 실제 시스템 카타로그를 사용하여 설명하였다. 세부적인 내용으로는 1) 포인터를 사용한 테이블간의 관계성 정의, 2) 관계성에 의한 데이터들의 항해, 3) 포인터 연산에 대한 일관성 지원, 4) 균일한 처리 시간 보장, 5) 객체지향 특성 지원, 6) 다중 테이블간 인덱스 공유에 대해 설명하였다. 기존의 연구 결과로서, 여러 데이터 모델이 제안된 바 있으나, Tachyon 모델과 같이 복잡하고 점진적으로 변화하는 응용 환경에 맞는 데이터 모델에 대해서는 아직까지 제시되지 않고 있다.

현재, Tachyon 모델은 IPv6 데이터를 표현하고 처리하기 위한 여러 정의들을 확장하고 있으며, 추가로 다수 네트워크 자원 관리를 위한 하부 저장 시스템으로 Tachyon을 사용하기 위해, XML 형태의 데이터 포맷을 지원하고 MIB 스키마를 Tachyon 모델로 사상하는 연구가 진행중이다. 우리의 모델은 다양한 네트워크 데이터 타입들을 지원하며, 복잡한 형태의 관계성을 쉽게 정의할 수 있다는 점에서, 여러 응용에 매우 효과적으로 적용될 수 있으며 향상된 성능을 보장할 수 있으리라 예상된다.

참 고 문 헌

[1] Jan Lindstrm, Tiina Niklander, Pasi Porkka, and Kimmo Raatikainen, A Distributed Real-Time Main-Memory Databas

e for Telecommunication, Databases in Telecommunications, 1999.
[2] Bran Selic, Jim Rumbaugh, "Using UML for Modeling Complex Real-time System," uml-rt 1998.
[3] H. Garcia-Molina and K. Salem, "Main Memory Database Systems: An Overview," IEEE Trans. On Knowledge and Data Engineering, Vol. 4, No. 6, pp. 509-516, 1992.
[4] Neal Sample et al., "Managing Complex and Varied Data with the IndexFabric," Proc. of the 18th Int'l Conf. On Data Engineering, pp. 492-493, 2002.
[5] V.F. Wolfe, J.J. Prichard, L. DiPippo and J. Black, "Real-Time Database Systems: The RTSORAC Real-Time Object-Oriented Database Prototype," Kluwer Academic Publishers, pp. 279-303, Dec. 1997.
[6] Yong Yuan, "SMOS : A memory- resident Object Store" master thesis, Univ. of Rhode Island, Kingston, RI, 1997.
[7] S. I. Jun et al., "SROS: A Dynamically Scalable Distributed Real-time Operating System for ATM Switching Network," In Proc. IEEE Global Telecommunications Conference, pp. 2918-2923, 1998.
[8] S. W. Kim et al., "Design and Implementation of the Index Manager in the Main Memory DBMS Tachyon," In Proc. Int'l Symp. On Database and Applications (DBA2002), 2002.
[9] 김상욱, 이경태, 김원영, 최완, "주기억장치 DBMS Tachyon을 위한 캐쉬 인덱스 관리자", The 12th Joint Conf. on Comm. and Information(JCCI), April 2002.
[10] S. W. Kim et al., "Design and Implementation of the Concurrency Control Manager in the Main Memory DBMS Tachyon," COMSAC, Aug. 2002(to appear).
[11] P. Bernstein, V. Hadzilacos, and M. Goodman, Concurrency Control and Recovery in Database Systems, Addison Wesley, 1987.
[12] Y. M. Park, M. N. Bae, W. Choi, and B. S. Lee, "Design and Implementation of Object-Relational Queries and Data types for the Object-Relational Real-time DBMS(S)," The 8nd Int'l Conf. on Real-Time Computing and Applications, pp. 153-157, Keio University. Mita Campus Tokyo, Japan, 2002.
[13] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-time Synchronization," IEEE TOC, Vol. 39, No. 9, pp. 1175-1185, 1990.
[14] Abraham Silberschatz et al., Database System Concepts, 3rd Edition, 2001.
[15] Polyhedra Plc., Bloor Research Overview of Polyhedra, White Paper, Polyhedra Plc.
[16] TimesTen Performance Software, In-Memory Data Management Technical White Paper, White paper, TimesTen Performance Software.
[17] ObjectStore, eXcelon Inc., http:// www.exln.com/
[18] UniSQL, KCC, http://www.unisql. com/
[19] Wan Choi et al., "Tachyon: the Object-Relational Real-time DBMS for Telecommunication systems, The 6th Int'l Conf. on Electronics, Information, and Comm. (ICEIC 2002), Ullaanbaatar, Mongolia, July 2002.
[20] Weiyi Meng, Aqueo Kamada, and Yu His Chang, "Transformation of Relation Schemas to Object-Oriented Schemas,: In Proc. on Computer Software and Application, pp. 356-361, 1995.
[21] GigaBASE, Object-Relational Database Management System, http://www.ispras.ru/~knizhnik/ gigabase.html.
[22] Michael Stonebreaker and Paul Brown, "Object-Relational DBMSs: Tracking the Next Great Wave," San Francisco: Morgan Kaufmann Publishers, 1999.
[23] Y. I. Yoon et al., "Scalable Distributed

Real-time Database Management for Switching System," Int'l Switching Symposium(ISS 97), Toronto, Canada, pp. 539-545, Sep. 1997.
[24] K. L. Eddie Law, "XML on LDAP network database,"2000 Canadian Conference on Electrical and Computer Engineering, Vol. 1, pp. 469-473, 2000.
[25] Peter Boncz, Stefan Manegold, Martin Kersten, "Data Architecture Optimized for the New Bottleneck: Memory Access," In Proceeding of VLDB Conference, 1999.
[26] 배명남, 최완, 이동춘, "통신환경에서 비정형적 구조를 갖는 데이터세트의 효과적인 제어 방법", 한국정보처리학회 논문지, 제 9-C권, 1호, pp. 31-38, 2002.
[27] RDM Database Manager, Mbrane Inc., http://www.mbrane.com.

배 명 남(Myung-Nam Bae) 정회원

1991년 2월 : 전북대학교 전산통계학과 졸업
1993년 2월 : 전북대학교 전산통계학과 석사
1998년 2월 : 전북대학교 전산통계학과 박사
1998년 3월~2003년 2월 : 한국전자통신연구원 네트워크연구소 네트워크S/W플랫폼팀
2003 3월~현재 : 한국전자통신연구원 컴퓨터소프트웨어연구소 바이오정보연구팀

<주관심분야> 객체지향 시스템, 데이터베이스 모델, 질의 처리, UML, XML, 바이오 데이터베이스 모델링.

최 완(Wan Choi) 정회원

1981년 2월 : 경북대학교 전자공학과(잔자계산학전공) 졸업
1983년 2월 : 한국과학기술원 전산학과 석사
1988년 : 정보처리기술사(전자계산응용) 자격획득
1983년 ~ 1985년 : 한국과학기술원 전산학과 연구조교
1985년 ~ 현재 : 한국전자통신연구원 컴퓨터소프트웨어연구소 컴퓨터시스템연구부

<주관심분야> 실시간 소프트웨어, DBMS, OS, Compiler