임베디드 DBMS 환경에서의 동기화를 위한 프레임워크†

강원대학교 김상욱* · 오세봉
포디홈네트(주) 손성용 · 이진호

1. 서 론

포스트 PC 시대가 도래함에 따라 PDA(personal digital assistant), 이동 전화(mobile phone), HPC (hand-held PC), PPC(pocket PC) 등의 소형 이동 단말기(mobile device)의 보급이 급속도로 진행되고 있다. 이러한 이동 단말기의 보급은 무선 인터넷 기술과 결합되어 사용자들이 시간과 공간의 제약이 없는 풍부한 정보 공유를 가능하도록 한다. 임베디드 DBMS(embedded database management system)란 이러한 이동 단말기 내의 데이터를 보다 효율적으로 검색하고 저장할 수 있도록 개발된 소형 DBMS이다 [1-5].

임베디드 DBMS가 동작하는 이동 단말기는 그 저장 용량이 매우 작으므로, 다수의 사용자가 대용량의 공유 데이터베이스를 관리하기에는 적합하지 않다. 대신, 임베디드 DBMS 환경에서는 해당 이동 단말기의 사용자가 다수 사용자의 공유 데이터베이스를 관리하는 하나의 서버 DBMS로부터 필요한 데이터를 임베디드 DBMS 내에 다운로드(download) 받아 사용하는 방식을 사용한다. 이 결과, 다운로드 된 데이터는 서버 DBMS와 임베디드 DBMS 양쪽에 모두 저장되므로, 데이터 중복(data replication)이 발생한다.

중복된 데이터는 서버 DBMS 혹은 임베디드 DBMS에서 모두 변경될 수 있다. 이러한 변경을 양쪽 모두에 일관되게 반영하지 않는 경우, 데이터의 비일관성(data inconsistency) 문제로 인하여 응용의 올바른 수행을 보장할 수 없다. 동기화(synchronization)란 이러한 중복된 데이터의 일관성(consistency)을 유지시켜 주기 위한 임베디드 DBMS 환경에서의 핵심 기능이다.

본 논문에서는 임베디드 DBMS 환경을 위한 동기화에 관하여 논의하고자 한다. 물론, 기존의 상용 임베디드 DBMS들은 이러한 동기화 기능을 기본적으로 지원하고 있다[6-9]. 그러나, 이 기능에 관한 내용이 논문의 형태로 발표되지 않아 개발자들이 그 구체적인 이슈 및 지원 방안에 관하여 참조하기가 매우 어렵다. 현재, 포디홈네트(주)와 강원대학교 데이터 및 지식공학 연구실에서는 정보통신부의 지원으로 이동 단말기 및 정보 가전을 위한 임베디드 DBMS를 공동 개발 중에 있다. 본 논문의 주요 목적은 동기화 관리자(synchronization manager)의 개발 결과 중 일부를 유사 분야의 개발자 및 학자들과 공유하기 위한 것이다.

본 논문에서는 임베디드 DBMS 환경에서 동기화를 지원하기 위한 프레임워크를 제안한다. 먼저, 동기화 지원을 위하여 해결해야 하는 주요 이슈들을 지적하고, 이에 대한 해결책으로서 우리 시스템에서 채택한 방법들을 소개한다.

본 논문의 구성은 다음과 같다. 제 2장에서는 관련 연구로서 임베디드 DBMS 환경을 위한 동기화 모델을 소개하고, 충돌의 유형에 관하여 논의한다. 제 3장에서는 본 연구에서 제안하고자 하는 동기화 관리자의 설계 목표를 제시한다. 제 4장에서는 제안하는 동기화 관리자를 위한 자료 구조, 동기화 절차, 충돌 감지, 충돌 해결 전략을 제시한다. 끝으로, 제 5장에서는 본 논문을 요약하고 결론을 내린다.

2. 동기화

본 장에서는 본 연구에 관한 배경 지식으로서 임

† 본 내용은 정보통신부 정보통신연구진흥원의 2002년 선도기술개발 사업(과제명 : 인터넷 정보가전용 내장형 DBMS 개발, 과제번호 2002-S-12)의 위탁과제(과제명 : Embedded DBMS 개발을 위한 핵심 시스템 기술 개발) 연구 결과입니다.
* 종신회원

베디드 DBMS 환경을 위한 동기화 모델과 충돌의 유형에 관하여 설명한다.

2.1 동기화 모델

그림 1은 임베디드 DBMS 환경을 위한 동기화 모델을 나타낸 것이다. 전체 환경은 하나의 서버와 다수의 클라이언트들이 유선 혹은 무선의 네트워크로 연결된 형태를 갖는다. 서버는 응용 분야에 따라 메인프레임 컴퓨터 혹은 워크스테이션을 고려할 수 있으며, 클라이언트는 PDA, HPC, PPC, 이동 전화 등의 이동 단말기와 대응된다.

서버 내에는 모든 클라이언트들이 공유하는 서버 데이터베이스가 저장되며, 이것은 서버 DBMS에 의하여 관리된다. 클라이언트 내에는 이 이동 단말기의 사용자의 관심 대상인 클라이언트 데이터베이스가 저장되며, 이것은 클라이언트에 내장된 임베디드 DBMS에 의하여 관리된다.

클라이언트 내에 저장된 데이터베이스는 사용자에 의하여 생성될 수도 있으나, 서버 데이터베이스 중에서 사용자가 필요한 일부를 다운로드 받음으로써 생성되는 경우가 대부분이다. 이 결과, 다운로드된 데이터는 서버 데이터베이스와 클라이언트 데이터베이스 양쪽에 모두 존재하게 되므로 데이터 중복(data replication)이 발생된다.

일반적으로, 다운로드 후에는 서버와 클라이언트 간의 연결이 끊어진다. 연결이 끊어진 후, 클라이언트 데이터베이스는 임베디드 DBMS에 의하여 변경될 수 있다. 이러한 변경은 클라이언트가 서버와 추후 연결되었을 때, 서버 데이터베이스에 일관되게 반영되어야 한다. 또한, 연결이 끊어진 동안 발생된 서버 데이터베이스 내의 변경 역시 클라이언트 데이터베이스 내에 반영되어야 한다. 동기화 관리자(synchronization manager)란 이러한 중복된 데이터의 일관성(consistency)을 유지시켜 주기 위한 임베디드 DBMS 환경에서의 핵심 컴포넌트이다.

2.2 충 돌

둘 이상의 클라이언트가 동일한 데이터를 서버 데이터베이스로부터 다운로드 한 후, 이를 서로 다른 방식으로 변경할 수 있다. 이 때, 두 변경을 서버 데이터베이스 내에 올바르게 반영하지 못하는 상황이 발생할 수 있는데, 이를 충돌(conflict)이라 한다[10]. 본 논문에서는 이러한 충돌의 유형을 다음과 같이 삽입 충돌, 삭제 충돌, 갱신 충돌의 세 가지로 분류한다.

삽입 충돌(insertion conflict)은 동일한 레코드 R을 서로 다른 두 클라이언트에서 삽입하고자 하는 경우에 발생한다. 즉, 서로 다른 두 이동 단말기에서 R을 자신의 클라이언트 데이터베이스에 각각 삽입한 후, 제각기 동기화 하는 과정에서 발생하는 충돌이다. 별다른 조치 없이 서버 데이터베이스에 그대로 반영하는 경우, 동일한 레코드가 두 개 생성되는 문제가 발생한다.

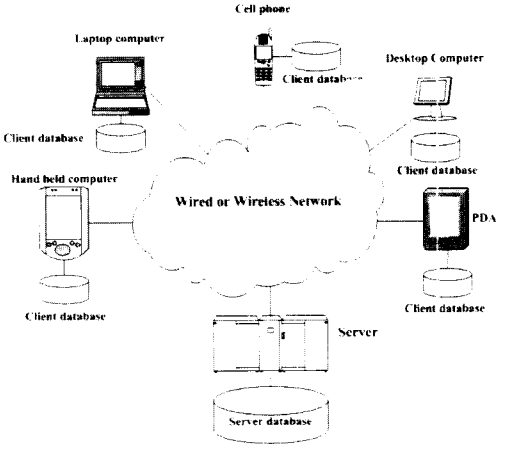


그림 1 임베디드 DBMS 환경

삭제 충돌(deletion conflict)은 동일한 레코드 R에 대하여 한 클라이언트는 삭제하고, 또 다른 클라이언트는 갱신하고자 하는 경우에 발생한다. 즉, R을 클라이언트 데이터베이스 내에 가진 두 이동 단말기 중 하나는 R을 삭제하고, 다른 하나는 R을 갱신한 후, 제각기 동기화 하는 과정에서 발생하는 충돌이다. 별다른 조치 없이 서버 데이터베이스에 그대로 반영하는 경우, 이미 삭제된 레코드를 변경하고자 시도하는 문제가 발생한다.

갱신 충돌(modification conflict)은 동일한 레코드 R을 서로 다른 두 클라이언트에서 다른 방식으로 변경하고자 하는 경우에 발생한다. 즉, R을 클라이언트 데이터베이스 내에 가진 두 이동 단말기에서 R을 각각 변경시킨 후, 제각기 동기화 하는 과정에서 발생하는 충돌이다. 별다른 조치 없이 서버 데이터베이스에 그대로 반영하는 경우, 두 클라이언트 중 하나의 변경이 무시되는 문제가 발생한다.

동기화 관리자는 클라이언트와 서버간의 동기화를 수행할 때, 이와 같은 각 유형의 충돌을 감지해야 하며, 충돌이 감지 된 경우 이를 응용에서 원하는 방식으로 해결해야 한다.

기존의 상용 임베디드 DBMS에서 사용하는 동기화는 크게 타임스탬프(time-stamp)를 이용하는 방식[11][9]과 변경 전의 값(old value)을 이용하는 방식[12]으로 분류된다[1]. 본 논문에서는 이 두 방식의 영문 이니셜을 따서 간략히 TS 방식과 OV 방식으로 칭한다. 그림 2는 두 가지 방식의 동작 원리를 표현한 것이다.

TS 방식은 서버와 클라이언트에 존재하는 동기화 대상이 되는 각 레코드에 타임스탬프 필드를 추가하는 것이다. 타임스탬프는 서버에 존재하는 레코드의 최종 변경 시점을 표현하는 값이다. 클라이언트는 서버로부터 데이터를 다운로드 할 때, 각 레코드와 대응되는 타임스탬프 값을 함께 다운로드받는다. 이 값은 클라이언트의 임베디드 DBMS 내에서 유지되다가 동기화 시에 충돌 감지를 위하여 사용된다.

OV 방식은 클라이언트가 서버로부터 다운로드 한 변경 전의 각 레코드 값을 이용한다. 클라이언트에서 변경이 일어난 후에도 이 값은 클라이언트의 임베디드 DBMS 내에서 그대로 유지되다가, 동기화 시에 충돌 감지를 위하여 사용된다.

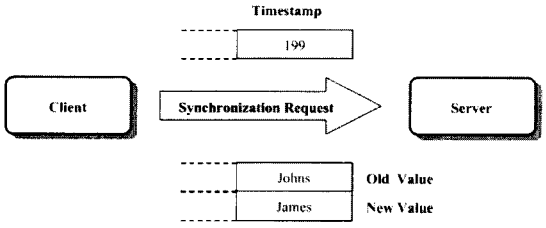


그림 2 TS 방식과 OV 방식의 비교

3. 설계 목표

본 장에서는 본 연구에서 동기화 관리자를 개발할 때 추구하였던 설계 목표를 제시한다.

1) 이러한 상용 임베디드 DBMS에서 사용 중인 동기화 원리는 현재 논문의 형태로 발표된 것이 아니므로, 참고문헌 [11][9][12] 등과 같은 매뉴얼 혹은 브로셔에서 그 개념만 간단히 파악할 수 있었다. 특히, 이러한 참고 문헌에는 전술한 충돌의 유형 중, 갱신 충돌에 주요 초점이 맞추어져 있으며, 삽입 충돌 및 삭제 충돌에 관해서는 언급되어 있지 않다.

(1) 동기화를 위하여 요구되는 클라이언트 내 저장 공간의 오버헤드를 최소화한다. 이동 전화, PDA 등 소형 이동 단말기는 특성상 주기억장치 및 디스크 등의 저장 공간이 매우 작다. 따라서 동기화를 위해서 추가로 요구되는 저장 공간이 지나치게 크다면, 실용성 측면에서 문제가 있다.

(2) 클라이언트와 서버간의 각 연결 시에 전송되는 정보량을 최소화한다. 클라이언트와 서버간의 각 연결시의 정보량이 많은 경우, 서버에서 동기화를 요청하는 각 클라이언트에 관한 별도의 연결 정보를 에 관한 오버헤드가 커질 뿐만 아니라 네트워크 장애로 인한 동기화 실패도 발생할 가능성이 높다. 따라서 클라이언트와 서버간의 각 연결 시에 전송되는 정보량을 줄이는 것은 빠른 동기화를 위하여 매우 중요하다.

(3) 동기화 관리자를 서버 DBMS에 종속되지 않도록 한다. 동기화 관리자가 특정한 상용 서버 DBMS에 종속된다면, 그 외의 상용 서버 DBMS는 서버에서 사용할 수 없거나, 서버 DBMS마다 다른 동기화 관리자를 개발해야 한다. 따라서 서버 DBMS에 종속되지 않도록 함으로써 플랫폼 독립적인 동기화 관리자를 개발할 수 있다.

(4) 충돌에 대한 해결책을 응용에서 결정할 수 있도록 하기 위한 다양한 기능을 제공한다. 충돌에 대한 가장 기본적인 해결책은 충돌을 감지한 클라이언트의 모든 변경 요청을 무시하는 것이다. 그러나 몇몇 응용에서는 충돌을 감지한 클라이언트의 중요도에 따라 그러한 충돌을 묵인하고, 변경 요청을 받아들여야 하는 경우가 발생한다. 따라서 충돌의 감지는 동기화 기능에서 담당하고, 충돌에 대한에 대한 대책은 응용에서 결정하도록 하는 것이 바람직하다.

4. 제안하는 기법

본 장에서는 위의 설계 목표를 고려하여 개발된 동기화 관리자에 관하여 상세히 논의한다. 제 4.1절에서는 사용되는 자료 구조를 제시하고, 제 4.2절에서는 동기화를 위한 절차를 제안한다. 제 4.3절에서는 충돌의 감지 방법을 제안하고, 끝으로 제 4.4절에서는 충돌 감지 후의 해결 정책을 제시한다.

4.1 자료 구조

충돌 감지 방식 중 OV 방식이 TS 방식보다 저장 공간의 오버헤드가 크다. TS 방식은 변경되는 다운

로드 된 각 레코드 당 타임스탬프 필드 하나만을 추가하는 반면, OV 방식은 각 레코드의 모든 필드에 대하여 변경 전 값과 변경 후 값을 모두 저장해야 하기 때문이다. 본 연구에서는 설계 목표 (1)인 저장 공간의 오버헤드 측면을 고려하여 TS를 기반으로 하는 방식을 사용하였다[2].

서버 데이터베이스 내의 모든 레코드들은 원래의 필드들 외에 충돌 감지와 관련된 타임스탬프 필드를 추가한다.

타임스탬프는 레코드가 서버 데이터베이스 내에서 변경된 가장 최근 시점을 의미하는 값이다. 이러한 변경은 클라이언트에 의한 동기화 요청이나 서버에 의한 자체 변경 요청으로 발생하게 된다. 각 타임스탬프 값은 해당 레코드 각각에 대하여 고유의 의미를 갖는다. 이것은 서로 다른 두 레코드가 동일한 타임스탬프 값을 가진다고 하더라도 동일한 시점에 변경이 발생하였다는 의미는 아니라는 것이다.

서버 데이터베이스 내에 새로운 레코드가 삽입이 되는 경우, 그 최소의 타임스탬프 값은 1로 세트된다. 서버 데이터베이스 내에 존재하는 레코드가 갱신되는 경우, 이 레코드의 현재의 타임스탬프 값은 하나 증가된다. 이것은 이 레코드가 새로운 버전이 됨을 의미한다. 서버 데이터베이스에 존재하던 레코드가 삭제되는 경우에는 타임스탬프 값의 변경 없이 해당 레코드를 즉시 삭제한다.

본 연구에서는 타임스탬프를 위한 타입으로서 정수를 사용한다. 정수는 DBMS에서 제공하는 일반 타임스탬프 타입보다 적은 공간을 차지한다. 뿐만 아니라, 별도의 함수 없이도 값을 비교하거나 변경할 수 있으므로 덧셈과 비교 연산과 같은 기본적인 연산만으로도 효율적으로 충돌을 감지할 수 있다. 정수 타입이 가지는 양수의 범위는 1~MAX(4바이트 int인 경우 2^{31}-1)이므로 충분한 수의 변경을 지원할 수 있다. 뿐만 아니라, 서버 내에 요구되는 변경의 수가 MAX를 초과하는 경우에도 타임스탬프 값을 1에서부터 다시 시작하면 되므로, 타임스탬프로 인하여 변경의 횟수가 제한되는 상황은 발생하지 않는다.

클라이언트 데이터베이스 내의 모든 레코드들은 원래 필드들 외에 충돌 감지를 위한 타임스탬프 필드와 변경 연산 추적을 위한 변경 상태 필드를 추가한다.

표 1 서버 내에서 변경에 따른 타임스탬프 값의 변화

변경 연산	삽입	갱신	삭제
Timestamp 값의 변화	1	Timestamp+1	레코드 삭제

클라이언트 데이터베이스 내에 존재하는 레코드의 타임스탬프 값은 삽입, 삭제, 변경 등의 어떠한 변경 연산에도 그대로 유지된다. 이 타임스탬프는 가장 최근의 동기화(동기화가 없는 경우에는 최초의 다운로드) 시점에 참조했던 서버 데이터베이스 내의 해당 레코드의 버전을 나타내는 값이다. 이 값은 다음 동기화 시에 충돌 감지를 위해서 서버 데이터베이스 내의 해당 레코드의 타임스탬프 값과 비교된다.

동기화 시, 충돌 감지를 위하여 사용되는 정보는 클라이언트 데이터베이스 내에서 변경된 레코드들이다. 본 개발에서는 설계 목표 (2)인 클라이언트 서버 간의 동기화를 위한 전송 정보의 최소화를 위하여 충돌 감지 시 클라이언트에서 이들 변경된 레코드만을 서버로 전송하는 전략을 채택한다. 따라서 가장 최근의 동기화(동기화가 없는 경우에는 최초의 다운로드) 시점 이후에 변경된 클라이언트 데이터베이스 내의 레코드들을 별도로 관리해야 한다.

제안하는 기법에서는 이러한 변경 연산의 추적을 위하여 각 레코드 존재하는 변경 상태 필드를 이용한다. 변경 상태 필드는 크기가 1바이트인 CHAR 타입으로서 클라이언트 데이터베이스 내의 해당 레코드에 발생한 변경 연산의 유형을 기록한다. 변경 상태 필드는 표 2에 나타난 바와 같이, 변경 연산이 삽입인 경우에는 'I', 갱신인 경우에는 'U', 삭제인 경우에는 'D', 변경이 발생하지 않은 경우에는 'N'을 갖는다. 동기화 시, 동기화 관리자는 충돌 감지를 위한 정보로서 클라이언트 데이터베이스에서 변경 상태 필드가 'N'이 아닌 레코드들만을 추출한다. 이것은 동기화 관리자가 클라이언트 DBMS가 제공하는 SQL 문을 이용하여 간단하게 처리할 수 있다.

최초 삽입된 레코드가 다음 동기화 이전에 변경되는 경우에는 변경 상태 필드를 'U'로 바꾸지 않고, 'I'로 유지된다. 이것은 다음 동기화 시에 서버 데이터베이스에 이 레코드가 새롭게 삽입되어야 하기 때문이다. 또한, 클라이언트 데이터베이스 내에 레코드 삭제가 발생하는 경우에는 실제로 해당 레코드를 삭

2) 전술한 바와 같이, 이러한 참고 문헌 [11][9][12]에는 동기화의 개념만 간단히 기술되어 있으며, 본 논문에서 제시하는 구체적인 충돌 감지 및 충돌 해결 전략에 관해서는 언급되어 있지 않다.

제하지 않고, 이 변경 상태 필드의 값만을 'D'로 바꾸게 된다. 이것은 다음 동기화 시에 서버 데이터베이스로부터 이 레코드를 삭제하기 위해서는 이 레코드를 클라이언트 데이터베이스로부터 제거해서는 안되기 때문이다. 클라이언트에서의 레코드 검색 시 이러한 레코드는 결과에 나타나서는 안되므로, 질의 처리 시에 이와 같이 변경 상태 필드 값이 'D'인 레코드들은 최종 검색 결과에서 제외시킨다. 최초 삽입된 레코드가 다음 동기화 이전에 삭제되는 경우에는 변경 상태 필드를 'D'로 바꾸지 않고 해당 레코드를 즉시 클라이언트 데이터베이스로부터 삭제한다. 이것은 서버 데이터베이스에 이미 이 레코드가 존재하지 않으므로, 다음 동기화까지 이 레코드에 관한 정보를 유지할 필요가 없기 때문이다.

표 2 클라이언트 데이터베이스의 변경에 따른 변경 상태 값의 변화

변경 연산	삽입	갱신	삭제	없음
변경 상태 필드 값	I	U	D	N

4.2 동기화 절차

본 절에서는 제안하는 기법의 동기화 절차에 관하여 논의한다. 그림 3은 동기화와 연관된 전체 과정을 스케치한 것이다. 이와 같은 동기화 절차는 클라이언트 DBMS 및 서버 DBMS에 대한 트랜잭션의 수행으로 진행된다. 즉, 동기화를 위하여 클라이언트 데이터베이스 및 서버 데이터베이스 내의 레코드들을 액세스할 때, 각각의 DBMS가 지원하는 SQL 문을 이용한다. 이렇게 함으로써 동기화 관리자가 특정 서버 DBMS에 종속되지 않는 플랫폼 독립성을 확보하게 되므로 설계 목표 (3)을 만족시킬 수 있다.

① 클라이언트 데이터베이스 내에서 레코드의 변경이 발생하면, 변경된 레코드의 변경 상태 필드는 표 2와 같이 변경된다. 이와 같은 변경은 클라이언트 데이터베이스 내에서 반복적으로 발생할 수 있다.

② 클라이언트가 서버로 동기화를 요청한다. 먼저, 클라이언트는 자신의 데이터베이스 내에서 서버 데이터베이스에 새롭게 반영되어야 할 레코드들을 선정한다. 이들은 가장 최근의 동기화 (동기화가 없는 경우에는 최초의 다운로드) 이후에 변경된 레코드들로서 표 2의 기준에서와 같이 해당 변경 상태 필드 값이 'N'이 아닌 레코드들이다. 동기화 요청 시에 클라이언트는 두 가지 종류의 리스트를 서버로 전송한다.

그 하나는 위에서 선정된 레코드들의 전체 필드 값들과 타임스탬프 필드 값의 리스트로서 우리는 이를 클라이언트 변경 레코드 리스트라 정의한다. 클라이언트 변경 레코드 리스트는 클라이언트 내의 변경을 서버 데이터베이스 내에 반영하기 위하여 사용된다. 설계 목표 (2)인 클라이언트 서버간의 동기화를 위한 전송 정보의 최소화를 위하여 클라이언트 변경 레코드 리스트에는 변경 상태 필드가 포함시키지 않는다. 클라이언트 데이터베이스 내에서는 변경 상태 필드가 삽입, 삭제, 갱신, 무 변경 등 네 가지 유형을 구분하기 위하여 필요하다. 반면, 클라이언트 변경 레코드 리스트에는 변경되지 않은 레코드는 포함되지 않으므로 삽입, 삭제, 갱신의 세 가지 유형만을 구분하면 된다. 이를 위하여 제안한 기법에서는 타임스탬프 필드를 이용하여 이를 구분한다. 즉, 삽입 시에는 0, 삭제 시에는 음의 값, 갱신 시에는 양의 값을 갖도록 함으로써 이러한 세 가지 유형을 구분할 수 있다.

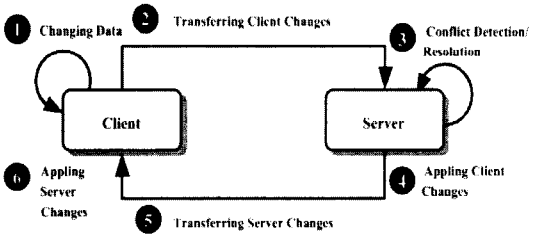


그림 3 동기화 과정

또 다른 하나는 클라이언트 데이터베이스 내의 모든 레코드들의 <기본 키, 타임스탬프> 쌍들의 리스트로서 우리는 이를 전체 레코드 요약 리스트라 정의한다. 이 전체 레코드 요약 리스트는 클라이언트 데이터베이스에 반영해야 할 서버 데이터베이스 내에서 새로운 변경이 발생하였는가의 여부를 판별하기 위하여 사용된다. 전체 레코드 요약 리스트를 이용한 서버 데이터베이스 내의 새로운 변경 검사 방법은 ④에서 자세히 설명한다.

③ 서버는 클라이언트로부터 전송된 클라이언트 변경 레코드 리스트를 이용하여 충돌이 발생하였는지의 여부를 판단한다. 또한, 충돌의 발생을 감지한 경우, 이를 해결한다. 제 4.3절과 제 4.4절에서는 이러

한 충돌 감지 및 해결에 관하여 보다 상세하게 설명한다.

④ 서버는 클라이언트 변경 레코드 리스트를 참고하여 변경된 레코드들을 서버 데이터베이스 내에 반영한다. 즉, 전송 받은 레코드의 타임스탬프 값이 양수인 경우에는 갱신, 음수인 경우에는 삭제, 0인 경우에는 삽입을 서버 데이터베이스에 수행한다. 이와 같이, 클라이언트에서의 변경이 반영된 서버 데이터베이스 내의 레코드들의 타임스탬프 값은 표 1을 기준으로 조정된다.

⑤ 서버는 전체 레코드 요약 리스트를 참고하여 서버 데이터베이스 내에서 그 클라이언트의 가장 최근의 동기화 (동기화가 없는 경우에는 최초의 다운로드) 이후에 변경된 레코드들의 리스트를 작성하는데, 이를 서버 변경 레코드 리스트라 정의한다. 이러한 서버 변경 레코드 리스트는 클라이언트 데이터베이스 내에 반영해야 하는 새로운 변경을 의미하며, 클라이언트로 전송된다. 서버 변경 레코드 리스트는 전술한 클라이언트내의 변경 레코드 리스트와 동일한 구조를 갖는다.

레코드 갱신 : 전체 레코드 요약 리스트 내의 각 <기본 키 PK, 타임스탬프 TS>에 대하여 이 PK 값을 갖는 서버 데이터베이스 내의 레코드 R을 찾아 R의 타임스탬프가 TS와 같은 값을 갖는가를 검사한다. 만일, 같지 않다면 최근의 동기화 이후에 서버 데이터베이스 내에서 해당 레코드에 갱신이 발생한 것이다. 이 경우에는 갱신된 레코드를 서버 변경 레코드 리스트에 포함시킨다. 리스트 내의 이 레코드의 타임스탬프 값은 원래 서버 데이터베이스 내 레코드의 타임스탬프 값인 양수로 그대로 유지한다.

레코드 삽입 : 전체 레코드 요약 리스트 내에 존재하지 않는 기본 키를 갖는 레코드가 서버 데이터베이스 내에 존재하는가를 검사한다. 만일, 이러한 레코드가 존재한다면, 이는 최근의 동기화 이후에 새롭게 삽입된 것이다. 리스트 내의 이 레코드의 타임스탬프 값은 원래 서버 데이터베이스 내 레코드의 타임스탬프 값을 음수로 바꾸어 유지한다. 클라이언트 변경 레코드 리스트와는 달리 음수의 타임스탬프 값을 사용하는 이유는 클라이언트 데이터베이스 내에서 이 레코드가 서버 데이터베이스 내의 어떤 버전과 대응되는가를 추적하는 것이 필요하기 때문이다.

레코드 삭제 : 전체 레코드 요약 리스트 내의 각 <기본 키 PK, 타임스탬프 TS>에 대하여 PK 값을 갖는 레코드가 서버 데이터베이스 내에 존재하는가를 검사한다. 만일, 이러한 레코드가 존재하지 않는다면, 이는 최근의 동기화 이후에 새롭게 삭제된 것이다. 이 경우, 삭제된 레코드를 서버 변경 레코드 리스트에 포함시키며, 리스트 내의 이 레코드의 타임스탬프 값은 0이 된다. 클라이언트 변경 레코드 리스트와는 달리 타임스탬프 값으로 단지 0을 사용하는 이유는 클라이언트 데이터베이스 내에서는 충돌 감지 없이 단지 이 레코드를 삭제만 하면 되기 때문이다.

⑥ 클라이언트는 서버 변경 레코드 리스트를 참고하여 변경된 레코드들을 클라이언트 데이터베이스 내에 반영한다. 즉, 전송 받은 레코드의 타임스탬프 값이 양수인 경우에는 갱신, 음수인 경우에는 삽입, 0인 경우에는 삭제를 클라이언트 데이터베이스에 수행한다. 단, 삽입된 레코드의 타임스탬프 값은 다시 양수로 조정된다.

최종적으로, 클라이언트 데이터베이스 내의 레코드들 중 변경 상태 필드가 'D'인 레코드들을 모두 삭제함으로써 동기화를 종료한다. 이러한 레코드들은 동기화의 성공적인 종료 이후에는 유지할 필요가 없기 때문이다.

4.3 충돌 감지

클라이언트로부터 동기화 요청을 받은 서버는 클라이언트가 전송한 클라이언트 변경 레코드 리스트를 검사하여 충돌의 발생 여부를 검사한다.

첫째, 삽입 충돌의 감지이다. 서버 데이터베이스 내에 동일한 기본 키를 가지는 레코드가 이미 존재하는 경우에는, 서버 DBMS에서 무결성 제약 조건(integrity constraint)이 위배되므로 삽입이 거부된다. 동기화 관리자는 기본 키를 이용하여 삽입 충돌을 감지한다. 즉, 전송된 클라이언트 변경 리스트에서 타임스탬프 값이 0인 레코드의 기본 키와 동일한 기본 키 값을 갖는 레코드가 서버 데이터베이스 내에 존재하는지를 검사함으로써 삽입 충돌을 감지한다.

둘째, 갱신 충돌의 감지이다. 갱신 충돌은 크게 두 가지로 구분된다. 그 하나는 이미 서버 데이터베이스에서 삭제된 레코드를 갱신하려는 경우이다. 또 다른 하나는 클라이언트 A가 다운로드 한 레코드 R을 다른 클라이언트 B가 서버 데이터베이스 내에 갱신한 이후에 클라이언트 A가 갱신된 R을 서버 데이터베이스 내에 반영하려는 경우이다. 전자의 경우, 전송된 클라이언트 변경 리스트에서 타임스탬프 값이 양

수인 레코드의 기본 키 값을 갖는 레코드가 서버 데이터베이스 내에 존재하는지 검사함으로써 갱신 충돌을 감지할 수 있다. 즉, 이러한 레코드가 존재하지 않는 경우, 이 레코드는 이미 삭제된 것이다. 후자의 경우, 전송된 클라이언트 변경 리스트 내 타임스탬프 값이 양수인 레코드의 기본 키 값과 같은 기본 키 값을 갖는 서버 데이터베이스 내 레코드의 타임스탬프 값이 동일한가를 검사함으로써 갱신 충돌을 감지한다. 즉, 이 두 값이 서로 다른 경우, 서버 데이터베이스 내의 레코드는 이미 다른 버전으로 변경된 것으로 간주하는 것이다.

셋째, 삭제 충돌의 감지이다. 클라이언트 A가 다운로드 한 레코드 R을 다른 클라이언트 B가 서버 데이터베이스 내에 갱신한 이후에 클라이언트 A가 R을 클라이언트 데이터베이스로부터 삭제하고, 이를 동기화 작업 시 서버 데이터베이스 내에 반영하려는 경우이다. 이 경우에는 전송된 클라이언트 변경 리스트 내 타임스탬프 값이 음수인 레코드를 먼저 찾고, 이 레코드와 같은 기본 키 값을 갖는 서버 데이터베이스 내 레코드를 찾아 두 레코드의 타임스탬프 필드의 절대값이 동일한가를 검사함으로써 삭제 충돌을 감지한다. 즉, 이 두 값이 서로 다른 경우, 서버 데이터베이스 내의 레코드는 이미 다른 버전으로 변경된 것으로 간주하는 것이다.

4.4 충돌 해결

동기화를 요청한 클라이언트가 충돌을 감지하는 경우, 데이터베이스의 일관성을 완벽하게 보장하는 유일한 방법은 충돌을 유발시킨 변경을 포기하는 것이다. 충돌에도 불구하고 서버 데이터베이스 내에 이러한 변경을 강행하는 경우, 서버 데이터베이스의 일관성에 문제가 발생할 수 있기 때문이다. 따라서 제안된 기법에서 충돌 감지시의 기본적인 해결 방식으로서 변경을 포기하는 방식을 채택한다.

그러나 클라이언트 내에서의 변경을 모두 포기한다는 것은 그동안 수행한 모든 작업을 무효화하는 것을 의미하므로, 사용자의 불만이 커질 수 있다는 문제가 있다. 완벽한 서버 데이터베이스의 일관성을 보장 측면보다는 사용자의 만족도를 높이는 측면을 더 중요시하는 응용 분야도 존재한다. 따라서 제안된 기법에서는 기본 전략 이외에도 변경의 포기 혹은 반영에 관한 결정을 일정 수준의 권한을 가지는 사용자가 선택할 수 있는 방식을 채택한다. 단, 충돌에 대한 대책을 수립하기 위하여 필요한 다양한 정보를 동기화 관리자에서 제공함으로써 응용의 의사 결정을 도울 수 있다. 이러한 전략은 응용이 요구하는 다양한 충돌 해결 방식을 지원할 수 있도록 하기 위한 설계 목표 (4)를 반영한 것이다.

5. 결 론

동기화 기능은 클라이언트 데이터베이스와 서버 데이터베이스에 공존하는 이러한 중복된 데이터의 일관성을 유지시켜 주기 위한 임베디드 DBMS 환경에서의 핵심 기능이다. 기존의 상용 임베디드 DBMS들이 이러한 동기화 기능을 지원하고 있으나 [6][7][8][9], 이 기능을 지원하기 위한 기술적 이슈 및 지원 방안이 발표된 바 없어 임베디드 DBMS 개발자들이 이를 참조하기가 어려운 실정이다. 본 논문에서는 포디홈네트(주)와 강원대학교 데이터 및 지식공학 연구실에서 공동으로 개발 중인 임베디드 DBMS에서 채택하는 동기화 프레임워크에 관하여 논의하였다. 본 논문은 동기화 관리자의 개발 결과 중 일부를 유사 분야의 개발자 및 학자들과 공유하기 위한 것이며, 이를 통하여 향후 임베디드 DBMS 개발자들의 시행 착오를 줄일 수 있으리라 생각된다.

참고문헌

[1] Michael A. Olson, "Selecting and Implementing an Embedded Database System," IEEE Computer Magazine, pp.27-34, Sept. 2000.

[2] Sixto Ortiz, Jr., "Embedded Databases Come out of Hiding," IEEE Computer Magazine, Mar. 2000, pp. 16-19.

[3] 최윤석, "모바일 환경을 위한 초경량 데이터베이스 Oracle9i Lite," 데이터베이스 연구회지, 17권 3호, 2001년 9월.

[4] 김도연, "포스트 PC 시대의 데이터 관리 - IBM INFORMIX CLOUDSCAPE," 데이터베이스 연구회지, 17권 3호, 2001년 9월.

[5] 이상윤, 박순영, 이미영, 김명준, "이동 DBMS의 데이터 동기화 기술 분석," 데이터베이스 연구회지, 17권 3호, 2001년 9월.

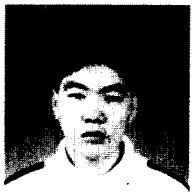
[6] Oracle, Oracle 8i Introduction, Oracle's White Paper

[7] IBM, DB2 Solutions for Mobile Computing, IBM's White Paper
[8] Informix, Informix CloudSync, Informix's White Paper
[9] Sybase, Synchronization Technologies for Mobile and Embedded Computing, A White Paper from Sybase, Inc.
[10] IBM DB2 Sync Server 관리 안내
[11] IBM, DB2 복제 안내 및 참고서
[12] Oracle, Oracle 8i Replication

김 상 욱

1989 서울대학교 컴퓨터공학과 졸업(학사)
1991 한국과학기술원 전산학과 졸업(석사)
1994 한국과학기술원 전산학과 졸업(박사)
1991 Stanford University, Computer Science Department, Summer Student
1994~1995 KAIST 정보전자연구소 위촉연구원
1999~2000 미국 IBM T.J. Watson Research Center, Post Doc.
1995~현재 강원대학교 컴퓨터정보통신공학부 부교수
관심분야:DBMS, 주기억장치 DBMS, 임베디드 DBMS, GIS, 트랜잭션 관리, 데이터 마이닝, 시퀀스 매칭, 바이오 정보공학, 웹 데이터베이스
E-mail:wook@kangwon.ac.kr

오 세 봉



2003 강원대학교 컴퓨터정보통신공학부 졸업 예정
관심분야:DBMS, 임베디드 DBMS, 시퀀스 매칭
E-mail:kw95216020@future.kangwon.ac.kr

손 성 용

1990 한국과학기술원 생산공학과 졸업(학사)
1992 한국과학기술원 정밀공학과 졸업(석사)
1992~1995 LG 소프트웨어
1995~1996 생산기술연구원 연구원 II
2000 미국 The University of Michigan, Mechanical Engineering졸업(박사)
2000~현재 포디홈네트(주) 연구소장
관심분야:홈네트워크, 임베디드 DBMS, 원격관리
E-mail: xtra@4dhome.net

이 진 호

1991 한국과학기술원 학부과정 컴퓨터공학 졸업
1993 포항공과대학교 일반대학원 컴퓨터공학 졸업
1993~2000 포항공과대학교 정보통신연구소 연구원 재직
1995~현재 포항공과대학교 일반대학원 컴퓨터공학과 박사과정 재학 중
2000~현재 포디홈네트(주) 경영기획본부 담당이사
2001~현재 주식회사 알티캐스트 경영기획실 담당이사
E-mail:zino@4dhome.net