테이블의 모든 레코드 삭제의 효율적 수행†

차명훈*, 박준현**, 박영철***
영진전문대학*, 한국컴퓨터통신(주)**, 경북대학교***
mhcha@yeungjin.ac.kr, jhpark@unisql.com, ycpark@knu.ac.kr

An Efficient Scheme of Deleting All Records of a Table

Myung Hoon Cha*, Jun Hyun Park**, Young Chul Park***
Yeungjin Junior College*, KCOM**, Kyungpook National University***

요 약

테이블의 모든 레코드들을 삭제하는 연산과 그 연산을 취소하는 작업은 그 수행에 드는 비용이 매우 크다. 상용 데이타베이스 관리 시스템들은 테이블의 모든 레코드들을 삭제하는 연산을 신속하게 수행하기 위하여 TRUNCATE TABLE문을 제공한다. 그 문을 수행하면 레코드들을 개별적으로 삭제하지 않고 그 화일에 할당된 디스크 공간을 반환하며 그 삭제 연산을 독립된 트랜잭션으로 처리하여 즉시 완료시킨다. 따라서, 그 연산은 일단 실행되고 나면 복귀가 불가능하다. 본 논문은 DELETE문을 WHERE절 없이 사용하는 경우에 대하여 모든 레코드들을 삭제하고자 하는 화일과 동일한 구조를 가지는 하나의 빈(empty) 화일을 생성하고, 기존 화일과 새로이 생성한 화일 간에 화일 설명자의 내용과 디스크 공간을 교체한 후, 새로이 생성한 화일 자체를 삭제함으로써 로깅 부담을 줄이면서 신속히 모든 레코드들을 삭제하고 그 연산을 복귀할 수 있도록 하는 기법을 제시한다.

1. 서 론

데이타베이스 관리 시스템에서 테이블의 모든 레코드들을 삭제하는 연산과 그 연산을 취소하는 작업은 매우 큰 수행 부담을 요구한다.

Oracle, Sybase 등의 상용 데이타베이스 관리 시스템들에서 테이블의 모든 레코드들을 삭제함으로써 빈 테이블만 남아있도록 하기 위해 사용가능한 방법은 세가지로 요약할 수 있다[1, 2, 3].

첫번째 방법은 DROP TABLE문을 사용하여 테이블을 삭제한 후 CREATE TABLE문과 CREATE INDEX문을 이용하여 그 테이블과 그 테이블의 색인들을 다시 생성하는 방법이다. 이 방법에서 DROP TABLE문은 일단 실행되면 복귀가 되지 않으며 그 테이블에 대해 설정해둔 뷰, 저장 프로시저, 트리거, 참조무결성의 유효성에 영향을 미치며 그 테이블과 그 테이블에 연계된 객체들에 대해 지니고 있는 사용자들의 권한 등에 영향을 미치게 되어 선택할 수 없는 방법이다. 두번째 방법은 DELETE문을 WHERE절 없이 사용함으로써 테이블의 구조는 그대로 두고 그 테이블의 모든 레코드들만 삭제하는 방법이다. 이 방법은 상용 데이타베이스 관리 시스템들과 기존 바다-II[4, 5, 6, 7, 8]가 채택한 방법으로써 각 레코드마다 개별적인 삭제를 수행하고 이에 대해 로깅을 수행하므로 상당한 자원을 소모한다. 이 방법의 장점은 그 삭제 연산을 복귀할 수 있다는 것이다. 세번째 방법은 Oracle, Sybase 등의 상용 시스템들이 제시하는 방법으로써 TRUNCATE TABLE문을 사용하여 레코드들을 개별적으로 삭제하지 않고 해당 데이타 화일에 할당된 디스크 공간을 반환하는 방법이다. Oracle과 Sybase는 이 작업동안 그 디스크 공간에 수록된 내용을 로깅하지 않는다고 명시하고 있으며 구체적인 수행 방법에 대해서는 언급하지 않고 있다. 이 방법은 모든 레코드들을 삭제하기 위한 효율적인 방법인 반면, 현재의 트랜잭션을 즉시 완료(commit)하기 때문에 그 문을 일단 실행하면 그 연산을 복귀할 수 없다.

본 논문은 DELETE문을 WHERE절 없이 사용하는 경우에 대하여 그 테이블의 모든 레코드들을 일일이 삭제하지 않고 해당 데이타 화일과 그 화일을 바탕으로 생성된 색인들의 디스크 공간을 반환함으로써 그 테이블의 모든 레코드들을 삭제한 것과 동일한 효과를 도출하면서 이를 수행한 트랜잭션을 즉시 완료시키지 않는 방법을 제시한다.

본 논문의 나머지 부분의 구성은 다음과 같다. 제 2장에서 모든 레코드들을 삭제할 테이블의 데이타 화일과 그 테이블에 생성된 모든 색인들의 색인 화일들의 내용을 비어있는 상태로 만드는 기법을 제시한다. 제 3장에서 본 논문의 결론을 맺는다.

2. 모든 레코드 삭제의 기법

바다-II는 각 화일에 대하여 시스템 내에서 그 화일을 유일하게 식별하기 위한 화일 식별자(File IDentifier : FID)를 부여하고, 그 화일에 대한 제반 정보들의 모임인 화일 설명자(File Descriptor : FD)를 유지하며, 그 화일의 데이타를 저장하는 공간으로서 하나 이상의 익스텐트(extent)를 할당한다. 하나의 익스텐트는 연속한 페이지들의 모임이며, 하나의 페이지는 데이타베이스 공간의 가장 작은 단위로서 디스크와 메모리 사이의 입출력 단위로 사용된다. 바다-II에서 하나의 색인 화일은

†본 논문은 한국과학재단 목적기초연구(과제번호 : 2000-2-51200-002-3) 지원으로 수행되었음.

그 색인이 바탕으로 하는 데이타 화일에 종속된다. 따라서, 색인 화일은 시스템 전체에서 그 색인을 유일하게 식별하는 화일 식별자 외에 그 색인이 기반으로 하는 데이타 화일에서 그 색인을 유일하게 식별하는 색인 번호를 별도로 부여받는다. 색인 화일의 화일 설명자를 특별히 색인 설명자(Index Descriptor : ID)라 한다. 화일 설명자는 <화일 유형, 화일에 할당된 페이지들의 리스트에서 첫번째 페이지와 마지막 페이지의 페이지 식별자들, 화일에 할당된 페이지들의 수, 화일에 있는 레코드들의 수, 화일에 생성된 색인들의 수, 생성된 각 색인에 대해 색인 번호와 그 색인의 색인 설명자를 저장한 레코드의 레코드 식별자>로 구성된다. 색인 설명자는 <색인이 속한 데이타 화일의 화일 식별자, 그 데이타 화일 내의 색인 번호, 색인 화일의 화일 식별자, 기타 색인 관련 정보>로 구성된다. 바다-II는 각 화일에 할당된 디스크 공간을 관리하기 위하여 화일 맵(File Map : FM)을 둔다. 화일 맵의 i번째 엔트리는 화일 식별자 값이 i인 화일에 할당된 익스텐트들의 리스트에서 첫번째 익스텐트의 번호를 가진다.

그림 1은 모든 레코드들을 삭제할 데이타 화일(file_old라 하자)에 대한 모든 레코드들을 삭제하기 이전의 상태를 나타낸다. 그림 1에서 file_old의 화일 식별자 값은 i로 할당되어 있으며, 화일 맵의 i번째 엔트리인 FM_i는 file_old에 할당된 익스텐트들의 리스트의 첫번째 익스텐트의 번호가 a임을 나타낸다. 화일 설명자의 값은 <xx,..,p>이며, xx는 file_old의 제반 정보들을 의미하며 p는 file_old에 생성되어 있는 하나의 색인 index_old에 대한 색인 설명자의 레코드 식별자 값이다. 색인 설명자 ID_p는 레코드 식별자 값이 p인 색인 설명자를 나타내며 그 값인 <i,ss,m>에서 i는 그 색인이 바탕으로 하는 데이타 화일의 화일 식별자 값을 나타내고, ss는 그 색인의 제반 정보를 나타내며, m은 그 색인 화일의 화일 식별자 값을 나타낸다. 화일 맵의 m번째 엔트리인 FM_m은 index_old에 할당된 익스텐트들의 리스트의 첫번째 익스텐트의 번호가 r임을 나타낸다.

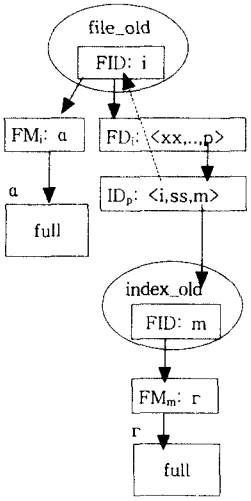

그림 1. 모든 레코드들의 삭제 연산 수행 이전의 상태

데이타 화일 file_old의 모든 레코드들을 삭제하기 위해 본 논문이 제시하는 알고리즘은 다음과 같다. 첫째, file_old의 구조와 동일한 구조를 가지면서 하나의 레코드도 포함하지 않는 비어있는 데이타 화일(file_new)을 생성하고 file_old의 각 색인(index_old)에 대하여 그 색인의 구조와 동일한 구조를 가지면서 하나의 키도 가지지 않는 비어있는 색인(index_new)을 file_new에 생성한다. 둘째, file_new의 각 색인의 색인 설명자에서 그 색인이 속한 데이타 화일의 화일 식별자를 file_new의 화일 식별자에서 file_old의 화일 식별자로 변경한 후 file_old의 <화일 설명자의 내용, 화일 맵 엔트리의 내용>과 file_new의 그것을 교체한다. 셋째, file_new와 새로이 file_new의 색인으로 설정된 색인 화일들을 화일 삭제 방법에 따라 삭제한다.

그림 2는 모든 레코드 삭제의 첫번째 과정을 수행한 후의 모습이다.

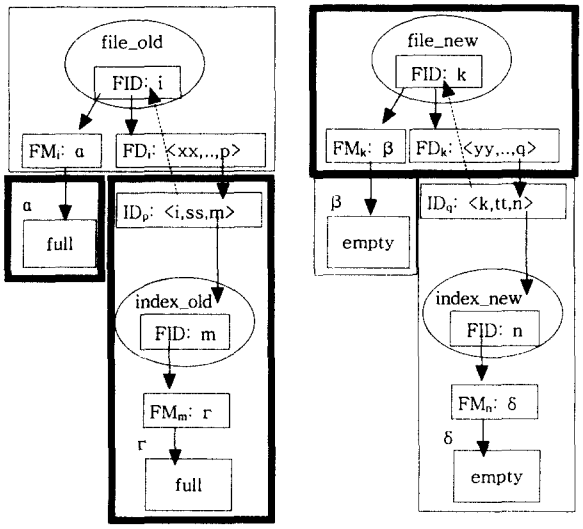

그림 2. 비어있는 새로운 화일의 생성 이후

비어있는 화일 file_new를 생성하는 절차는 다음과 같다. 먼저, 새로운 화일 식별자를 할당한 후, 하나의 익스텐트를 할당하고 그 익스텐트의 번호를 할당한 화일 식별자에 해당하는 화일 맵의 엔트리에 설정한다. 그리고 나서, file_old의 각 색인에 대하여 그 색인의 구조와 동일한 구조를 가지면서 하나의 키도 가지지 않는 비어있는 색인을 file_new에 생성한다. 마지막으로, 이들을 반영하는 화일 설명자를 설정한 후 디스크에 저장한다. 비어있는 색인 화일을 생성하는 과정은 색인 설명자의 저장을 제외하고는 데이타 화일을 생성하는 경우와 동일하다. 바다-II는 비어있는 화일을 생성하더라도 하나의 익스텐트를 할당한다.

그림 3은 모든 레코드 삭제의 두번째 과정으로서 file_old의 화일 맵 엔트리의 내용 및 화일 설명자의 내용을 file_new의 그것들과 교체한 후의 모습이다. 이의 수행 방법은 다음과 같

다. file_new에 새로이 생성된 빈 색인들의 색인 설명자의 그 색인이 속한 데이타 화일의 화일 식별자 값을 file_old의 화일 식별자 값으로 설정한다. file_old의 기존 색인들은 삭제될 화일이므로 이들 색인에 대한 색인 설명자는 변경할 필요가 없다. file_old와 file_new의 화일 설명자들의 내용을 상호 교체하고, file_old의 화일 맵 엔트리 값과 file_new의 화일 맵 엔트리 값을 교체한다.

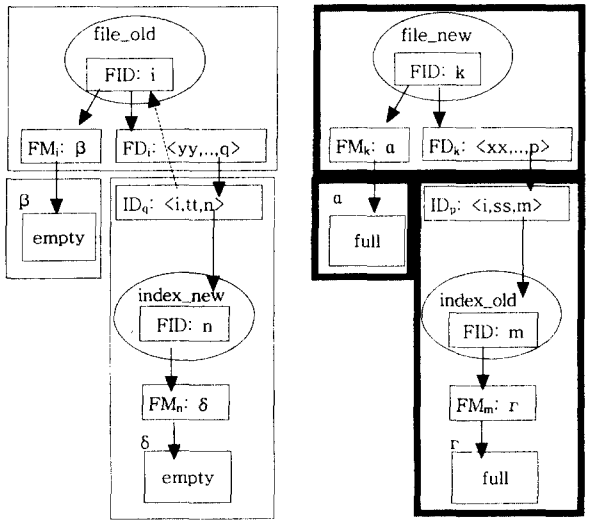

그림 3. 화일 맵 엔트리와 화일 설명자를 교체한 이후

그림 3에서 file_old가 가지는 디스크 공간은 file_new가 가지고 있던 디스크 공간이 되며, file_new가 가지는 디스크 공간은 file_old가 기존에 가지고 있던 디스크 공간이 된다. file_old와 file_new의 화일 식별자들의 값은 변경되지 않고 기존 값들인 i와 k를 유지하기 때문에 화일 맵 엔트리와 화일 설명자의 기존 위치는 변경되지 않으며, 화일 맵 엔트리의 내용과 화일 설명자의 내용들만 변경됨을 알 수 있다.

모든 레코드 삭제의 세번째 과정으로서, 새로이 설정된 file_new에 해당하는 데이타 화일과 그 화일에 종속된 모든 색인 화일들을 삭제한다. 화일 삭제를 즉시 연산과 미처리 연산으로 구분하여 수행하는 알고리즘의 상세 사항들은 [8]에 제시되어 있다.

모든 레코드들을 삭제하는 연산을 수행한 트랜잭션이 복귀할 경우에는 그 트랜잭션이 생성한 로그들에 대한 취소 연산을 수행함으로써 트랜잭션 수행 이전의 상태로 복귀한다[8, 9].

3. 결론

데이블의 모든 레코드들을 삭제하는 연산은 로깅에 따른 부담으로 인하여 트랜잭션의 수행과 복귀에 많은 자원을 소모한다. 본 논문은 모든 레코드들을 삭제하고자 하는 데이타 화일에 대하여 동일한 구조를 가지면서 내용이 없는 빈 데이타 화일을 생성하고 화일 설명자의 내용 등을 교체한 후 새로이 생성한 데이타 화일을 삭제하는 기법을 사용함으로써 로깅 부담을 줄이면서 신속히 모든 레코드들을 삭제하며, 디스크 공간을 반환하는 작업을 미처리 연산으로 수행함으로써 화일 삭제 연산의 복귀에 따른 부담을 최소화하며 그 연산을 복귀시킬 수 있는 알고리즘을 제시하였다.

데이타 화일의 모든 레코드들을 효율적으로 삭제하기 위해 본 논문에서 제시한 방법이 모든 경우에 최적으로 동작하는 것은 아니다. 예를 들어, 데이타 화일에 레코드가 하나밖에 없는 경우에는 그 레코드만 삭제하는 것이 효율적이다. 따라서, 본 논문에서 제시하는 방법을 사용할 경우에 소요되는 비용과 몇 개의 레코드들을 순차적으로 삭제할 경우의 비용을 비교하는 과정이 필요하며, 어떤 특정 수효 이상의 레코드들을 지닌 화일의 모든 레코드들을 삭제하는 경우에 한하여 본 논문에서 제시한 방법을 사용하여야 한다.

참고문헌

[1] SYBASE SQL Server Reference Manual: Volume 1, Sybase, Inc., 1996.

[2] ORACLE 8i Administrator's Guide, Oracle Corporation, 1999.

[3] ORACLE 8i SQL Reference Volume 2, Oracle Corporation, 1999.

[4] S.H. Kim, M.S. Jung, J.H. Park, and Y.C. Park, "A Design and Implementation of Savepoints and Partial Rollbacks considering Transaction Isolation Levels of SQL2," Proceedings of 6th International Conference on DASFAA, pp. 303~312, April 1999.

[5] 박영철, 박준현, 김준, 이진수, "허상 트랜잭션: 퍼지검사점에 따른 회복의 문제점," 정보과학회논문지(B), 제25권3호, pp. 426~443, 1998년 3월.

[6] 박준현, 박영철, "상태미반영 로그 레코드: 퍼지 검사점과 트랜잭션의 비동기 수행에 따른 회복의 문제점," 정보과학회논문지(B), 제26권 5호, pp. 621~638, 1999년 5월.

[7] 박준현, 박영철, "바다-II에서 저장점과 부분 복귀의 설계와 구현," 정보과학회논문지(B), 제27권 9호, pp. 578~600, 2000년 9월.

[8] 박준현, 박영철, "데이타베이스 화일의 삭제를 위한 미처리 연산의 효율적 수행 기법," 정보과학회논문지:데이타베이스, 게재예정, 2001년.

[9] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging," ACM TODS, Vol. 17, No. 1, pp. 94~162, March 1992.