주기억 데이터베이스 시스템 질의 엔진의 설계 및 구현*

이경식[0], 김경창
홍익대학교 컴퓨터공학과
{ksyi[0], kckim}@cs.hongik.ac.kr

The Design and Implementation of Query Engine for Main Memory Database System

Kyung-Shik Yi, Kyung-Chang Kim
Dept. of Computer Engineering, Hongik University

요 약

본 논문에서는 주기억 데이터베이스 시스템을 위한 질의처리 엔진의 설계 및 구현에 대해서 설명하였다. 이를 바탕으로 Embedded SQL 지원을 위한 Pre-compiler 구현 방법, 단순 질의 및 Join 과정에서의 주기억 장치의 효율적인 사용 방법, Cursor, Dynamic SQL 처리 방법에 대해 소개하였다.

1. 서 론

본 논문은 주기억 데이터베이스 시스템의 질의처리 엔진에 대한 설계 및 구현방법을 제시한다. 본 논문의 구성은 2장에서 주기억 데이터베이스의 특징을 보고, 3장에서 구현된 질의 처리 엔진의 구성과 설계를 살펴본다. 4장에서는 질의 처리 엔진 구현에 사용된 환경에 대해 살펴보도록 한다. 마지막으로 5장에서는 연구의 결론을 내고, 향후 연구 방향을 생각해보고 마치도록 하겠다.

2. 주기억 데이터 베이스의 특징

2.1 주기억 데이터베이스의 출현 배경

고성능 및 실시간 데이터 처리를 전제로 하는 분야에서는 기존의 디스크 기반 데이터베이스로는 필요한 성능을 얻기에 부족함이 있었다. 따라서 이러한 분야에 사용되는 응용 프로그램들을 기존의 디스크 기반 데이터베이스보다 성능이 뛰어난 주기억 데이터베이스를 사용하는데, 많은 노력이 필요한 주기억 장치에서의 자료저장, 동시성 제어, 회복 기능의 반복 구현을 제거하여 개발비용을 절감하고 보다 안정된 응용프로그램을 만들어내고자 하는 요구에 대응하기 위하여 메모리 상주 데이터베이스 시스템 (Main Memory Database Management System, MMDBMS) [1,2]이 출현하였다.

2.2 주기억 데이터베이스의 구성 요소

이 논문에서는 주기억 데이터베이스 시스템의 구성 요소(그림 1) 중, Interactive SQL 인터페이스, 질의 처리를 위한 Query Engine, Embedded SQL 처리를 위한 Pre-compiler를 중심으로 살펴보고, 이를 논문에서의 구현 범위로 하고자 한다.[3]

* 본 연구는 한국과학재단 특정기초연구 (과제번호: R01-2001-00540) 지원으로 수행되었음.

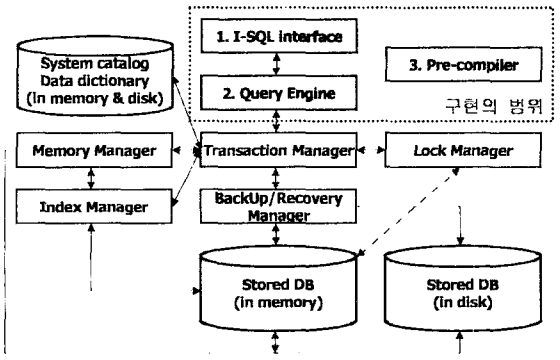

그림 1: 일반적인 주기억 데이터베이스의 구성 요소

3. 질의 처리 엔진 구조

질의 처리 엔진은 크게 SQL 문자열을 입력받아 실행하는 모듈, pre-compiler 모듈, E-SQL 실행을 위한 API의 3가지 요소로 나뉘어 구현되었다. (그림 2)

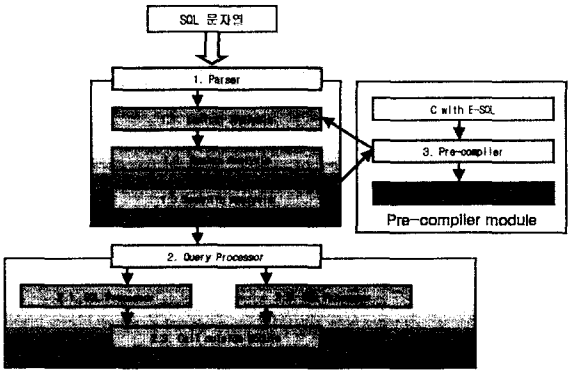

그림 2: 질의 처리 엔진 구성

3.1 질의 처리 엔진의 구성 요소

3.1.1 Parsing 모듈

SQL 문장을 해석하는 모듈이다. 이 모듈은 Lexical analysis, Syntax analysis, Semantic analysis의 3가지 하부 모듈로 이루어져 있다.

Lexical analyzer는 입력받은 SQL 문장을 token으로 변환하는 모듈로서, 입력이 "SELECT DISTINCT S.sname, S.age FROM Sailor S;"일 때, {"SELECT", "DISTINCT", "S", ".", "sname", "S", ".", "age", "FROM", "Sailor", "S"}의 11개 token으로 변환하여 그 결과를 LIST(토큰 리스트) 구조체의 포인터로 반환한다.

Syntax analyzer는 Lexical analysis 과정에서 생성한 token 리스트를 입력으로 받아 Parse tree를 생성하는 모듈이다. token 리스트가 {"SELECT", "DISTINCT", "S", ".", "sname", "S", ".", "age", "FROM", "Sailor", "S"} 인 경우, 이를 그림 3과 같은 tree 구조로 변환한다.

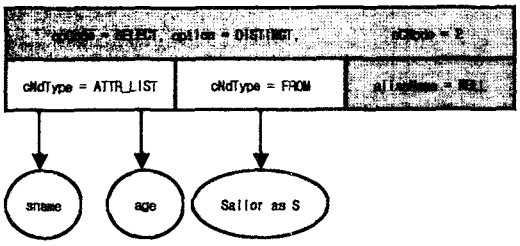

그림 3: Parse tree 생성

Semantic analyzer는 Syntax analysis의 결과인 parse tree를 입력으로 받아 SQL 문장의 의미를 결정하며 문장의 오류를 검사한다.

이러한 3단계의 분석기를 거치는 동안 오류가 발생하지 않으면, 입력으로 들어온 SQL 문장은 QUERY 구조체 포인터의 형태로 변환되어 Query Processor로 전달된다. 이 때, SQL 문장이 Table 생성 및 삭제, 속성 추가 및 삭제, 색인 생성 및 삭제인 경우 DDL Processor 모듈을 호출하게 되고, 튜플 삽입 및 삭제, 변경과 SELECT 질의, Cursor 관련, Dynamic SQL 관련 명령인 경우 DML Processor를 호출한다.

3.1.2 Query Processor

Parser에 의해 해석된 SQL 문장을 실행하는 Module이다. Embedded SQL도 이 Query Processor Module을 이용하여, ARCHYS 주기억 데이터베이스의 API를 호출하게 된다. 이 모듈은 크게 DDL processor, DML processor, Query Optimizer, Call outside module로 구성된다.

DDL processor는 테이블 생성, 삭제, 속성 추가, 삭제, 색인 생성, 삭제의 기능을 수행한다.

DML processor는 튜플 삽입, 삭제, 변경과 SELECT 문장에서 수행되는 Selection, Projection, Join을 처리하는 모듈이다. 또한, E-SQL에서 사용되는 Cursor 지원에 사용되는 DECLARE, OPEN, FETCH, CLOSE 명령, Dynamic SQL 처리를 위한 PREPARE, EXCUTE 명령을 처리한다. Selection은 속성에 대한 색인이 존재하지 않는 경우, 테이블에 임시 색인을 생성하였으며, 속성에 기존 색인이 존재하는 경우, 색인 정보를 이용하여 조건을 만족하는 레코드를 검색한다. Projection의 처리를 위해서는 우선 메모리에 존재하는 테이블을 읽어 들인 후, 불필요한 속성들을 제거하고, 중복된 튜플을 제거한다.[4,5] 구현에서는 정렬 후 중복을 제거하는 방법을 이용한다. 해당 속성에 색인이 존재하지 않는 경우 임시적인(On the fly) 색인을 생성한다. 색인이 존재하는 경우는, 기존의 색인을 이용하여 중복된 튜플을 제거한다. Join의 경우 처음에는 구현이 용이한 Nested loop Join 방식을 이용하였으나 성능 평가 단계에서 성능 저하가 심한 이유로 Sort-Merge 방식으로 대체하였다.

Call outside module은 입력받은 SQL 문장을 주기억 데이터베이스의 각 관리자 모듈을 이용하여 실행한다. atomic, consistency, isolation, durability를 만족시키고, 동시성 제어가 가능하며, 시스템 이상이 생길 경우 데이터를 보호할 수 있도록 트랜잭션 관리자를 통해 메모리 관리자, 색인 관리자를 실행하게 된다. 또한 Selection, Projection, Join 시에 생성하는 임시 저장 테이블을 관리한다.

3.2 Embedded SQL 지원을 위한 Pre-compiler

Embedded SQL로 작성된 응용 프로그램을 실행하기 위해서는 기존의 컴파일러를 사용하여 실행 파일을 만들 수 있는 형태로 변환해 주는 Pre-compiling 단계와, Pre-compiler를 통해서 컴파일 가능한 C 코드로 변환되어 실행 파일이 생성된 후, 실제로 SQL을 실행하여 데이터베이스를 이용할 수 있는 API를 제공하는 단계로 진행된다. Pre-compiler의 입출력 내용은 다음과 같다.

입력: EXEC SQL 등의 키워드로 SQL 문장을 실행하는 코드가 추가되어 있는 C 문법 기반의 파일
출력: C 컴파일러를 이용, 컴파일 가능하게 변환된 파일

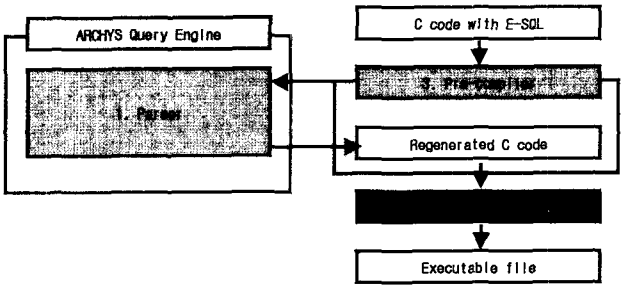

그림 4: Pre-compiler와 Parser

3.3 단순 질의 및 Join 처리

SQL 명령에서 가장 자주 사용되는 문장은 SELECT 명령으로 이루어진 문장이다. SELECT 문장을 처리하는 방법에 따라 성능의 차이가 크며, 특히 SELECT 문장의 WHERE 절에서 Join이 일어나는 경우, 주기억 데이터베이스에서의 주기억 장치의 사용 공간의 크기에 큰 영향을 미치게 된다. 이러한 SELECT 문장의 처리 순서를 살펴보면 다음

과 같다.[6]

(1) WHERE 절의 조건을 만족시키는 레코드를 검색하는 selection 과정
(2) GROUP BY 절의 column 리스트에 따라 레코드를 grouping하는 과정
(3) 나누어진 그룹에 대해 HAVING 절을 수행
(4) ORDER BY 절의 column 리스트를 이용하여 정렬 수행
(5) SELECT 다음에 위치하는 column 리스트를 이용하는 projection
(6) DISTINCT 옵션이 있을 때의 중복 제거

SELECT 질의를 처리할 때, WHERE 절의 조건에 따라 Join 조건이 없는 단순 조건 질의와, Join 조건과 단순 조건이 섞여있는 Join 질의로 나누어 처리하였다.

3.4 Embedded SQL에서의 Cursor 처리

Interactive SQL의 경우와 달리, Embedded SQL에서는 질의 결과 레코드에 대한 조작을 하기 위해 Cursor를 사용하여야 한다. 이를 위해 DECLARE 문장을 이용하여 Cursor를 선언하고, OPEN 문장을 이용하여 선언한 Cursor를 불러오며, FETCH 문장을 이용하여 Cursor가 위치하는 레코드의 값을 host variable에 저장한다. DECLARE 문장을 이용해 Cursor를 선언한 경우 CURSOR_LIST 구조체에 선언된 Cursor의 이름과 Cursor에 대한 Parse tree를 저장하며, 등록된 Cursor의 수를 1만큼 증가시킨다. OPEN 문장을 이용해 Cursor를 불러올 때, 실제 테이블에서 값을 이용하여 Cursor를 생성하게 된다. Cursor가 생성되면, Cursor가 가리키는 테이블 포인터와 레코드 포인터, 레코드 수에 대한 값이 구조체에 채워진다. OPEN 문장이 실행된 후 FETCH 문장이 실행되면, CURSOR_LIST 구조체의 nFetched 변수의 값을 1 증가시키고, fetchPtr을 증가시킨다. CLOSE 문장이 실행된 경우에는 Cursor가 가리키는 테이블이 임시 테이블인 경우는 테이블을 삭제하고, Cursor 이름과 Parse tree 값을 제외하고 CURSOR_LIST 구조체의 값을 초기화한다.

3.5 Dynamic SQL

Embedded SQL로 작성된 응용 프로그램에 따라, 사용자로부터 입력받은 질의 조건에 따라 항상 다른 질의를 처리해야 하는 경우가 있다. 이 때 질의 조건에 맞는 적당한 SQL 문장을 생성해야 하는데, 이러한 질의를 Dynamic SQL이라고 한다. Dynamic SQL을 지원하는 명령은 2가지가 있다. PREPARE, EXECUTE 명령으로 PREPARE 명령이 수행된 경우 SQL 문장을 저장한 host variable을 등록하며, EXECUTE 명령이 수행되면, 등록된 SQL 문자열을 이용해 실제 질의를 처리한다.

4. 구현 환경

4.1 사용 하드웨어 및 운영체제

구현에 사용된 하드웨어와 운영체제는 Compaq DS 10(Tru64 UNIX 5.0), IBM-PC(Linux kernel 2.4.2)의 2종류이다. Compaq DS 10의 경우, 64비트 주소 체계를 지원하는 Tru64 UNIX를 운영체제로 탑재하고 있다. IBM-PC의 경우 32비트 주소 체계를 지원하며 Linux를 운영체제로 탑재한 환경을 이용하였다.

4.2 사용 주기억 데이터베이스

구현에 사용된 주기억 데이터베이스는 ARCHYS 3.0 DBMS[7]로, 1998년에 상용화된 관계형 데이터 모델에 기반을 둔 주기억 데이터베이스 시스템이다.

5. 결론

본 논문의 결과로 Interactive SQL과 Embedded SQL을 지원하는 인터페이스 관리자를 구현하였으나, 여러 가지 면에서 개선해야할 부분이 많았다. 특히 질의 최적화 기법이 전혀 사용되지 않았다는 것이 앞으로 개선해야할 점이다. 기존의 디스크 기반 데이터베이스에 사용되는 비용 모델(cost model)에서는 디스크 입출력에 주안점을 둔데 반해서, 주기억 데이터베이스에서는 주기억 장치의 효율적 사용 측면을 강조한 새로운 비용 모델을 가지고 질의 최적화 기법을 만들어야 될 것이다. 또한, 임시 테이블을 생성하면, 주기억 장치의 가용 공간이 줄어드는 문제뿐만이 아니라, 테이블의 레코드 검색 시 원 테이블에 존재하는 색인을 사용하지 못하는 관계로 성능의 떨어지는 단점이 있다. 이 역시 임시 테이블을 줄이는 방향으로 개선되어야 할 것이다.

참 고 문 헌

[1] Margaret H. Eich "Main Memory Database Research Directions" Technical Report 88-CSE-35
[2] H. V. Jagadish, Daniel Lieuwen, Rajeev Rastogi, Avi Silberschatz "Dali: A High Performance Main Memory Storage Manager", Proceedings of the 20th VLDB Conference Santiago, Chile, 1994
[3] Tobin J. Lehman, Eugene J. Shekita, Luis-Felipe Cabrera, "An Evaluation of Starburt's Memory Resident Storage Component", IEEE Trans. on Knowledge and Data Eng., Vol. 4, No. 6, Dec., 1992
[4] T. Lehman and M. Carey, "A study of index structures for main memory database management systems." in Proc. 12th Int. Conf. on VLDB, pp. 294-303, Aug. 1986
[5] Le, Gruenward, Margaret H. Eich, "MMDB Reload Algorithms", Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data, pp. 397-405, May 1991
[6] Michael Stonebraker, "Readings in Database Systems", 2nd, Morgan Kaufmanm, 1994
[7] ARCHYS "www.archys.com"