효율적인 Nested Loops Join을 위한 조인순서 선정 및 인덱스 구성에 관한 연구

Chen Liu*, 여정모**
*부경대학교 컴퓨터공학과
**부경대학교 컴퓨터공학과
*e-mail:liuchenchina@naver.com
**e-mail:yeo@pknu.ac.kr

The Study of the Method that to Choice Efficient Nested Loops Join Order and the Index Design

Chen Liu*, Jeong-mo Yeo**
*Dept of Computer Engineering, Pukyong National University
**Dept of Computer Engineering, Pukyong National University

요 약

정보시스템의 기반이 되는 관계형 데이터베이스에서는 데이터의 양에 따라 성능 차이가 발생한다. 데이터베이스에 관한 여러 가지 기능에 대한 이해가 부족하여 많은 성능 저하 문제를 유발하는데, 그 중에 조인 성능문제가 큰 비중을 차지하고 있다. 아주 드문 경우가 아니라면 대부분의 데이터 처리는 하나 이상의 테이블이 필요하기 때문이다. 조인을 정확하게 사용하면 성능 개선에 큰 이점을 가져 올 수 있다. 본 연구는 관계형 데이터베이스 기반의 가장 기본적인 조인방식인 Nested Loops Join 방식을 효율적으로 수행하기 위한 조인순서 선정 및 인덱스 구성에 관한 연구를 하였다. 연구 결과를 평가하기 위해서 SQL Trace을 추출한 후 성능을 비교함으로써 선정된 조인순서가 효율적인 것을 입증하였다. 또한 기존의 응답시간을 기준으로 성능평가방법보다 액세스한 데이터 블록 수를 기준으로 한 성능평가방법이 더 근본적으로 조인 성능을 개선할 수 있음을 증명하였다. 차후에는 더 복잡한 조인 형태 및 다른 조인방식의 성능개선 방법에 관한 연구를 진행할 것이다.

1. 서론

오늘날 데이터의 양이 전보다 훨씬 더 빠른 속도로 증가하고 있는 추세이기 때문에 관계형 데이터베이스의 성능에 대한 요구가 갈수록 증가하고 있다는 사실은 명확하다. 대량의 데이터를 처리하기 위한 새로운 기술들이 나타나고 있지만 아직도 관계형 데이터베이스가 시장에서 높은 점유율을 차지하고 있는 이유는 새로운 기술들보다 많은 장점을 가지고 있기 때문이다. 관계형 데이터베이스의 성능을 제대로 사용한다면 새로운 기술을 사용하지 않고도 대량의 데이터를 충분히 처리할 수 있지만 아직도 관계형 데이터베이스를 제대로 사용하지 못하고 있다.

관계형 데이터베이스의 성능에 영향을 미치는 요소는 여러 가지가 있지만 대부분의 데이터 처리는 두 개 이상의 테이블을 필요로 하기 때문에 조인성능에 대한 연구는 필수적이다. 조인을 정확하게 사용한다면 성능 개선에 큰 이점을 가져 올 수 있다. 또한 조인을 효율적으로 수행하기 위해서는 적절한 조인 방법을 선택하는 것뿐만 아니라 조인 방식에 따라 적절한 인덱스를 구성하는 것도 필연적이다. 따라서 본 연구에서는 조인의 성능을 개선하기 위해서 여러 가지 조인방식 중 Nested Loops Join에 초점을 맞추어 조인 순서의 선정 및 해당 조인순서에 대한 적절한 인덱스를 구성하는 방법에 관한 연구를 진행 하고자 한다.

2. 관련 연구

2.1 Nested Loops Join

Nested Loops Join(이하 간단히 NL조인)은 모든 관계형 데이터베이스에 존재한다. 조인을 접하는 개발자들이 가장 처음 듣는 용어가 아마도 NL조인일 것이다. 여러 가지 조인 방식 중에 가장 먼저 소개된 방식이며 그 만큼 효용 가치가 있는 조인 방식이다. 또한 조인은 데이터베이스 안에서 항상 사용되며 OLTP(On-Line Transaction Processing)시스템에서 조인을 튜닝할 때에도 일차적으로 NL조인부터 고려한다. 왜냐하면 NL조인이 가장 기본적인 조인 방식이기 때문이다. 만약 NL조인으로 좋은 성능을 얻을 수 없으면 다른 조인 방식을 고려하게 된다.

NL조인을 정확하게 사용하려면 먼저 조인이 이루어지는 과정을 숙지해야 한다. NL조인은 조인할 두 집합 중에서 선행 집합(Outer Table)을 선정하고 선행 집합을 기준으로 각 Row들을 순차적으로 읽으면서 후행 집합(Inner Table)의 모든 데이터와 비교해서 결합시킨다. 이런 과정 중에서 성능저하를 유발하는 부하지점이 3가지 있다. 첫

째, Outer Table에서 해당 필터조건을 만족하는 데이터를 검색할 때 발생하는 비용이다. 둘째, 연결고리 컬럼의 인덱스를 통해서 조인조건에 만족하는 데이터를 검색할 때 발생하는 비용이다. 셋째, Inner Table에서 조인조건을 만족하고 해당 필터조건도 만족하는 데이터의 조인 가능여부를 체크한 후 조인할 때 발생하는 비용이다. 이 세 가지 부하지점에서 발생하는 비용을 합치면 NL조인의 총비용이 된다. 또한 Outer Table을 검색하는 비용은 SQL문장을 실행할 때 INDEX로 스캔하는지 Full Table Scan방식으로 스캔하는지에 따라 다르다. 두 가지 방식 중에 비용이 작은 방식을 선택한다. 구체적인 계산방법은 본 연구에서 다루지 않기로 했다.

<표 1> NL조인 방식의 수행 구조

```
for(i=o; i<100; i++)
{   --Outer Loop
        for(j=0; j<100; j++)
        {   --Inner Loop
                //Do Anything...
        }
}
```

NL조인 방식의 가장 큰 특징은 <표 1>과 같은 수행구조를 가지기 때문에 선행 집합(Outer Loop)의 크기가 조인의 효율을 결정하는 데 많은 영향을 미친다. 또한 반드시 연결고리이상이 없어야 한다. 만약 연결고리가 이상하다면 선행 집합의 건수만큼 후행 집합을 Full Table Scan 해야 하기 때문에 성능저하를 가져온다.

2.2 인덱스 구성전략

인덱스는 책의 색인과 같이 어떤 자료를 빨리 찾기 위한 목적으로 사용하고 있다. 현업에서 더 좋은 성능을 위하여 인덱스를 섣불리 사용함으로써 성능저하 및 인덱스 관리 문제가 많이 발생한다. 그렇기 때문에 인덱스를 정확하고 전략적으로 구성해서 사용해야 한다.

인덱스를 설계할 때는 여러 가지 요소를 고려해야 한다. 예를 들어 인덱스 Column의 분포도, Clustering Factor, 결합인덱스 구성 Column 순서 등이 있다. 이런 요소들은 인덱스를 통해서 액세스할 때의 효율을 결정한다. 또한 인덱스를 생성한 후 유지 보수하는데 투자비용도 인덱스 구성 전략을 세울 때 고려해야 한다. 왜냐하면 인덱스를 생성한 후 테이블에 데이터를 Insert, Update, Delete할 때 인덱스까지 변경되기 때문이다. 그러므로 인덱스 구성전략을 세울 때 업무나 개별SQL를 모두 고려해야 할 뿐만 아니라 인덱스의 수량도 고려해야 한다.

본 연구의 초점은 NL조인에 국한되어 있기 때문에 NL조인의 과정 중에 인덱스를 어떻게 이용해서 보다 더 나은 성능을 얻을 수 있는지에 대해서만 다루기로 한다. 본 연구에서 설계된 인덱스들은 추후 인덱스 구성전략을 세우는 기반이 된다.

2.3 성능 평가의 방법

일반적으로 데이터베이스 성능을 평가할 때 SQL문장의 실행시간을 가지고 판단한다. 실행시간은 SQL문장을 실행할 때마다 다를 수도 있고 병렬처리를 사용했는지 등 많은 요소에 따라 다를 수 있다. 예를 들어 조인할 때 병렬처리를 사용하면 빠른 시간에 사용자에게 최종결과를 보여 줄 수 있지만 처리하는 과정이 비효율적일 수 있다. 속도가 빠른 것은 보다 더 많은 자원을 사용하였다는 것이다. 만약 빠른 속도를 얻기 위해서 모든 처리가 다 이런 방식으로 진행하고 수행과정 중에서 비효율을 제거해 주지 않는다면 시스템 자원 경쟁 때문에 큰 문제가 될 뿐만 아니라 시스템의 전체 성능이 떨어진다. 그렇기 때문에 실제 액세스하는 과정 중 액세스하는 블록 개수를 기준으로 조인의 처리 중 비효율이 존재하는지 체크하고 제거해야 한다.

본 연구에서 실제 액세스하는 데이터 블록 수를 기준으로 성능을 판단하였다. 또한 기존 성능 평가 방법 중 응답시간만 기준으로 평가하는 것과 비교하여 테스트하였다. 본 연구는 응답시간이 비슷하더라도 처리과정 중에서 성능문제가 존재하는 것을 보여줄 것이다.

3. 본론

3.1 용어 정의

데이터베이스 안에서 조인이 이루어질 수 있는 이유는 값의 비교를 통해 매핑할 수 있는 컬럼이 존재하기 때문이다. 조인이 이루어질 때 테이블들의 연결 컬럼들을 연결고리라고 한다.

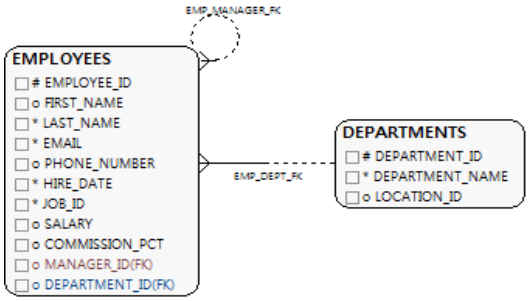
NL조인에서 선행 테이블을 Outer Table 또는 Driving Table이라하고 후행 테이블을 Inner Table 또는 Driven Table이라 한다. Inner Table 측에 있는 연결고리 컬럼에 인덱스가 없다면 Outer Table의 값을 상속시켜 Inner Table에서 테이블 전체를 스캔하게 된다. 이 때 불필요한 비용이 많이 발생한다. Inner Table의 연결고리 컬럼에 인덱스가 없어서 유발되는 성능저하 문제를 연결고리이상이라고 한다.

조인할 때 SQL문장의 WHERE 절에서 여러 가지 조건이 있을 것이다. WHERE 절의 조건들은 연결을 위한 조건도 있고, 조건에 맞는 값을 체크하여 여과시키는 조건도 있다. 값을 체크하여 여과시키는 조건을 필터조건이라고 한다. 조인을 수행하여 나온 최종결과는 필터조건을 만족하는 데이터들로 만들어진 것이다. 실제 조인할 때 필터조건에 만족하는 데이터들을 조인참여집합이라 한다.

3.2 NL조인의 순서 선정 및 인덱스 구성

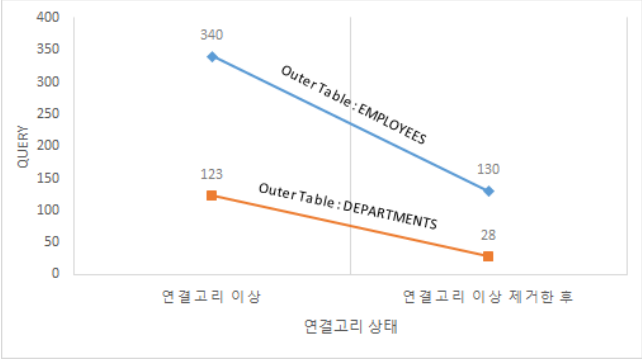
본 연구에서 NL조인을 위한 조인순서 선정 및 인덱스 구성에 관한 연구를 위해서 세 단계의 테스트를 하였다. 첫 번째 단계는 연결고리이상 유무에 따른 성능 비교이다. 두 번째 단계는 NL조인의 조인순서를 선정, 변경할 전후의 성능비교이다. 마지막으로 더 나은 성능을 위해 결합인덱스를 이용한 개선 방법에 관한 테스트를 하였다.

3.2.1 연결고리이상 유무에 따른 성능 비교



(그림 1) 연결고리이상 유무에 따른 성능 테스트 ERD

(그림 1)의 ERD 모델의 테이블을 NL조인할 때 조인순서에는 두 가지가 있다. EMLOYEES 테이블을 Outer Table로 선정하고 DEPARTMENTS 테이블을 Inner Table로 선정하거나 이와 반대 방향으로 선정하는 경우가 있다. (그림 1)의 모델과 같이 무결성을 준수하여 제약조건을 정확하게 생성하면 DEPARTMENTS 테이블을 Outer Table로 선정할 때만 연결고리이상 현상이 발생한다. 만약 제약조건이 정확하게 만들어지지 않았다면 연결고리이상 현상이 어느 방향이든 나타나게 된다. (그림 1) 모델과 같은 경우에 EMPLOYEES 테이블이 DEPARTMENTS에게 상속받은 FK인 DEPARTMENT_ID 컬럼을 포함하는 인덱스를 만들어 주면 연결고리이상 현상을 제거할 수 있다. (그림 2)를 보면 두 개 테이블이 서로 다른 방향으로 조인할 때 연결고리 유무에 따른 성능 차이를 확인할 수 있다.

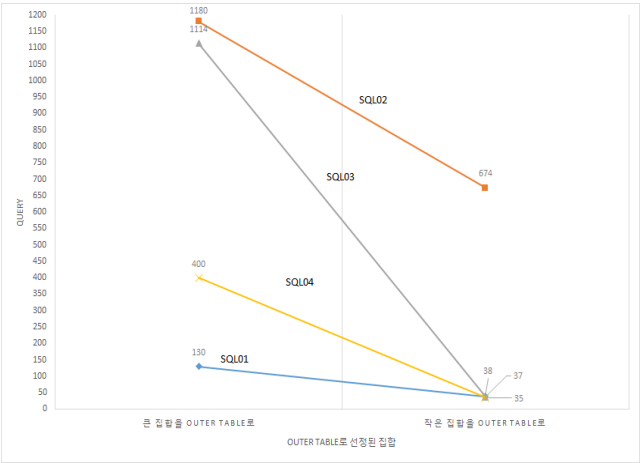


(그림 2) 연결고리이상 유무에 따른 성능 비교

그러므로 NL조인을 할 때 연결고리이상이 없다면 보다 더 나은 성능을 얻을 수 있다. 따라서 NL조인을 할 때 가장 먼저 판단해야 할 요소는 연결고리이상 유무이다. 만약 연결고리이상 상태라면 적절한 인덱스를 구성하여 연결고리이상을 제거해야 한다.

3.2.2 NL조인의 조인순서 선정

본 연구에서는 연결고리이상이 없다고 가정하고 테스트를 진행한다. 연결고리이상이 없을 때 먼저 고려해야 할 것은 조인참여집합의 크기이다. 또한 조인 SQL의 형태는 여러 가지가 있는데, 본 연구에서는 조인 조건이 "="인 경우로 국한한다. 필터조건이 없을 때, 한 쪽만 필터조건이 있을 때, 그리고 양 쪽 다 필터조건이 있을 경우 총 세 가지를 나누어서 테스트를 진행하였다. 각 SQL마다 NL방식으로 조인 순서를 바꾸며 테스트하였고 SQL Trace를 통해서 실제 SQL을 실행했을 때의 실행 계획 및 통계정보를 수집한 후 액세스한 데이터 블록 수를 비교한 결과는 (그림 3)과 같다.



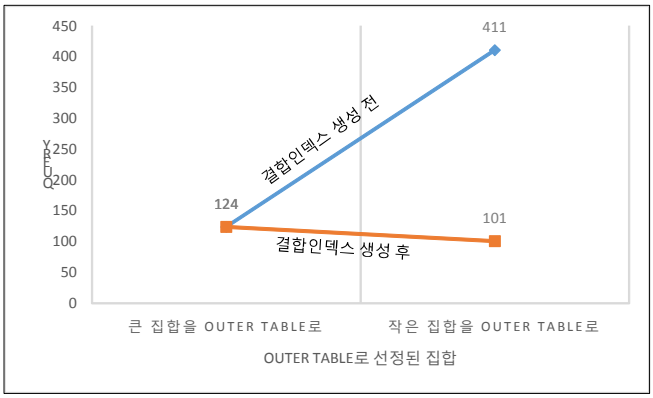
(그림 3) NL조인 순서 변경 전후 성능 비교

(그림 3)의 결과에서 확인할 수 있듯이 작은 집합을 Outer Table로 선정할 때 성능이 좋은 것을 알 수 있다.

3.2.3 결합인덱스를 이용한 NL조인성능 개선

NL조인을 할 때 Inner Table에 필터조건이 있고 이 필터조건 컬럼에 단일 컬럼 인덱스가 있거나 인덱스가 없는 경우에는 Outer Table을 읽고 난 후 Inner Table에 조인 가능한 모든 데이터를 검색하고 조인시킨다. 그 후 조인된 모든 데이터들에 필터 조건을 적용하여 필터링 작업이 진행되고 필터조건을 만족하지 않은 데이터들은 최종 결과 집합에서 제외된다. 이러한 과정 중에서 Inner Table의 필터조건을 만족하지 않는 데이터까지 조인했기 때문에 불필요한 비용이 발생한다. 그러므로 Inner Table의 필터조건을 만족하지 않은 데이터들을 조인하기 전에 미리 제외시키고 필터조건을 만족하는 데이터만 조인하게 되면 성

능을 향상시킬 수 있다. 이 때 Inner Table의 연결고리 컬럼과 필터조건 컬럼의 조합으로 결합 인덱스를 생성하면 Outer Table과 Inner Table이 조인하기 전에 결합 인덱스 내부에서 체크하면서 필터조건을 만족하지 않은 데이터들을 제외시킨다. 결합인덱스를 생성하고 테스트한 결과는 (그림 4)와 같다.

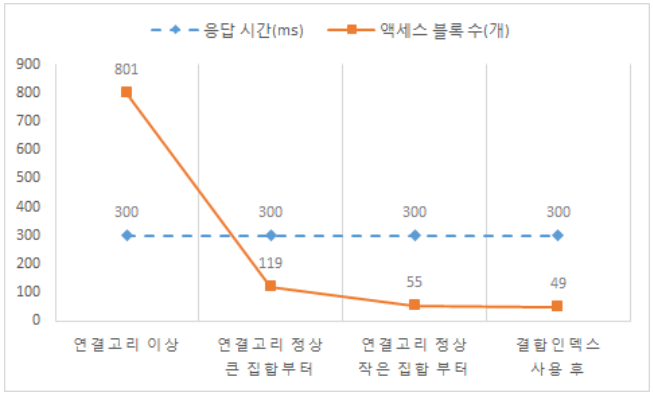


(그림 4) 결합인덱스 유무에 따른 NL조인 성능 비교

(그림 4)에서 Inner Table에 필터 조건이 있을 때 결합 인덱스를 생성하지 않으면 작은 집합을 Outer Table로 선정했지만 큰 집합을 Outer Table로 선정하는 것보다 성능이 더 나쁘다는 것을 확인할 수 있다. 반면에 결합 인덱스를 생성해 주면 작은 집합을 Outer Table로 선정하는 것이 더 성능이 좋다는 것을 확인할 수 있다.

3.3 액세스한 데이터 블록 수 기준으로 한 성능 평가 방법

기존에 사용하고 있는 여러 가지 성능 평가방법들은 응답시간을 이용하여 성능을 평가하였다. 그러나 조인 SQL을 튜닝 할 때 응답시간도 중요하지만 실제 조인 진행하는 과정 중 비효율이 존재할 수 있다.



(그림 5) 응답시간을 기준으로 한 성능 평가방법 및 액세스한 데이터블록 수 평가방법 비교

(그림 5)과 같이 본 연구에서 제안하는 조인 SQL 튜닝 방법을 사용하여 각 단계별로 응답시간과 액세스한 데이터 블록 수를 비교하였다. 또한 이런 단계에 따라서 실제적으로 처리하고 있는 데이터 블록 수가 점차 줄어든 것도 확인할 수 있다.

4. 결론

본 연구는 관계형 데이터베이스의 세 가지 조인 방식 중에 NL조인을 수행하는 과정에서 성능을 개선할 수 있는 방법에 관한 연구를 하였다. NL조인할 때 조인순서를 선정하는 방법은 첫째, 연결고리이상을 제거해야 한다. 둘째, 작은 집합을 Outer Table로 선정하는 것이 효율적인 NL조인을 수행할 수 있다. 셋째, Inner Table의 연결고리 컬럼과 필터조건 컬럼을 결합인덱스로 생성하면 더 나은 성능을 보장한다. 또한 조인 SQL을 튜닝 할 때 응답시간보다 실제 액세스하는 데이터 블록 수가 더 중요한 평가 기준이 된다. 본 연구에서는 NL조인 성능에 대한 테스트를 두 개의 테이블을 대상으로 진행하였고 두 개 이상의 테이블인 경우는 언급하지 않았다. 또한 세 가지 조인 방식 중에 Sort Merge Join 및 Hash Join 방식의 성능개선 방법에 대한 연구는 진행 중이다.

참고문헌

[1] 이화식, “새로운 쓴 대용량 데이터베이스 솔루션 I”, 엔코아 컨설팅, 2008

[2] 조시형, “오라클 성능 고도화 원리와 해법 II”, 비투엔 컨설팅, 2011

[3] 권순용, “실행 계획으로 배우는 고성능 데이터베이스 튜닝”, 러닝스페이스, 2009

[4] Jonathan Lewis, “Cost-Based Oracle Fundamentals”, 사이텍미디어, 2010

[5] Huaiyuan Tan, “How to Make Your Oracle Fly”, Publishing House of Electronics Industry, 2010

[6] Dan Tow, “SQL Tuning”, O’Reilly, 2003

[7] 신명주, 김용성, “관계형 데이터베이스에서 테이블 연산시 효율 향상을 위한 성능 평가 방법”, 한국정보과학회 학술발표논문집, Vol.30 No.1A, pp.731-733, 2003

[8] 이원조, “관계형 데이터베이스에서 셀프조인 쿼리를 위한 성능평가”, 연구논문집, 제28권 제2호, pp.33-36, 2001

[9] Mishra, Chaitanya, “Join Reordering by Join Simulation”, IEEE 25th International Conference, 2009

[10] P. Griffiths Selinger, M. M. Astrahan, Chamberlin, “Access Path Selection in a Relational Database Management System”, http://pages.cs.wisc.edu/~nil/764/Relat/11_selinger.pdf, 2003

[11] Kenji Imasaki, Sivarama Dandamudi, “Performance Evaluation of Nested-loop Join Processing on Networks of Workstations”, Seventh International Conference, 2000