철저한 게시판 관리가 웹 커뮤니티 인터페이스의 핵심

다중처리 방법 채택 통해 오버헤드 줄여야…

이현호 교수 / 안양과학대학 컴퓨터정보학부

연 재 순 서

1. 웹환경에서 더욱 위력적인 Oracle8i의 새로운 기능 (이번호)
2. 웹 커뮤니티 시스템 모델링(1)
3. 웹 커뮤니티 시스템 모델링(2)
4. Oracle8i의 새로운 기능을 이용한 웹 커뮤니티 시스템 구현(1)
5. Oracle8i의 새로운 기능을 이용한 웹 커뮤니티 시스템 구현(2)

지난 호까지 웹 커뮤니티 시스템의 요구사항을 분석하여 데이터모델링을 완성하고, 이에 근거한 데이터베이스 설계를 통하여 물리적인 테이블까지 도출하였다.

이번 호에서는 완성된 모델을 바탕으로 첫 회에서 소개한 오라클 8i에 도입된 새로운 기능– analytic function, function-based index, materialized view –을 이용한 웹 커뮤니티 시스템을 구현에 대해서, '게시판의 부분범위처리', '집계성 데이터의 효율적인 관리' 등의 핵심적인 기능을 중심으로 소개하고자 한다.

참고적으로, 클라이언트 개발도구는 PHP, JSP, ASP 등 여러 가지가 있을 수 있으나 본 글에서는 PHP 4.0에서 Oracle 8 OCI library function을 사용하는 것을 기준으로 설명할 것임을 밝혀둔다.

1. 동적 SQL(Dynamic SQL)과 정적 SQL(Static SQL)

커뮤니티의 핵심 인터페이스인 게시판의 구현을 논하기에 앞서, 꼭 짚어야 할 부분이 있어 잠시 언급하겠다. 오라클에서의 SQL 수행과정을 간략하게 살펴보면, 크게 Parsing, Executing, Fetching 과정으로 분류할 수 있다.

Parsing은 수행하고자 하는 SQL을 오라클 SGA(System Global Area) 내의 공유 SQL 영역(shared SQL pool)에 넣고, 문법적 오류(syntax error)를 찾음과 동시에 데이터 딕셔너리

(data dictionary) 테이블을 검색하여 SQL의 유효성을 확인한 다음, 대상 테이블이나 인덱스 구조에 따른 실행계획(Excution Plan)을 작성하는 과정이다.

Executing은 SQL을 실행계획대로 수행하는 과정이며 Fetching은 수행된 결과 데이터를 DB로부터 가져오는 과정을 말한다.

```
$sql = "select dname from dept where dno = '".$dno."' ";
$stmt = OCIParse($conn, $sql);
OCIExecute($stmt);
$nrows = OCIFetchStatement($stmt, $results); ····· (1)
```

정적 SQL은 SQL statement 자체에 조건에 따라 유동적인 부분은 변수로 정의하여, SQL 수행시 변수 바인딩(binding)을 통하여 SQL statement를 완성하는 방식을 의미한다.

```
$sql = "select dname from dept where dno = :v_dno";
$stmt = OCIParse($conn, $sql);
OCIBindByName($stmt,":v_dno",&$dno,32);
OCIExecute($stmt);
$nrows = OCIFetchStatement($stmt, $results); ····· (2)
```

본 절에서 강조하고자 하는 내용은 질의의 성능향상을 위해서 parsing 오버헤드를 줄이라는 것이다.

그 방안으로, parsing 오버헤드를 줄이기 위해서 동적 SQL보다는 정적 SQL을 사용하고 fetching 오버헤드를 줄이기 위해서 다중처리 방법을 채택해야 한다. 우선, parsing 오버헤드를 줄이는 방안에 대해서 살펴보자.

이를 위해, 동적 SQL과 정적 SQL의 개념에 대해서 간략히 살펴보면, 동적 SQL은 클라이언트 프로그램에서 조건값에 따라 dynamic하게 SQL statement를 만들어 나가는 방식을 의미한다.

오라클은 SGA 메모리 영역 내에 parsing된 SQL을 일정기간 보관하게 된다. 만약, 동일 SQL의 수행요구가 들어오면, 다시 parsing하지 않고 재사용하기 위해서다.

오라클이 판단하는 동일 SQL의 기준은 SQL statement 상으로 정확히 일치하는 SQL을 말한다. 예를 들어, "select * from emp"와 "select * from EMP"는 논리적으로 같은 SQL이지만, 오라클 옵티마이져는 이 두 SQL을 동일 SQL로 보지 않는다.

즉, 두 SQL이 차례로 수행될 경우, 두 번의 parsing을 피할 수 없다. 그러므로, 동적 SQL은 절대 동일 SQL로 취급받을 수 없다. 그러므로, A 프로그램이 수행될 때마다 옵티마이져는 parsing을 시도할 것이다. SGA내의 공유 SQL 영역을 유한하다. 그러므로, 동적 SQL 방식의 수행은 다른 모듈의 수행에도 악영향을 미치게 될 것이다.

이에 반해, B 프로그램은 조건값이 :v_dno라는 변수로 정의되어 있어 SQL statement 자체는 오직 한 종류이다. 나중에 executing시에 바인딩이 일어날 뿐이다. 그러므로, 아무리 반복수행이 된다고 하더라도 1번의 parsing 밖에는 일어나지 않는다.

본 저자는 모회사의 빌링(billing) 시스템 튜닝 시에, 단순히 프로그램 방식을 동적 SQL 방식에서 정적 SQL 방식으로 바꾸어 주기만 했는데도 3배의 성능향상 효과를 거둔 적이 있다. 그만큼 parsing 오버헤드는 무시할 수 없는 수준이며, 개발툴이 정적 SQL 방식을 지원한다면 반드시 정적 SQL 방식을 채택하라고 권고하고 싶다.

2. Analytic function과 Function-based index를 이용한 게시판의 부분범위처리

1) 제안배경

웹 커뮤니티 인터페이스의 핵심은 게시판이다. 그런데, connectionless(stateless)의 기본 성격을 가지는 웹의 구조적인 문제 때문에, 웹은 DB의 이전 상태(state)를 기억할 수 없는 근본적인 한계 때문에 게시판의 일부분을 뿌려주기 위해서 내부적으로는 게시판 전체를 읽어야 하는 비효율의 문제를 내재하고 있다.

물론, 이를 해결하기 위하여 middleware(application server)를 도입한 3-Tier환경으로의 전환이나 Java 등의 전혀 새로운 architecture가 제시되었다. 그러나, 현실적으로 미들웨어(middleware)는 비용부담의 측면을 차치하고서라도 network balancing의 역할 이상을 기대하기 어렵고 Java는 속도면에 많은 문제를 안고 있다.

또한, 더 근본적으로 웹의 게시판은 'random page jump' 속성을 가지고 있어 일반 클라이언트 프로그램에서 page up/down 시의 부분범위처리와는 성격이 전혀 다르다.

이러한 현 실적인 어려움 때문에 대부분의 게시판은 사용자

액션(page click) 이 들어올 때마다, 게 시판 DB table 전체를 읽고 나서 필요한 부분만 잘라서 보여주고 있다. 그러나, 이러한 방식은 게시판의 글 건수가 증가하면 증가할 수록 응답속도가 떨어질 수밖에 없는 뻔한 결론에 도달한다.

본 글에서는, 현재 운영 중인 한 포탈사이트(운영환경 : OS - Linux, Web/Client ? Apache/PHP, DB - Oracle 8.1.6)에서 웹게시판의 부분범위처리를 구현하여 성공적으로 적용되고 있는 방법을 소개하고자 한다. 본 글에서는 응답글이 없는 게시판(공지사항 등...)을 대상으로 한 방법을 다루고 다음 회에서 응답글이 있는 게시판에 대해서 다루기로 한다.

2) 구현원리

앞에서도 언급한 바와 같이, 웹게시판은 DB의 이전 state를 기억할 수 없다는 점과 'random page jump'의 속성을 가지고 있다는 것이 부분범위처리를 가로막는 주요 원인이다. 그러나, 웹의 구조와 웹 게시판의 전형적인 인터페이스 구조를 자세히 살펴보면, 이러한 문제가 전혀 해결할 수 없는 문제가 아님을 알 수 있다.

① 웹이 DB의 이전 state를 기억할 수 없다는 점에 대해서

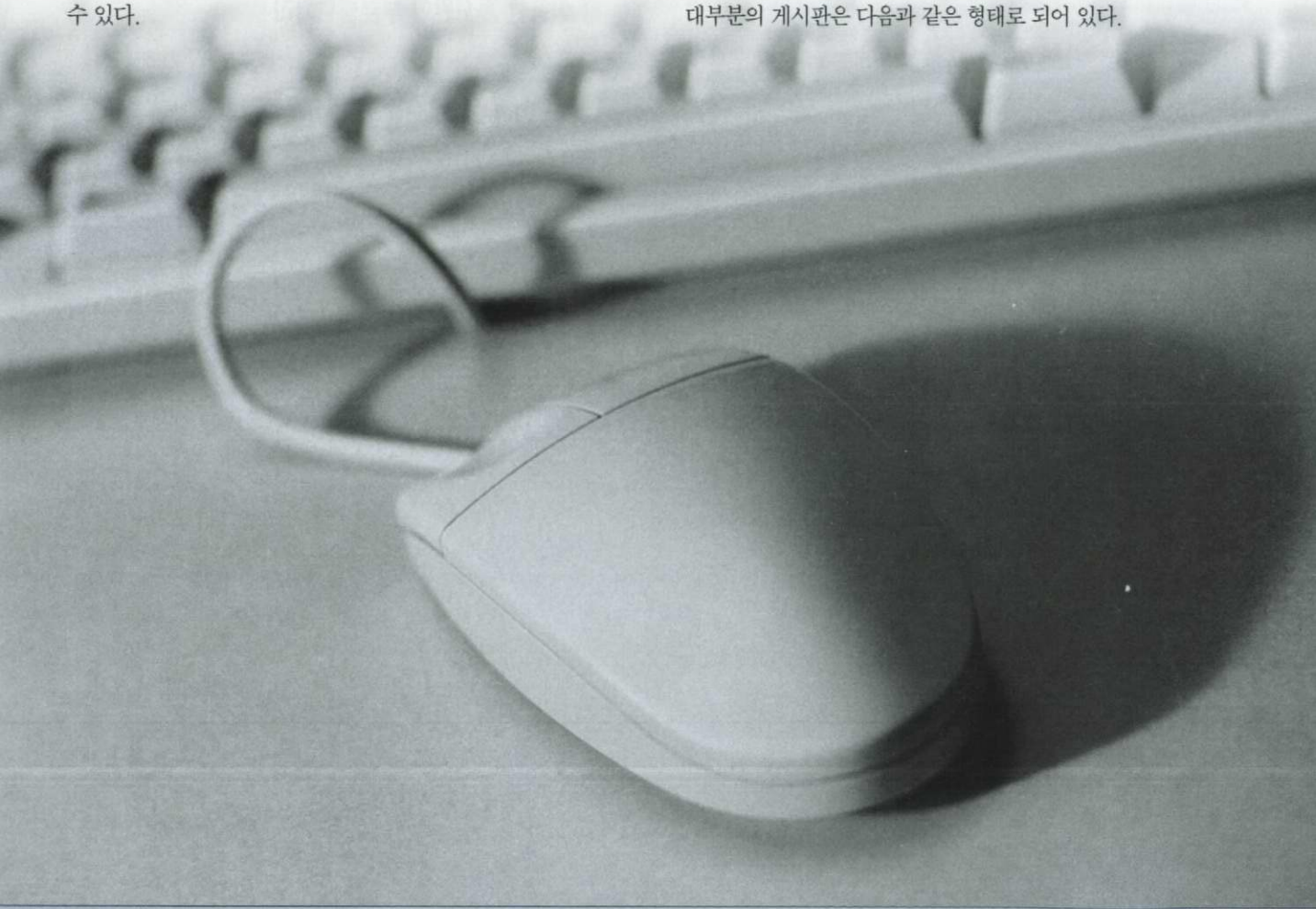
웹이 기본적으로 connectionless 구조를 가지는 것은 사실이지만, 이를 보완할 수 있는 구조가 전혀 없는 것은 아니다.

그 대표적인 사례가 쿠키(Cookie)이다. 쿠키는 사용자 로그온 시부터 로그아웃 시까지 사용자별로 지속적으로 참조해야하는 정보를 담을 수 있는 오브젝트이다. 또한, 제한적이기는 하지만, cgi parameter도 이러한 역할을 할 수 있다.

이를 웹게시판에 이용하면, 현 page view 시에 읽은 데이터의 시작점과 끝점 등의 필요한 정보를 다음 page view 시에 이용할 수 있다.

② 웹 게시판 의 'random page jump' 속성에 대해서

웹 게시판은 1 page에서 2 page로만 갈 수 있는 것이 아니고 바로 10 page로 jump하여 게시글을 볼 수도 있는 구조를 가지고 있다. 그러나, 웹게시판의 전형적인 인터페이스 구조를 곰곰히 살펴보면, page down/up에 해당하는 특성을 찾을 수 있다. 대부분의 게시판은 다음과 같은 형태로 되어 있다.



◀ 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 ▶

한 번에 보여줄 수 있는 page들의 단위를 screen이라고 하고, 한 screen은 10 page로 구성되어 있다고 가정하자. 그러면, 우리는 "1 page에서 10page로는 바로 갈 수 있지만, 1 page에서 31 page로는 바로 갈 수 없다"는 중요한 점을 발견할 수 있다. 즉, page는 random jump를 하지만 screen은 순차적인 page down/up 구조와 동일하다는 것이다.

이와 같이, 웹 게시판의 부분범위처리를 가로막는 두 요인에 대한 해결책은 분명히 있다. 그러면, 어떻게 구현할 것인가? (앞으로, 한 screen은 10page, 1 page는 10개의 글로 구성되어 있다는 전제로 서술하겠다.)

● 기본적인 mechanism

웹 게시판은 random jump의 범위가 전체 page가 아니라 해당 screen 내라는 중요한 점을 발견하였다. random은 말그대로 어디로 갈 지 모르기 때문에, 해당 범위를 다 읽을 수밖에 없다. 그러나, 한 screen에 해당하는 데이터량만 읽으면 된다. 만약, 1 page가 10건의 글로 구성되어 있고 1 screen이 10 pages로 구성되어 있다면, 100건을 읽고 그 중 보고자 하는 page에 해당하는 10건을 뿌려주는 것이다. 이와같이 하면, 100건 중에 10건밖에 사용하지 못하는 비효율은 어쩔 수 없다 하더라도, 게시판에 등록된 총 글수가 몇 개이던지 간에 원하는 page를 읽기 위해서 항상 100건만 읽으면 된다는 결론을 얻을 수 있다.

● 부분범위 처리를 위해 상태(state) 이동 시, 무슨 정보가 필요하나?

우선, 현 page가 속한 screen의 시작 점과 끝점의 정보를 cookie나 cgi parameter에 담아야 한다. 웹게시판은 게시된 순서의 역순으로 출력하는(응답글이 있는 게시판은 좀 다르지만) 것이 일반 적이므로 screen의 시작점(가장 큰 글번호)는 screen up시에 출발점으로 활용해야하고 screen의 끝점(가장 작은 글번호)는 screen down시에 출발점으로 활용해야 한다.

또한, 현 screen에 해당하는 글 수를 알아야 한다. 왜냐하면, 몇 번째 screen이 마지막 screen이 될 지 모르는 상태에서 마지막 screen에서는 몇 page를 뿌려야 할 지를 알아야 하기 때문이다. 또한, 마지막 screen에서는 '다음' 페이지를 나타내는 '▶'를 disable시켜야 한다.

이번 screen이 마지막 screen인지 아닌지를 알려면, 100건만 읽어서는 알 수 없고 그 100건 다 음에 글이 있느냐? 없느냐?를 알아야하기 때문에 101건을 읽어야 한다. 그래서, stop key(rownum)를 이용하여 101건으로 끊었을 때, 실제로 101건을 읽으면 다음 screen이 존재하는 것이고 101건보다 작은 건수를 읽으면 마지막 screen이 되는 것이다. 마지막 screen으로 판명되면 실제 읽은 건수로 몇 page까지 뿌려야할 지를 알 수 있다.

3) 1-SQL로의 구현

지난 회에서 각 게시판의 글정보를 저장하는 테이블로 게시판 BODY 테이블을 설계하였다.

글 내용은 필요에 따라 파일시스템을 이용하고 그 위치(path) 정보만을 DB에 저정하는 방식을 택할 수 있으나 여기서는 DB에 글내용 자체도 저장하는 것으로 가정한다.

```
create table 게시판BODY (
동호회ID varchar2(8),

게시판ID varchar2(8),

글번호 varchar2(8),
작성자 varchar2(8),
글제목 varchar2(255),
작성일시 date,

조회수 number,   — 글방문횟수

글내용 varchar2(2000)
);
```

그리고, 다음이 웹 게시판 부분범위처리를 위한 1-SQL이다. 인덱스는 게시판BODY_PK(동호회ID+게시판ID+글번호)가 primary key로서 잡혀 있는 것으로 가정한다.

① screen down 시와 첫 번째 screen을 뿌릴 시,

select 글번호, 작성자, 글제목, 작성일시, 조회수, min_글번

```
호, max_글번호, cnt
from (
select /*+ index_desc(a 게시판BODY_PK) */ 글번호, 작성자, 글제목, 작성일시, 조회수,
min(글번호) over () as min_글번호, -- screen의 끝점(가장 작은 글번호)
max(글번호) over () as max_글번호, -- screen의 시작점(가장 큰 글번호)
count(*) over () as cnt, -- screen의 글 건수
rownum rnum
from 게시판BODY a
where :v_pagedown = 1 and 동호회ID = :v_cafeid and 게시판ID = :v_bbsid

and 글번호 <= decode(trunc(:v_page/11),0,'99999999',:v_min_index)
and rownum <= 101 )
where rnum between 10*(decode(mod(:v_page,10),0,9,mod(:v_page,10)-1))+1
and 10*decode(mod(:v_page,10),0,10,mod(:v_page,10))
union all
```

② 한 screen 내에서 random page jump를 할 시,

```
select 글번호, 작성자, 글제목, 작성일시, 조회수, min_글번호, max_글번호, cnt
from (
select /*+ index_desc(a 게시판BODY_PK) */ 글번호, 작성자, 글제목, 작성일시, 조회수,
min(글번호) over () as min_글번호, -- screen의 끝점(가장 작은 글번호)
max(글번호) over () as max_글번호, -- screen의 시작점(가장 큰 글번호)
count(*) over () as cnt, -- screen의 글 건수
rownum rnum
from 게시판BODY a
where :v_pagedown = 0 and 동호회ID = :v_cafeid and 게시판ID = :v_bbsid
and 글번호 <= decode(trunc(:v_page/11),0,'99999999',:v_max_index)
and rownum <= 101 )

where rnum between 10*(decode(mod(:v_page,10),0,9,mod(:v_page,10)-1))+1
and 10*decode(mod(:v_page,10),0,10,mod(:v_page,10))
union all
```

③ creen up 시,

```
select 글번호, 작성자, 글제목, 작성일시, 조회수, min_글번호, max_글번호, cnt
from (
select 글번호, 작성자, 글제목, 작성일시, 조회수, min_글번호, max_글번호, cnt, rownum rnum
from (
select 글번호, 작성자, 글제목, 작성일시, 조회수,
min(글번호) over (order by 글번호 desc
rows between unbounded preceding and unbounded following) as min_글번호,
-- screen의 끝점(가장 작은 글번호)
max(글번호) over () as max_글번호, -- screen의 시작점(가장 큰 글번호)
count(*) over () cnt -- screen의 글 건수
from 게시판BODY a
where :v_pagedown = -1 and 동호회ID = :v_cafeid and 게시판ID = :v_bbsid

and c_index >= :v_max_index and rownum <= 101 ) )
where rnum between 10*(decode(mod(:v_page,10),0,9,mod(:v_page,10)-1))+1
and 10*decode(mod(:v_page,10),0,10,mod(:v_page,10))
```

본 SQL은 글번호의 순서와 글 게시시각의 순서가 일치한다는 전제 하에서 정상적으로 동작한다. 응답 글이 없는 게시판은 일반적으로 이러한 성질을 가진다. 즉, 게시 시각의 순서로 글번호가 매겨지며, 이의 역순으로 출력된다는 것이다.

본 SQL에서 cookie나 cgi parameter에 담아야 할 정보는 min_c_index와 max_c_index이다. 이 두 정보만 있으면, 다음 screen의 시작점을 쉽게 알 수 있다.

본 SQL에서 특히 주의해야 할 점은, 첫 번째 screen(1 page - 10 page)를 뿌릴 때는 항상 :v_pagedown = 1(page down mode)로 주고 뿌려야 한다는 것이다. 왜냐하면, 웹게시판은 우리가 글을 보는 사이에도 끊임없이 새로운 글이 올라온다는 사실 때문이다. 이렇게 하지 않으면, 글을 보는 사이에 올라온 새로운 글은 게시판을 처음부터 다시 들어오지 않는 한 절대 볼 수 없다는 문제가 생긴다. 물론, 댓가는 있다.

두 번째 screen에서 첫 번째 screen으로 갈 때는 새로 게시된 글 수만큼 기존 글 을 볼 수 없다는 것이다. 그러나, 이 정도는 감수할 수 있다. 보이지 않는 글은 이미 오래된 글이며, 다시 두 번째 screen으로 이동할 때 보이기 때문이다. 'c_index 〈= decode(trunc (:v_page/11),0,' 99999999' ,:v_max_index)' 부분이 이를 구현한 것이다.

본 SQL은 screen의 시작점과 끝점, screen의 글 건수를 구하기 위해 인라인뷰(inline view)에서 analytic function ? max() over, min() over, count() over을 사용하였다. Analytic function에 관한 자세한 내용은 본 지의 2002년 1월호의 본 기술연재란을 참조하기 바란다.

그런데, Analytic function은 오라클 8.1.6 이상의 버전(version)에서만 동작하므로, 버전이 낮은 오라클 사용자나 다른 DBMS 사용자는 사용이 불가하다. 그러나, 걱정할 필요는 없다. 테이블을 101건씩 두 번 읽으면 된다. 한 번은 일반적인 글 정보를 출력하기 위해, 다른 한 번은 screen의 시작점과 끝점, screen의 글 건수를 구하기 위해 사용하면 된다. 이렇게 하더라도, 매번 게시판 전체를 읽는 비효율보다는 훨씬 낫다.

다음 회에서는 응답글이 있는 웹 게시판에서의 부분범위처리와 materialized view를 이용한 집계성 데이터의 처리에 대해서 살펴보겠다.

응답글이 있는 게시판의 부분범위처리는 응답글이 없는 단순 게시판에 비해 훨씬 복잡하다. 그 근본적인 원인은 글이 게시되는 순서와 출력되는 순서가 응답 글이 없는 웹게시판과 같이 일치하지 않는다는 데에 있다.

또한, 응답글의 depth가 얼마나 깊어질지 모르기 때문에 순환구조로 풀지 않으면 안되는 maintenance의 복잡성이 있다. 그러나, 이것도 약간의 추가정보를 첨가하면 1-SQL로 처리가 가능하다. 여기에서 FBI(Function Based Index)의 개념도 소개된다. 또한, 게시판의 총글수나 사용자별 마일리지합 정보 등 집계성이나 항상 사용자가 온라인상에서 보고자하는 정보들을 효율적으로 관리하기 위해 materialized view를 적용하는 방법에 대해서 알아보겠다.

참고문헌 및 참고사이트

[1] 엔코아정보컨설팅 홈페이지(www.en-core.com) 솔루션웨어하우스(solution warehouse)

[2] PHP Manual, Stig Sæther Bakken 외, http://kr.php.net/manual/en/, 2002.