제20회 한국정보처리학회 추계학술발표대회 논문집 제10권 제2호 (2003. 11)

가상 디렉토리 확장 해시 색인: 확장 해싱에서의 새로운 디렉토리 구조를 이용한 저비용 해시 색인

°박상근*, 박순영, 김명근, 배해영
인하대학교 전자계산공학과
e-mail : {skpark, sunny, kimmkeun}@dblab.inha.ac.kr, hybae@inha.ac.kr

Virtual Directory Extendible Hash index: An Economic Hash Index Using New Directory Structure

°Sang-Keun Park*, Soon-Young Park, Myung-Keun Kim, Hae-Young Bae
Dept. of Computer Science & Engineering, Inha University

요 약

데이터베이스 관계 연산자 중 프로젝션(projection)과 집단 연산(aggregate function) 시 사용되는 GROUP BY 절, 그리고 동등 조인(equi join)에 대한 질의 처리는 중복된 튜플, 중복된 GROUP BY 필드, 조인 중 발생하는 임시결과에 대한 제거나 집단 연산, 임시 결과의 저장을 위해 정렬이나 해싱 기반 알고리즘을 적용하고 있다. 이 중 해싱 기반 알고리즘은 데이터에 대한 직접적인 접근 방법과 정렬비용이 없다는 장점으로 인해 자주 사용하게 된다. 그러나 이러한 해싱(extendible hashing)[1] 기반 알고리즘은 키 값이 저장되는 버켓(bucket) 페이지의 넘침(overflow)으로 인해 분할(split)이 발생하는 경우, 분할을 야기시킨 버켓 페이지에 대한 정보를 제외한 동일한 내용의 기존 디렉토리 구조를 배로 확장해야 하는 공간 확장과, 확장된 디렉토리 구조의 유지를 위해 많은 비용을 소모하게 된다.

본 논문에서는 다량의 데이터에 대한 접근 기법과 디렉토리 구조의 저장공간, 유지 비용 절감 및 중복 해시 값을 지니는 데이터를 처리하기위한 해시 색인인 가상 디렉토리 확장 해시 색인을 제안한다. 가상 디렉토리 확장 해시 색인은 디렉토리 구조를 다단계 구조로 유지함으로써, 넓은 저장공간을 필요로 하는 다량의 데이터에 대한 접근경로 문제를 해결하였고, 가상 디렉토리 레벨이라는 새로운 구조를 통해, 기존 디렉토리 구조의 공간 낭비 및 유지 비용을 최소화 시켰으며, 버켓 페이지를 리스트(list) 구조로 유지함으로써 중복 해시 값에 의한 디렉토리 구조의 연쇄적 분할 문제를 해결하였다.

1. 서론

데이터베이스 관계연산자 수행 과정 중 프로젝션과 AVG, MIN, MAX, SUM, COUNT 등의 집단 연산 시 사용되는 GROUP BY 절, 그리고 연산의 비용이 집중되는 조인질의(동등 조인)에서는 중복된 튜플, 중복된 GROUP BY 필드, 조인 중 발생하는 임시 결과에 대해, 중복 제거, <그룹짓기-값, 실행중-정보> 구조[2]를 이용한 집단 연산, 임시 결과의 저장을 위해 정렬이나 해싱 기반 알고리즘을 적용하게 된다. 이 중 해싱 기반 알고리즘은 키에 대한 해시 값을 이용, 해당 데이터로의 직접적인 접근 방법을 지니며 정렬비용이 없다는 특징으로 인해 자주 사용하게 되는데, 이러한 해싱 기반 알고리즘으로 버켓의 수가 고정되어 있어 파일의 축소, 확장에 대한 공간낭비와 오버플로우 체인이 생기는 정적 해싱보다는 버켓을 가리키는 디렉토리 구조의 추가로 인해 파일의 축소, 확장에 효율적으로 대응할 수 있는 확장 해싱이 보다 유용하다고 할 수 있다. 그러나 이러한 동적 확장 해싱은 버켓의 넘침으로 인해 분할이 발생하는 경우 기존 디렉토리 구조를 두배로 확장하기 위한 저장 공간과 확장된 디렉토리 구조의 유지를 위한 자원의 낭비문제 지니게 된다. 특히 이러한 디렉토리 구조는 확장 당시 생성된 버켓의 위치정보를 지니는 디렉토리 레코드를 제외하고는, 기존 디렉토리 구조의 버켓 페이지 위치정보 값을 그대로 복사해서 확장하기 때문에 디렉토리 구조가 확장될수록 이를 위한 저장 공간 및 유지 비용이 더욱 확장되게 된다..

본 논문[1]에서는 다량의 데이터를 고려한 디렉토리 구조와 이를 위한 저장 공간 및 유지 비용 절감, 중복 해시 값을 지니는 데이터를 처리하기 위한, 가상 디렉토리 확장 해시 색인(이하 VDEHI) 기법을 제안한다.

VDEHI는 디스크 기반 해시 색인이며, 색인의 디렉토리 구조를 다단계 구조로 하여 다량의 데이터에 대한 접근 경로 문제를 해결하였고, 가상 디렉토리 레벨을 추가하여 확장해야 할 디렉토리 구조를 최소화함으로써 디렉토리 구조를 위한 저장공간과 유지 비용을 절감하였으며 중복 해시 값에 대한 버켓 페이지를 리스트 구조로 유지함으로써 디렉토리 구조의 연쇄적인 분할을 방지하는 저비용의 확장 해시 색인 기법이다.

본 논문의 구성은 다음과 같다. 2장에서는 해싱의 충돌 해결을 위한 기법들에 대해 설명하고, 3장에서는 VDEHI의 전체 구조와 VDEHI를 위한 자료구조 및 해시 값을 이용한 페이지 내 접근 방법에 대해 설명한다. 4장에서는 다단계 디렉토리 구조를 이용한 삽입과 분할 및 디렉토리 페이지의 생성 알고리즘에 대해 설명하며, 마지막으로 5장에서 결론을 맺기로 한다.

2. 관련 연구

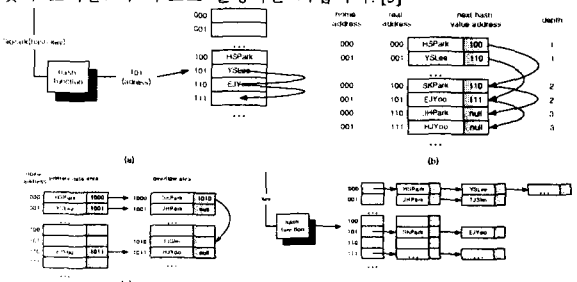
해싱 알고리즘은 레코드의 저장분포를 확산시킬 수는 있으나 충돌을 방지 할 수는 없다. 따라서 동일 해시 값을 지니는 레코드의 오버플로우 문제를 해결하기 위해서는 추가적인 기법이 필요하게

[1] 본 연구는 대학 IT연구센터 육성.지원사업의 연구 결과로 수행되었음

된다.

2.1 정적 해싱의 충돌 해결을 위한 기법

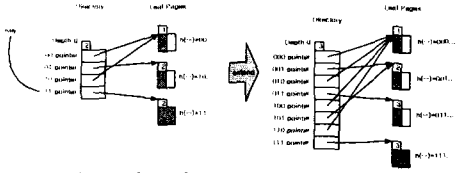
[그림2-1]은 정적 해싱에서의 충돌 해결을 위한 기법들을 나타내고 있다. 이 중 (a)는 오버플로우 레코드를 저장하기 위해 빈 주소를 발견할 때까지 그 다음의 주소들을 연속적으로 탐색하여, 처음으로 나오는 빈 주소를 오버플로우 레코드의 주소로 선택하는 선형 조사 기법을 나타낸다. (b)는 선형 조사의 변형된 기법으로, 유사키들이 포인터로 함께 연결되어 검색 시 이러한 유사키들 지니는 레코드들만이 접근되는 특징을 지니는 체인으로 연결된 선형 조사 기법(chained progressive overflow)을 나타낸다. (c)는 홈(home) 주소에 저장될 레코드와 오버플로우 레코드를 주 데이터 영역(prime data area)과 오버플로우 영역(overflow area)으로 구분하여 저장하는, 별도의 오버플로우 영역을 체인으로 연결한 구조를 나타내며, (d)는 해시 파일에 레코드가 아닌 레코드를 가리키는 포인터를 저장하는 기법으로, 간단한 색인을 제공하며 한번의 접근으로 탐색이 가능한 산재 테이블 구조를 나타낸다. 그밖에 레코드의 군집화(clustering) 문제를 해결하기 위한 방법으로 오버플로우 레코드를 홈 주소로부터 멀리 떨어지게 저장하기 위한 이중 해싱(double hashing) 기법이 있다. 이중 해싱 기법은 오버플로우 레코드가 발생하면, 두 번째 해시 함수에 키를 적용하여 소수(prime)인 p를 생성하고, 이 값을 홈 주소에 더해서 오버플로우 주소를 생성하며, 생성된 주소에서 오버플로우가 발생하는 경우 p 값을 더하는 과정을 반복하여 빈 주소를 찾아 오버플로우 주소로 결정하는 기법이다. [3]



[그림2-1] 정적 해싱의 충돌 해결을 위한 기법

2.2 확장 해싱의 충돌 해결을 위한 방법

확장 해싱은 파일의 크기가 증가하여도 확장이 불가능한 정적 해싱의 확장성 문제를 해결하기 위해, 동적이고 시간이 지나면서 크기가 증가하는 파일을 고려하여 해싱을 적용한 해싱 기법이다. 이러한 확장 해싱은 버켓 페이지의 넘침이 발생하는 경우 전역 깊이(global depth)와 지역 깊이(local depth)가 일치하지 않으면 버켓 페이지를 할당하여 저장 데이터를 분할하게 되고 전역 깊이와 지역 깊이가 같은 경우 버켓 페이지 할당 외에 디렉토리 구조를 두배로 확장함으로써 데이터의 접근 주소와 저장 공간을 확장하여 충돌문제를 해결하고 있다. [그림2-2]는 후자의 확장형태를 나타낸다



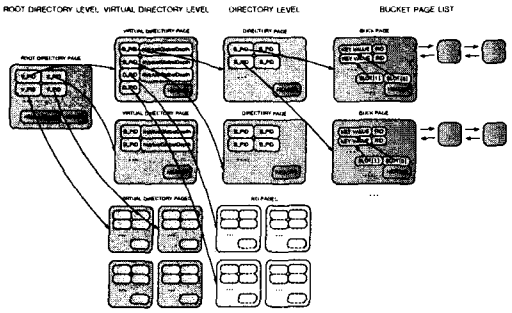
[그림2-2] 확장 해싱의 충돌 해결을 위한 기법

이러한 확장 해싱은 정적 해싱의 단점인 확장 파일에 대한 해싱 문제를 해결하였으나, 데이터의 양이 보다 많아지는 경우 주소 지정에 대한 한계와 중복된 값을 지니는 디렉토리 구조의 확장으로 인해 증가되는 공간 낭비 및 유지비용 문제을 여전히 지니게 된다.

3. VDEHI

3.1 VDEHI 전체 구조

VDEHI의 전체 구조는 [그림3-1]과 같으며 루트 디렉토리 레벨(ROOT DIRECTORY LEVEL), 가상 디렉토리 레벨(VIRTUAL DIRECTORY LEVEL), 디렉토리 레벨(DIRECTORY LEVEL)로 구성된 3종류의 디렉토리 레벨과 양방향 리스트 구조인 버켓 페이지 리스트로 이루어 진다.



[그림3-1] VDEHI 전체 구조

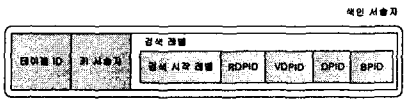
VDEHI는 디렉토리 구조의 확장 시 한쪽 방향 만으로의 확장을 고려한 기존의 확장 해싱 디렉토리 구조를 양방향 확장을 고려한 다단계 디렉토리 구조로 변경함으로써 이진법을 이용해 저장위치를 나타내는 해시 값의 크기를 확장하였으며 이로 인해 다량의 데이터에 대한 저장 공간 범위와 해당 키로의 접근 경로 문제를 해결하였다.

VDEHI의 디렉토리 레벨 페이지들은 동적 확장 해싱과 같은 분할 시점에서 디렉토리 레벨이 두배로 확장하지 않고 단지 가상 디렉토리 레벨의 레코드들만 확장하며 확장 당시의 전역 깊이를 이 레코드에 저장함으로써 나중에 디렉토리 페이지의 정보를 필요로 하는 경우 가상 디렉토리 페이지의 전역 깊이정보 즉, 가상 전역 깊이를 이용한 디렉토리 페이지 정보의 검색을 가능하게 하고 있다.

3.2 VDEHI를 위한 자료구조

3.2.1 색인 서술자

VDEHI에 대한 접근은 [그림3-2]의 색인 서술자 정보로부터 시작된다.



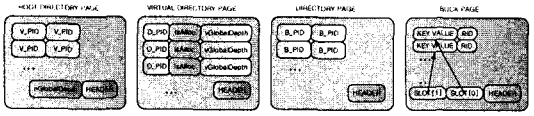
[그림3-2]색인 서술자

색인 서술자의 구조는 본 색인을 사용하는 테이블에 대한 '테이블 ID', 본 색인에 사용되는 키에 대한 정보를 지니는 '키 서술자', 검색이나 삽입을 위해 본 색인 구조로 접근하는 경우, 검색을 시작하는 레벨 정보인 '검색 시작 레벨'(RDLEVEL, VDLEVEL, DLEVEL, BLEVEL)과 시작 레벨의 페이지 ID 값을 저장한 구조인 '검색 페이지ID' 정보를 지니게 된다. 해시 값을 이용한 접근방법을 사용하는 본 기법에서는 분할이 되지 않은 레벨이 지니고 있는 위치정보는 색인 서술자 구조만으로도 접근이 가능하므로, 검색 레벨과 이에 해당하는 페이지 정보를 이용, 디스크 접근 비용을 줄일 수 있게 하였다.

3.2.2 VDEHI를 위한 페이지 구조

[그림3-4]에서 설명하는 VDEHI의 페이지 구조 중 루트 디렉토리 페이지는 디렉토리 레벨의 분할 횟수인 전역 깊이 정보와 가상 디렉토리 페이지들에 대한 위치 정보 등을 저장하고, 가상 디렉토리 페이지는 디렉토리 페이지들에 대한 위치정보와 각각의 디렉토리 페이지들에 대해, 확장 해싱의 경우와 같은 분할 시점의 전역 깊이 정보를 지니고 있게 된다. 이러한 가상 전역 깊이는 [알고리즘4-3]에서 가상 디렉토리 페이지를 이용해 미 할당된 디렉토리 페이지 내 버켓 페이지 위치 정보를 구하기 위한 중요 요소가 된다. 디렉토리 페이지는 버켓 페이지에 대한 위치정보를

지니게 되며 버켓 페이지에는 해시 키 값이 되는 실제 데이터와 이 데이터에 대한 레코드 ID인 RID 값을 저장하게 되고 키 값의 다양한 길이를 고려해 고정길이 슬롯 구조를 이용한 접근 방법을 사용하고 있다.



[그림3-4] VDEHI의 페이지 구조

3.3 해시 값을 이용한 VDEHI 페이지 내부경로 접근

색인 서술자에서 접근 시작 디렉토리 레벨과 해당 레벨의 페이지 ID를 얻은 후, 페이지 내 해당 데이터에 대한 오프셋(offset) 정보를 구하면 하위 레벨의 접근을 위한 페이지 ID 정보를 지닌 레코드에 접근 할 수 있게 된다. 이러한 접근 방법을 위해 각 레벨에 대한 접근 정보인 해시 값은 다단계 디렉토리 레벨의 해당 페이지 내 오프셋 정보 제공을 위해 사용하게 되는데 투트 디렉토리 페이지 내 오프셋 정보인 RD_{Offset}은 [수식3-1]을 이용해 얻을 수 있고, 가상 디렉토리 페이지 내 오프셋 정보인 VD_{Offset} 은 [수식3-2]를 통해 얻을 수 있으며, 마지막으로 디렉토리 페이지 내 오프셋 정보인 D_{Offset} 은 [수식3-3]을 이용해 얻을 수 있다.

- $RD_{Offset} = (h / DITEM_{Item}) / VDITEM_{Item}$ .....[수식3-1]
- $VD_{Offset} = (h / DITEM_{Item}) \% VDITEM_{Item}$ .....[수식3-2]
- $D_{Offset} = h \% DITEM_{Item}$ .....[수식3-3]

위의 수식에서 h는 유효한 해시 값을 나타내며 $DITEM_{Item}$ 은 디렉토리 페이지 내 최대 DIR ITEM(버켓 페이지 위치 정보) 갯수를, $VDITEM_{Item}$ 은 가상 디렉토리 페이지 내 최대 VIR ITEM(디렉토리 페이지 위치 정보) 갯수를 나타낸다.

4. VDEHI의 삽입, 분할, 검색 알고리즘

4.1 삽입 알고리즘

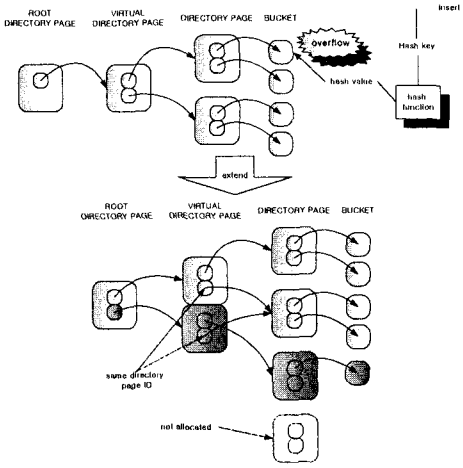
VDEHI의 삽입 과정은 확장된 레벨의 상태 정보를 이용, 삽입 레벨을 결정하며 삽입을 위한 디렉토리 구조의 접근은 투트 디렉토리 페이지, 가상 디렉토리 페이지, 디렉토리 페이지, 버켓 페이지 순으로 진행된다. 가상 디렉토리를 통한 간단한 삽입 과정은 [알고리즘4-1]과 같으며, 각 레벨에 속하는 페이지의 접근은 다음 레벨의 접근을 유도하게 되고 이러한 과정 중 가상 디렉토리 페이지로의 접근 모듈은, 가상 디렉토리 페이지 내 아이템을 실 디렉토리 페이지 할당을 통해 맵핑(mapping)시키는 디렉토리 페이지 유효화 과정을, 버켓 페이지로의 접근은 버켓 페이지의 넘침 시 분할 과정을 담당하게 된다.

[알고리즘4-1] ***INSERT***

```
Input
    IndexDesc: index description
    Key: index key value
    RID: record ID

Procedure Insert(IndexDesc)
Begin
    RDPAGE.Fetch(IndexDesc.RDPID);
    DEPTH_Global = RDPAGE.DEPTH_Global;

    h_Validate = GetValidateHashVal (Key, DEPTH_Global);

    Switch(IndexDesc.LevelState)
    Begin
    Case(RDLEVEL):
        IndexDesc.VDPID = RDPAGE.GetVDPID(h_Validate);
    Case(VDLEVEL):
        VDPAGE.Fetch(IndexDesc.VDPID);
        IndexDesc.DPID = VDPAGE.GetDPID(h_Validate);
    Case(DLEVEL):
        DPAGE.Fetch(IndexDesc.DPID);
        IndexDesc.BPID = DPAGE.GetBPID(h_Validate);
    Case(BLEVEL):
        BPAGE.Fetch(IndexDesc.BPID);
        BPAGE.Insert(Key, RID);   /* treat bucket overflow */
        break;
    End-switch
End
```

4.2 분할 알고리즘

버켓 페이지로의 키 값 삽입 과정 중 넘침이 발생하는 경우 가상 디렉토리 구조의 전역 깊이보다 버켓 페이지의 지역 깊이가 작은 경우 버켓 페이지만 분할 하고 해당 디렉토리 페이지에 버켓 페이지 정보를 기록하게 된다. 이 때 가상 전역 깊이가 전역 깊이보다 낮으면 [알고리즘4-3]의 과정에 의해 해당 디렉토리 정보를 기록하게 된다. 반면 버켓 페이지 분할 시 전역 깊이와 지역 깊이가 같게 되면 버켓 페이지와 가상 디렉토리 구조 모두를 확장 하게 되는데 [그림4-1]과 같이 디렉토리 페이지는 확장 되지 않고 가상 디렉토리 페이지만 두배로 확장한 후 확장된 가상 디렉토리 페이지 내 아이템에 확장 이전 아이템 값을 설정하게 된다.



[그림4-1] VDEHI 디렉토리 구조의 확장

마지막으로 분할된 버켓 페이지의 정보를 디렉토리 페이지에 기록하게 되는데 이 때 이전 과정에서 미 할당된 페이지가 필요하게 되는 경우 해당 디렉토리 페이지 만을 할당하고 가상 디렉토리 레벨 중 이를 지칭하는 가상 디렉토리 아이템의 정보만을 갱신하게 된다. 확장 해싱의 경우 해시 값이 같을 때의 삽입은 버켓 페이지의 재 분할을 야기시키기도 하는데 이런 과정은 디렉토리 구조의 연쇄적 분할을 초래할 수 있기 때문에, VDEHI는 이런 경우 버켓 페이지를 리스트로 확장하여 디렉토리 구조의 연쇄적 확장을 방지하고 있다. VDEHI의 분할 알고리즘은 [알고리즘4-2]와 같다.

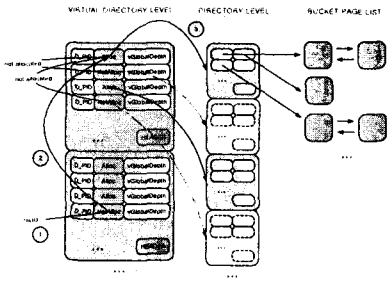
[알고리즘4-2] ***SPLIT***

```
Input
  PREVDITEM_i: pre-virtual page item

Procedure Split()
Begin
/* DEPTH_Global is equal to DEPTH_Local */
DEPTH_VirtualGlobal = DEPTH_Global + 1;

If DITEM_Num * 2 are less equal then DITEM_max then
    /* number of extended DITEM is less then coverage of Directory Page */
    For each i in DITEM_Num do
        ExtendDItemInADPage();
    End-For
    DITEM_Num *= 2;
Else
    /* module in ExtDPageToMPages() */
    If VDITEM_Num * 2 are less equal then VDITEM_max then
        /* number of extended VDITEM is less then coverage of Virtual Directory Page */
        For each VDITEM_i do /* i is 0 to VDITEM_Num * 2 */
            VDITEM_i'->DPID = PREVDITEM_i->DPID;
            VDITEM_i'->DEPTH_VirtualGlobal = DEPTH_VirtualGlobal;
        End-For
    Else
        /* number of extended VD_ITEM is more then coverage of Virtual Directory Page */
```

```
        For each RDITEMi do /* i is 0 to RDITEMNum */
            VDPAGEAlloc = AllocPage();
            For each VDITEMi do   /* i is 0 to VDITEMmax */
                PREVPAGE = GetPreVDPAGE(hValidate);
                VDPAGEAlloc->VDITEMi->DPID = PREVPAGE->PREVDITEMi->DPID;
                VDPAGEAlloc->DEPTHVirtualGlobal = DEPTHVirtualGlobal;
            End-For
            RDPAGE->RDIDEMi+RD_ITEM = PID of VDPAGEAlloc;
        End-For
        RDITEMNum *= 2;
    End-If
End-If

DEPTHGlobal ++;
InsertBPIDInDPage(): /* insert Bucket PID in Directory Page */
End
```

4.3 디렉토리 페이지 생성 알고리즘

VDEHI를 이용한 검색 과정 중 [그림4-2]의 가상 디렉토리 페이지 내 ①의 경로로 접근하게 되면 가상 디렉토리 레벨의 확장 시 기록된 ②와 동일한 디렉토리 페이지 위치 정보를 이용하여, 디렉토리 레벨로의 접근을 수행하며, 삽입 과정 중 분할이 발생하여 ①의 경로로 접근하게 되면 ①의 아이템에 기록된 가상 전역 깊이와 전역 깊이를 비교, 가상 전역 깊이가 보다 작으면 해당 디렉토리 페이지가 존재하지 않는 경우이므로 ②에 기록된 아이템이 지니는 위치 정보가 가리키는 디렉토리 페이지 정보를, 생성된 디렉토리 페이지에 복사하여 유효화 과정을 거친 후 ①에 기록된 디렉토리 페이지 위치 정보를 할당된 디렉토리 페이지 정보로 갱신하게 된다. 마지막으로 분할을 야기시킨 버켓 페이지 위치 정보를 할당된 디렉토리 페이지 내 해당 아이템에 기록하면 된다.



[그림4-2] 가상 디렉토리를 이용한 위치정보 검색

삽입 과정 시 디렉토리 페이지를 할당하는 방법은 [알고리즘4-3]과 같다.

```
[알고리즘4-3] Get DIRECTORY PID from VIRTUAL DIRECTORY PAGE
Input:
    VDITEM: Directory Information in Virtual Directory Page
    VDPID: Virtual Directory Page ID
Output:
    DPAGEAlloc PID: New Directory Page ID

Proccdure GetDPIDFromVDPage()
Begin
/* Direcoty Page is Not Allocated */
If VDITEM-> DEPTHVirtualGlobal is less then DEPTHGlobal then
    /* allocate Directory Page and write the Bucket Page information in the Directory Page */
    DPAGEAlloc = AllocPage();
    DPAGE = Fetch(DPIDAllocated);
    For each i in DITEMNum do
        DPAGEAlloc->DITEMi = DPAGE->DITEMi;
    End-For
    VDPAGE->VDIDEM[VDITEMOffset] = DPAGEAlloc;
Else
    return;
End-if
```

5. 성능 평가

본 장에서는 공간 데이터베이스 관리시스템인 GMS[10]를 이용, 본 논문에서 제안한 VDEHI와 일반적인 확장형 해시 색인인 Extendible Hash(EH)와의 동등 조인에 대한 성능평가를 하였다. 시스템 환경은 Pentium Ⅳ 1.8 CPU, 512 M memory 이며 테이블 구성과 질의는 [그림5-1]과 같고.질의 결과는 [그림5-2]와 같다.

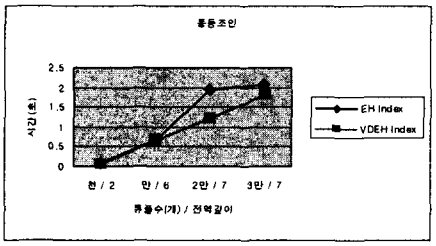
id	fname	mname	lname	birth

(a) 테이블 구성

```
select *
from author1, author2
where author1.id = author2.id;
```

(b) 검색 질의

[그림5-1] 테이블 구성 및 검색 질의



[그림5-2] EH Index VS. VEDH Index(동등 조인)

[그림5-2]의 결과를 보면 전역 깊이가 증가할수록 제안 기법이 기존의 확장 해시 색인 기법보다 더 나은 성능을 보인다는 것을 알 수 있으며, 이는 제안 기법이 전역 분할 시 발생하는 디렉토리 구조의 생성과 유지 비용을 절감하였기 때문이다.

6. 결론 및 향후 연구방향

본 논문에서 제안한 VDEHI는 다량의 데이터에 대한 접근 경로 지정, 해시 디렉토리 구조의 경제적 확장과 유지, 해시 값의 충돌 시 재 확장 방지를 위한, 확장된 해시 색인이다.

VDEHI는 확장된 경로 지정을 위해 다단계 디렉토리 구조를 사용하고 있으며, 디렉토리 구조의 경제적 확장과 유지를 위해 새로 생성된 버켓 페이지에 대한 위치 정보를 지니는 디렉토리 페이지에 대한 정보만을 지니는 가상 디렉토리 구조를 사용하였고 디렉토리 구조의 연쇄적 확장 방지를 위해, 버켓 페이지가 연속 분할을 야기시키는 경우 버켓 페이지를 리스트 구조로 확장하도록 하였다.

향후 연구 과제로는 VDEHI의 디렉토리 구조에 대한 bottom up tree 구조로의 확장 방안이 연구 중에 있다.

참고문헌

[1] RONALD FAGIN, JURG NIEVERGELT , NICHOLAS PIPPENGER , H. RAYMOND STRONG , "Extendible Hashing-A Fast Access Method for Dynamic Files," ACM Transactions on Database Systems, Vol. 4, No. 3, September 1979.

[2] Ramakrishnan, J. Gehrke, "Database Management Systems," McGraw Hill College Div, 2002.

[3] M. Folk, B. Zoellick, G. Riccardi, "File Structures," Addison-Wesley, 1998

[4] R. J. ENBODY, H. C. DU, "Dynamic Hashing Schemes," ACM Computing Surveys, Vol. 20, No. 2, June 1988.

[5] Hector Garcia-Molina, Jeffry D Ulman, Jennifer Widom, "Database System Implementation," Prentice Hall, 2000.

[6] W. Litwin. Linear hashing: A new tool for file and table addressing. In Proc. of VLDB, 1980.

[7] MARKOWSKY, G, CARTER, J.L., AND WEGMAN, M. Analysis of a universal class of hash functions.
Lecture Notes in Computer Science 64,1978, pp. 345-354.

[8] MATTSON, R., GECSEI, J., SLUTZ, D., AND TRAIGER, I. Evaluation techniques for storage hierarchies. IBM Syst. J. 9,2 (1970), 78-117.

[9] WALKER, W.A. Hybrid trees as a data structure. Ph.D. Diss., Dept. Comptr. Sci., U. of Toronto,
Toronto, Ont., Can

[10] 박상근, 박순영, 정원일, 김명근, 배해영, "GMS: 공간 데이터베이스 관리 시스템", 2003. 03