

Ch15. 데이터베이스 파일 조직: 레코드들의 비순서, 순서 및 해시된 파일

File

- 디스크 저장 매체
- 레코드들로 구성된 파일
- 파일에 대한 연산
- 비순서 파일(Heap)
- 순서 파일
- 해시 파일
 - 동적이고 확장 가능한 해싱 기술
- RAID 기술

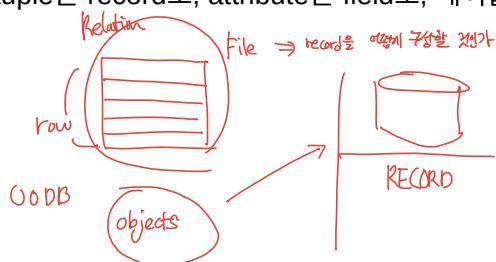
Access
Method Indexing

covers (f, g)

F covers $G \rightarrow \text{True}$

물리적 DB에서 혼동을 일으키는 용어들

- FILE: OS File이 아니다. Disk-based Data structure로 이해해야 한다. 레코드의 집합체
- Data structure라는 용어보다 Data organization이라는 용어를 전통적으로 사용한다.
- Key: 논리적 데이터 모델에서 이야기하는 키가 아니다. search term이면서, 파일에서 탐색을 위해 사용되는 필드이다. key는 주고 value를 가져온다
- tuple은 record로, attribute는 field로, 테이블은 파일로 매팅된다.



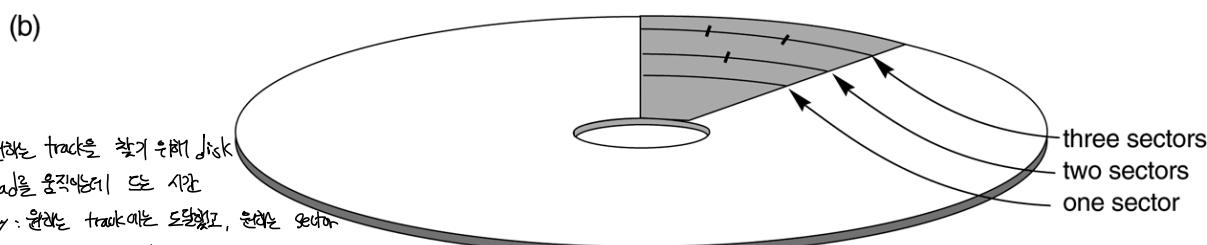
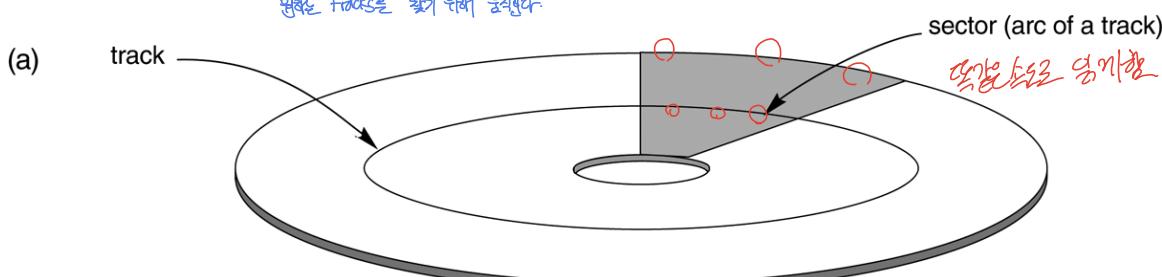
디스크 기억 장치

- 자기 디스크는 적은 비용으로 방대한 양의 데이터를 저장하기 위하여 사용된다.
- 디스크에 저장하는 가장 기본적인 데이터의 단위는 비트이며, 특정 방법으로 디스크상의 한 영역을 자기화함으로써 0 또는 1의 비트값을 표현
- 디스크 팩은 여러 장의 자기 디스크를 뮤어서 구성
- 디스크는 디스크 표면상의 동심원인 트랙으로 나누어지며 각 트랙은 4 ~ 50 Kbytes를 기록할 수 있음.
- 트랙은 블록으로 나누어지며 블록 크기는 한 시스템 내에서 고정되어 있으며 일반적인 블록 크기는 512byte에서 4096byte이다. 주기억장치와 디스크간의 데이터 이동은 블록 단위로 수행
- 디스크 상의 여러 섹터 구조 :
 - (case a) 일정각으로 분할된 섹터,
 - (case b) 동일한 기록 밀도를 유지하는 섹터

Block (or page) : sectors의 그룹 (read/write의 Unit)
일반적으로 4K bytes

- Disk는 magnetic platters의 stack
- 각 platter의 표면은 여러개 tracks로 구성
- tracks는 sectors로 나눠진다.
- Sector: Storage의 기본 Unit

disk heads는 spindle의 원통을 찾기 위해 회전하는 동안 (동일한 cylinder내)
원하는 tracks를 찾기 위해 움직인다.



- 판독/기록 헤드가 전송할 블록이 있는 트랙으로 이동하고 자기 디스크면이 회전하여 원하는 블록이 판독/기록 헤드 아래에 위치하면 그 블록을 읽거나 쓴다. ↗
- 디스크 접근 시간 = 탐색시간 (seek time) + 회전 지연시간 (또는 지연시간 rotational delay) + 블록 전송 시간(block transfer time)
→ 가장 시간이 짧게 걸림. ↗ 한바퀴 돌아와서 일정시간 ↗ bit의 disk block을 전송하기 위해 사용되는 시간
- 물리적 디스크 블록 주소는 표면번호, 표면내 트랙번호, 트랙내 블록번호로 구성
- 디스크 블록의 읽기나 쓰기에서 탐색시간과 회전 지연 시간이 대부분의 시간을 차지한다.

磁気盤은 Commit 했는데 렌즈에 있으면
Reto
whe.

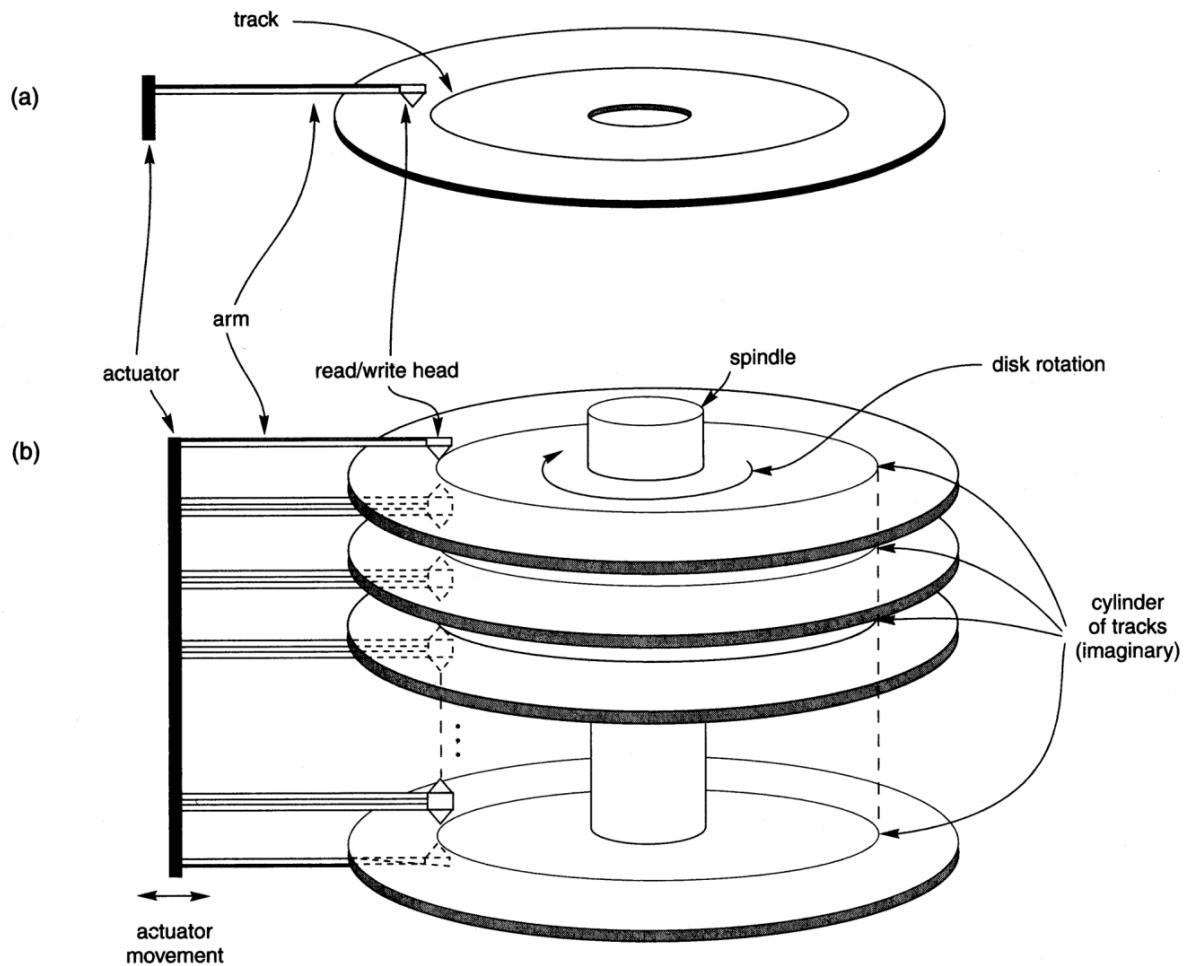
Disk read

- 처리할 disk block의 LBA가 disk drive buffer(cache)에 복사
- cache에 이미 해당 LBA가 있다면 copy X

Disk write

- buffer (main memory buffer)의 contents가 LBA와 함께 disk block로 복사됨
- Disk read와 write는 Synchronous 하여 성능이 같지만, flash의 경우는 두 경로의 성능이 차이가 있다.

(a) 판독/기록 하드웨어를 장착한 단일면 디스크 (b) 판독/기록 하드웨어를 장착한 디스크



Making Data Access More Efficient on Disk - 7th Edition in English

We list some of the commonly used techniques to make accessing data more efficient on HDDs.

1. Buffering of data
2. Proper organization of data on disk:
 - keep related data on contiguous blocks
 - Doing so avoids unnecessary movement of the read/write arm and related seek times
3. Reading data ahead of request
4. Proper scheduling of I/O requests:
 - **elevator algorithm**
5. Use of log disks:
 - A single disk may be assigned to just one function called logging of writes.
 - All blocks to be written can go to that disk sequentially, thus eliminating any seek time.
6. Use of SSDs or flash memory for recovery purposes

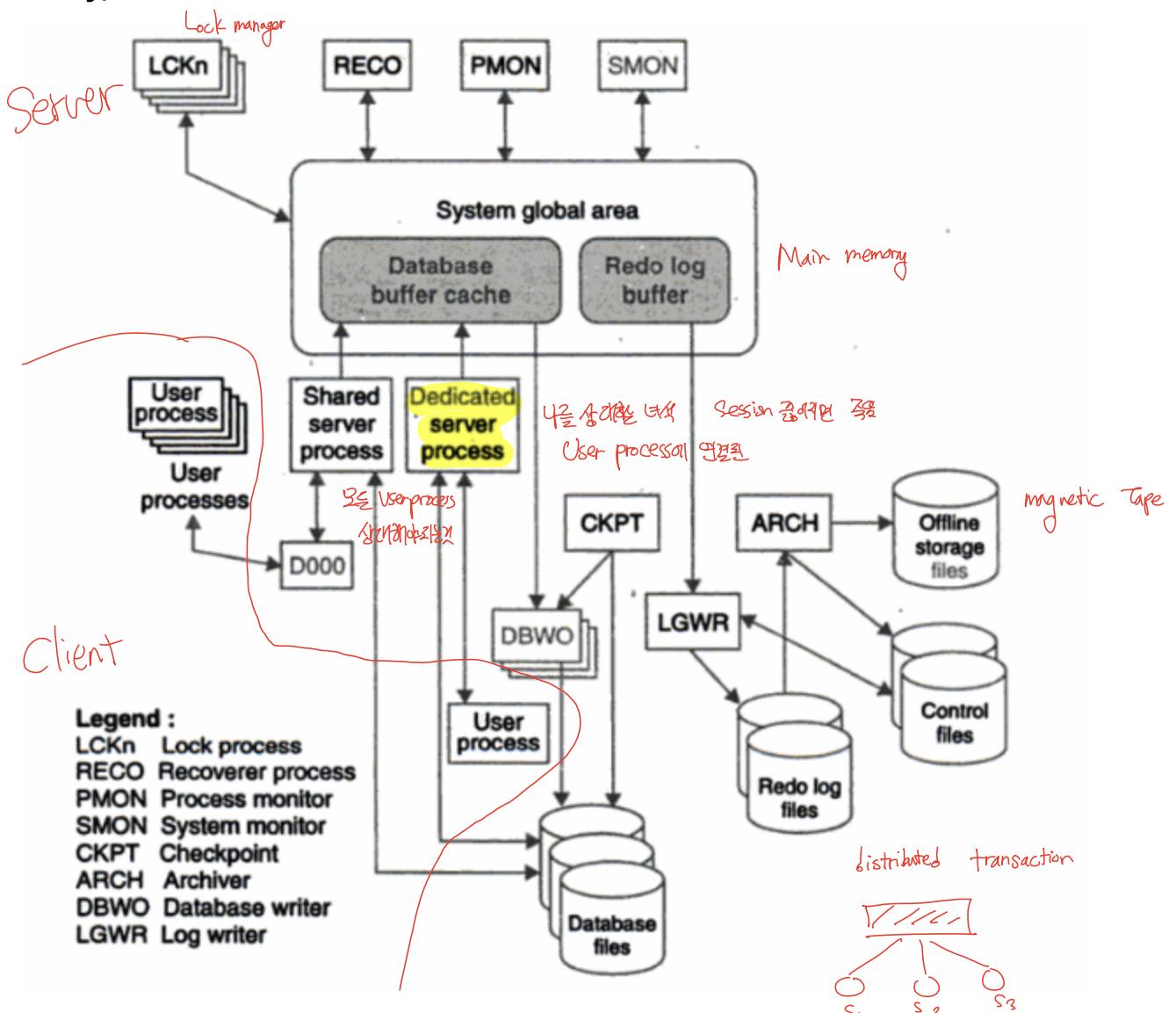
Oracle DBA Guide

https://docs.oracle.com/cd/B28359_01/server.111/b28310/toc.htm
(https://docs.oracle.com/cd/B28359_01/server.111/b28310/toc.htm).

Oracle Performance Tuning Guide

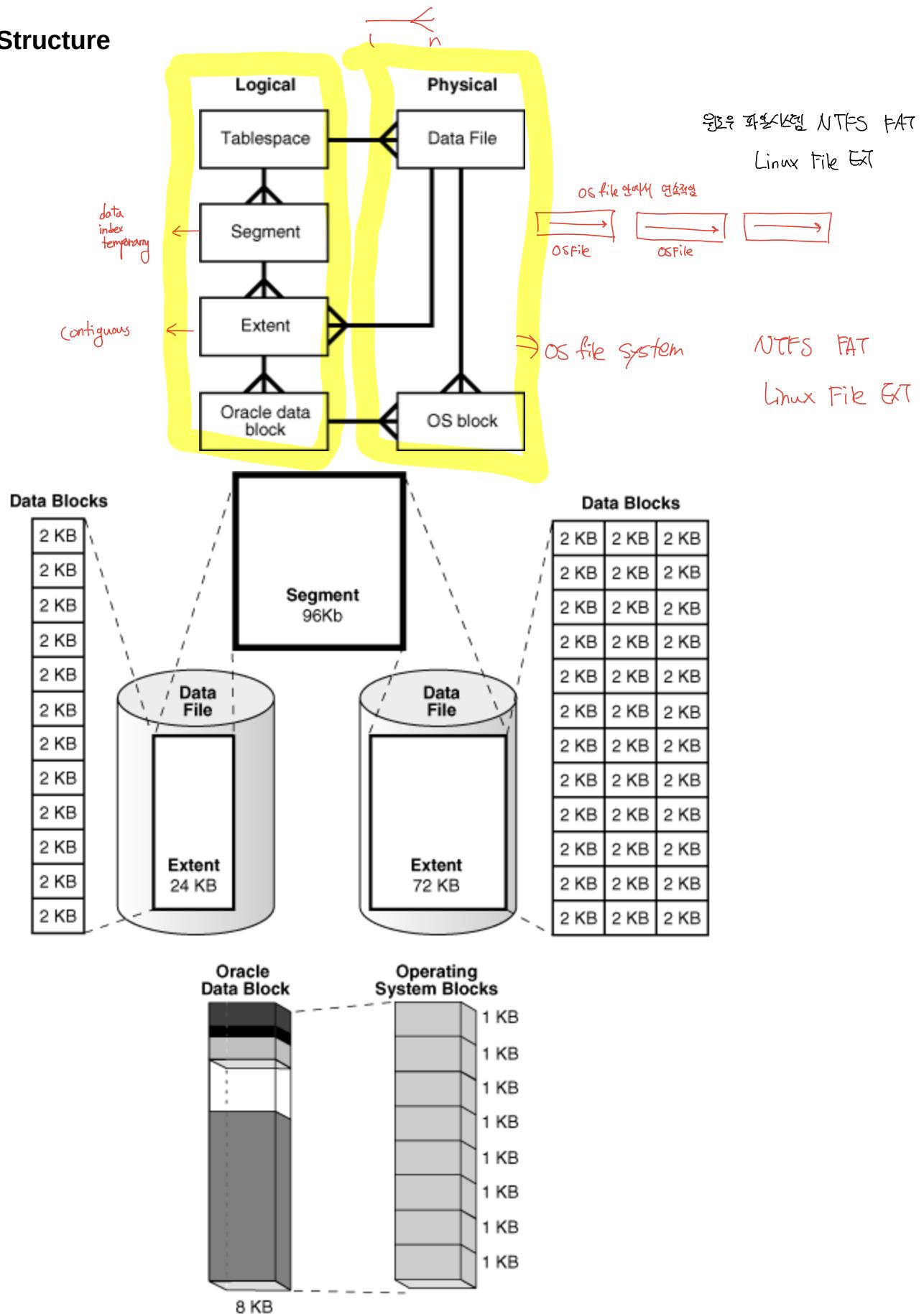
https://docs.oracle.com/cd/B28359_01/server.111/b28274/toc.htm
(https://docs.oracle.com/cd/B28359_01/server.111/b28274/toc.htm).

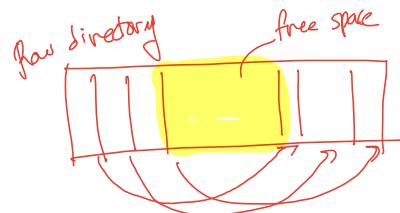
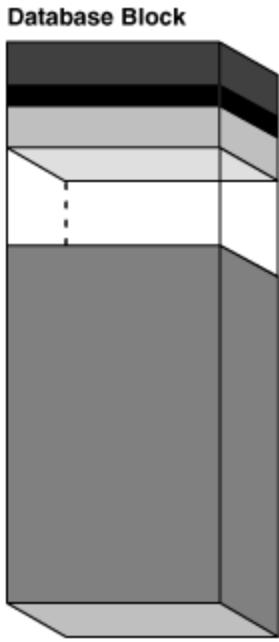
Oracle Memory, Disk and Processes



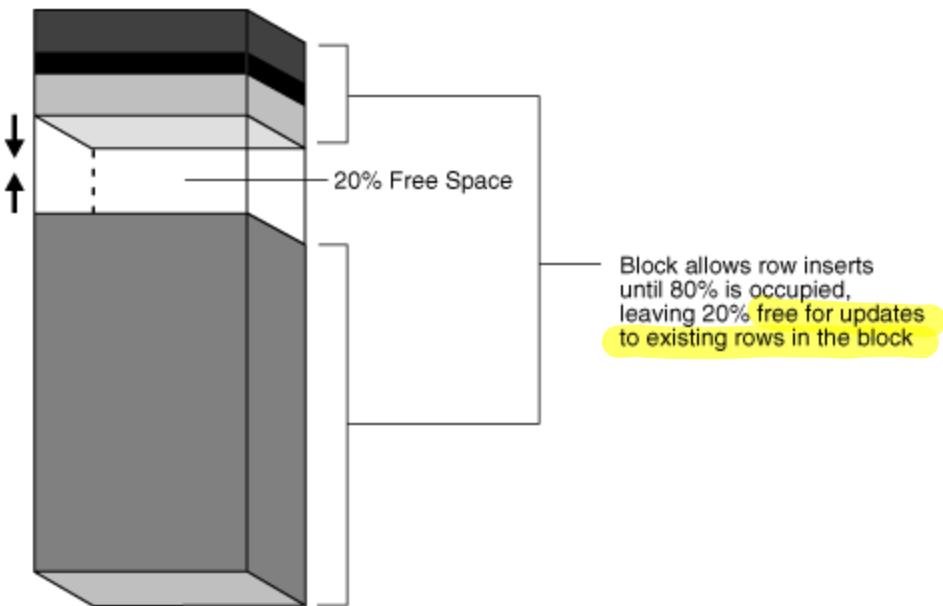
- SMON is responsible for system recovery at instance start-up
- PMON is responsible for user transaction recovery, reclaim cache, and other resources when user transaction fails
- RECO is responsible for distributed transactional recovery
- A dedicated server process, which services only one user process
- A shared server process, which can service multiple user processes

Oracle Storage Structure



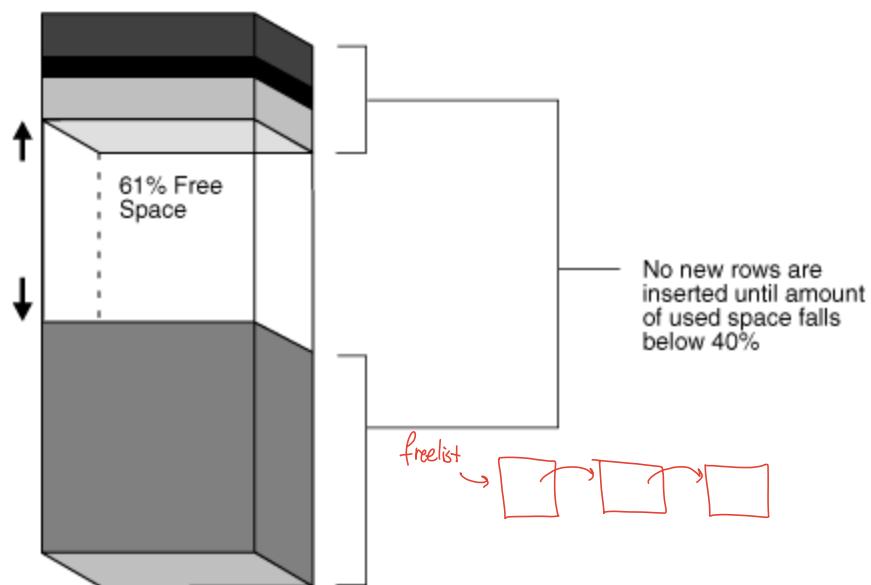


Data Block
PCTFREE = 20



한국어 번역본

Data Block
PCTUSED = 40



```

Create Tablespace my_tbl_space
Datafile
' /data/Bigtbsa01.dbf '
Size 100M
AUTOEXTEND On
Next 10M
MAXSIZE 200M,

' /data/Bigtbsa02.dbf '
Size 100M
AUTOEXTEND On
Next 10M
Maxsize 200M,

' /data/Bigtbsa03.dbf '
Size 100M
AUTOEXTEND On
Next 10M
MAXSIZE 200M,

' /data/Bigtbsa04.dbf '
Size 100M
AUTOEXTEND On
Next 10M
Maxsize 200M
Blocksize 16K;

```

Record and Record Type

- 데이터는 일반적으로 레코드 형태로 저장되며, 각 레코드는 연관된 데이터 값이나 항목들로 구성된다.
- 각 값은 하나 이상의 바이트로 구성되며 레코드의 특정 필드에 해당된다.
- 레코드는 엔티티와 엔티티의 애트리뷰트들을 기술한다.
- 한 필드의 데이터 타입은 프로그래밍에서 사용되는 표준 데이터 타입들 중의 하나이다.
- 표준 데이터 타입에는 수치, 문자열, 부울(0과 1 또는 TRUE와 FALSE) 데이터 타입이 있고, 때로는 특별하게 코드화된 날짜, 시간 데이터 타입들도 사용된다.
- 이미지, 비디오, 오디오, 긴 텍스트를 나타내는 방대한 양의 비구조적인 객체들은 BLOB(Binary Large Object)라고 하며, 대개 별도의 디스크 블럭들의 풀(pool)에 저장되고 레코드에는 그 포인터만 저장한다.

enum { 'Spring', 'Summer', 'Autumn', 'Winter'
 0 1 2 3

Epoch 시즌의 탱생점

Blocking

- 디스크와 주기억 장치 사이의 데이터 전송 단위는 하나의 블록이므로 한 파일의 레코드들을 디스크 블록들에 할당하여야 한다.
- 블록 크기가 레코드 크기보다 크다면 각 블록에 여러 개의 레코드를 저장한다.
- 블록크기가 B 바이트이라 하고 크기가 R 바이트인 고정 길이 레코드들로 구성된 파일에 대하여 $B \geq R$ 인 경우에 블록당 $bfr = \lfloor B/R \rfloor$ 개의 레코드를 저장할 수 있다. B : block size R : record size bfr : blocking factor for the file
- 일반적으로 B 를 R 로 정확히 나눌 수 없으므로 각 블록마다 $B - (bfr \times R)$ 바이트만큼의 사용되지 않는 공간이 있다.
- 비사용 공간을 사용하기 위하여 레코드의 일부분을 한 블록에 저장하고 나머지 부분을 다른 블록에 저장할 수 있다.

bfr many records per block을 사용할 수 있다.

$$b = \lceil r/bfr \rceil \text{ blocks } (r: \text{file} \text{ read } \ell)$$

Q How many blocks are needed to store a file of 100 reads?

Record size : 16 bytes

Block size : 4KB bytes

Unused space : 0 bytes

bfr ?

$$0 = 4096 - (bfr * 16)$$

$$bfr = 4096 / 16 = 256$$

Record and File

- 파일은 데이터 값이나 항목들로 구성된 레코드의 집합
- 파일 내의 레코드들이 모두 같을 때 이 파일이 고정 길이 레코드들로 구성되어 있다고 말한다.
- 파일 내의 레코드들의 크기가 서로 다를 때 이 파일은 가변 길이 레코드들로 구성되어 있다고 말한다.
- 가변 길이 레코드 파일에서는 가변 길이 필드의 바이트 수를 결정하기 위해 분리 문자를 사용하거나, 필드값 앞에 가변 길이 필드의 바이트 길이를 기록한다.
- 레코드를 하나 이상의 블록에 걸쳐서 저장하는 방식을 신장(spanned)조직이라 한다
- 레코드 하나의 크기가 한 블록의 크기보다 클 경우 신장조직을 사용하여야 한다.
- 레코드가 블록 경계를 넘지 않도록 저장하는 방식을 비신장(unspanned) 조직이라 한다.
- 가변길이 레코드의 경우에는 신장 또는 비신장 조직을 사용할 수 있다.
- 평균 레코드 크기가 클 경우에는 각 블록내의 손실 공간을 줄이기 위하여 신장조직을 사용하는 것이 유리하다.

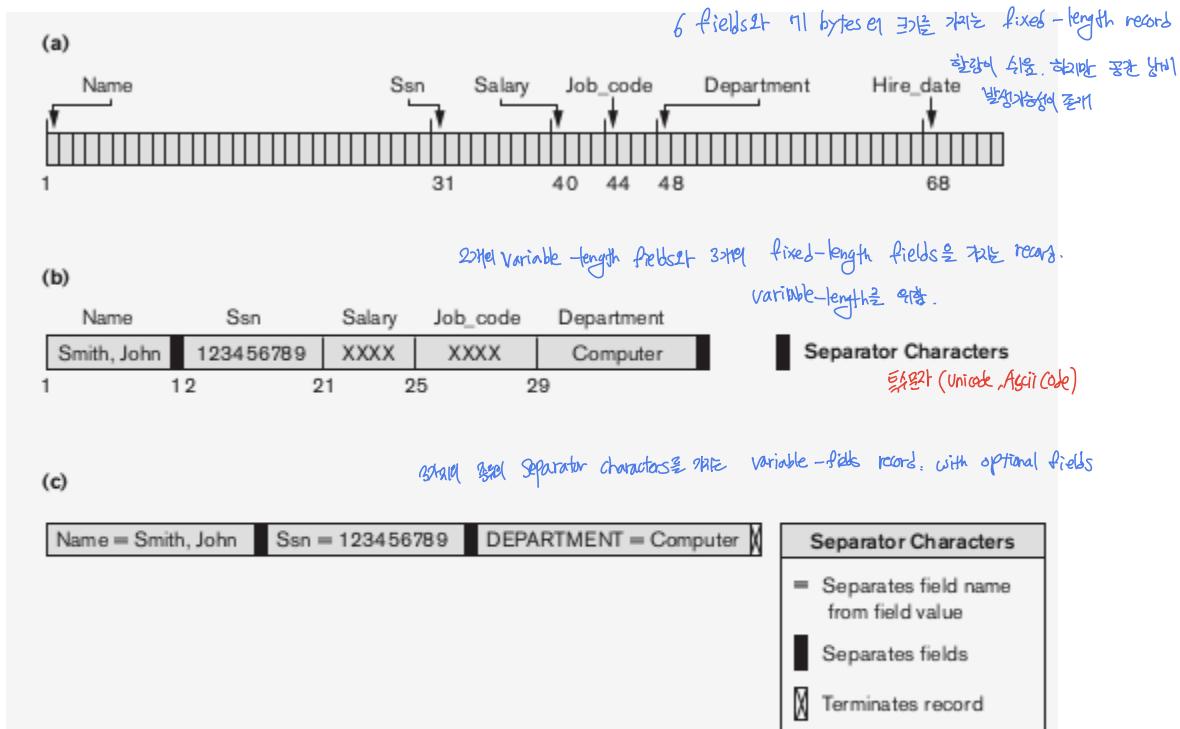


Figure 16.5

Three record storage formats. (a) A fixed-length record with six fields and size of 71 bytes. (b) A record with two variable-length fields and three fixed-length fields. (c) A variable-field record with three types of separator characters.

record의 크기가 block보다 작은 경우

pointer를 사용하지 않아 빠르지만, 공간이 낭비된다.

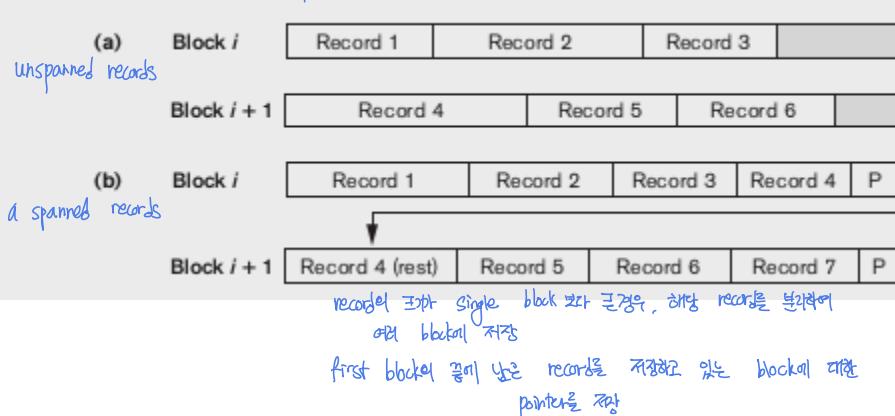


Figure 16.6

Types of record organization.

(a) Unspanned.

(b) Spanned.

file의 records는 무조건 disk blocks에 할당되어야 한다.

• block은 disk와 memory 사이의 data transfer의 기본위

• block의 크기가 record의 크기와 크거나 같다면, 한 block에 여러 개의 records가 저장될 수 있다

화일에 대한 연산

65/8 처음보는거

- 한 번에 한 레코드 연산
 - Open : 판독과 기록을 위해 화일을 개방하고, 디스크로부터 파일 블록들을 유지하기 위해 적절한 버퍼들을 할당한 후 파일 헤더를 검색한다. 파일 포인터를 파일의 시작 위치로 설정한다.
 - Reset : open된 파일의 파일 포인터를 파일의 시작 위치로 설정한다.
 - Find : 조건을 만족하는 첫 번째 레코드를 탐색하고, 탐색된 레코드를 현재 레코드로 지정한다.
 - FINDNEXT : 현재 레코드로부터 조건을 만족하는 다음 레코드를 탐색하고, 탐색된 레코드를 현재 레코드로 지정한다.
 - READ : 현재 레코드를 프로그램 변수로 복사한다.
 - Delete : 현재 레코드를 삭제하고, 파일을 갱신한다.
 - Modify : 현재 레코드의 필드값을 수정하고, 파일을 갱신한다.
 - Insert : 새로운 레코드를 파일에 삽입한다.
 - Close : 버퍼 해제와 기타 필요한 제거 연산을 수행하여 파일 접근을 완료한다.
- 한 번에 레코드들의 집합 연산 : 데이터베이스 시스템에서 사용하는 고수준의 연산
 - FindAll : 탐색 조건을 만족하는 파일 내의 모든 레코드를 찾는다.
 - Find n : 검색 조건을 만족하는 첫 번째 레코드를 검색한 후 동일한 조건을 만족하는 나머지 n-1개의 레코드를 연속해서 찾는다.
 - FindOrdered : 특정 순서대로 파일 내의 모든 레코드를 검색한다.
 - Reorganize : 파일을 재조직한다.
- **화일 조직(File Organization)**은 기억 장치 매체상에 레코드와 블록들을 저장하고 서로 연결시키는 방법.
- **접근 방법(Access Method)**은 파일에 적용할 수 있는 연산들의 그룹을 제공 - scan or indexed access
- 파일에 자주 사용되는 연산들을 가능한 한 효율적으로 수행해야 성능이 우수한 파일 조직이라 하겠다.

HEAP FILE

file의 어느곳이든 record를 저장 가능

- 비순서(unordered) 파일. pile이라 불리기도 함.
- 새로운 레코드는 파일의 마지막에 삽입하므로 **삽입은 매우 효율적이다**.
- 레코드를 탐색하기 위해서는 선형 탐색 기법을 사용해야 한다. 이 기법은 **평균 파일 블록의 반**을 탐색하기 때문에 비효율적이다.
- 어떤 필드값의 순서대로 레코드를 판독하기 위해서는 해당 파일의 레코드들을 정렬시켜야 한다.
- 비저장 블록과 연속할당 방식을 사용하는 비순서 고정길이 레코드들로 구성된 파일에서는 레코드의 위치(순번)을 이용하여 레코드를 접근할 수 있다. 이런 방식으로 접근할 수 있는 파일을 상대 파일이라고 한다.

un internal sorting

외부 External "

Oracle create table syntax

```
CREATE TABLE [schema.]tablename
  ((columnname datatype [DEFAULT (default_constant|NULL)] [col_constr {col_constr. . .}])
   | table_constr) -- choice of either columnname-definition or table_constr
  [, (columnname (repeat DEFAULT clause and col_constr list) | table_constr) . . .])
  [ORGANIZATION HEAP | ORGANIZATION INDEX (this has clauses not covered)]
  [TABLESPACE tblspacename]
  [STORAGE ([INITIAL n [K|M]] [NEXT n [K|M]] [MINEXTENTS n] [MAXEXTENTS {n|UNLIMITED}]
            [PCTINCREASE n]) (additional STORAGE options not covered)]
  [PCTFREE n] [PCTUSED n]
  [disk storage and other clauses (not covered, or deferred)]
  [AS subquery]
```

ORACLE Relational Create Table Statement Syntax with Partial Indexing Syntax

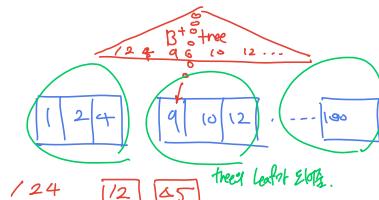
Ordered File

각 record Search key 값에 따라 순차적으로 records를 저장

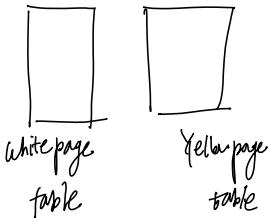
정렬을 위한 key field 필요

이진 탐색 Binary Search (key)

- 순서 파일. 순차화일 (sequential file)로 불리우기도 한다.
- 파일 내의 레코드들을 순서 필드의 값에 따라 정렬된다.
- 레코드들이 정렬된 형태로 추가되어야 하기 때문에 삽입 비용이 비싸다. 더욱 효율적으로 작업하기 위해 오버플로 파일 또는 트랜잭션 파일을 사용한다. 단, 이 방식은 정기적으로 주 파일에 합병시켜야 한다.
- 이진 탐색은 레코드의 순서 필드값에 따른 탐색의 경우 사용된다.
- 평균적으로 $\log_2 b$ 개의 블록을 접근하는 이진 탐색은 선형 탐색보다 더 좋은 성능을 보인다.** (b는 블록의 총 개수).
- 정렬된 키 값의 순서대로 판독하는 연산이 매우 효율적이다. (why?)
- (dynamic file) 레코드 삽입과 삭제는 레코드를 물리적으로 정돈된 상태를 유지해야 하므로 비용이 매우 크다. (static file인 경우, no problem. ISAM)
- 순서 파일 방식에서 비순서 필드 값을 기반으로 레코드들을 임의 접근하거나 순서 접근하기 어렵다. 이 경우 임의 접근 (random access)은 선형 탐색을 사용하고, 순서 접근(sequential access)은 파일을 정렬한 복사본을 사용해야 한다.
- 삽입을 효율적으로 수행하기 위해 오버플로 파일 또는 트랜잭션 파일이라는 임시 비순서 파일에 기록한 후 주 파일 또는 마스터 파일과 정기적으로 합병**
- 레코드 삭제시 빈 공간을 메우기 위해 레코드들을 이동하거나 삭제 표시자를 사용할 수 있다. 생략하면 맹거야짐. 시간이 되면 merge 할 때 처리
- 순서 필드 값을 수정하는 연산은 그 레코드를 삭제하고 새로운 순서 필드 값을 갖는 레코드를 삽입한다. 포함해도 찾는게 merge
- 기본 인덱스라고 부르는 부가적인 접근 경로와 함께 사용되는 것이 일반적이다. (Oracle primary index)



white page and Yellow page example
only one ordered file



정렬되려면 잎은 꼭. 블록의 개수만큼 끊어져야함
정렬되려면 잎이 많을경우 레코드 개수만큼 끊어져야함



ordering key : fileid key field or ordering field

NAME 필드를 순서키로 갖는 EMPLOYEE 레코드들로 구성된 순서 화일의 일부 목록

정렬되어져 있을 경우

	Name	Ssn	Birth_date	Job	Salary	Sex
Block 1	Aaron, Ed					
	Abbott, Diane					
		:				
	Acosta, Marc					
Block 2	Adams, John					
	Adams, Robin					
		:				
	Akers, Jan					
Block 3	Alexander, Ed					
	Alfred, Bob					
		:				
	Allen, Sam					
Block 4	Allen, Troy					
	Anders, Keith					
		:				
	Anderson, Rob					
Block 5	Anderson, Zach					
	Angeli, Joe					
		:				
	Archer, Sue					
Block 6	Arnold, Mack					
	Arnold, Steven					
		:				
	Atkins, Timothy					
		:				
Block $n-1$	Wong, James					
	Wood, Donald					
		:				
	Woods, Manny					
Block n	Wright, Pam					
	Wyatt, Charles					
		:				
	Zimmer, Byron					

Figure 16.7

Some blocks of an ordered (sequential) file of EMPLOYEE records with Name as the ordering key field.

heap files와 비교하여 가지는 단점

정렬의 필요 X

다음 record 찾는데 block의

마지막 record가 아니면 초기화한
block access가 필요해 같다.

(정렬되어 있기 때문에 다음 record
는 같은 block 위에 존재)

binary search 가능

기본 파일 구조에 대한 평균 접근 시간

TABLE 13.2 AVERAGE ACCESS TIMES FOR BASIC FILE ORGANIZATIONS

TYPE OF ORGANIZATION	ACCESS/SEARCH METHOD	AVERAGE TIME TO ACCESS A SPECIFIC RECORD
Heap (Unordered)	Sequential scan (Linear Search)	$b/2$
Ordered	Sequential scan	$b/2$
Ordered	Binary Search	$\log_2 b$

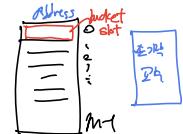
$$10^3 = 2^{10} \quad \log_2 10^6 = \log_2 2^{10} \cdot 2^{10} = 20$$

20번의 접근이 필요함

Optimal search

File organization + Access Method

Hash h(key)



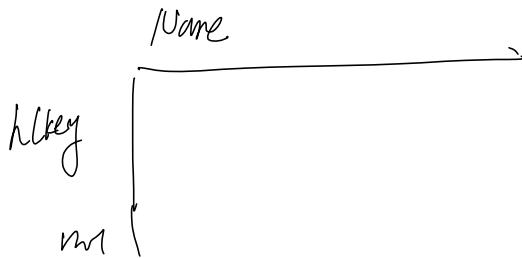
Hashing

hash function을 사용하여 해시 값에 따라 record 위치

$h(k) \rightarrow v$

randomize

- 해싱을 기반으로 조직된 파일을 해시 파일 또는 직접 파일(direct file)이라고 한다.
- 해시 함수 또는 난수화 함수라고 하는 함수 h 를 레코드의 해시 필드값에 적용하여 레코드를 저장하고 있는 디스크 블록의 주소를 산출한다.
- 대부분의 레코드에 대해 한 번의 블록 접근으로 원하는 레코드를 검색하는 매우 효율적인 방식이다.
- 파일의 키 필드로 해시 필드를 사용할 경우 이를 해시키라고 한다.
- 프로그램에서 어떤 필드값을 사용하여 레코드들의 그룹을 배타적으로 접근하고자 할 때 내부 탐색 구조로 해싱을 사용한다.
- 내부 파일에서 해싱은 일반적으로 레코드들의 배열을 이용하여 해시 테이블로 구현한다.
- 해시 필드 공간이 주소 공간보다 매우 크기 때문에 대부분의 해시 함수는 서로 다른 해시 필드값을 항상 서로 다른 주소로 사상하지는 못하는 문제점이 있다. \Rightarrow Collision
- 삽입하려는 새로운 레코드의 해시 필드 값이 다른 레코드가 이미 점유하고 있는 주소로 해싱될 때 충돌(collision)이 발생. 이 경우 새로운 레코드를 삽입할 다른 위치를 찾아야하는데 이를 충돌 해결이라 함.
- 해시 테이블을 70%에서 90% 정도 사용하도록 유지할 때 충돌 횟수도 줄이고, 공간 낭비도 줄일 수 있다.
- 충돌(Collision) 해결 기법
 - 개방 주소 지정(Open addressing): 해시 주소가 가리키고 있는 위치부터 시작하여 빈 위치를 찾을 때까지 순차적으로 그 이후의 위치들을 조사한다.
 - 체인(Chaining): 오버플로 영역에 새로운 레코드를 저장하고 다른 레코드가 이미 점유하고 있는 해시 주소 영역의 포인터에 오버플로 영역의 주소를 지정하여 충돌을 해결한다.
 - 다중 해싱(Multiple hashing): 충돌이 발생하지 않을 때까지 여러 개의 해시 함수를 적용한다. 충돌 시 필요하면 개방 주소 지정 방법을 사용한다.



(a)

Name Ssn Job Salary

Figure 16.8

Oracle Hash Cluster Syntax

```
CREATE CLUSTER [schema.]clusternname
  (columnname datatype [, columnname datatype ...] -- this is the cluster key
   [cluster_clause { cluster_clause ...}]);
```

The cluster_clauses that specify cluster characteristics are chosen from the following:

```
[PCTFREE n] [PCTUSED n]
[STORAGE ([INITIAL n [K|M]] [NEXT n [K|M]] [MINEXTENTS n] [MAXEXTENTS n]
          [PCTINCREASE n])]

[SIZE n [K|M]]                                     -- defaults to one disk block
[TABLESPACE tblspacename]
[INDEX | [SINGLE TABLE] HASHKEYS n [HASH is expr]]
[other clauses not covered];
```

이것은 디스크 파일에 대한
슬롯의 개수 또는 크기를 정하는
문장입니다.

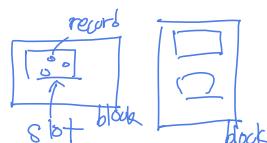
To delete an existing cluster, the DROP CLUSTER statement is used:

```
DROP CLUSTER [schema.]clustername [INCLUDING TABLES [CASCADE CONSTRAINTS]];
```

ORACLE Create Cluster Statement Syntax

```
CREATE TABLE [schema.]tablename
  (column definitions as Basic SQL, see Figure 7.1 or Figure 8.5)
  CLUSTER clusternname (columnname [, columnname...]) -- table columns map to cluster key
  [AS subquery];
```

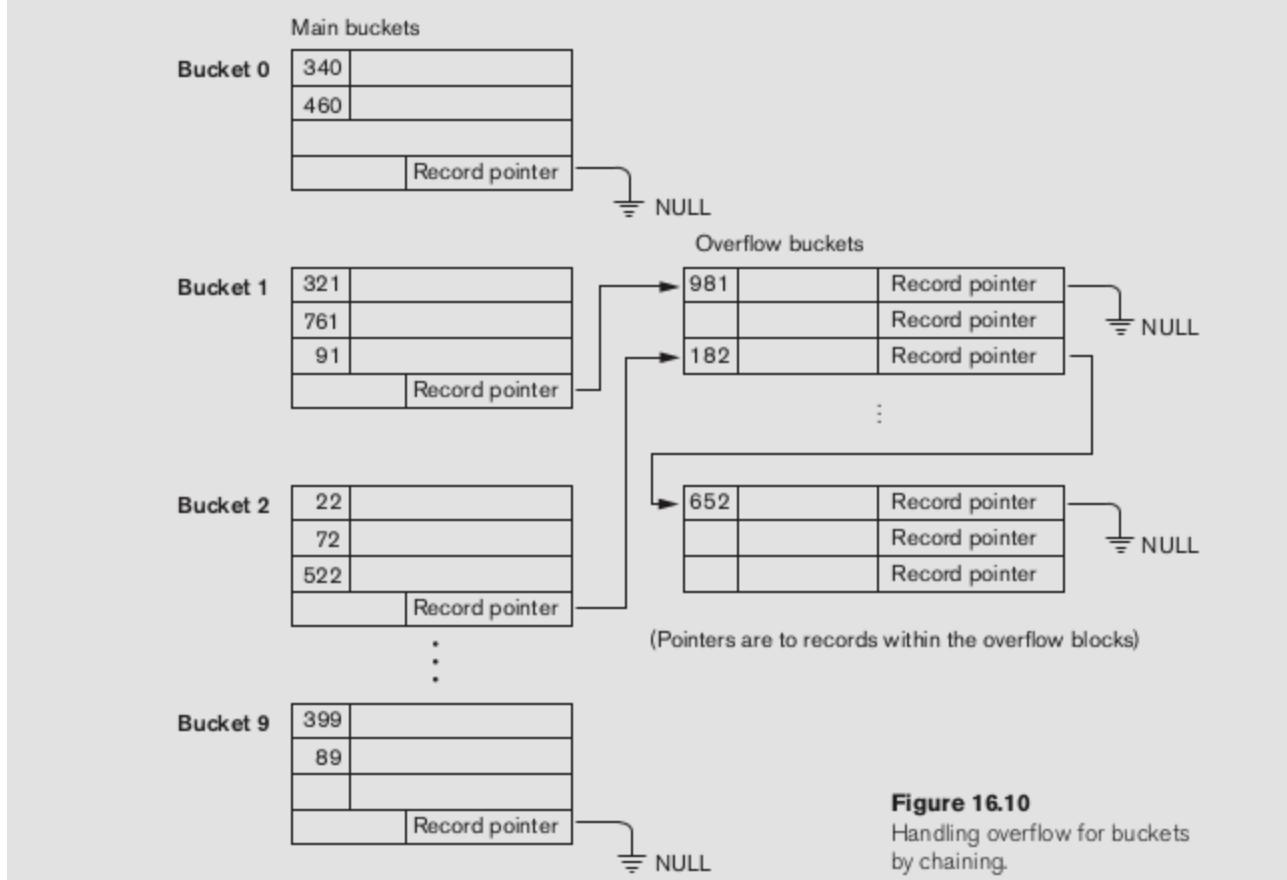
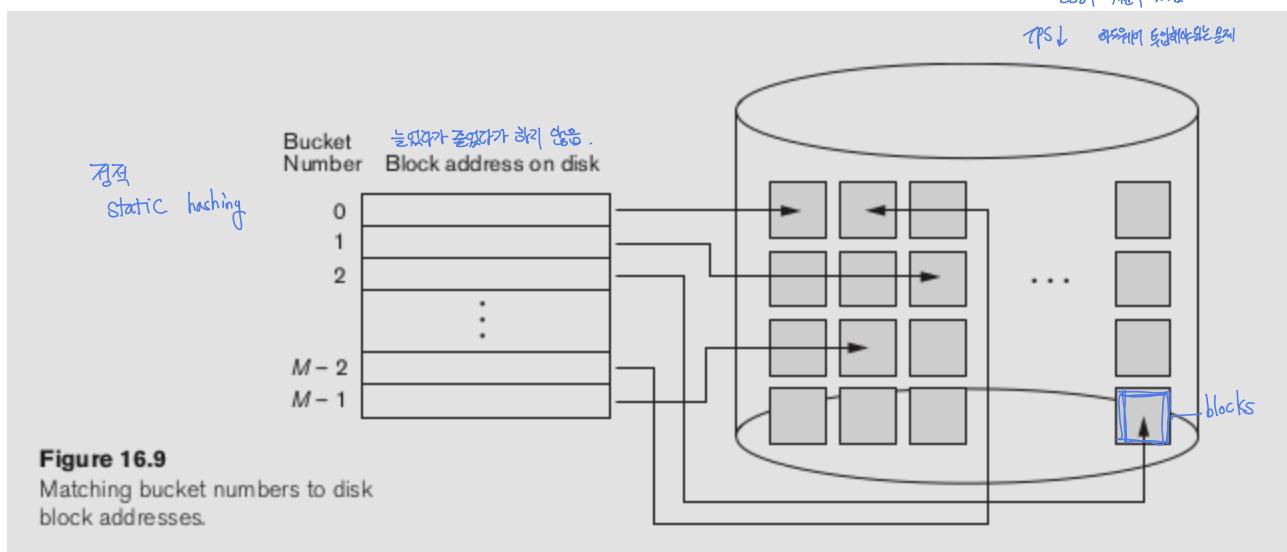
Create Table Statement for Table to Be Contained in a Cluster



Hashing File

- 목표 주소 공간을 같은 크기의 버켓들로 구성하고, 각 버켓에는 다수의 레코드들을 유지함.
- 버켓은 디스크 블록 한 개나 연속적인 블록인 클러스터 한 개의 크기를 갖는다.
- 해시 키가 K 인 레코드는 해시 함수 $i = h(K)$ 에 의해 결정된 버켓 i 에 저장
- 가득 찬 버켓으로 새로운 레코드가 해시 될 경우 충돌이 발생되며 충돌된 레코드는 오버플로우(Overflow) 버켓에 저장, 각 버켓에서 오버플로우된 레코드들은 연결 리스트로 구성
- 단점
 - 버켓의 수가 고정되어 있기 때문에 파일의 레코드수가 버켓 공간에 비해 너무 적거나(공간 낭비 초래) 너무 많으면 (빈번한 충돌 발생) 문제가 발생한다.
 - 해시 키에 따른 순차적 접근은 매우 비효율적이며 레코드의 정렬이 요구된다.

디스크 I/O가
extra 밸류, 충돌이
일어나므로
TPS ↓ 해도 유리한 방법이 있는지



Types of Query

1. Point Query

```
SELECT balance  
FROM accounts  
WHERE number = 1023;
```

2. Multipoint Query

```
SELECT balance  
FROM accounts  
WHERE branchnum = 100;
```

3. Range Query

```
SELECT number  
FROM accounts  
WHERE balance > 10000;
```

4. Join Query

```
SELECT *  
FROM accounts, branch  
WHERE a.branchnum = b.branchnum;
```

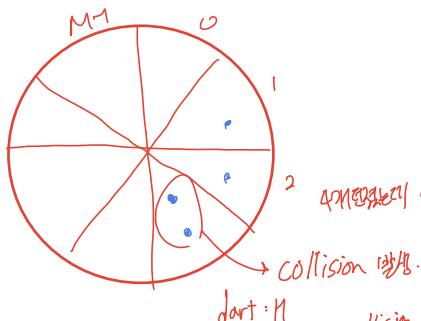
equijoin accounts \bowtie branch

Hashing File은 어떤 질의에 효과적일까? Why?

Mysql의 selfjoin은 Hash join
hash join 한정

Throwing Darts at Random slots

- number of darts : N
- number of random slots : M
- How many slots on the board have a dart in them?



$$\text{collision rate} = \frac{N-1}{N}$$

충돌은 예상한 상태로 충돌

예상과 차이점은 충돌을 빼면 → 실제 충돌수

365-24*365 = 329 static hashing 개수
충돌

$(1 - \frac{1}{M})^N$ N개를 충돌을 예상한 확률

충돌 확률은 M-1개의 충돌을 예상한 확률

$$M \cdot (1 - \frac{1}{M})^N : \text{기대값}$$

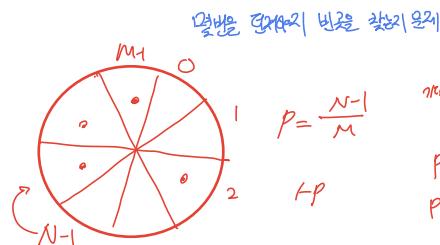
$$M - M \cdot (1 - \frac{1}{M})^N : M-1개의 충돌$$

충돌 확률

$$= M \left(1 - \left(1 - \frac{1}{M} \right)^N \right) = \frac{N-M(e^{-1})}{N} = 1 - e^{-1}$$

Slot Occupancy of ONE : number of Retries (Rehash chain)

- One row in any single slot
- Rehash the key value to a sequence of slots until an empty slot is located
- Dart version:
 - N darts in M slots
 - Slot occupancy 1
 - Retries until all darts in board
 - What is the expected number of retries?



$$\begin{aligned}
 f(p) &= 1+2p+3p^2+\dots \\
 -p f(p) &= p+2p^2+\dots \\
 (\cancel{p}) f(p) &= 1+p+p^2+\dots \\
 &= \frac{1}{1-p} \quad f(p) = \frac{1}{(1-p)^2} \\
 P(1) &= 1-p \\
 P(2) &= p \cdot (1-p) \\
 P(3) &= p \cdot p \cdot (1-p) \\
 &\vdots \\
 P(n) &= p^{n-1} \cdot (1-p)
 \end{aligned}$$

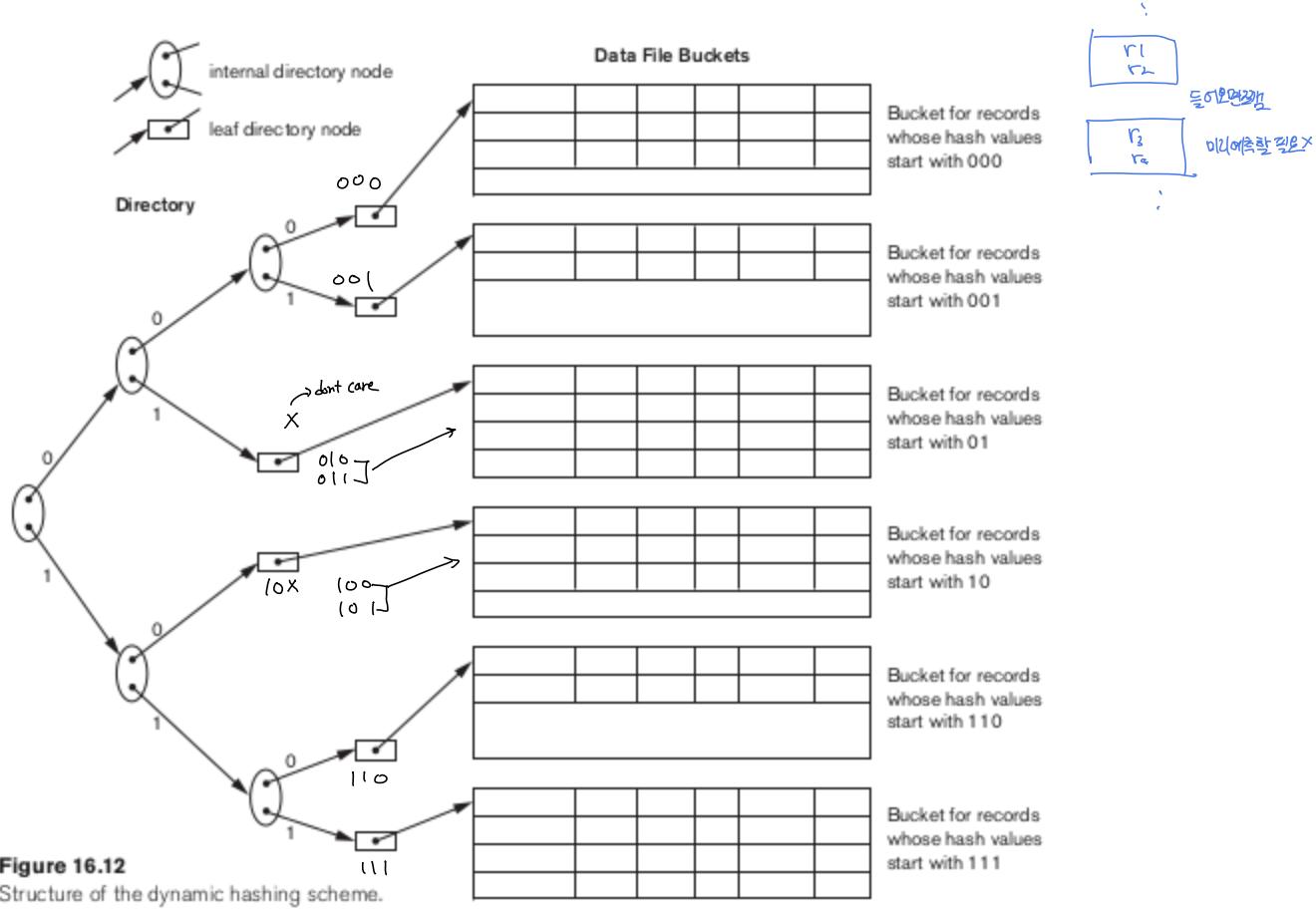
Expected length = $1 \cdot P(1) + 2 \cdot P(2) + \dots$

동적(dynamic) 및 확장가능(extendable) 해싱 화일

- 정적 해싱은 해시 주소 공간이 고정되어 화일을 동적으로 확장하고 축소하는 것이 어렵다. 이 문제를 해결하기 위한 기법으로는 확장가능 해싱, 선형 해싱이 있다.
- 이런 해싱 기법들은 해시 함수 결과 (아래에서 K로 표시함)를 이진수로 표현할 수 있고, 이진수 표현을 기반으로 접근 구조를 구성한다. 이런 이진수 표현을 레코드에 대한 해시값이라 한다. 이진수 표현은 비트열로 구성된다. 레코드 해시값에서 선행하는 비트들의 값을 기반으로 레코드들을 버켓들에 분배한다.

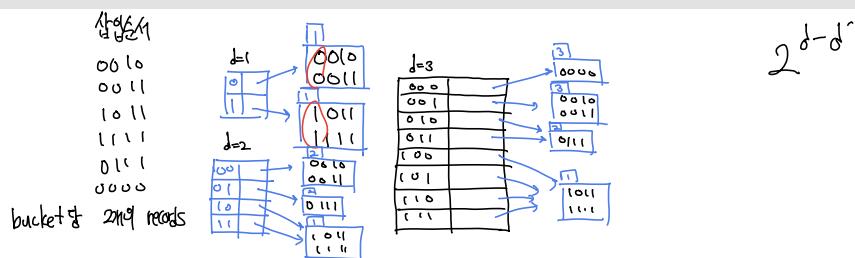
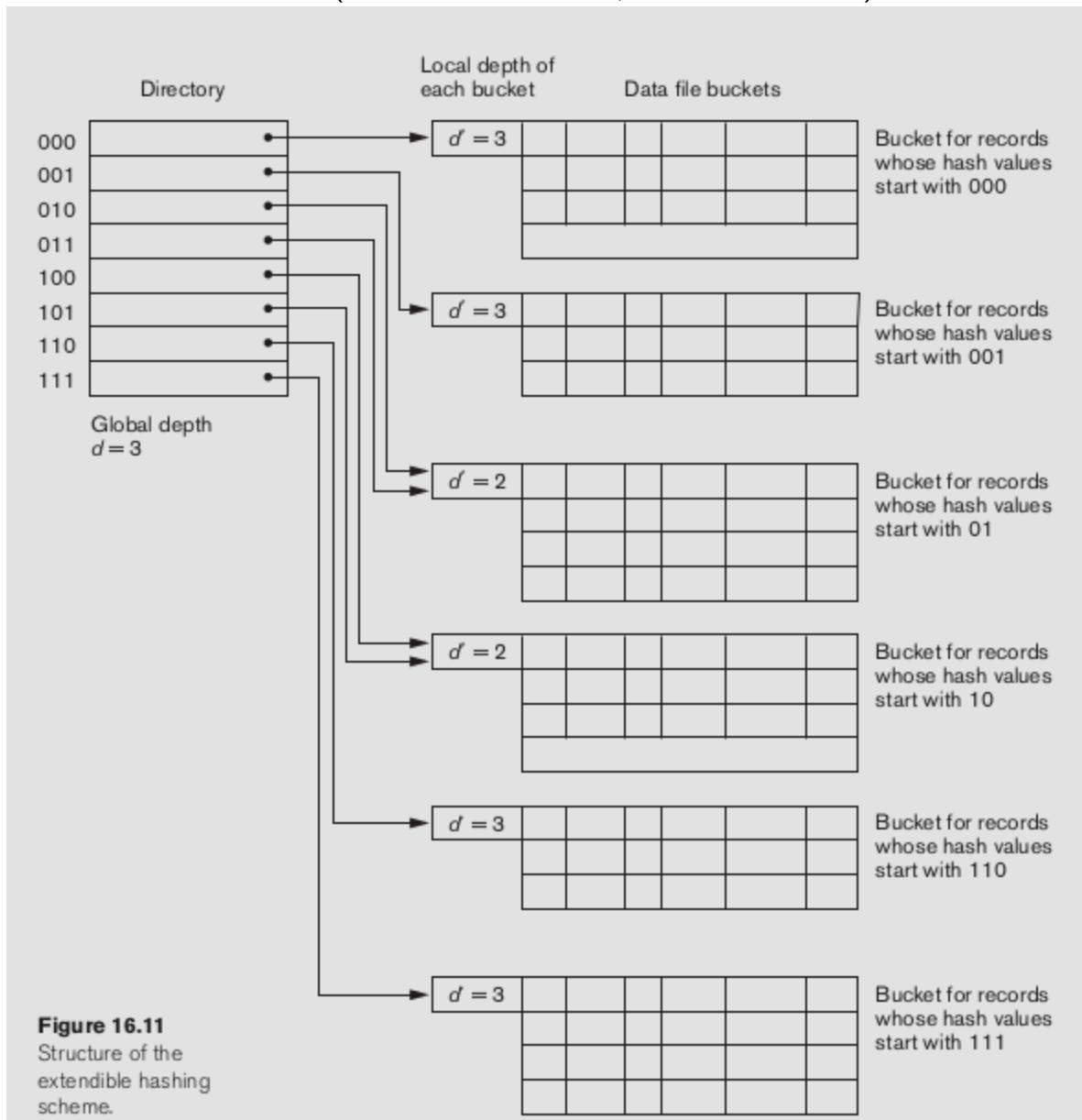
동적 해싱(Dynamic Hashing)

- 확장가능해싱과의 차이점은 디렉토리를 구성하는 방식이다.
- 두 개의 포인터를 갖는 내부 노드는 (해시된 주소에서) 0비트에 해당하는 왼쪽 포인터와 1비트에 해당하는 오른쪽 포인터를 가진다.
- 디스크 기반 자료구조가 아니다.
- 이를 변형하여 디스크 기반 자료구조로 만든 것이 확장가능해싱이다.
- 트리는 "거의" 균형트리다. Why?



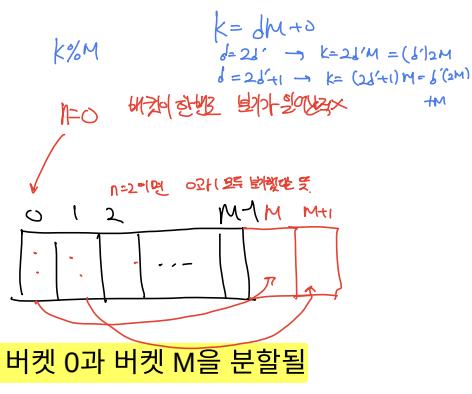
확장가능 해싱(Extendable Hashing)

- 디렉토리는 2^d 개의 버켓 주소를 갖는 배열이다. 이때 d 는 전역 깊이(global depth)라 하며, 해시 값의 처음 d 개 비트를 디렉토리 배열의 인덱스 값을 결정하는데 사용한다.
- d 개 디렉토리 엔트리들은 서로 다른 버켓 주소를 유지할 필요는 없다.
- 처음 d' 개의 비트 값이 갖는 레코드가 하나의 버켓에 저장될 수 있으면 여러 디렉토리 엔트리가 같은 버켓 주소를 유지한다. 각 버켓내의 레코드가 기반으로 하는 비트 수 d' 을 지역 깊이(local depth)라고 한다.
- 지역 깊이 d' 과 전역 깊이 d 가 같은 버켓에서 오버플로가 발생할 경우 디렉토리 배열 내의 엔트리 수는 2배가 된다.
- 어떤 레코드를 삭제한 후, 모든 버켓에 대해 $d > d'$ 인 경우
 - 디렉토리 배열내의 엔트리 수는 절반이 된다.
- 대부분의 레코드 검색은 두 개의 블록 접근(디렉토리에 대한 블록접근, 버켓에 대한 블록 접근)을 필요로 한다.



선형 해싱(Linear Hashing)

- 디렉토리를 사용하지 않고, 해시 파일의 버켓수를 동적으로 늘리거나 줄인다.
- 초기에는 M 버켓 사용: 0, 1, ..., M - 1
- 초기 해시 함수 $h_i(K) = K \bmod M$
- 계속되는 충돌로 인해 오버플로 레코드가 발생하면, 버켓 0, 1, 2, ..., n 순서로 분할한다.
- 버켓 0은 버켓 0과 버켓 M으로 분할한다. 버켓 1은 버켓 1과 버켓 M+1으로 분할한다. ...
 - 중요한 특성:** h_i 를 기반으로 하여 버켓 0에 해시된 모든 레코드는 h_{i+1} 을 기반으로 하여 버켓 0과 버켓 M을 분할될 수 있다. why?
- 해시 함수는 다음과 같다: $h_{i+1}(K) = K \bmod 2M$
- $h_i(K) < n$ 이면 해시 함수는 h_{i+1} 이고, 그렇지 않으면 해시 함수는 h_i 이다.
- j번 분할하면 해시 함수는 다음과 같다: $h_{i+j}(K) = K \bmod (2^j M)$
- 특정 버켓의 오버플로우가 발생할 때마다 분할하지 않고, 다음과 같이 분할할 수도 있다.
 - 적재인수 $r/(bfr * N)$ 임계값(예, 0.9)을 넘어가면 분할한다.
 - 적재인수가 임계값(예, 0.9)을 넘어가면 합병한다)



Algorithm 16.3. The Search Procedure for Linear Hashing

```

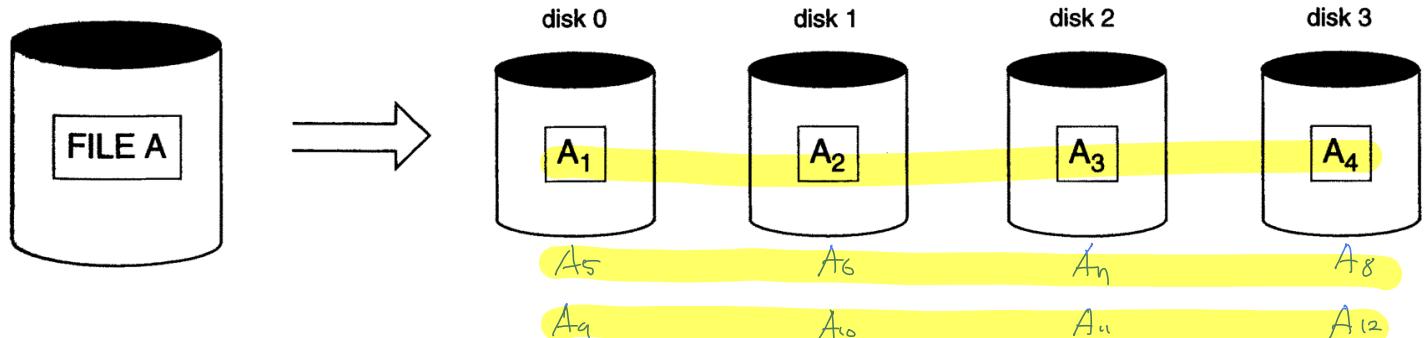
if  $n = 0$ 
  then  $m \leftarrow h_j(K)$  (*  $m$  is the hash value of record with hash key  $K$ *)
else begin
   $m \leftarrow h_j(K)$ ;
  if  $m < n$  then  $m \leftarrow h_{j+1}(K)$ 
end;

```

search the bucket whose hash value is m (and its overflow, if any);

RAID 기술을 이용한 병렬 디스크 접근

- 보조 기억 장치 기술의 성능과 신뢰도를 프로세서 기술의 수준으로 올려야 한다.
- 보조 기억 장치 기술의 주요한 발전은 RAID(Redundant Arrays of Inexpensive Disks)의 개발로 대표된다.
- RAID의 주요 목표는 메모리와 마이크로프로세서의 성능 향상과 균형을 맞출 수 있도록 디스크의 성능을 획기적인 비율로 향상시키는 데 있다.
- 자연스러운 해결책은 여러 개의 작고 독립적인 디스크를 배열로 구성하여 하나의 고성능 디스크처럼 동작하도록 하는 것이다. 여기에는 디스크의 성능 향상을 위해 병렬성을 사용하는데, 이 개념을 데이터 스트라이핑(data striping)이라 한다.
- 데이터 스트라이핑은 여러 개의 디스크가 하나의 크고 빠른 디스크처럼 보이도록 데이터를 다중 디스크로 투명하게 분산시킨다.
- Data striping: 파일 A가 네 개의 디스크에 스트라이핑



Reliability

- 신뢰성을 향상시키는 첫 번째 방법은 데이터를 두 개의 동일한 물리적 디스크에 중복해서 기록하는 반사(mirroring) 또는 그림자(shadowing) 기법을 사용하는 것이다.
- 두 번째 방법은 디스크 오류 발생시 손실된 정보를 재구축하는 데 사용할 부가 정보를 저장하는 것이다.
 - 여분의 정보를 계산하기 위해 패리티 비트나 해밍 코드 같은 특별한 코드를 포함한 에러 검출 코드를 사용한다.
 - 중복된 정보를 디스크 배열상에 분산시키기 위해 몇 개의 디스크에 여분의 정보를 저장하는 방식과, 모든 디스크에 균등하게 여분의 정보를 저장하는 방식이 있다. 두 번째 방식이 더 좋은 부하 균등을 제공한다.

Data striping granularity

- Bit-level data striping : 데이터의 각 바이트를 비트들로 분할하여 비트들이 서로 다른 디스크에 분산함으로 더 작은 단위를 데이터 전송에 사용하는 방식이다.
- Block-level data striping : 데이터를 분산하는 단위로 파일의 블록을 사용하는 방식이다. 하나의 블록에 다수의 요청을 각 디스크에 의해 병렬로 처리할 수 있어서, I/O 요청 대기 시간이 줄어든다.

RAID Level

- RAID의 구조는 데이터 분산(스트라이핑)의 단위와 여분의 정보를 계산하는 데 사용되는 패턴의 두 가지 요인의 조합 방법에 따라 서로 구별된다.
 - RAID 레벨 0은 여분의 데이터를 갖지 않아서 데이터의 간신히 중복되지 않으므로 가장 좋은 쓰기 성능을 가진다.
 - RAID 레벨 1은 디스크들을 복사한다.
 - RAID 레벨 5는 블록 레벨 데이터 스트라이핑을 이용하며, 데이터와 패리티 정보를 모든 디스크에 분산시킨다.
- (a) RAID Level 1 (b) RAID Level 5

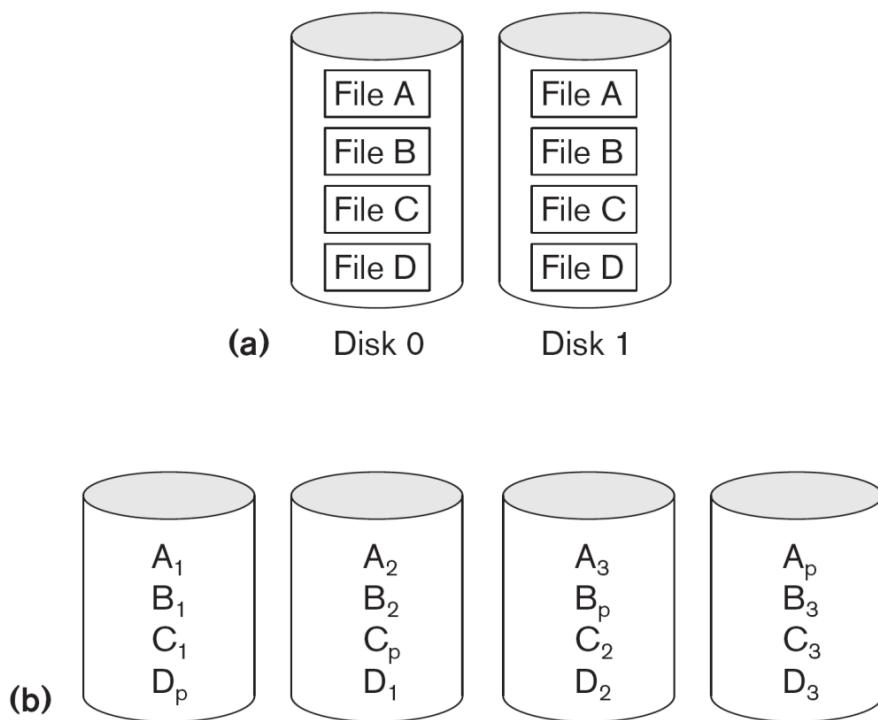


Figure 16.14

Some popular levels of RAID.
 (a) RAID level 1: Mirroring of data on two disks. (b) RAID level 5: Striping of data with distributed parity across four disks.

- RAID 레벨 1에서 디스크 오류를 복구하는 것이 가장 쉽다.
 - **레벨 1은 로그 저장 같은 중요한 응용을 위해 사용한다.**
- RAID 레벨 5는 대용량의 데이터 저장을 위해 주로 사용되며, 높은 전송률을 제공한다.
- 현재 가장 널리 사용되고 있는 RAID 기술은 스트라이핑을 지원하는 레벨 0, 반사 기능을 가진 레벨 1, 패리티를 위한 추가적인 디스크를 가진 레벨 5이다.
- 주어진 여러 응용을 위한 RAID 설정을 설계하기 위해서는 RAID의 레벨, 디스크의 수, 패리티 기법의 선택, 블록 레벨 스트라이핑을 위한 디스크 그룹핑 방법 등 여러 가지 사항을 고려해야 한다.

