

11_강 해싱과 특수 인덱스

■ ■ ■ ■ 컴퓨터과학과 정재화

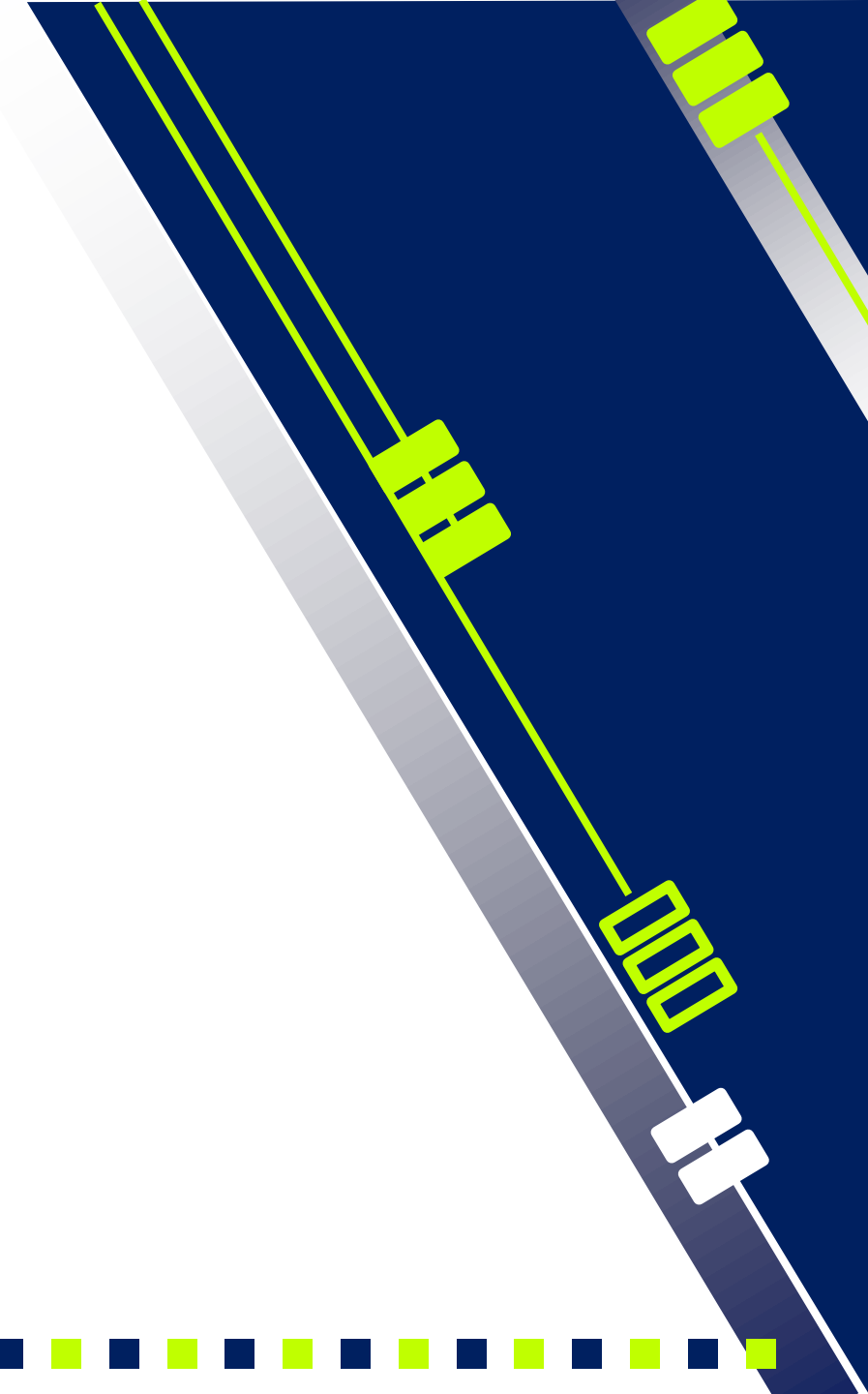
목 차



01. 정적 해싱

02. 동적 해싱

03. 비트맵 인덱스



01

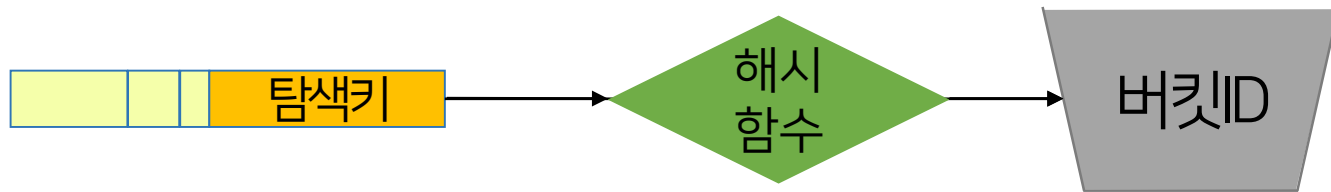
■ ■ ■ ■ 11강 해싱과 특수 인덱스

정적 해싱

- 해시의 구조
- 해시 파일 구조
- 해시 인덱스

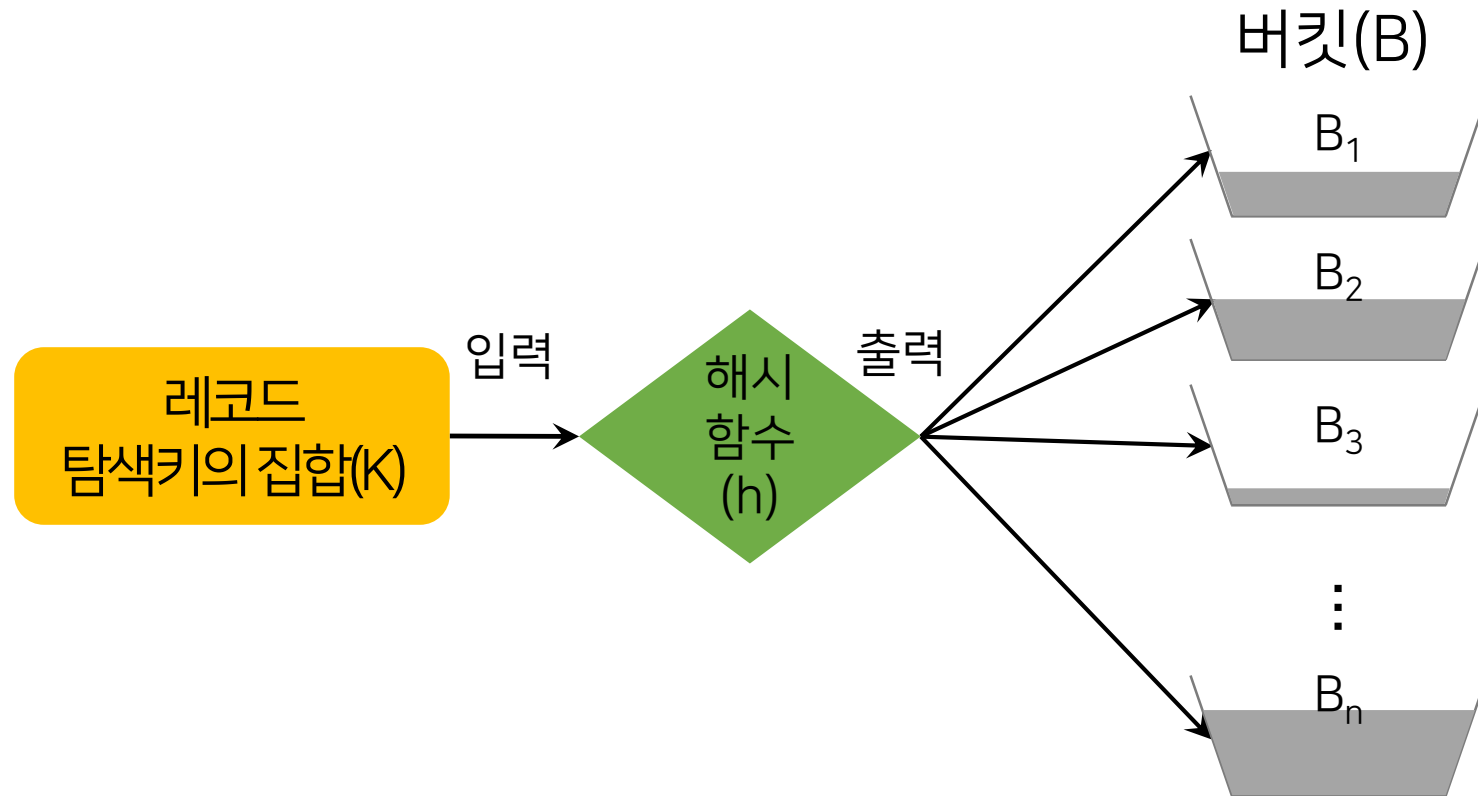
1. 해시(hash)

- ▶ 탐색키에 산술적인 연산을 통해 버킷의 주소를 계산하는 해시 함수를 사용하여 데이터 배분 및 접근하는 기법

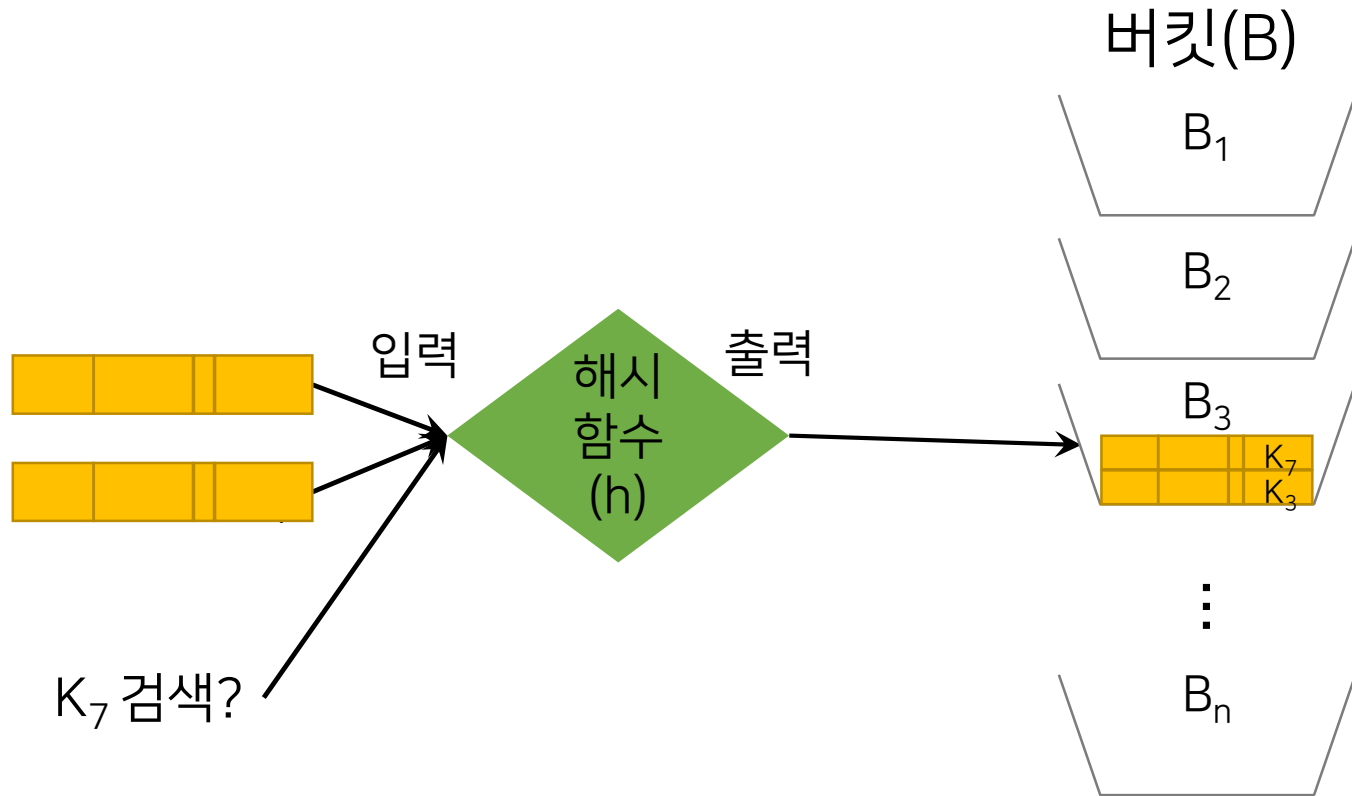


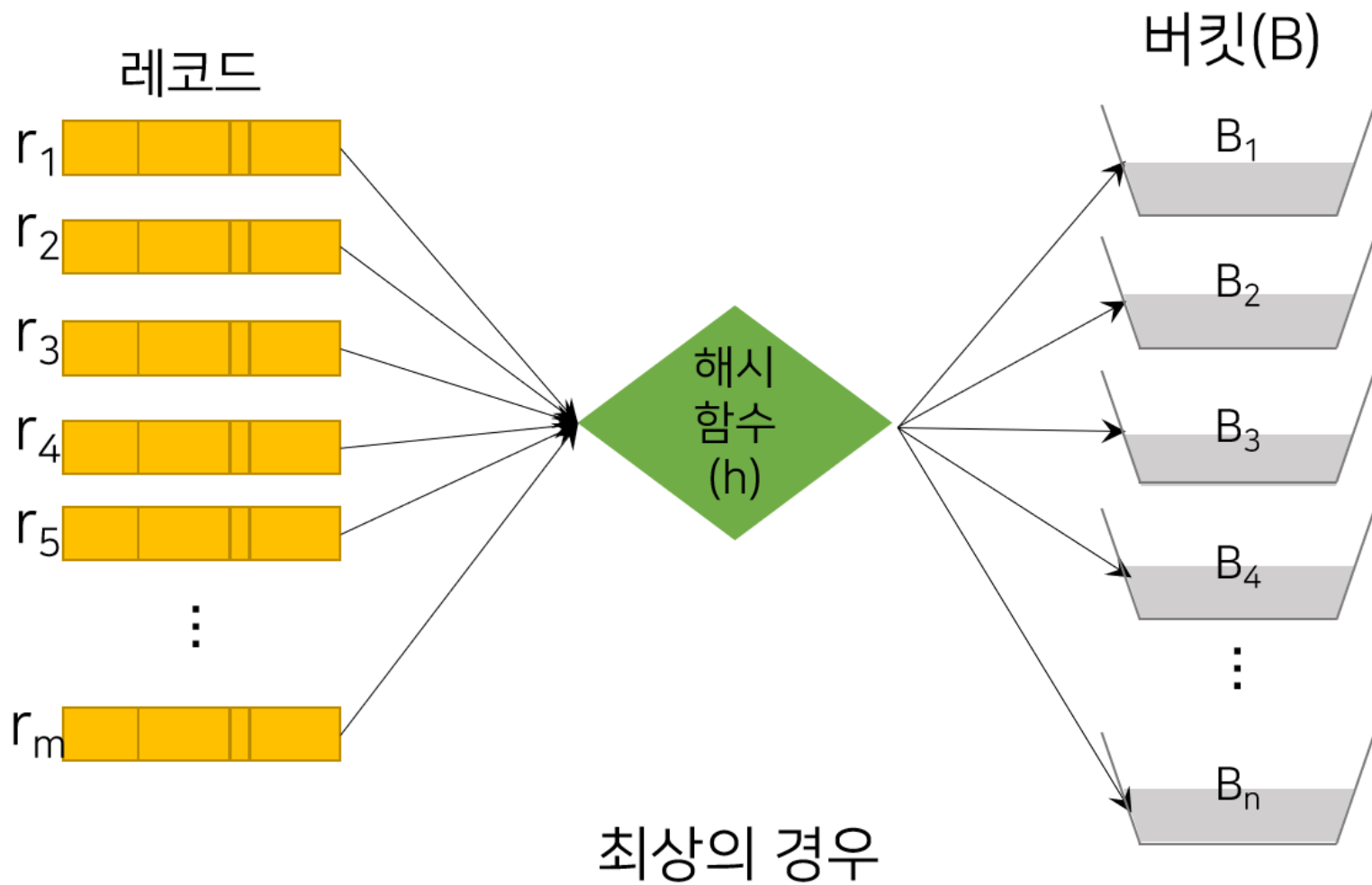
2. 버킷(bucket)

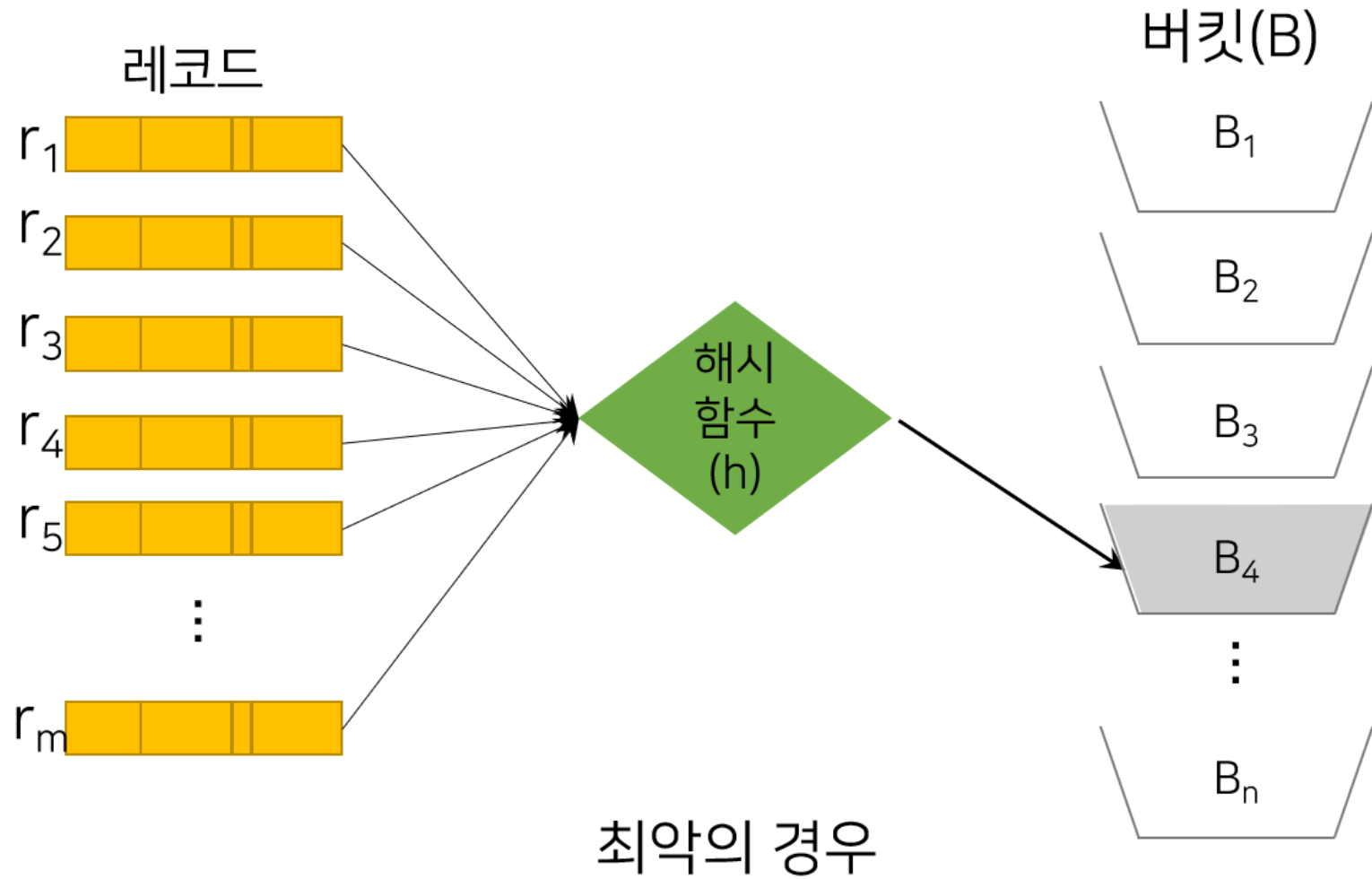
- ▶ 한 개 이상의 레코드를 저장할 수 있는 저장공간의 단위
- ▶ 크기는 일반적으로 디스크 블록의 크기와 일치

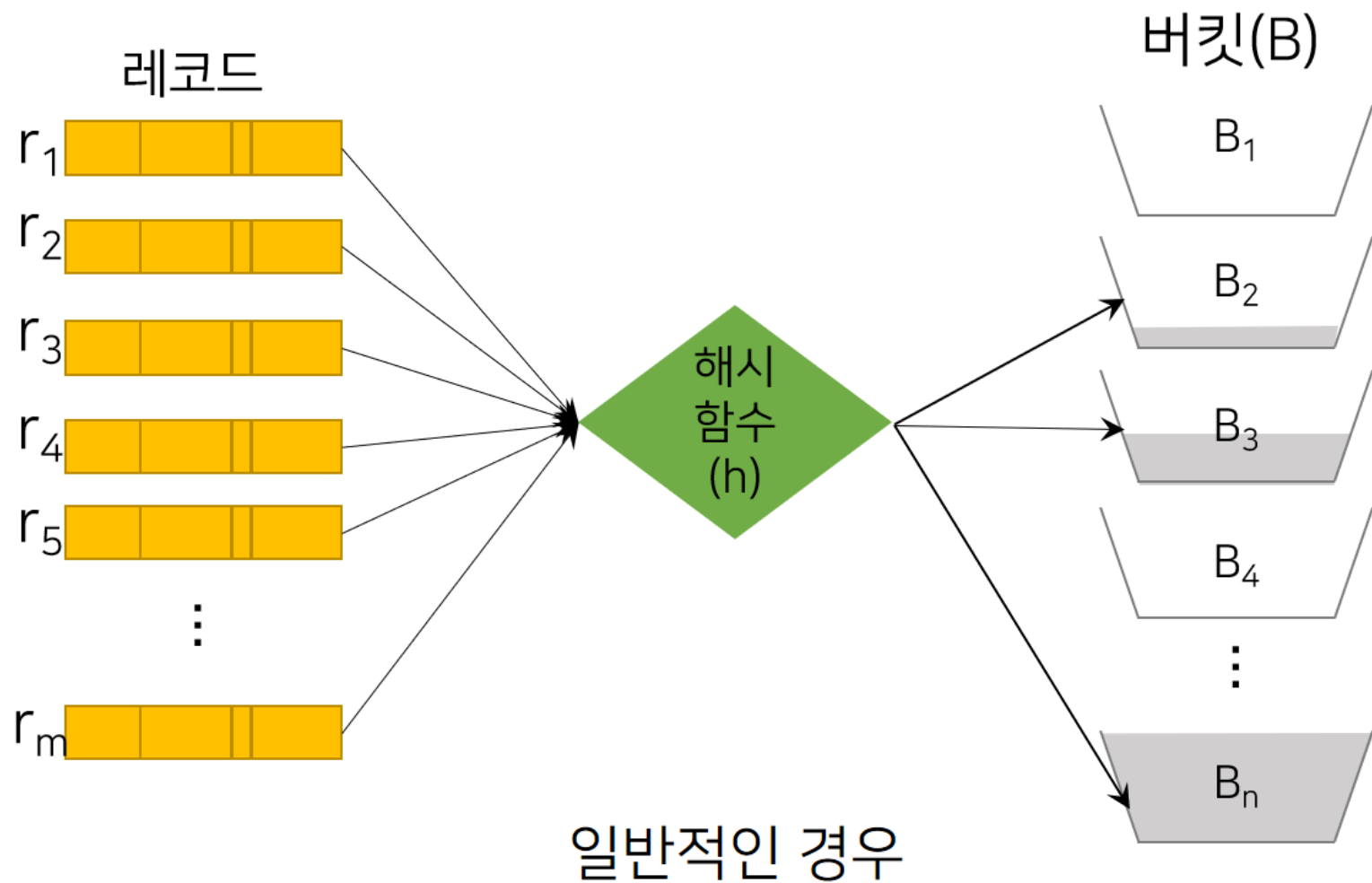


1. $h(K_3) = h(K_7) = 3$



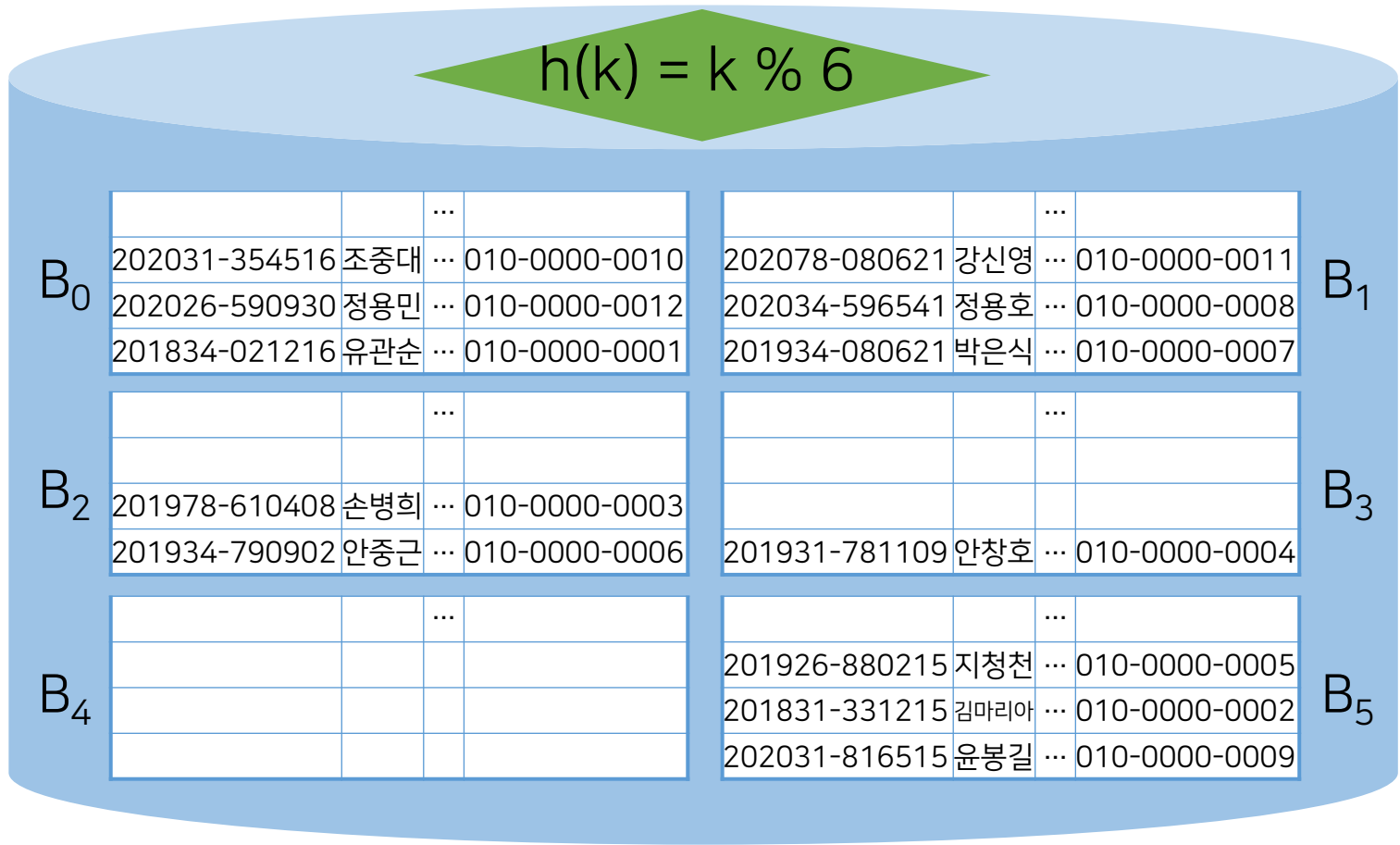






$$h(k) = k \% 6$$

학생번호	학생이름 ...	전화번호
201831-331215	김마리아 ...	010-0000-0002
201834-021216	유관순 ...	010-0000-0001
201926-880215	지청천 ...	010-0000-0005
201931-781109	안창호 ...	010-0000-0004
201934-080621	박은식 ...	010-0000-0007
201934-790902	안중근 ...	010-0000-0006
201978-610408	손병희 ...	010-0000-0003
202026-590930	정용민 ...	010-0000-0012
202031-354516	조중대 ...	010-0000-0010
202031-816515	윤봉길 ...	010-0000-0009
202034-596541	정용호 ...	010-0000-0008
202078-080621	강신영 ...	010-0000-0011



1. 버킷의 개수가 고정된 해싱 기법

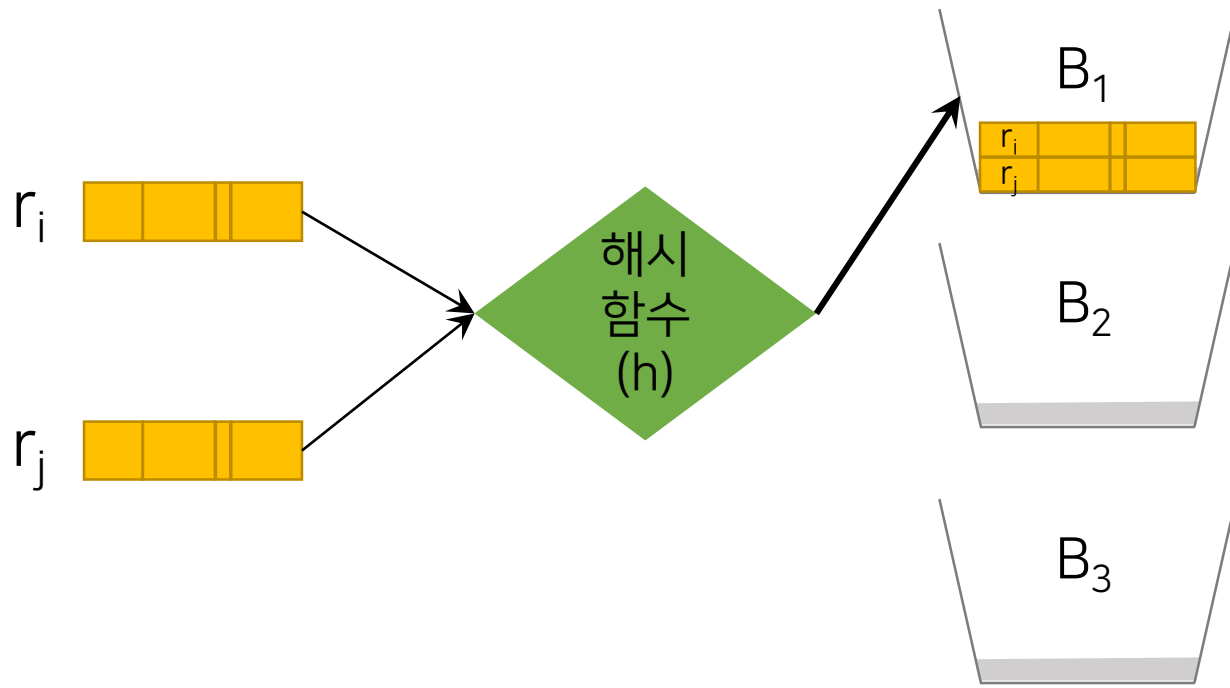
2. 키 값이 K_i 인 레코드 삽입

- ▶ $h(K_i)$ 를 통하여 K_i 에 대응하는 버킷 주소를 생성하고 레코드를 해당 버킷에 저장

3. 키 값이 K_i 인 레코드 검색

- ▶ $h(K_i)$ 을 통하여 버킷 주소를 생성하고 버킷에 저장된 레코드 접근
- ▶ $h(K_i) = h(K_j) = m$ 인 경우가 발생하기 때문에 버킷 m 에 저장된 모든 레코드를 탐색하여 선택하는 과정이 필요

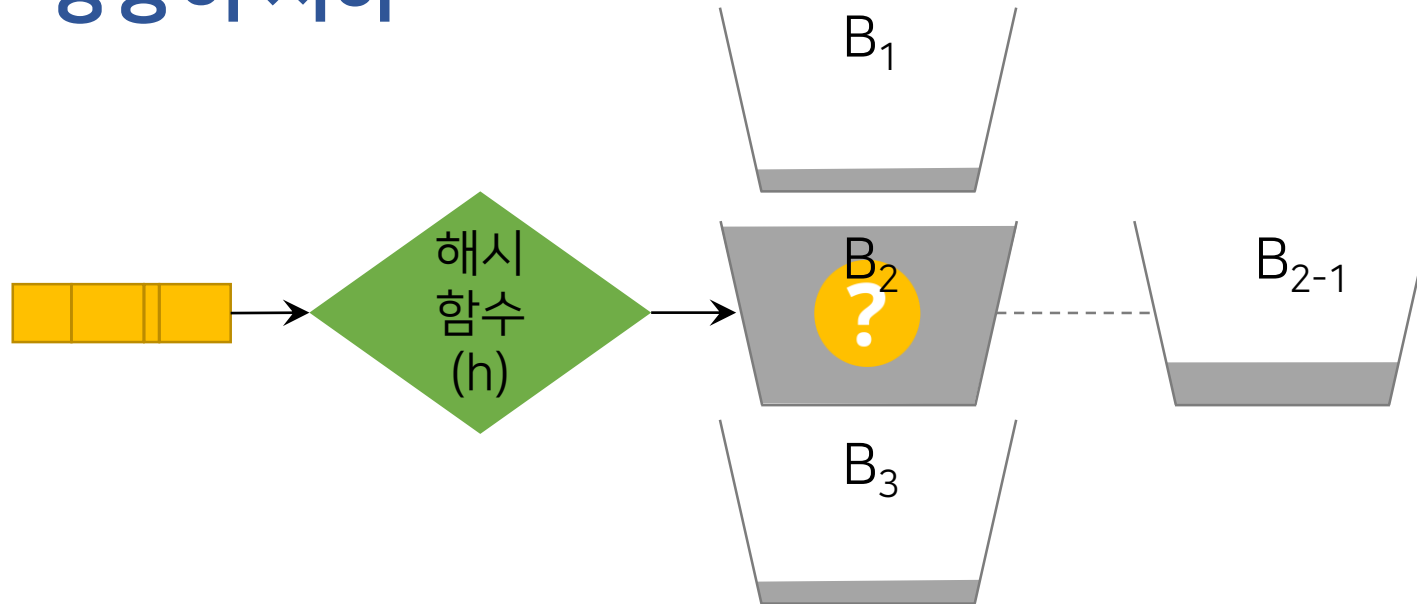
1. 충돌: 서로 다른 두 레코드가 동일한 버킷에 대응
2. 동거자: 충돌에 의해 같은 버킷 주소를 갖는 레코드



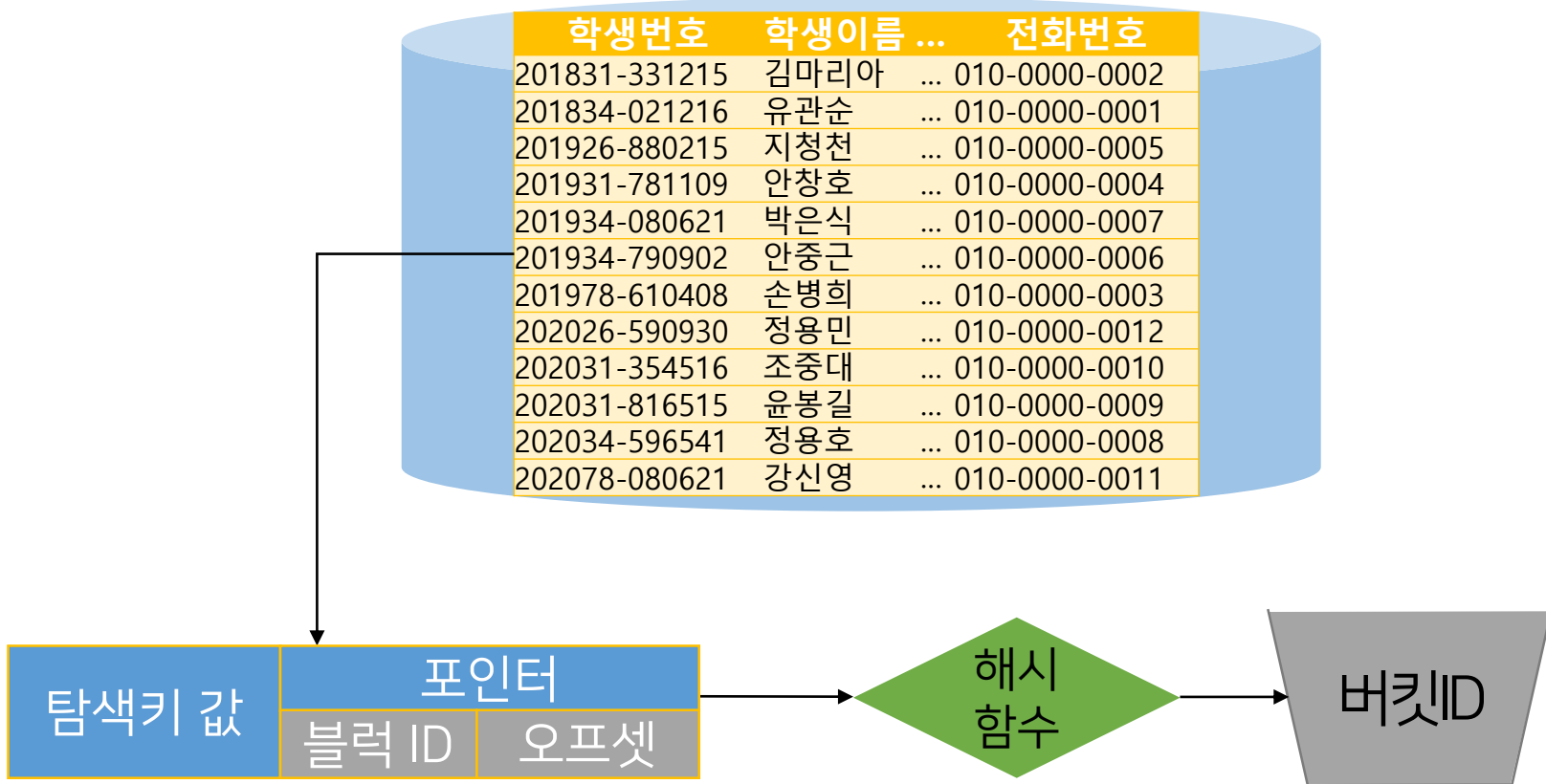
■ ■ ■ ■ 오버플로(overflow)

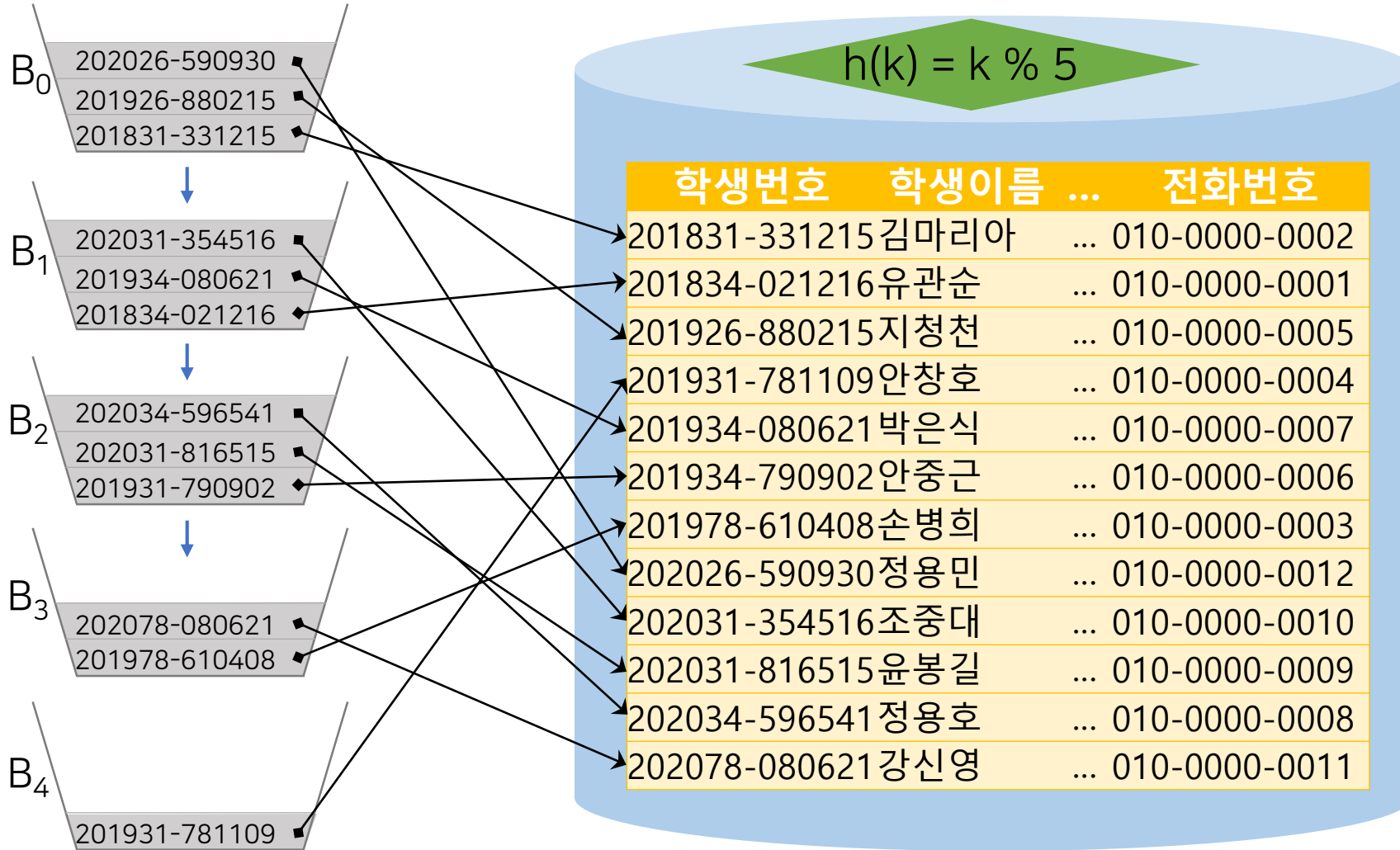
정적 해싱

1. 버킷에 레코드를 저장할 수 있는 여유 공간이 없는
상황에 발생
2. 추가적인 버킷을 할당 또는 다음 버킷에 할당하여 처리
3. 오버플로우가 발생할수록 접근시간이 길어지고 해시
성능이 저하

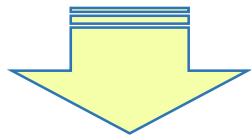


1. 해시 파일 구조와 동작 방식을 레코드가 아닌 인덱스 엔트리에 적용한 인덱스





1. 데이터베이스의 크기가 커짐에 따른 성능 감소
2. 미리 큰 공간을 잡을 경우 초기에 상당한 양의 공간이 낭비
3. 재구성 시 새롭게 선택된 해시 함수를 사용하여 모든 레코드에 대하여 다시 계산하고 버킷에 할당하는 대량의 비용이 발생



해시 구조의 크기가 동적으로 결정되는
동적 해싱 기법 제안

02

■ ■ ■ ■ 11강 해싱과 특수 인덱스

동적 해싱

- 동적 해싱의 개념
- 확장성 해싱
- 확장성 해싱의 구조

1. 동적 해싱의 정의

- ▶ 버킷의 개수를 가변적으로 조절할 수 있는 해싱 기법
- ▶ 데이터베이스의 크기에 따라 버킷의 크기가 비례

2. 데이터베이스의 증대 혹은 축소에 따른 인덱스의 구조를 조절하기 위해 해시 함수를 동적 변경하는 기술

3. 확장성 해싱

- ▶ 동적 해싱의 일종으로 디렉터리와 버킷의 2단계 구조
- ▶ 디렉터리는 디스크에 저장되는 버킷 주소 테이블
- ▶ 디렉터리 깊이를 의미하는 정수값 d 를 포함하는 헤더와 데이터가 저장된 버킷에 대한 2^d 개의 포인터로 구성

1. 모조키(pseudo key)

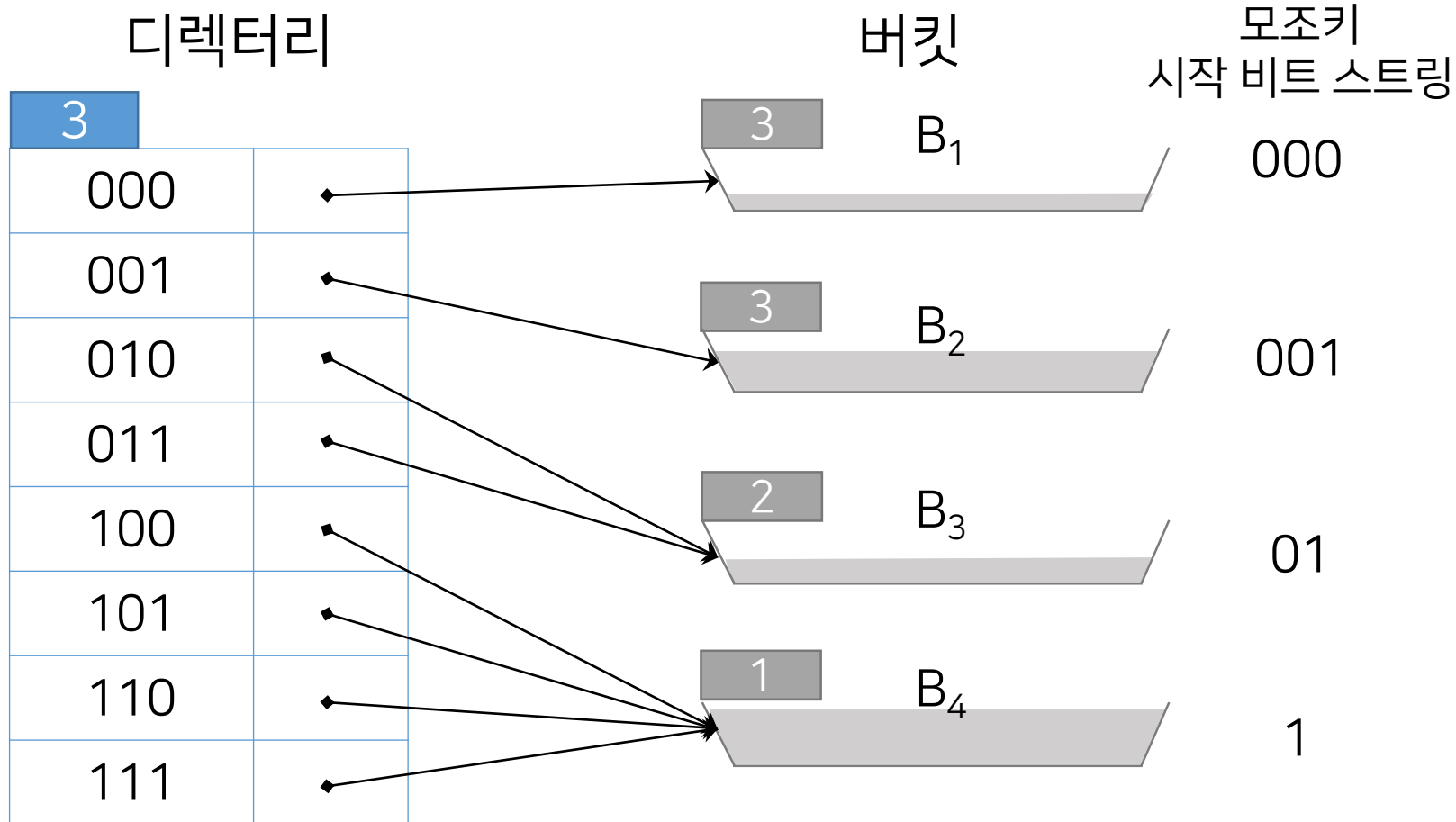
- ▶ 레코드의 탐색키 값이 해시 함수에 의해 일정 길이의 비트 스트링으로 변환된 키
- ▶ 모조키의 첫 d 비트를 사용하여 디렉터리에 접근

2. 버킷 헤더

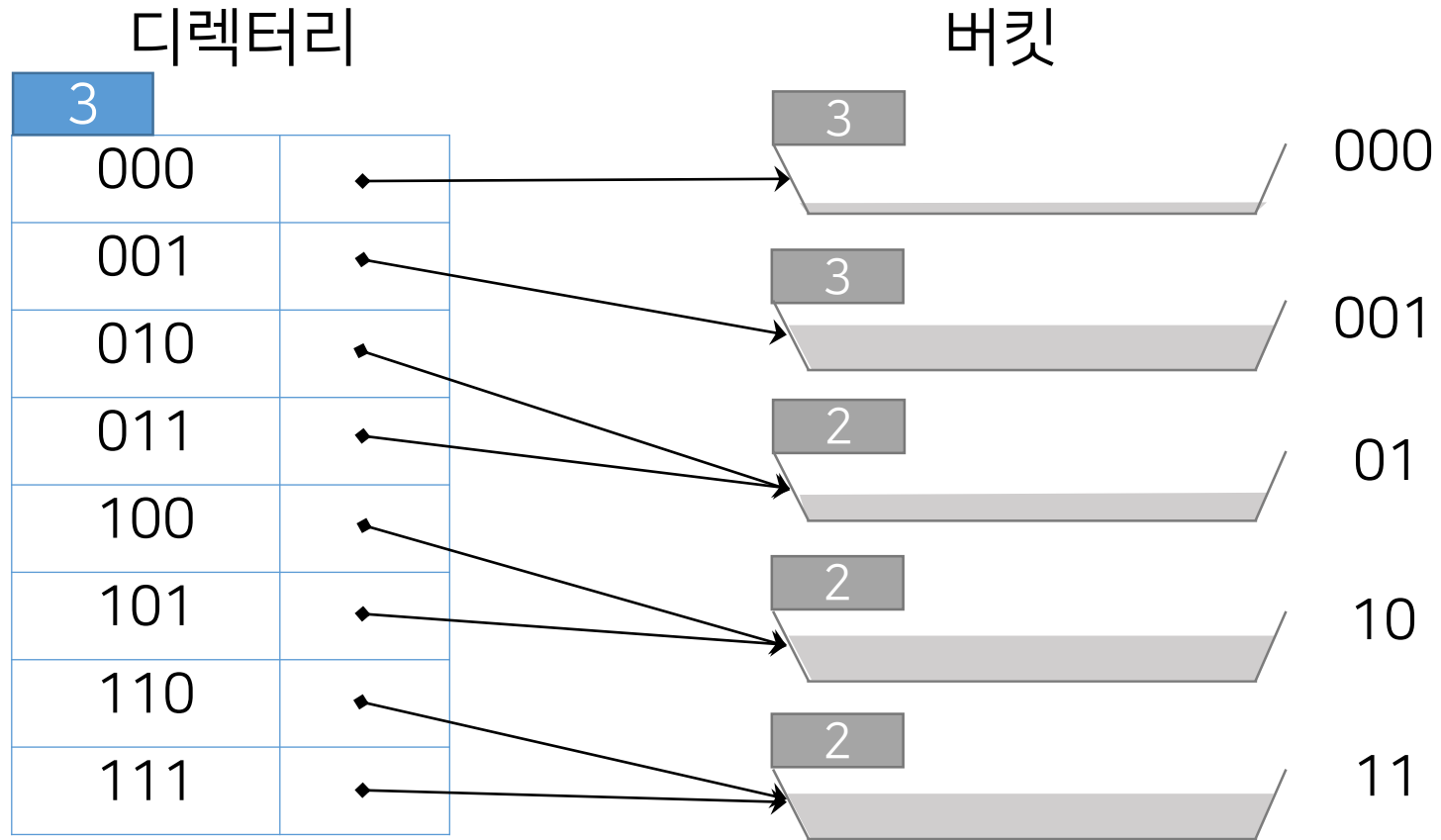
- ▶ 정수값 $i (\leq d)$ 가 저장되어 있음을 표시
- ▶ i 는 버킷에 저장되어 있는 레코드의 모조키들이 처음부터 i 비트까지 일치함을 표시



모조키
101000010001



1. 레코드 삽입에 의해 분할된 확장성 해싱 파일



비트맵 인덱스

- 비트맵 인덱스의 구성
- 비트맵 인덱스의 사용
- 비트맵 인덱스의 특징

1. 탐색키의 중복 비율이 높은 컬럼을 대상으로 하는 질의를 효율적으로 처리하기 위해 고안된 특수한 형태의 인덱스

2. 비트맵

- ▶ 간단한 비트의 배열
- ▶ 릴레이션 r 의 속성 A 에 대한 비트맵 인덱스는 A 가 가질 수 있는 값에 대해 비트맵을 구성
- ▶ 각 비트맵은 릴레이션에 있는 레코드의 수 n 개 만큼 n 개의 비트로 표현

1. i 번째 레코드가 컬럼 A 에 해당 값을 가지면 비트맵의 i 번째 비트를 1로, 그렇지 않으면 0으로 설정

번호	학번	성별	성적	남자	여자
0	12012	남자	B	1	0
1	12034	여자	C	0	1
2	13019	여자	A	0	1
3	13030	남자	B	1	0
4	13044	여자	D	0	1
5	14001	남자	F	1	0



생성된 성별 비트맵

남자	100101
여자	011010

1. i 번째 레코드가 컬럼 A 에 해당 값을 가지면 비트맵의 i 번째 비트를 1로, 그렇지 않으면 0으로 설정

번호	학번	성별	성적	A	B	C	D	F
0	12012	남자	B	0	1	0	0	0
1	12034	여자	C	0	0	1	0	0
2	13019	여자	A	1	0	0	0	0
3	13030	남자	B	0	1	0	0	0
4	13044	여자	D	0	0	0	1	0
5	14001	남자	F	0	0	0	0	1

1. i 번째 레코드가 컬럼 A 에 해당 값을 가지면 비트맵의 i 번째 비트를 1로, 그렇지 않으면 0으로 설정

번호	학번	성별	성적
0	12012	남자	B
1	12034	여자	C
2	13019	여자	A
3	13030	남자	B
4	13044	여자	D
5	14001	남자	F



생성된 성적 비트맵

A	001000
B	100100
C	010000
D	000010
F	000001

1. 성별이 남자이고 성적이 B인 학생의 정보를 출력

```
SELECT * FROM 학생
WHERE 성별 = '남자' AND 성적 = 'B'
```

2. 성별의 '남자'와 성적의 'B'의 비트열에 대한 비트 논리곱 연산을 수행

성별 비트맵	
남자	100101
여자	011010

성적 비트맵	
A	001000
B	100100
C	010000
D	000010
F	000001

1. 성별이 남자이고 성적이 B인 학생의 정보를 출력

```
SELECT * FROM 학생  
WHERE 성별 = '남자' AND 성적 = 'B'
```

2. 성별의 '남자'와 성적의 'B'의 비트열에 대한 비트 논리곱 연산을 수행

성별 = 남자

성적 = B

1 0 0 1 0 1

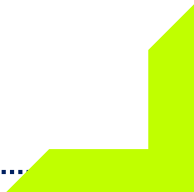
&

1 0 0 1 0 0

1 0 0 1 0 0



학번	성별	성적	
12012	남자	B	1
12034	여자	C	0
13019	여자	A	0
13030	남자	B	1
13044	여자	D	0
14001	남자	F	0



1. 비트맵의 활용

- ▶ 컬럼에 대한 값의 범위가 유한하고 비교적 개수가 적은 규모일 때 용이
- ▶ 적용: 직책, 학과, 혈액형 등

2. 비트맵 인덱스의 크기

- ▶ 레코드의 크기가 수백 바이트 이상이 되어도 비트맵 인덱스에서는 하나의 비트로 표시
- ▶ 실제 릴레이션 크기에 비해 매우 작은 것이 장점



다음 시간에는

12강 트랜잭션 을

학습하겠습니다.

