10

# CHAPTER

# 인덱스

# Contents

# 1-1 인덱스의 개요

- 인덱스의 개념
  - 찾아보기가 있는 책은 찾아보기에 주요 용어가 가나다순, 알파벳순으로 정렬되어 있고 용어 옆에 쪽수가 적혀 있어 해당 페이지를 펼치면 원하는 내용을 바로 찾을 수 있음
  - MySQL의 인덱스는 바로 이와 같은 찾아보기와 상당히 비슷한 개념
  - '데이터를 좀 더 빨리 찾을 수 있도록 도와주는 도구'


그림 10-1 책의 찾아보기

# 1-2 인덱스의 문제점

- 인덱스의 문제점
  - 책의 거의 모든 페이지에 나오는 단어를 찾아보기에 모두 표시하면 찾아보기의 분량이 엄청나게 많아져서 본문보다 더 두꺼워지는 난감한 상황이 될 수도 있음
  - 필요 없는 인덱스를 만들면 데이터베이스가 차지하는 공간만 늘어나고, 인덱스를 이용하여 데이터를 찾는 것이 전체 테이블을 찾아 보는 것보다 훨씬 느려짐

# 1-3 인덱스의 장단점

- 장점
  - 검색 속도가 매우 빨라짐(항상 그런 것은 아님)
  - 그 결과 해당 쿼리의 부하가 불어들어 결국 시스템 전체의 성능이 향상

- 단점
  - 인덱스를 저장할 공간이 필요(대략 데이터베이스 크기의 10% 정도 추가 공간이 필요)
  - 처음 인덱스를 생성하는 데 많은 시간이 소요
  - 데이터의 변경(삽입, 수정, 삭제) 작업이 자주 일어날 경우 오히려 성능이 나빠질 수 있음

# 2-1 인덱스의 종류

- 인덱스의 종류
  - MySQL에서 사용하는 인덱스에는 클러스터형 인덱스(clustered index)와 보조 인덱스(secondary index)가 있음

- 클러스터형 인덱스
  - 영어 사전처럼 책의 내용 자체가 순서대로 정렬되어 있어 인덱스가 책의 내용과 같음
  - 테이블당 하나만 생성할 수 있음
  - 행 데이터를 인덱스로 지정한 열에 맞춰서 자동으로 정렬함

- 보조 인덱스
  - 찾아보기에서 먼저 단어를 찾은 후 그 옆에 표시된 페이지로 이동하여 원하는 내용을 찾는 것과 같은 개념
  - 테이블당 여러 개를 생성할 수 있음

# 2-2 자동으로 생성되는 인덱스

- 클러스터형 인덱스
  - 기본키를 설정하면 자동으로 해당 열(아이디)에 클러스터형 인덱스가 생성

회원 테이블(userTBL)

| 아이디 | 이름 | 생년 | 지역 | 국번 | 전화번호 | 키 | 가입일 |
|---|---|---|---|---|---|---|---|
| YJS | 유재석 | 1972 | 서울 | 010 | 11111111 | 178 | 2008.8.8 |
| KHD | 강호동 | 1970 | 경북 | 011 | 22222222 | 182 | 2007.7.7 |
| KKJ | 김국진 | 1965 | 서울 | 019 | 33333333 | 171 | 2009.9.9 |
| KYM | 김용만 | 1967 | 서울 | 010 | 44444444 | 177 | 2015.5.5 |
| KJD | 김제동 | 1974 | 경남 | | | 173 | 2013.3.3 |
| NHS | 남희석 | 1971 | 충남 | 016 | 66666666 | 180 | 2017.4.4 |
| SDY | 신동엽 | 1971 | 경기 | | | 176 | 2008.10.10 |
| LHJ | 이휘재 | 1972 | 경기 | 011 | 88888888 | 180 | 2006.4.4 |
| LKK | 이경규 | 1960 | 경남 | 018 | 99999999 | 170 | 2004.12.12 |
| PSH | 박수홍 | 1970 | 서울 | 010 | 00000000 | 183 | 2012.5.5 |

그림 10-2 cookDB의 회원 테이블

```
CREATE TABLE userTBL
( userID char(8) NOT NULL PRIMARY KEY,
  userName varchar(10) NOT NULL,
  birthYear int NOT NULL,
  …
```

# [실습 10-1] 제약 조건으로 자동 생성되는 인덱스 확인하기

교재 348~354p 참고

1 테이블 만들고 자동으로 생성된 인덱스 확인하기

1-1 TBL1 테이블 생성하고 a 열을 기본키로 설정

```
USE cookDB;
CREATE TABLE TBL1
( a INT PRIMARY KEY,
  b INT,
  c INT
);
```

1-2 TBL1 테이블에 구성된 인덱스의 상태 확인

```
SHOW INDEX FROM TBL1;
```

| | Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | tbl1 | 0 | PRIMARY | 1 | a | A | 0 | NULL | NULL | | BTREE | |

# [실습 10-1] 제약 조건으로 자동 생성되는 인덱스 확인하기

교재 348~354p 참고

1-3 TBL2 테이블을 만들고 기본키와 UNIQUE 제약 조건 설정

```
CREATE TABLE TBL2
( a INT PRIMARY KEY,
  b INT UNIQUE,
  c INT UNIQUE,
  d INT
);
SHOW INDEX FROM TBL2;
```

| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
|---|---|---|---|---|---|---|---|---|---|---|
| tbl2 | 0 | PRIMARY | 1 | a | A | 0 | NULL | NULL | | BTREE |
| tbl2 | 0 | b | 1 | b | A | 0 | NULL | NULL | YES | BTREE |
| tbl2 | 0 | c | 1 | c | A | 0 | NULL | NULL | YES | BTREE |

1-4 TBL3 테이블을 만들고 기본키 없이 UNIQUE 제약 조건만 설정

```
CREATE TABLE TBL3
( a INT UNIQUE,
  b INT UNIQUE,
  c INT UNIQUE,
  d INT
);
SHOW INDEX FROM TBL3;
```

| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
|---|---|---|---|---|---|---|---|---|---|---|
| tbl3 | 0 | a | 1 | a | A | 0 | NULL | NULL | YES | BTREE |
| tbl3 | 0 | b | 1 | b | A | 0 | NULL | NULL | YES | BTREE |
| tbl3 | 0 | c | 1 | c | A | 0 | NULL | NULL | YES | BTREE |

# [실습 10-1] 제약 조건으로 자동 생성되는 인덱스 확인하기

교재 348~354p 참고

1-5 TBL4 테이블을 만들고 UNIQUE 제약 조건을 설정한 열 중 하나에 클러스터형 인덱스 생성

```
CREATE TABLE TBL4
( a INT UNIQUE NOT NULL,
  b INT UNIQUE,
  c INT UNIQUE,
 d INT
);
SHOW INDEX FROM TBL4;
```

| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
|---|---|---|---|---|---|---|---|---|---|---|
| tbl4 | 0 | a | 1 | a | A | 0 | NULL | NULL | | BTREE |
| tbl4 | 0 | b | 1 | b | A | 0 | NULL | NULL | YES | BTREE |
| tbl4 | 0 | c | 1 | c | A | 0 | NULL | NULL | YES | BTREE |

1-6 TBL5 테이블을 만들고 a 열에는 UNIQUE 제약 조건에 NOT NULL을 설정하고 d 열에는 기본키 설정

```
CREATE TABLE TBL5
( a INT UNIQUE NOT NULL,
  b INT UNIQUE,
  c INT UNIQUE,
  d INT PRIMARY KEY
);
SHOW INDEX FROM TBL5;
```

| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
|---|---|---|---|---|---|---|---|---|---|---|
| tbl5 | 0 | PRIMARY | 1 | d | A | 0 | NULL | NULL | | BTREE |
| tbl5 | 0 | a | 1 | a | A | 0 | NULL | NULL | | BTREE |
| tbl5 | 0 | b | 1 | b | A | 0 | NULL | NULL | YES | BTREE |
| tbl5 | 0 | c | 1 | c | A | 0 | NULL | NULL | YES | BTREE |

# [실습 10-1] 제약 조건으로 자동 생성되는 인덱스 확인하기

교재 348~354p 참고

## 2 클러스터형 인덱스의 정렬 확인하기

### 2-1 회원 테이블의 열만 정의

```
CREATE DATABASE IF NOT EXISTS testDB;
USE testDB;
DROP TABLE IF EXISTS userTBL;
CREATE TABLE userTBL
( userID char(8) NOT NULL PRIMARY KEY,
  userName varchar(10) NOT NULL,
  birthYear int NOT NULL,
  addr char(2) NOT NULL
);
```

### 2-2 데이터를 입력하고 확인

```
INSERT INTO userTBL VALUES ('YJS', '유재석', 1972, '서울');
INSERT INTO userTBL VALUES ('KHD', '강호동', 1970, '경북');
INSERT INTO userTBL VALUES ('KKJ', '김국진', 1965, '서울');
INSERT INTO userTBL VALUES ('KYM', '김용만', 1967, '서울');
INSERT INTO userTBL VALUES ('KJD', '김제동', 1974, '경남');
SELECT * FROM userTBL;
```

| userID | userName | birthYear | addr |
|---|---|---|---|
| KHD | 강호동 | 1970 | 경북 |
| KJD | 김제동 | 1974 | 경남 |
| KKJ | 김국진 | 1965 | 서울 |
| KYM | 김용만 | 1967 | 서울 |
| YJS | 유재석 | 1972 | 서울 |
| NULL | NULL | NULL | NULL |

# [실습 10-1] 제약 조건으로 자동 생성되는 인덱스 확인하기

교재 348~354p 참고

2-3 아이디 열의 기본키를 제거하고 이름(userName) 열을 기본키로 설정

```
ALTER TABLE userTBL DROP PRIMARY KEY;
ALTER TABLE userTBL
      ADD CONSTRAINT pk_userName PRIMARY KEY (userName);
SELECT * FROM userTBL;
```

| | userID | userName | birthYear | addr |
|---|---|---|---|---|
| ▶ | KHD | 강호동 | 1970 | 경북 |
| | KKJ | 김국진 | 1965 | 서울 |
| | KYM | 김용만 | 1967 | 서울 |
| | KJD | 김제동 | 1974 | 경남 |
| | YJS | 유재석 | 1972 | 서울 |
| * | NULL | NULL | NULL | NULL |

2-4 위 실습의 결론

- PRIMARY KEY로 지정한 열에 클러스터형 인덱스가 생성
- UNIQUE NOT NULL로 지정한 열에 클러스터형 인덱스가 생성
- UNIQUE 또는 UNIQUE NULL로 지정한 열에 보조 인덱스가 생성
- PRIMARY KEY와 UNIQUE NOT NULL이 같이 있으면 PRIMARY KEY로 지정한 열에 우선 클러스터형 인덱스가 생성
- PRIMARY KEY로 지정한 열을 기준으로 데이터가 오름차순 정렬

# 3-1 B-Tree의 개요

- B–Tree
  - '자료 구조'에 나오는, 범용적으로 사용되는 데이터 구조로 균형이 잡힌 트리
  - 트리 구조에서 데이터가 존재하는 공간을 노드라고 함
  - 노드의 종류
    - 루트 노드 : 가장 상위에 있는 노드, 모든 출발은 이 루트 노드에서 시작
    - 리프 노드 : 가장 말단에 있는 노드
    - 중간 수준 노드 : 루트 노드와 리프 노드의 중간에 끼인 노드
  - 페이지
    - MySQL에서는 노드를 페이지라고 함
    - 페이지는 최소한의 저장 단위로 크기가 16KB이며, 아무리 작은 데이터를 저장하더라도 1개의 페이지(16KB)를 사용

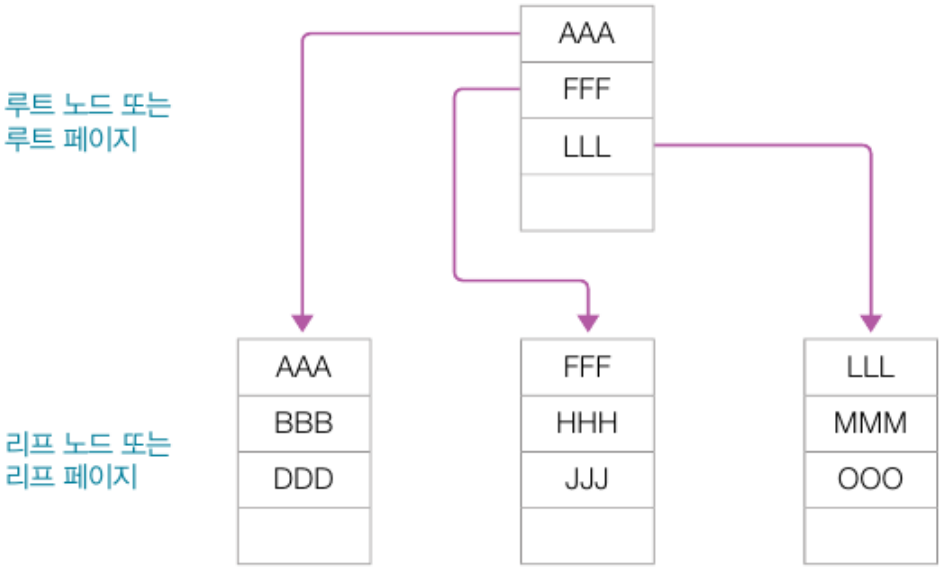



그림 10-10 B-Tree의 기본 구조

# 3-1 B-Tree의 개요

- MMM 이라는 데이터를 검색하는 경우
- (1) B – Tree 구조가 아니라면
  - 처음부터 검색하는 수 밖에. 8건의 데이터(3개의 페이지) 검색이 필요
- (2) B – Tree 구조에서
  - AAA, FFF, LLL이라는 데이터를 읽은 후 MMM이 LLL 다음에 나오므로 세 번째 리프 페이지로 직접 이동
  - 세 번째 리프 페이지에서 LLL, MMM이라는 데이터를 읽어 MMM을 찾음
  - 루트 페이지에서 3건(AAA, FFF, LLL), 리프 페이지에서 2건(LLL, MMM), 총 5건의 데이터를 검색하면 원하는 결과를 얻을 수 있다(2개의 페이지)

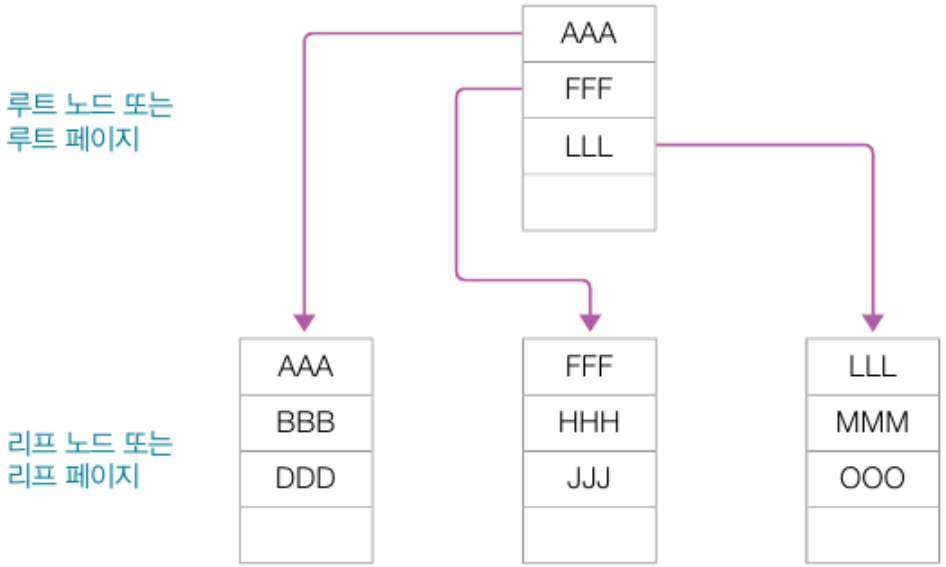



그림 10-10 B-Tree의 기본 구조

# 3-1 B-Tree의 개요

- B-Tree 단점 : 데이터를 삽입할 때 성능이 급격히 느려지는데, 이는 '페이지 분할 ' 이라는 작업 때문.  3가지 Case를 예로 보자.

- Case 1 : III라는 새로운 데이터를 삽입하는 경우

  -> JJJ가 한 칸 이동했을 뿐 큰 작업이 일어나지 않는다.

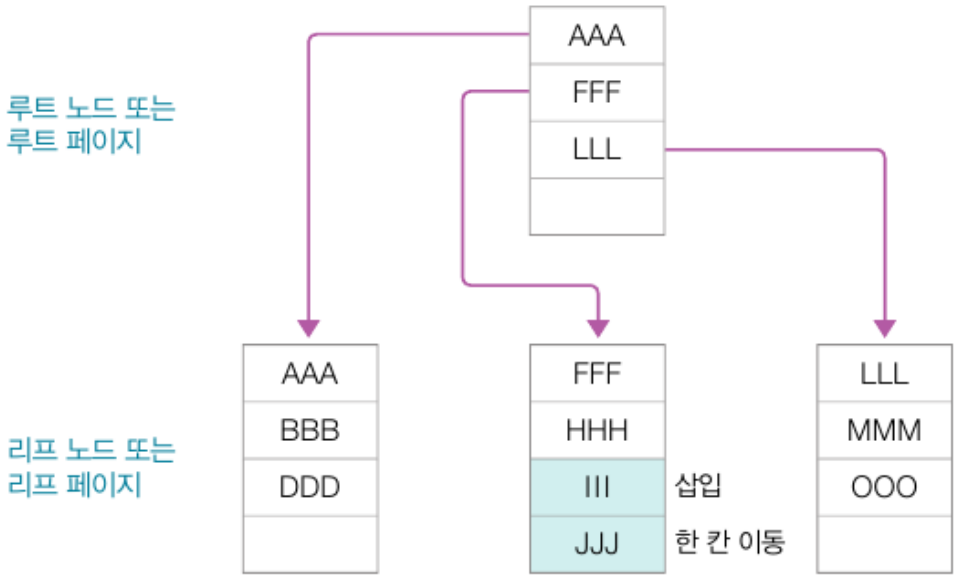



그림 10-11 III를 삽입하는 경우

# 3-2 페이지 분할

- Case 2 : GGG라는 새로운 데이터 삽입

  -> 페이지를 확보한 후 페이지 분할 작업을 하고, 루트 페이지에도 새로 등록된 페이지의 맨 위에 있는 데이터인 III가 등록된다.

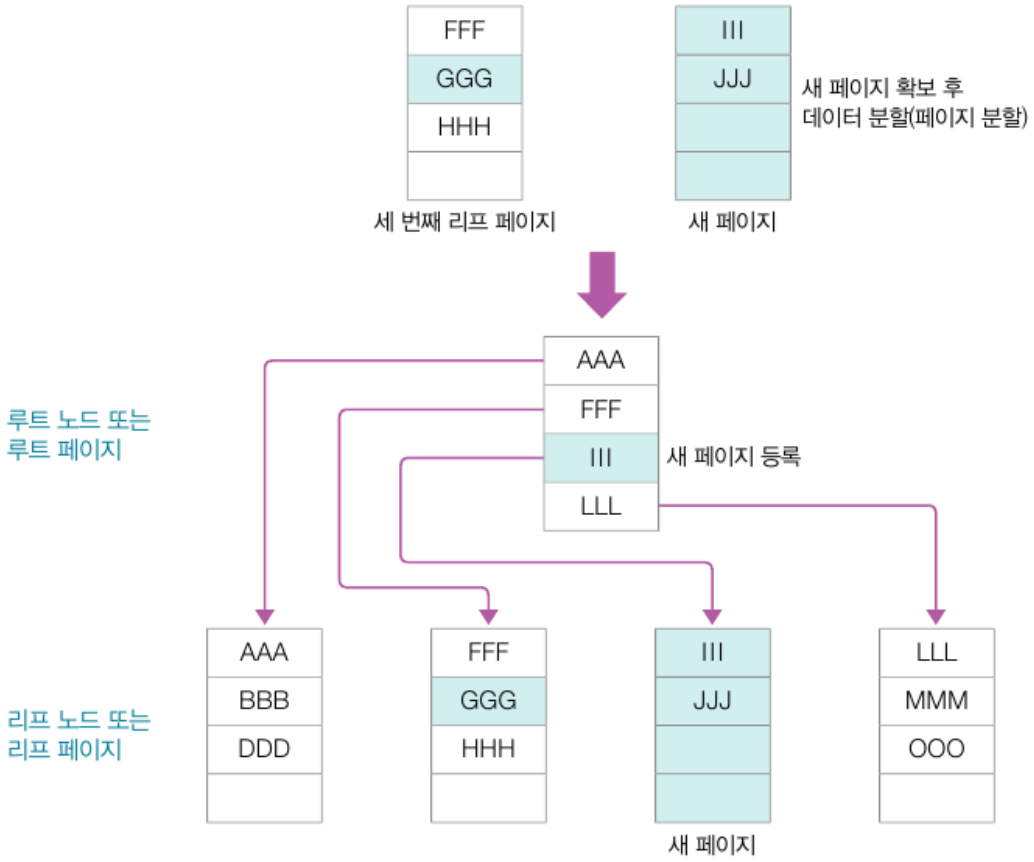



그림 10-12 GGG를 삽입하는 경우

# 3-2 페이지 분할

- Case 3 : PPP와 QQQ라는 데이터를 동시에 삽입

  -> 새로운 페이지 3개가 생성되고 2회의 페이지 분할이 발생한다.

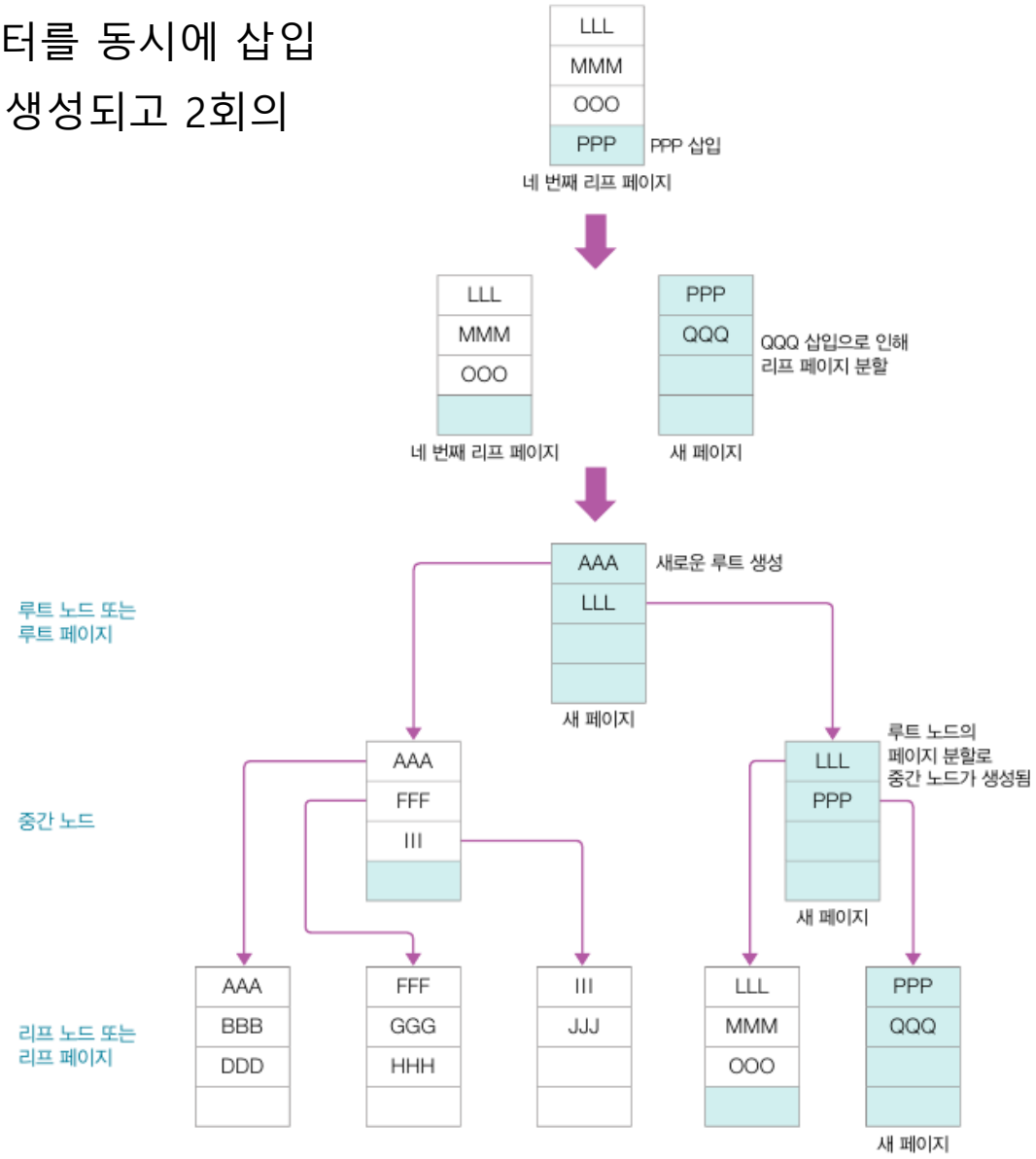



그림 10-13 PPP와 QQQ를 삽입하는 경우

# 3-3 클러스터형 인덱스와 보조 인덱스

- 인덱스 없이 테이블을 생성하고 다음과 같이 데이터를 입력하는 경우

```
CREATE DATABASE IF NOT EXISTS testDB;
USE testDB;
DROP TABLE IF EXISTS clusterTBL;
CREATE TABLE clusterTBL
( userID char(8),
  userName varchar(10)
);
INSERT INTO clusterTBL VALUES ('YJS', '유재석');
INSERT INTO clusterTBL VALUES ('KHD', '강호동');
INSERT INTO clusterTBL VALUES ('KKJ', '김국진');
INSERT INTO clusterTBL VALUES ('KYM', '김용만');
INSERT INTO clusterTBL VALUES ('KJD', '김제동');
INSERT INTO clusterTBL VALUES ('NHS', '남희석');
INSERT INTO clusterTBL VALUES ('SDY', '신동엽');
INSERT INTO clusterTBL VALUES ('LHJ', '이휘재');
INSERT INTO clusterTBL VALUES ('LKK', '이경규');
INSERT INTO clusterTBL VALUES ('PSH', '박수홍');
```

- 페이지당 4개의 행이 입력된다면 위의 데이터는 다음과 같이 구성

데이터 페이지 (Heap 영역)

1000

| | |
|---|---|
| YJS | 유재석 |
| KHD | 강호동 |
| KKJ | 김국진 |
| KYM | 김용만 |

1001

| | |
|---|---|
| KJD | 김제동 |
| NHS | 남희석 |
| SDY | 신동엽 |
| LHJ | 이휘재 |

1002

| | |
|---|---|
| LKK | 이경규 |
| PSH | 박수홍 |
| | |
| | |

# 3-3 클러스터형 인덱스와 보조 인덱스

- 데이터 확인

```
SELECT * FROM clusterTBL;
```

| userID | userName |
|---|---|
| YJS | 유재석 |
| KHD | 강호동 |
| KKJ | 김국진 |
| KYM | 김용만 |
| KJD | 김제동 |
| NHS | 남희석 |
| SDY | 신동엽 |
| LHJ | 이휘재 |
| LKK | 이경규 |
| PSH | 박수홍 |

- userID에 클러스터형 인덱스 구성

```
ALTER TABLE clusterTBL
      ADD CONSTRAINT PK_clusterTBL_userID
            PRIMARY KEY (userID);
```

- 다시 데이터 확인

```
SELECT * FROM clusterTBL;
```

| userID | userName |
|---|---|
| KHD | 강호동 |
| KJD | 김제동 |
| KKJ | 김국진 |
| KYM | 김용만 |
| LHJ | 이휘재 |
| LKK | 이경규 |
| NHS | 남희석 |
| PSH | 박수홍 |
| SDY | 신동엽 |
| YJS | 유재석 |

# 3-3 클러스터형 인덱스와 보조 인덱스

- 다시 데이터 확인

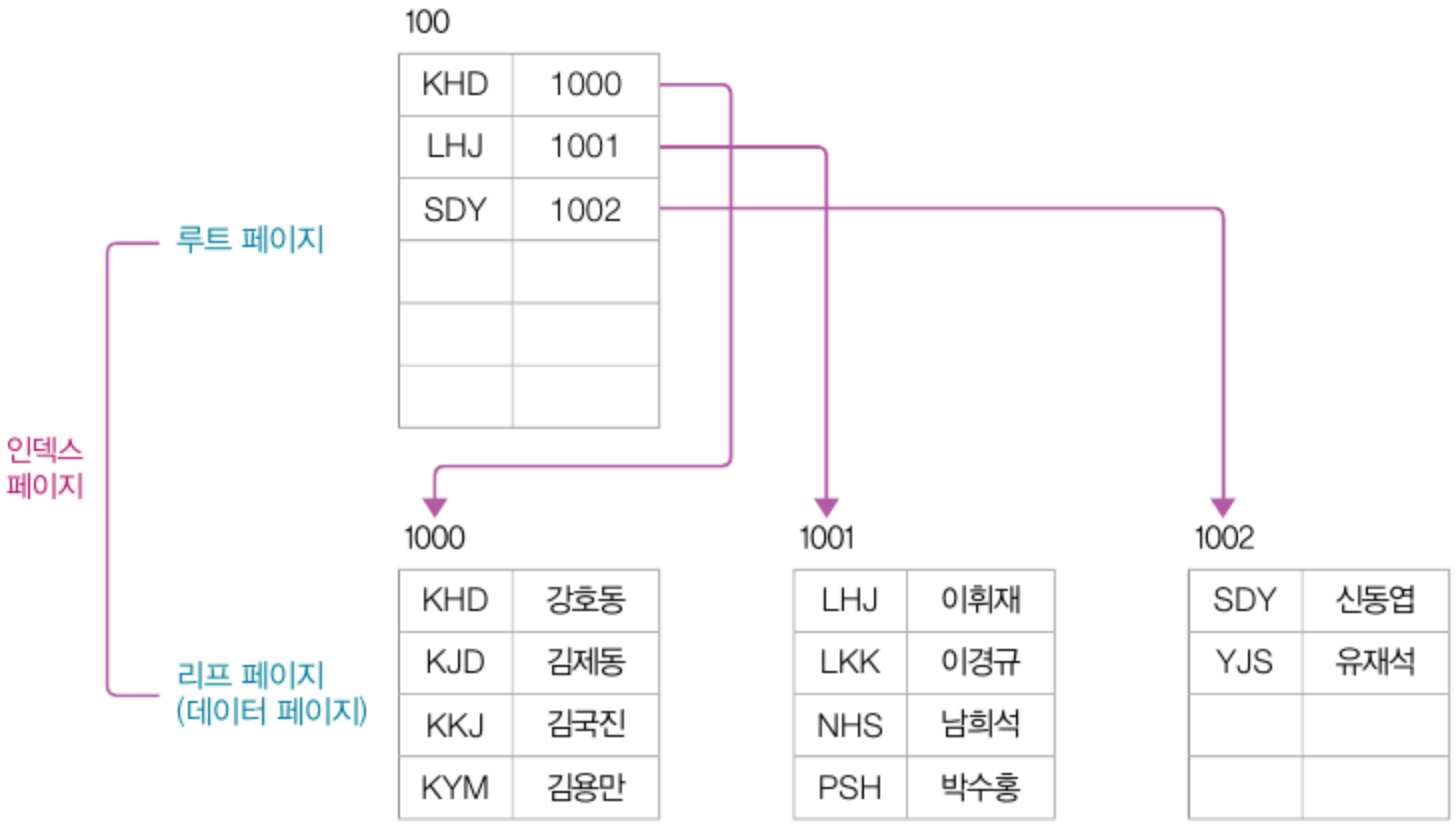



그림 10-17 클러스터형 인덱스를 구성하는 경우

# 3-3 클러스터형 인덱스와 보조 인덱스

- 보조 인덱스 만들기

```
CREATE DATABASE IF NOT EXISTS testDB;
USE testDB;
DROP TABLE IF EXISTS secondaryTBL;
CREATE TABLE secondaryTBL
( userID char(8),
  userName varchar(10)
);
INSERT INTO secondaryTBL VALUES ('YJS', '유재석');
INSERT INTO secondaryTBL VALUES ('KHD', '강호동');
INSERT INTO secondaryTBL VALUES ('KKJ', '김국진');
INSERT INTO secondaryTBL VALUES ('KYM', '김용만');
INSERT INTO secondaryTBL VALUES ('KJD', '김제동');
INSERT INTO secondaryTBL VALUES ('NHS', '남희석');
INSERT INTO secondaryTBL VALUES ('SDY', '신동엽');
INSERT INTO secondaryTBL VALUES ('LHJ', '이휘재');
INSERT INTO secondaryTBL VALUES ('LKK', '이경규');
INSERT INTO secondaryTBL VALUES ('PSH', '박수홍');
```

- userID 열에 UNIQUE 제약 조건 설정

```
ALTER TABLE secondaryTBL
      ADD CONSTRAINT UK_secondaryTBL_userID
            UNIQUE (userID);
```

# 3-3 클러스터형 인덱스와 보조 인덱스

- 데이터 순서 확인

```
SELECT * FROM secondaryTBL;
```

| | userID | userName |
|---|---|---|
| ▶ | YJS | 유재석 |
| | KHD | 강호동 |
| | KKJ | 김국진 |
| | KYM | 김용만 |
| | KJD | 김제동 |
| | NHS | 남희석 |
| | SDY | 신동엽 |
| | LHJ | 이휘재 |
| | LKK | 이경규 |
| | PSH | 박수홍 |

# 3-3 클러스터형 인덱스와 보조 인덱스

- 보조 인덱스의 내부 구성

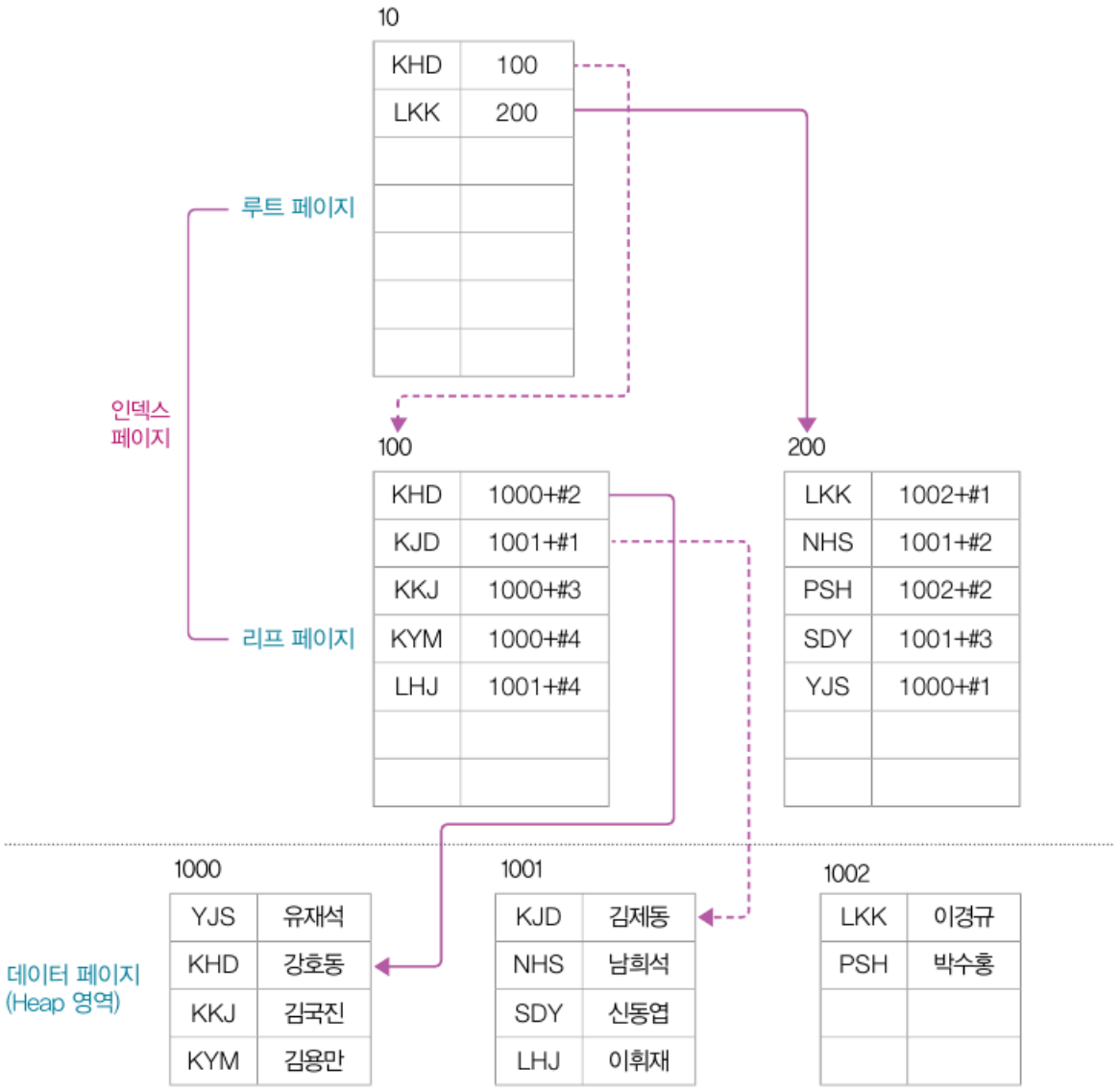



그림 10-19 보조 인덱스를 구성하는 경우

# 3-3 클러스터형 인덱스와 보조 인덱스

- 클러스터형 인덱스와 보조 인덱스에서 데이터 검색
  - 클러스터형 인덱스는 데이터 검색 속도가 보조 인덱스보다 빠름
- (1) 클러스터형 인덱스에서 NHS(남희석)를 검색하는 경우 루트 페이지(100번)와 리프 페이지(1001번), 총 2개 페이지만 읽어야 함

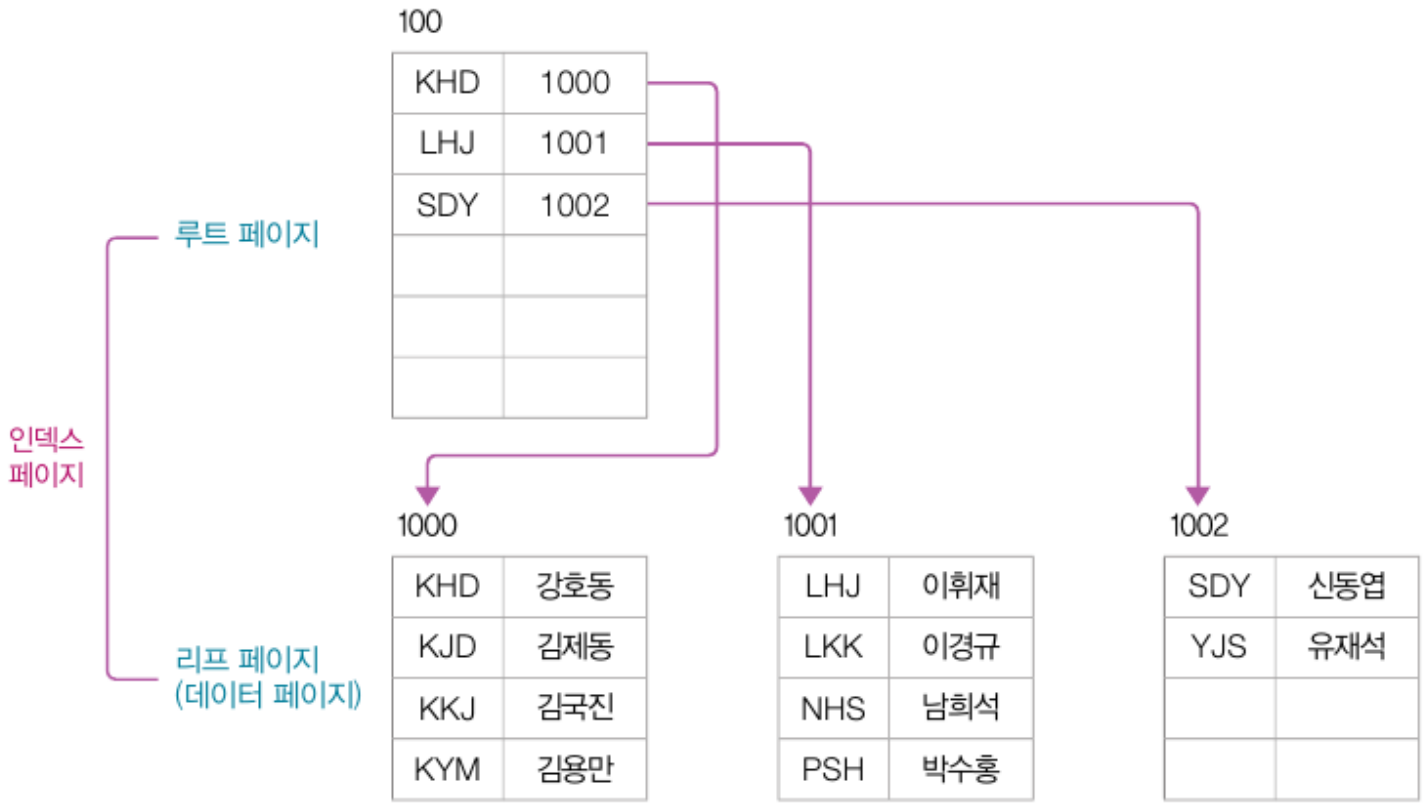



그림 10-17 클러스터형 인덱스를 구성하는 경우

# 3-3 클러스터형 인덱스와 보조 인덱스

- (2) 보조 인덱스에서 NHS(남 희석)를 검색할 때는 인덱스 페이지의 루트 페이지(10번), 리프 페이지(200번), 데이터 페이지 (1002번), 총 3개 페이지를 읽어야 함

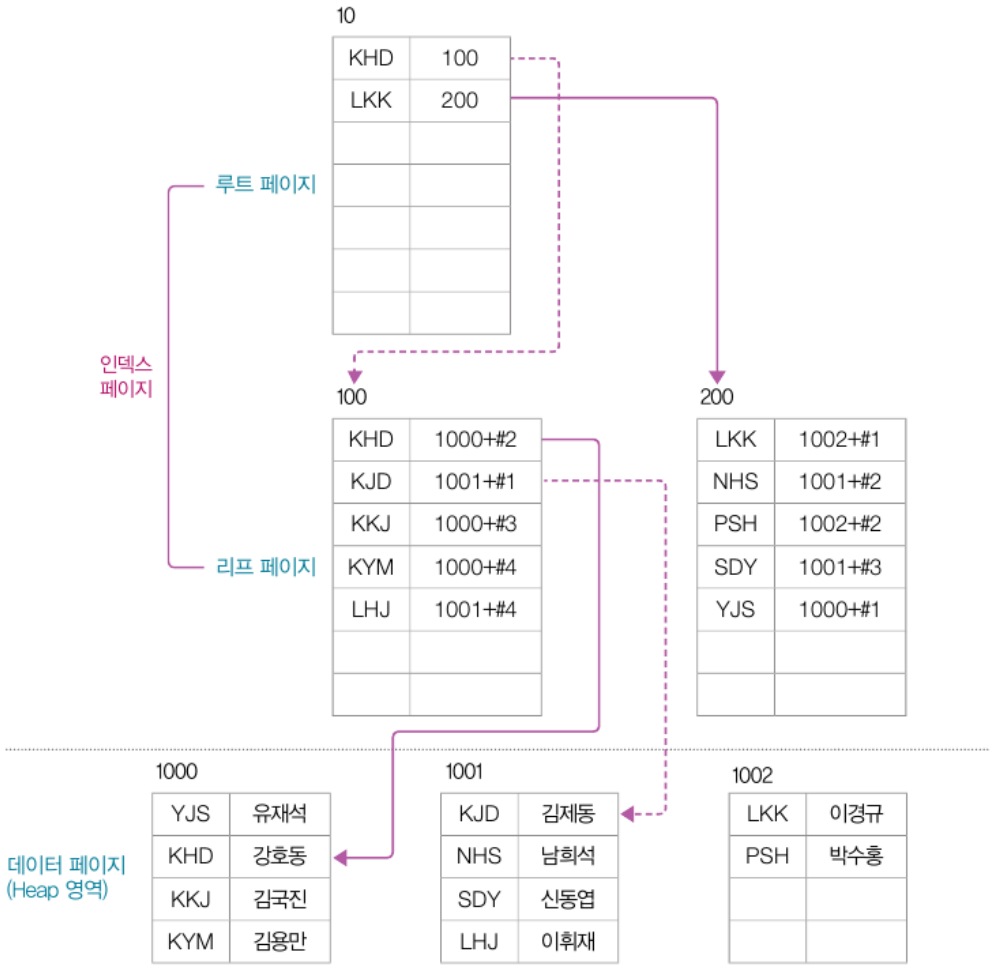



그림 10-19 보조 인덱스를 구성하는 경우

# 3-3 클러스터형 인덱스와 보조 인덱스

- 클러스터형 인덱스와 보조 인덱스에서 데이터 삽입
  - 보조 인덱스의 성능 부하가 클러스터형 인덱스보다 더 적음
- 클러스터형 인덱스에 새로운 데이터 삽입

```
INSERT INTO clusterTBL VALUES ('KKK', '크크크');
INSERT INTO clusterTBL VALUES ('MMM', '마마무');
```

-> 첫번째 리프 페이지와 두번째 리프 페이지에서 페이지 분할이 일어난다. 2개 페이지가 추가. 시간이 많이 걸린다.

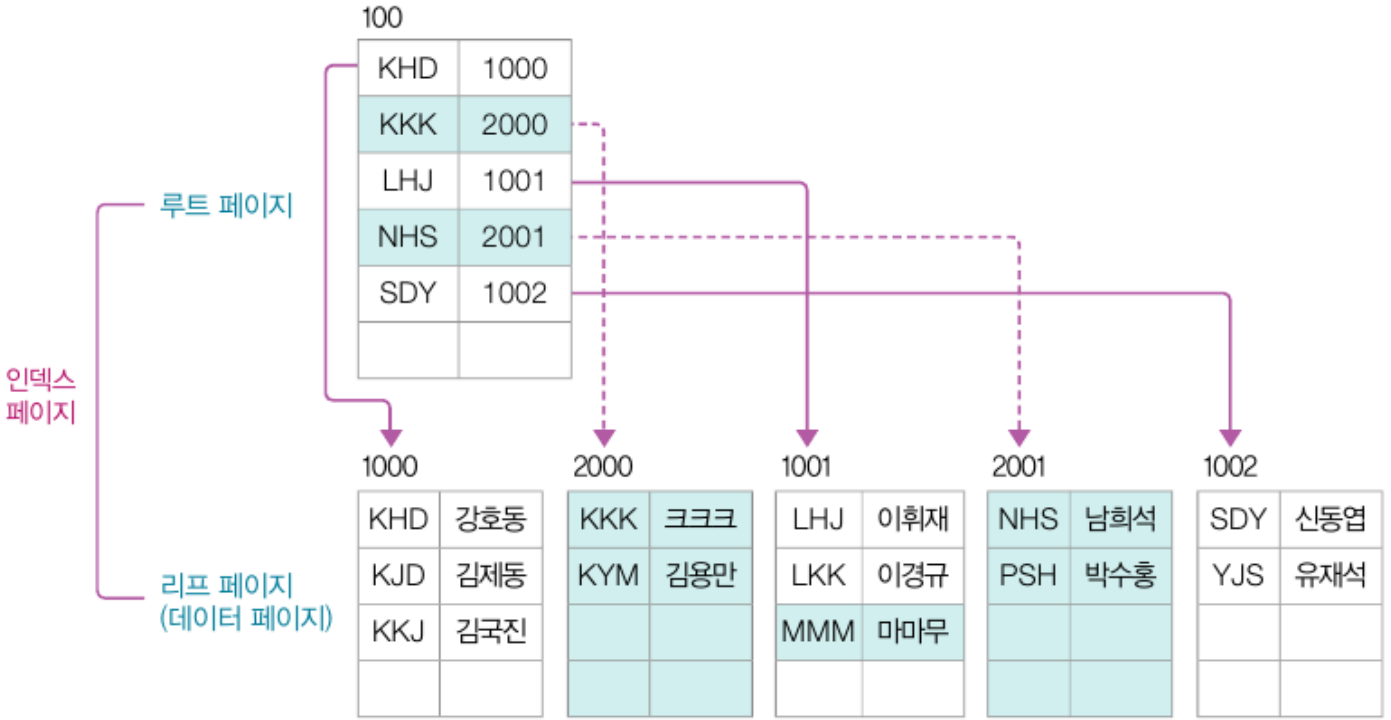



그림 10-20 클러스터형 인덱스에 2행을 추가하는 경우

# 3-3 클러스터형 인덱스와 보조 인덱스

- 보조 인덱스에 동일한 데이터 삽입

```
INSERT INTO secondaryTBL VALUES ('KKK', '크크크');
INSERT INTO secondaryTBL VALUES ('MMM', '마마무');
```

-> 데이터 페이지를 정렬하는 것이 아니므로 데이터 페이지의 뒤쪽 빈 부분에 데이터가 삽입된다. 약간의 순서 변경만 있을 뿐 페이지 분할이 일어나지 않는다.

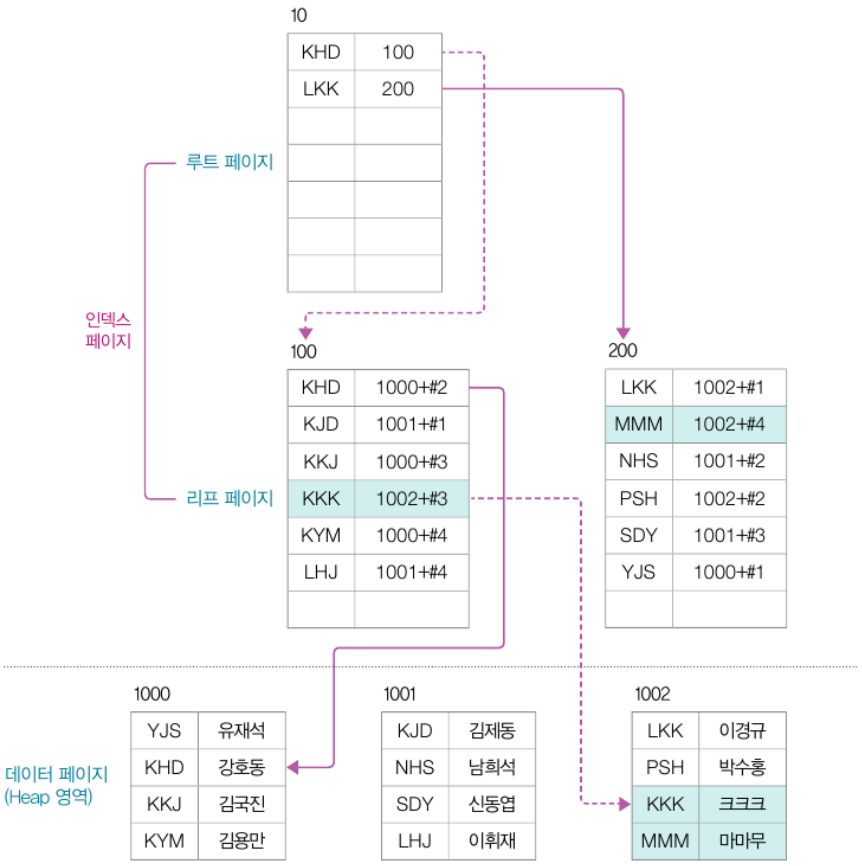



그림 10-21 보조 인덱스에 2행을 추가하는 경우

# 3-3 클러스터형 인덱스와 보조 인덱스

- 클러스터형 인덱스의 특징
  - 인덱스를 생성할 때 데이터 페이지 전체가 다시 정렬됨. 이미 대용량 데이터가 입력된 상태에서 중간에 클러스터형 인덱스를 생성하면 시스템에 심각한 부하를 줄 수 있음
  - 리프 페이지가 곧 데이터 페이지. 인덱스 자체에 데이터가 포함되어 있음
  - 보조 인덱스보다 검색 속도가 빠르고 데이터 변경(삽입, 수정, 삭제) 속도는 느림
  - 클러스터형 인덱스는 테이블에 하나만 생성할 수 있음. 어느 열에 클러스터형 인덱스를 생성하는지에 따라 시스템의 성능이 달라짐

- 보조 인덱스의 특징
  - 인덱스를 생성할 때 데이터 페이지는 그대로 둔 상태에서 별도의 페이지에 인덱스를 구성함
  - 리프 페이지에 데이터가 아니라 데이터가 위치하는 주소 값(RID)이 들어 있음
  - 데이터 변경(삽입, 수정, 삭제) 시 클러스터형 인덱스보다 성능 부하가 적음
  - 보조 인덱스는 한 테이블에 여러 개를 생성할 수 있음. 하지만 함부로 남용하면 오히려 시스템의 성능을 떨어뜨리는 결과를 초래할 수 있으므로 필요한 열에만 생성해야 함

# 4-1 인덱스 생성

- 인덱스 생성 형식

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
      [index_type]
      ON TBL_userName (index_col_userName, )
      [index_option]
      [algorithm_option | lock_option]

index_col_userName:
      col_userName [(length)] [ASC | DESC]

index_type:
      USING {BTREE | HASH}

index_option:
      KEY_BLOCK_SIZE [=] value
    | index_type
    | WITH PARSER parser_userName
    | COMMENT 'string'

algorithm_option:
      ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:
      LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

# 4-1 인덱스 생성

- 인덱스 생성 방법
  - CREATE INDEX 문으로 인덱스를 만들면 보조 인덱스가 생성
  - CREATE INDEX 문으로는 클러스터형 인덱스를 만들 수 없으며, 클러스터형 인덱스를 만들려면 앞에서 배운 ALTER TABLE 문을 사용해야 함
  - CREATE INDEX 문의 UNIQUE 옵션은 고유한 인덱스를 만들 때 사용
  - UNIQUE로 설정된 인덱스에는 동일한 데이터 값이 입력될 수 없음
  - ASC, DESC는 정렬 방식을 지정하는데, ASC가 기본 값이고 오름차순으로 정렬된 인덱스가 생성
  - index_type은 생략 가능하며, 생략할 경우 기본 값인 B - Tree 형식을 사용

# 4-2 인덱스 삭제

- 인덱스 삭제 형식

```
DROP INDEX index_name ON TBL_userName
      [algorithm_option | lock_option]

algorithm_option:
      ALGORITHM [=] {DEFAULT | INPLACE|COPY}

lock_option:
      LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

```
DROP INDEX 인덱스이름 ON 테이블이름;
```

  - 클러스터형 인덱스를 삭제하려면 위 구문의 인덱스 이름 부분에 PRIMARY를 넣음
  - 인덱스를 모두 삭제할 때는 보조 인덱스부터 삭제
  - 인덱스를 많이 생성해놓은 테이블의 경우 각 인덱스의 용도를 확인한 후 활용도가 떨어지는 인덱스를 삭제

# [실습 10-2] 인덱스 생성하고 활용하기

교재 368~373p 참고

1 cookDB 초기화하기

1-1 cookDB.sql 파일을 열어 실행

1-2 열린 쿼리 창을 모두 닫고 새 쿼리 창을 염

2 인덱스 생성하고 활용하기

2-1 회원 테이블(userTBL) 내용 확인

```
USE cookDB;
SELECT * FROM userTBL;
```

| userID | userName | birthYear | addr | mobile1 | mobile2 | height | mDate |
|---|---|---|---|---|---|---|---|
| KHD | 강호동 | 1970 | 경북 | 011 | 22222222 | 182 | 2007-07-07 |
| KJD | 김제동 | 1974 | 경남 | NULL | NULL | 173 | 2013-03-03 |
| KKJ | 김국진 | 1965 | 서울 | 019 | 33333333 | 171 | 2009-09-09 |
| KYM | 김용만 | 1967 | 서울 | 010 | 44444444 | 177 | 2015-05-05 |
| LHJ | 이휘재 | 1972 | 경기 | 011 | 88888888 | 180 | 2006-04-04 |
| LKK | 이경규 | 1960 | 경남 | 018 | 99999999 | 170 | 2004-12-12 |
| NHS | 남희석 | 1971 | 충남 | 016 | 66666666 | 180 | 2017-04-04 |
| PSH | 박수홍 | 1970 | 서울 | 010 | 00000000 | 183 | 2012-05-05 |
| SDY | 신동엽 | 1971 | 경기 | NULL | NULL | 176 | 2008-10-10 |
| YJS | 유재석 | 1972 | 서울 | 010 | 11111111 | 178 | 2008-08-08 |

# [실습 10-2] 인덱스 생성하고 활용하기

교재 368~373p 참고

2-2 회원 테이블에 어떤 인덱스가 설정되어 있는지 확인

```
SHOW INDEX FROM userTBL;
```

| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
|---|---|---|---|---|---|---|---|---|---|---|
| usertbl | 0 | PRIMARY | 1 | userID | A | 2 | NULL | NULL | | BTREE |

2-3 주소(addr) 열에 단순 보조 인덱스 생성

```
CREATE INDEX idx_userTBL_addr
    ON userTBL (addr);
```

2-4 생성된 보조 인덱스의 이름을 확인하고 결과에서 Non_unique 부분을 살펴보기

```
SHOW INDEX FROM userTBL;
```

| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
|---|---|---|---|---|---|---|---|---|---|---|
| usertbl | 0 | PRIMARY | 1 | userID | A | 2 | NULL | NULL | | BTREE |
| usertbl | 1 | idx_userTBL_addr | 1 | addr | A | 5 | NULL | NULL | | BTREE |

2-5 출생 연도(birthYear) 열에 고유 보조 인덱스 생성

```
CREATE UNIQUE INDEX idx_userTBL_birtyYear
      ON userTBL (birthYear);
```

**실행 결과**

Error Code: 1062. Duplicate entry '1971' for key 'idx_userTBL_birthYear'

# [실습 10-2] 인덱스 생성하고 활용하기

교재 368~373p 참고

2-6 이름(userName) 열에 고유 보조 인덱스를 생성하면 문제없이 생성

```
CREATE UNIQUE INDEX idx_userTBL_userName
    ON userTBL (userName);
SHOW INDEX FROM userTBL;
```

| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
|---|---|---|---|---|---|---|---|---|---|---|
| usertbl | 0 | PRIMARY | 1 | userID | A | 2 | NULL | NULL | | BTREE |
| usertbl | 0 | idx_userTBL_userName | 1 | userName | A | 10 | NULL | NULL | | BTREE |
| usertbl | 1 | idx_userTBL_addr | 1 | addr | A | 5 | NULL | NULL | | BTREE |

2-7 강호동과 이름이 같되 아이디가 GHD인 회원을 삽입

```
INSERT INTO userTBL VALUES ('GHD', '강호동', 1988, '미국', NULL, NULL, 172, NULL);
```

**실행 결과**

Error Code: 1062. Duplicate entry '강호동' for key 'idx_userTBL_userName'

2-8 이름(userName) 열과 출생 연도(birthYear) 열을 조합하여 인덱스 생성

```
CREATE INDEX idx_userTBL_userName_birthYear
    ON userTBL (userName, birthYear);
DROP INDEX idx_userTBL_userName ON userTBL;
SHOW INDEX FROM userTBL;
```

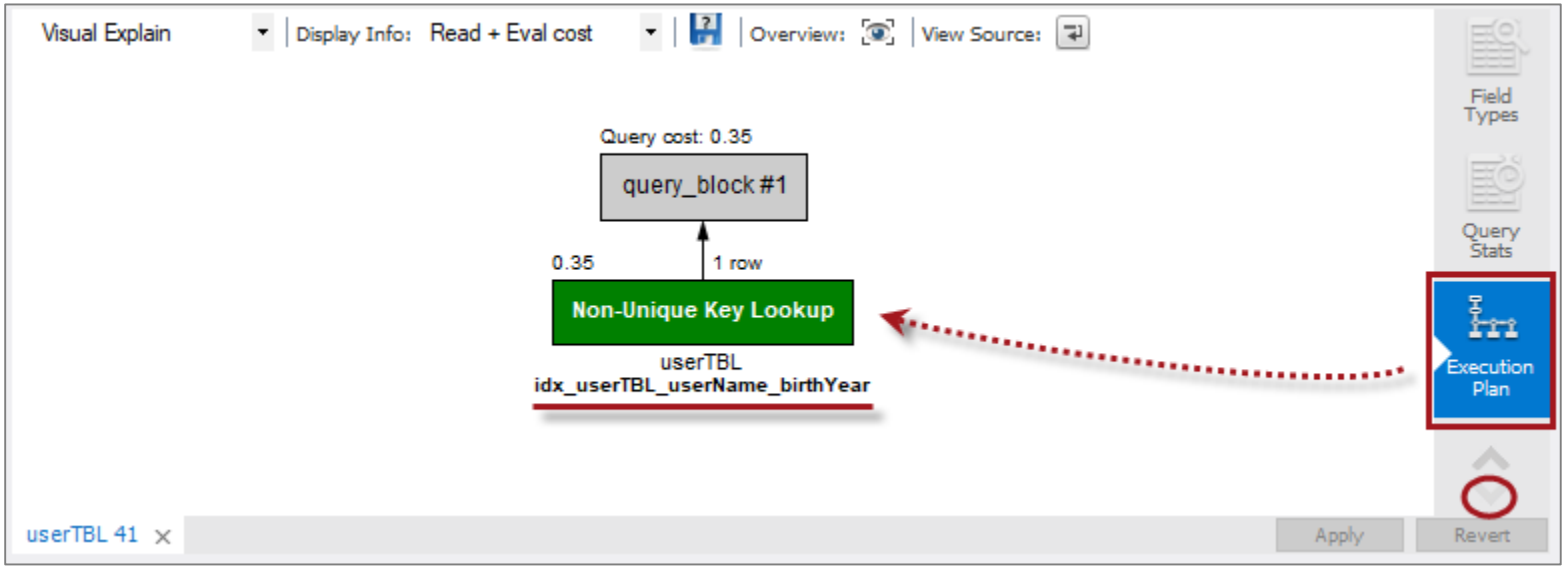
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type |
|---|---|---|---|---|---|---|---|---|---|---|
| usertbl | 0 | PRIMARY | 1 | userID | A | 2 | NULL | NULL | | BTREE |
| usertbl | 1 | idx_userTBL_addr | 1 | addr | A | 5 | NULL | NULL | | BTREE |
| usertbl | 1 | idx_userTBL_userName_birthYear | 1 | userName | A | 10 | NULL | NULL | | BTREE |
| usertbl | 1 | idx_userTBL_userName_birthYear | 2 | birthYear | A | 10 | NULL | NULL | | BTREE |

# [실습 10-2] 인덱스 생성하고 활용하기

교재 368~373p 참고

2-9 두 열이 조합된 조건문의 쿼리에도 해당 인덱스가 사용

```
SELECT * FROM userTBL WHERE userName = '신동엽' and birthYear = '1971';
```




2-10 휴대폰의 국번(mobile1) 열에 인덱스를 생성

```
CREATE INDEX idx_userTBL_mobile1
    ON userTBL (mobile1);
```

```
SELECT * FROM userTBL WHERE mobile1 = '011';
```

# [실습 10-2] 인덱스 생성하고 활용하기

교재 368~373p 참고

3 인덱스 삭제하기

3-1 회원 테이블(userTBL)에 생성된 인덱스의 이름 확인

```
SHOW INDEX FROM userTBL;
```

3-2 회원 테이블의 보조 인덱스 삭제

```
DROP INDEX idx_userTBL_addr ON userTBL;
DROP INDEX idx_userTBL_userName_birthYear ON userTBL;
DROP INDEX idx_userTBL_mobile1 ON userTBL;
```

3-3 자동으로 생성된 클러스터형 인덱스 삭제

```
ALTER TABLE userTBL DROP PRIMARY KEY;
```

# 5 인덱스의 생성 기준

- 인덱스 생성의 판단 기준
  - 인덱스는 열 단위에 생성
  - 인덱스는 WHERE 절에서 사용되는 열에 생성
  - WHERE 절에 사용되는 열이라도 자주 사용해야 가치가 있음
  - 데이터 중복도가 높은 열에는 인덱스를 만들어도 효과가 없음
  - 외래키를 설정한 열에는 자동으로 외래키 인덱스가 생성됨
  - 조인에 자주 사용되는 열에는 인덱스를 생성하는 것이 좋음
  - 데이터 변경(삽입, 수정, 삭제) 작업이 얼마나 자주 일어나는지 고려해야 함
  - 클러스터형 인덱스는 테이블당 하나만 생성할 수 있음
  - 테이블에 클러스터형 인덱스가 아예 없는 것이 좋은 경우도 있음
  - 사용하지 않는 인덱스는 제거