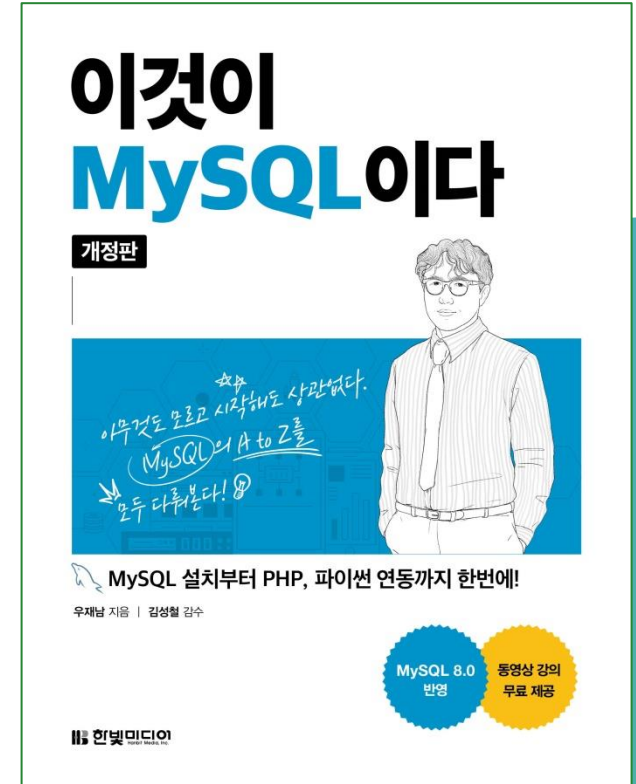


# 이것이 MySQL이다



저자 : 우재남

- CHAPTER 09 인덱스

- SECTION 01 인덱스의 개념
- SECTION 02 인덱스의 종류와 자동 생성

- 2.1 인덱스의 종류

- 2.2 자동으로 생성되는 인덱스

- SECTION 03 인덱스의 내부 작동

- 3.1 B-Tree(Balanced Tree, 균형 트리)

- 3.2 페이지 분할

- 3.3 클러스터형 인덱스와 보조 인덱스의 구조

- 3.4 클러스터형 인덱스와 보조 인덱스가 혼합되어 있을 경우

- CHAPTER 09 인덱스
  - SECTION 04 인덱스 생성/변경/삭제
    - 4.1 인덱스 생성
    - 4.2 인덱스 제거
  - SECTION 05 인덱스의 성능 비교
  - SECTION 06 결론 : 인덱스를 생성해야 하는 경우와 그렇지 않은 경우



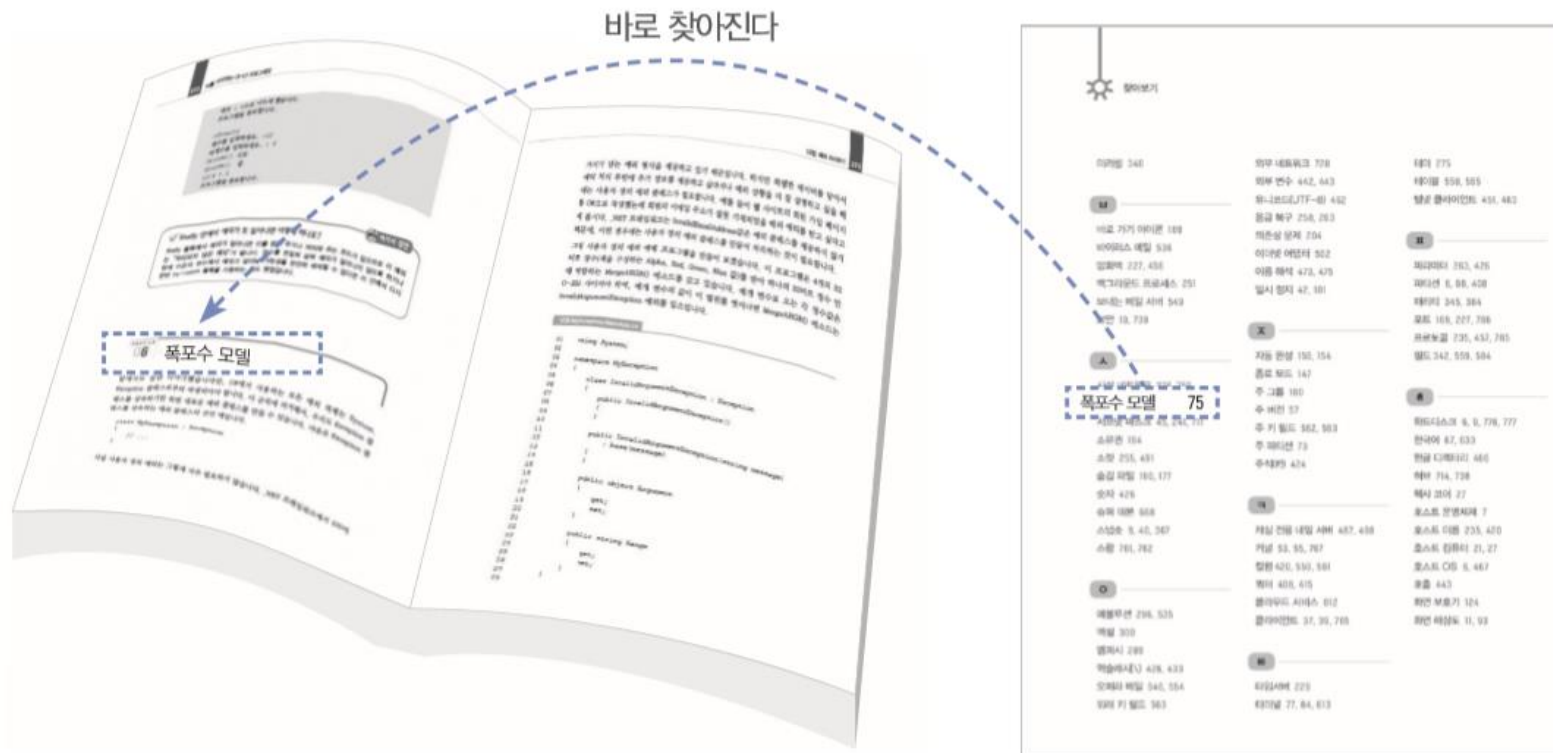
# CHAPTER 09 인덱스

데이터베이스 성능을 위해 중요한 역할을 하는 인덱스의 종류와 사용법에 대해서 살펴본다.

# SECTION 01 인덱스의 개념

## 인덱스(Index)란?

- 책의 <찾아보기>의 개념과 비슷
- 데이터를 좀 더 빠르게 찾을 수 있도록 해주는 도구



책의 제일 뒷부분

[그림 9-1] 책의 '찾아보기' 개념

# SECTION 01 인덱스의 개념

## 인덱스의 장단점

### ◦ 장점

- 검색 속도가 무척 빨라질 수 있음 (항상 그런 것은 아님)
- 쿼리의 부하가 줄어들어 시스템 전체의 성능 향상

### ◦ 단점

- 인덱스가 데이터베이스 공간을 차지해서 추가적인 공간 필요
  - 대략 데이터베이스 크기의 10% 정도의 추가 공간 필요
- 처음 인덱스 생성하는데 시간 소요
- 데이터의 변경 작업 (Insert, Update, Delete)이 자주 일어나는 경우 성능이 나빠질 수도 있음

# SECTION 02 인덱스의 종류와 자동 생성

## 인덱스의 종류

- 클러스터형 인덱스 (Clustered Index)
  - '영어 사전'과 같은 책
  - 테이블 당 한 개만 지정 가능
  - 행 데이터를 인덱스로 지정한 열에 맞춰 자동 정렬
- 보조 인덱스 (Secondary Index)
  - 책 뒤에 <찾아보기>가 있는 일반 책
  - 테이블당 여러 개도 생성 가능

## SECTION 02 인덱스의 종류와 자동 생성

### 자동으로 생성되는 인덱스

- 인덱스 실습 (P. 378 ~ 382)
  - sqlDB에서 작업 >> 테이블 생성
    - USE sqlldb;  
CREATE TABLE tbl1  
(  
    a INT PRIMARY KEY,  
    b INT,  
    c INT  
);
  - 인덱스 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	Index_comment
▶	tbl1	0	PRIMARY	1	a	A	0	NULL	NULL		BTREE		



# SECTION 02 인덱스의 종류와 자동 생성

## 자동으로 생성되는 인덱스

- 인덱스 실습 (P. 378 ~ 382)
  - Primary Key와 함께 Unique 제약 조건 생성
    - 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	Comment	Index_comment
▶	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		

## SECTION 02 인덱스의 종류와 자동 생성

### 자동으로 생성되는 인덱스

- 인덱스 실습 (P. 378 ~ 382)

- Primary Key 없이 Unique Key만 지정

- 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	Comment	Index_comment
▶	tbl3	0	a	1	a	A	0	HULL	HULL	YES	BTREE		
	tbl3	0	b	1	b	A	0	HULL	HULL	YES	BTREE		
	tbl3	0	c	1	c	A	0	HULL	HULL	YES	BTREE		

# SECTION 02 인덱스의 종류와 자동 생성

## 자동으로 생성되는 인덱스

- 인덱스 실습 (P. 378 ~ 382)

- UNIQUE에 클러스터형 인덱스 지정, UNIQUE에 NOT NULL이 포함되면 클러스터형 인덱스로 지정됨

- 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	Comment	Index_comment
▶	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		

## SECTION 02 인덱스의 종류와 자동 생성

### 자동으로 생성되는 인덱스

- 인덱스 실습 (P. 378 ~ 382)
  - UNIQUE에 NOT NULL과 PRIMARY KEY를 모두 지정
    - 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	Comment	Index_comment
▶	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		

## SECTION 02 인덱스의 종류와 자동 생성

### 자동으로 생성되는 인덱스

- 인덱스 실습 (P. 378 ~ 382)
  - 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,
  name    varchar(10) NOT NULL,
  birthYear  int NOT NULL,
  addr     nchar(2) NOT NULL
);
```

## SECTION 02 인덱스의 종류와 자동 생성

### 자동으로 생성되는 인덱스

- 인덱스 실습 (P. 378 ~ 382)

- 데이터 입력 및 확인

```
INSERT INTO usertbl VALUES('LSG', '이승기', 1987, '서울');  
INSERT INTO usertbl VALUES('KBS', '김범수', 1979, '경남');  
INSERT INTO usertbl VALUES('KKH', '김경호', 1971, '전남');  
INSERT INTO usertbl VALUES('JYP', '조용필', 1950, '경기');  
INSERT INTO usertbl VALUES('SSK', '성시경', 1979, '서울');  
SELECT * FROM usertbl;
```

	userID	name	birthYear	addr
▶	JYP	조용필	1950	경기
	KBS	김범수	1979	경남
	KKH	김경호	1971	전남
	LSG	이승기	1987	서울
	SSK	성시경	1979	서울

## SECTION 02 인덱스의 종류와 자동 생성

### 자동으로 생성되는 인덱스

- 인덱스 실습 (P. 378 ~ 382)

- userID 열의 Primary Key를 제거하고, name 열을 Primary Key로 지정

```
ALTER TABLE usertbl DROP PRIMARY KEY ;  
ALTER TABLE usertbl  
  ADD CONSTRAINT pk_name PRIMARY KEY(name);  
SELECT * FROM usertbl;
```

	userID	name	birthYear	addr
▶	KKH	김경호	1971	전남
	KBS	김범수	1979	경남
	SSK	성시경	1979	서울
	LSG	이슬기	1987	서울
	JYP	조용필	1950	경기

## SECTION 02 인덱스의 종류와 자동 생성

### 자동으로 생성되는 인덱스

- 인덱스의 특징

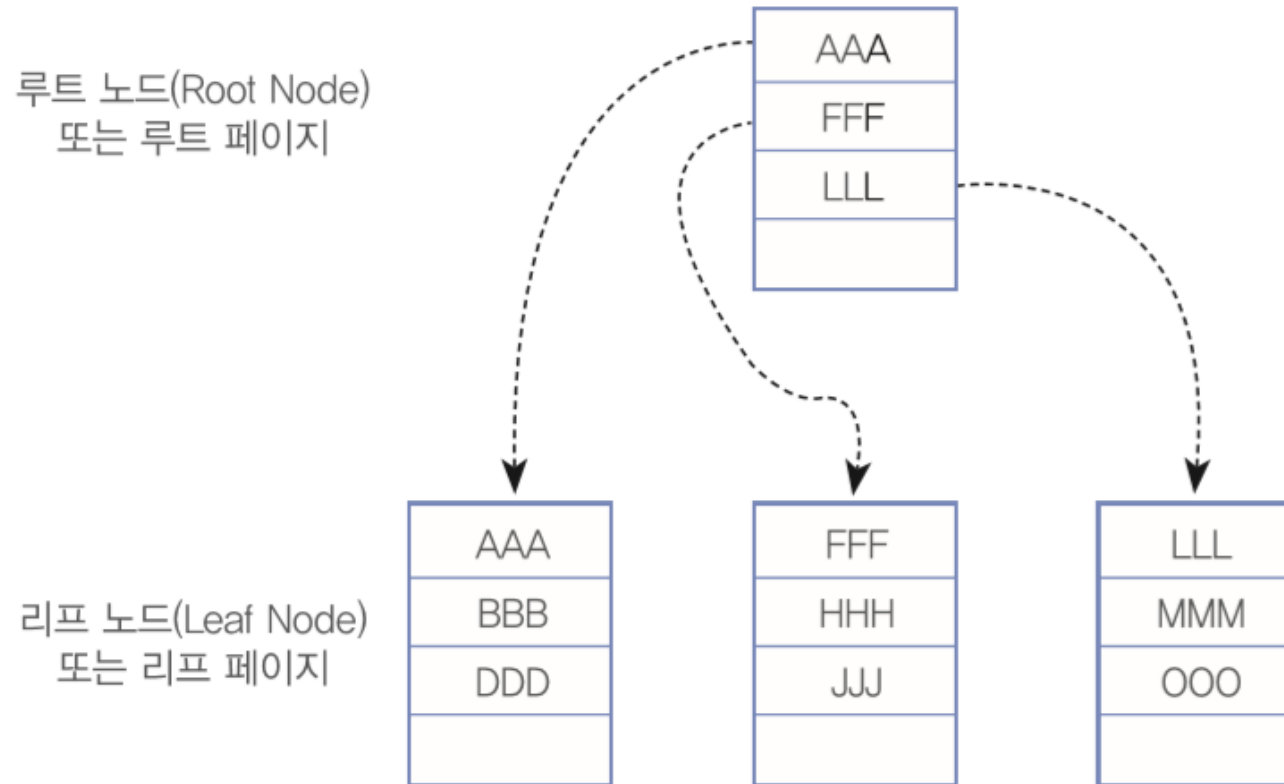
- PRIMARY KEY로 지정한 열은 클러스터형 인덱스가 생성
- UNIQUE NOT NULL로 지정한 열은 클러스터형 인덱스 생성
- UNIQUE(또는 UNIQUE NULL)로 지정한 열은 보조 인덱스 생성
- PRIMARY KEY와 UNIQUE NOT NULL이 존재
- PRIMARY KEY와 UNIQUE NOT NULL이 있으면 PRIMARY KEY에 지정한 열에 우선 클러스터형 인덱스 생성
- PRIMARY KEY로 지정한 열로 데이터가 오름차순 정렬



## SECTION 03 인덱스의 내부 작동

### B-Tree(Balanced Tree, 균형 트리)

- 자료 구조'에 나오는 범용적으로 사용되는 데이터 구조
- 인덱스 표현할 때와 그 외에도 많이 사용

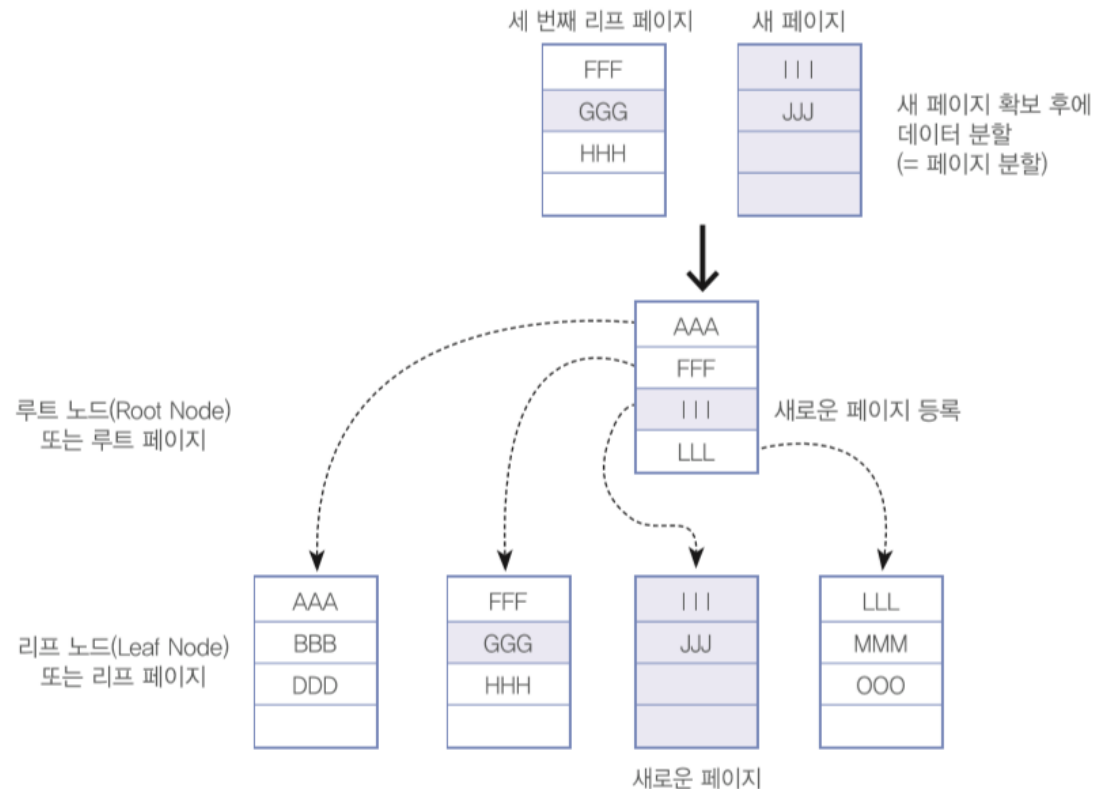


[그림 9-10] B-Tree의 기본 구조

## SECTION 03 인덱스의 내부 작동

### 페이지 분할

- 인덱스 구성시 SELECT 문의 효율성 향상
- 인덱스 구성시 INSERT 문이 일어날 경우 속도 저하되는 단점
  - 주어진 공간 이상으로 데이터 들어가면 페이지 분할 일어남



[그림 9-12] GGG 삽입 후

## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 인덱스 없는 테이블의 예시

```
CREATE DATABASE IF NOT EXISTS testdb;
USE testdb;
DROP TABLE IF EXISTS clustertbl;
CREATE TABLE clustertbl -- Cluster Table 약자
( userID CHAR(8) ,
  name VARCHAR(10)
);
INSERT INTO clustertbl VALUES('LSG', '이승기');
INSERT INTO clustertbl VALUES('KBS', '김범수');
INSERT INTO clustertbl VALUES('KKH', '김경호');
INSERT INTO clustertbl VALUES('JYP', '조용필');
INSERT INTO clustertbl VALUES('SSK', '성시경');
INSERT INTO clustertbl VALUES('LJB', '임재범');
INSERT INTO clustertbl VALUES('YJS', '윤종신');
INSERT INTO clustertbl VALUES('EJW', '은지원');
INSERT INTO clustertbl VALUES('JKW', '조관우');
INSERT INTO clustertbl VALUES('BBK', '바비킴');
```

## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 인덱스 없는 테이블의 예시

데이터 페이지  
(Heap 영역)

1000	
LSG	이승기
KBS	김범수
KKH	김경호
JYP	조용필

1001	
SSK	성시경
LJB	임재범
YJS	윤종신
EJW	은지원

1002	
JKW	조관우
BBK	바비킴

[그림 9-14] 인덱스 없는 테이블의 내부 구성

	userID	name
▶	LSG	이승기
	KBS	김범수
	KKH	김경호
	JYP	조용필
	SSK	성시경
	LJB	임재범
	YJS	윤종신
	EJW	은지원
	JKW	조관우
	BBK	바비킴

## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 클러스터형 인덱스 구성한 테이블 구조
  - userID를 Primary Key로 지정하면 클러스터형 인덱스로 구성됨

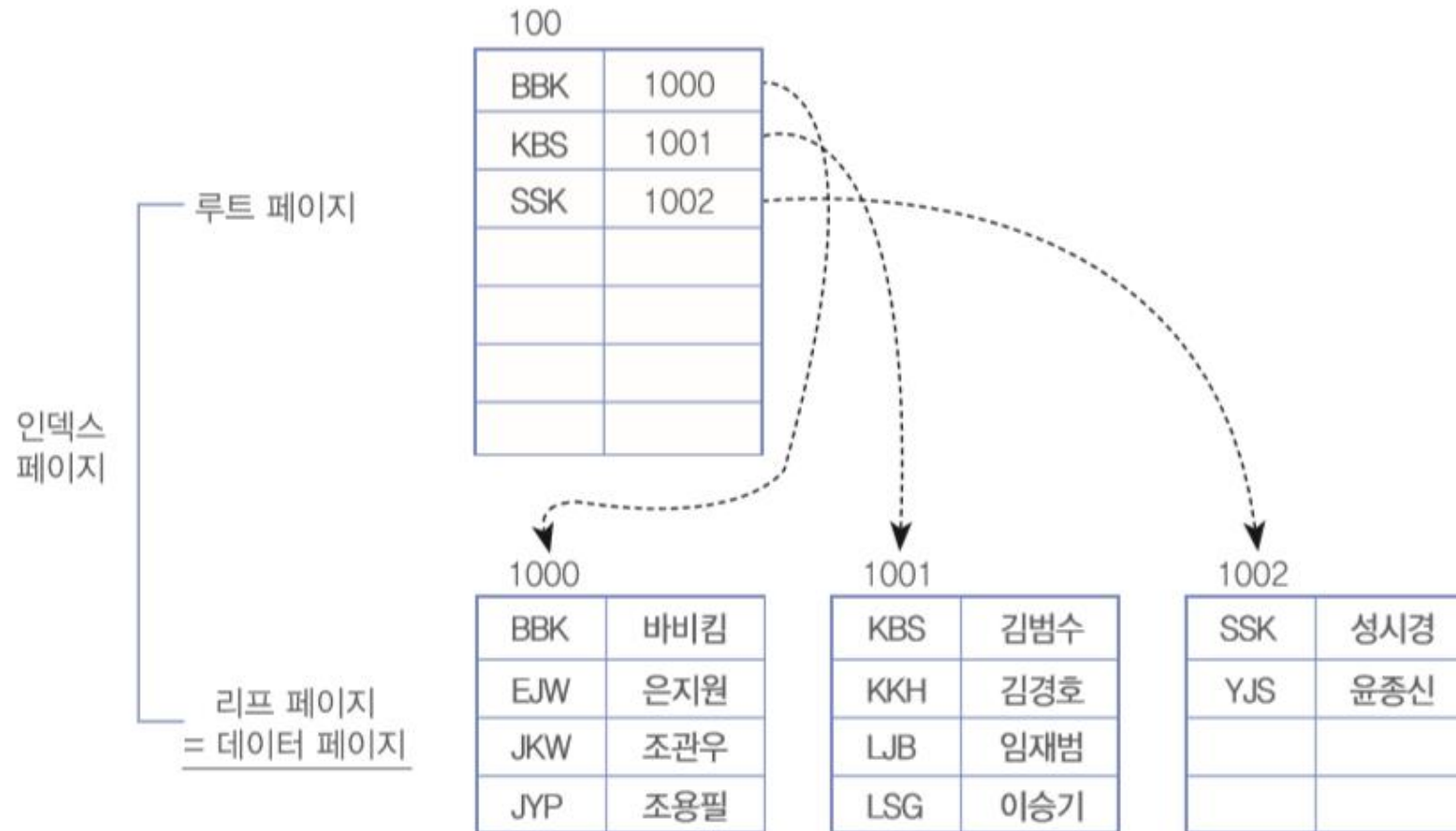
```
ALTER TABLE clustertbl  
  ADD CONSTRAINT PK_clustertbl_userID  
    PRIMARY KEY (userID);
```

	userID	name
▶	BBK	바비킴
	EJW	은지원
	JKW	조관우
	JYP	조용필
	KBS	김범수
	KKH	김경호
	LJB	임재범
	LSG	이승기
	SSK	성시경
	YJS	윤종신

## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 클러스터형 인덱스 구성한 테이블 구조



[그림 9-17] 클러스터형 인덱스의 구성 후

## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 보조 인덱스 구성한 테이블 구조

```
CREATE DATABASE IF NOT EXISTS testdb;
USE testdb;
DROP TABLE IF EXISTS secondarytbl;
CREATE TABLE secondarytbl -- Secondary Table 약자
( userID CHAR(8),
  name VARCHAR(10)
);
INSERT INTO secondarytbl VALUES('LSG', '이승기');
INSERT INTO secondarytbl VALUES('KBS', '김범수');
INSERT INTO secondarytbl VALUES('KKH', '김경호');
INSERT INTO secondarytbl VALUES('JYP', '조용필');
INSERT INTO secondarytbl VALUES('SSK', '성시경');
INSERT INTO secondarytbl VALUES('LJB', '임재범');
INSERT INTO secondarytbl VALUES('YJS', '윤종신');
INSERT INTO secondarytbl VALUES('EJW', '은지원');
INSERT INTO secondarytbl VALUES('JKW', '조관우');
INSERT INTO secondarytbl VALUES('BBK', '바비킴');
```

## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 보조 인덱스 구성한 테이블 구조

```
ALTER TABLE secondarytbl  
  ADD CONSTRAINT UK_secondarytbl_userID  
    UNIQUE (userID);
```

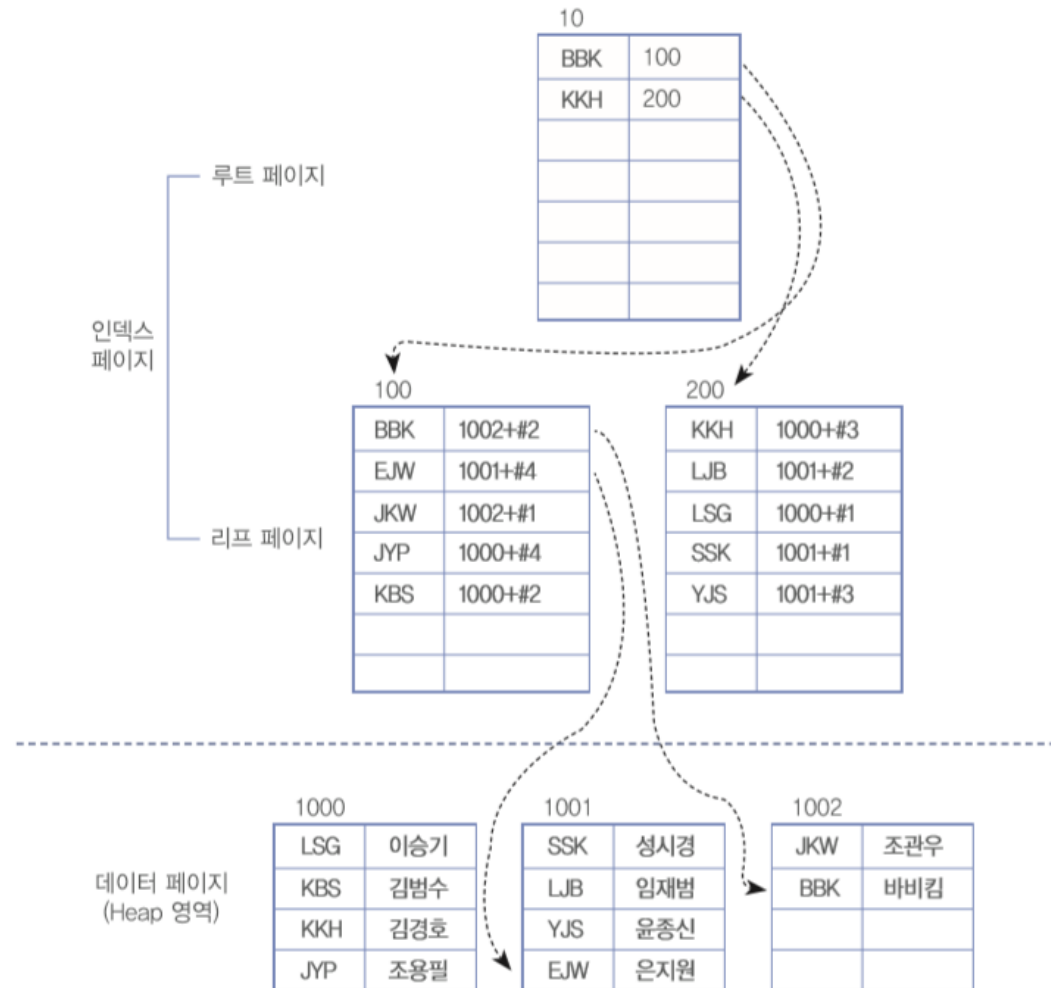
	userID	name
▶	LSG	이승기
	KBS	김범수
	KKH	김경호
	JYP	조용필
	SSK	성시경
	LJB	임재범
	YJS	윤종신
	EJW	은지원
	JKW	조관우
	BBK	바비킴



## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 보조 인덱스 구성한 테이블 구조

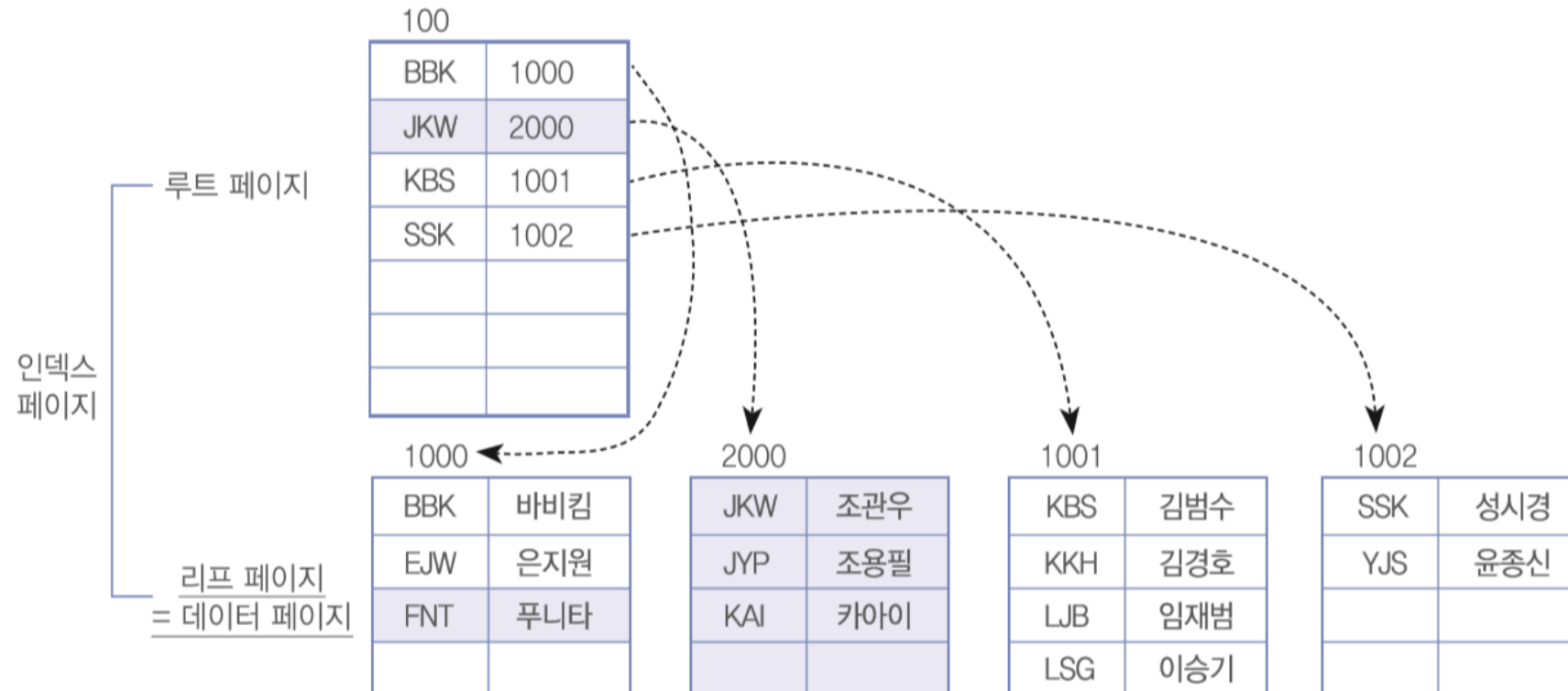


## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 클러스터 인덱스에 새로운 데이터 입력

```
INSERT INTO clustertbl VALUES('FNT', '푸니타');  
INSERT INTO clustertbl VALUES('KAI', '카아이');
```

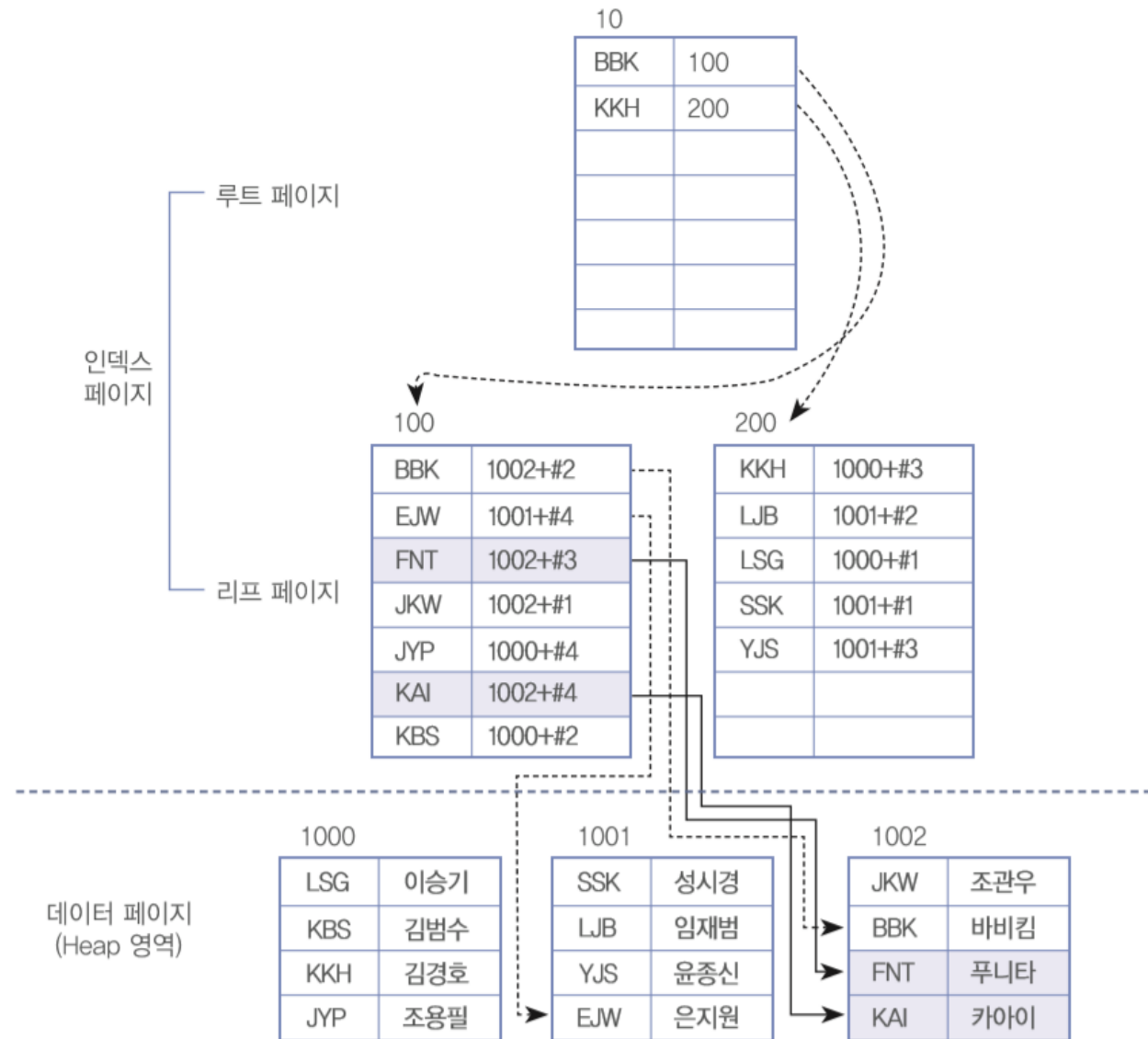


## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 보조 인덱스에 새로운 데이터 입력

```
INSERT INTO secondarytbl VALUES('FNT', '푸니타');  
INSERT INTO secondarytbl VALUES('KAI', '카아이');
```



[그림 9-21] 보조 인덱스에 두 행의 추가 후

## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 클러스터형 인덱스의 특징

- 클러스터형 인덱스의 생성 시에는 데이터 페이지 전체 다시 정렬
  - 이미 대용량의 데이터가 입력된 상태라면 업무시간에 클러스터형 인덱스 생성하는 것은 심각한 시스템 부하
- 인덱스 자체의 리프 페이지가 곧 데이터
  - 인덱스 자체에 데이터가 포함되어 있음
- 클러스터형 인덱스는 보조 인덱스보다 검색 속도는 더 빠름
  - 데이터의 입력/수정/삭제는 더 느림
- 클러스터형 인덱스는 성능이 좋지만 테이블에 한 개만 생성 가능
  - 어느 열에 클러스터형 인덱스 생성하는지에 따라 시스템의 성능이 달라짐

## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 보조 인덱스의 특징

- 보조 인덱스 생성시 별도의 페이지에 인덱스 구성
- 인덱스 자체의 리프 페이지는 데이터가 아니고 데이터가 위치하는 주소 값(RID)
- 클러스터형보다 검색 속도는 더 느림
  - 데이터의 입력/수정/삭제는 덜 느림
- 보조 인덱스는 여러 개 생성할 수 있음
  - 남용할 경우에는 시스템 성능을 떨어뜨리는 결과 발생

# SECTION 03 인덱스의 내부 작동

## 클러스터형 인덱스와 보조 인덱스의 구조

- 클러스터형 인덱스와 보조 인덱스가 혼합되어 있을 경우

```
CREATE DATABASE IF NOT EXISTS testdb;
USE testdb;
DROP TABLE IF EXISTS mixedtbl;
CREATE TABLE mixedtbl
( userID CHAR(8) NOT NULL ,
  name VARCHAR(10) NOT NULL,
  addr char(2)
);
INSERT INTO mixedtbl VALUES('LSG', '이승기', '서울');
INSERT INTO mixedtbl VALUES('KBS', '김범수', '경남');
INSERT INTO mixedtbl VALUES('KKH', '김경호', '전남');
INSERT INTO mixedtbl VALUES('JYP', '조용필', '경기');
INSERT INTO mixedtbl VALUES('SSK', '성시경', '서울');
INSERT INTO mixedtbl VALUES('LJB', '임재범', '서울');
INSERT INTO mixedtbl VALUES('YJS', '윤종신', '경남');
INSERT INTO mixedtbl VALUES('EJW', '은지원', '경북');
INSERT INTO mixedtbl VALUES('JKW', '조관우', '경기');
INSERT INTO mixedtbl VALUES('BBK', '바비킴', '서울');
```

데이터 페이지  
(Head 영역)

1000	1001	1002
LSG	SSK	JKW
KBS	LJB	BBK
KKH	YJS	
JYP	EJW	
이승기	성시경	조관우
김범수	임재범	바비킴
김경호	윤종신	
조용필	은지원	
서울	서울	경기
경남	서울	서울
전남	경남	
경기	경북	

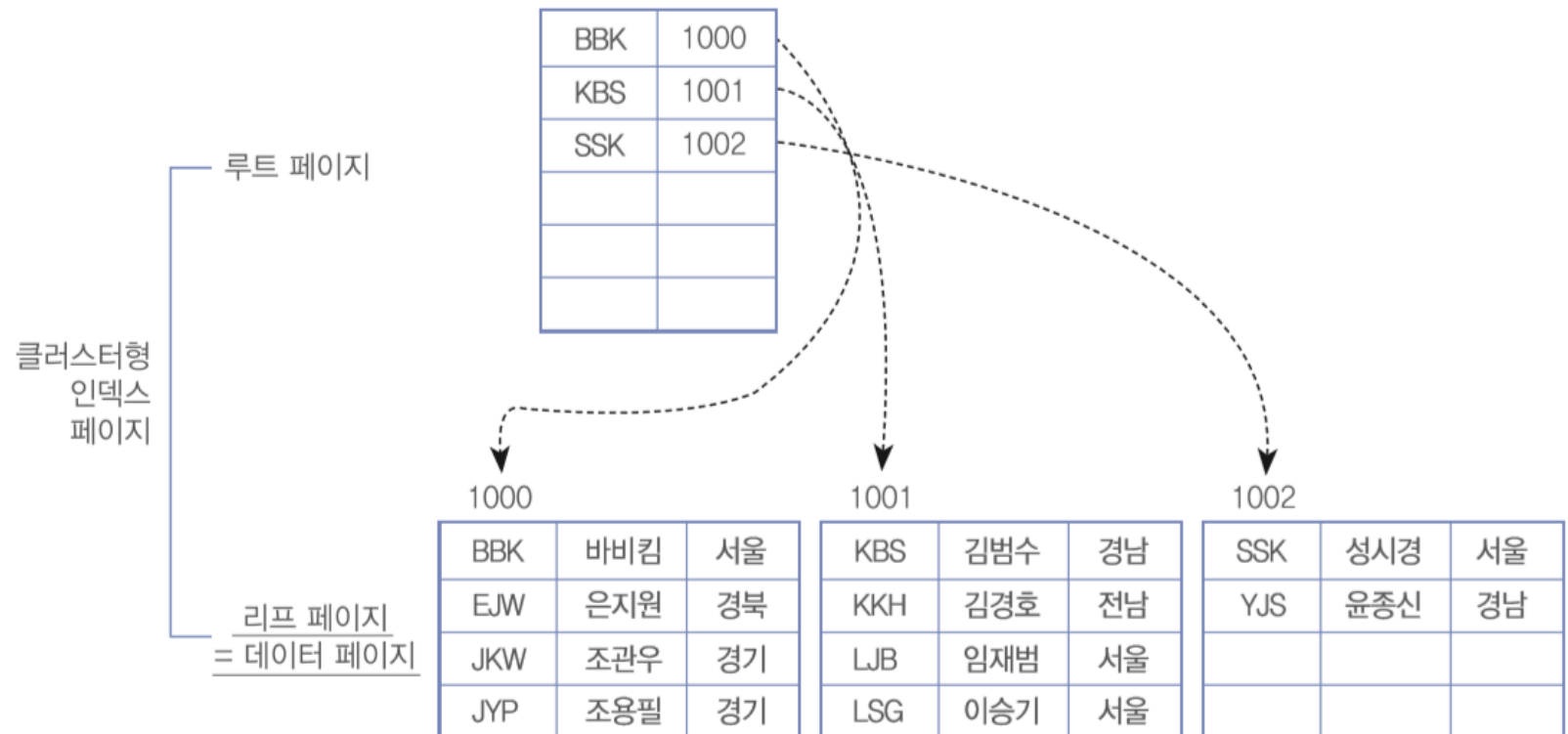
[그림 9-22] 인덱스가 없는 데이터 페이지

## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 클러스터형 인덱스와 보조 인덱스가 혼합되어 있을 경우

```
ALTER TABLE mixedtbl  
ADD CONSTRAINT PK_mixedtbl_userID  
PRIMARY KEY (userID);
```



## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 클러스터형 인덱스와 보조 인덱스가 혼합되어 있을 경우
  - UNIQUE 제약 조건으로 보조 인덱스 추가

```
ALTER TABLE mixedtbl  
  ADD CONSTRAINT UK_mixedtbl_name  
    UNIQUE (name) ;
```

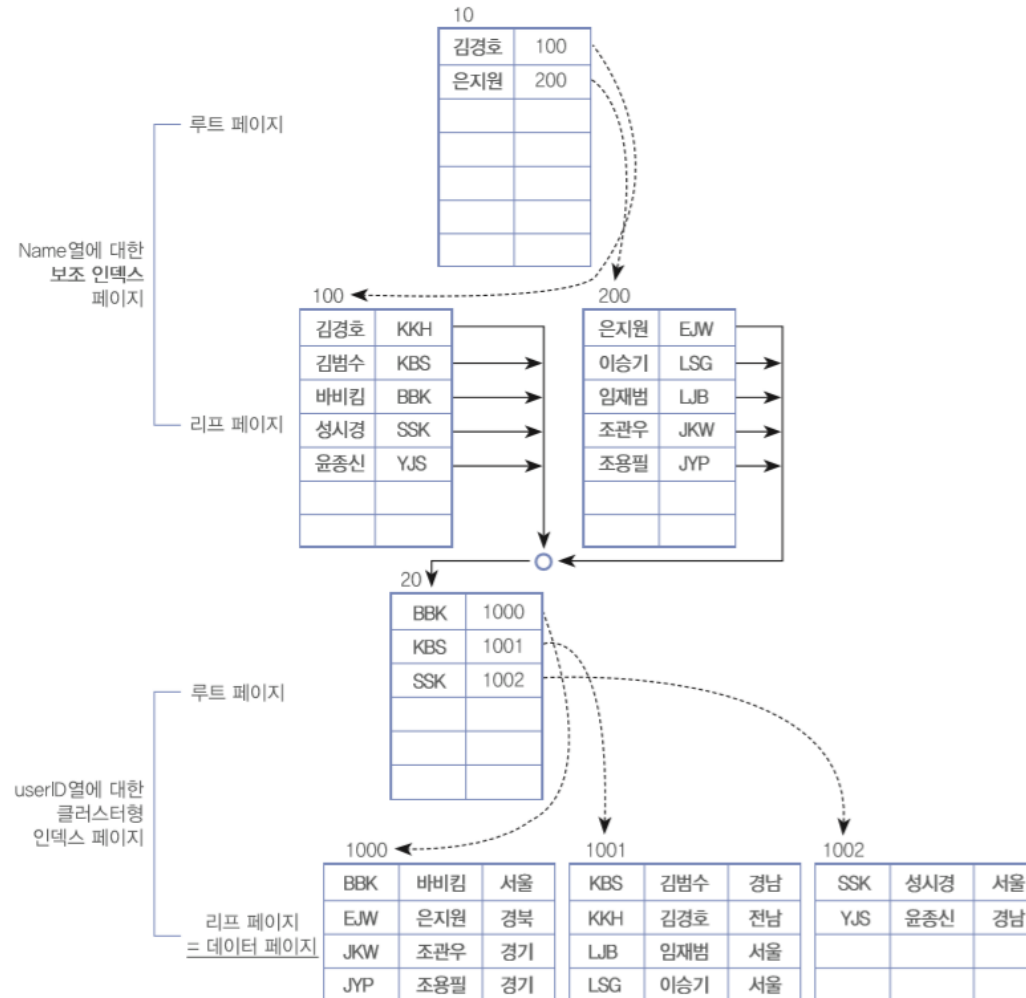
	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
▶	mixedtbl	0	PRIMARY	1	userID	A	10	NULL	NULL		BTREE		
	mixedtbl	0	UK_mixedTbl_name	1	name	A	10	NULL	NULL		BTREE		



## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 클러스터형 인덱스와 보조 인덱스가 혼합되어 있을 경우



## SECTION 03 인덱스의 내부 작동

### 클러스터형 인덱스와 보조 인덱스의 구조

- 클러스터형 인덱스와 보조 인덱스가 혼합되어 있을 경우
  - 보조 인덱스를 검색한 후에 다시 클러스터형 인덱스를 검색해야 하므로 약간의 손해를 볼 수도 있겠지만, 데이터의 삽입 때문에 보조 인덱스를 대폭 재구성하게 되는 큰 부하는 걸리지 않음
  - 보조 인덱스와 혼합되어 사용되는 경우 되도록이면 클러스터형 인덱스로 설정할 열은 적은 자릿수의 열을 선택하는 것이 바람직함
  - 인덱스를 검색하기 위한 일차 조건
    - WHERE절에 해당 인덱스를 생성한 열의 이름이 나와야 함
    - WHERE절에 해당 인덱스를 생성한 열 이름이 나와도 인덱스를 사용하지 않는 경우도 많음

# SECTION 04 인덱스 생성/변경/삭제

## 인덱스 생성

- 인덱스 생성 문법

형식:

```
CREATE [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name  
    [index_type]  
    ON tbl_name (key_part,...)  
    [index_option]  
    [algorithm_option | lock_option] ...
```

key\_part:

```
{col_name [(length)] | (expr)} [ASC | DESC]
```

index\_option:

```
    KEY_BLOCK_SIZE [=] value  
    | index_type  
    | WITH PARSER parser_name  
    | COMMENT 'string'  
    | {VISIBLE | INVISIBLE}
```

index\_type:

```
    USING {BTREE | HASH}
```

algorithm\_option:

```
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}
```

lock\_option:

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

## SECTION 04 인덱스 생성/변경/삭제

### 인덱스 제거

- 인덱스 삭제 형식

```
형식 :  
DROP INDEX index_name ON tbl_name  
    [algorithm_option ! lock_option] ...  
  
algorithm_option:  
    ALGORITHM [=] {DEFAULT|INPLACE|COPY}  
  
lock_option:  
    LOCK [=] {DEFAULT|NONE|SHARED|EXCLUSIVE}
```

- 간단히 인덱스 삭제하는 구문

```
DROP INDEX 인덱스이름 ON 테이블이름;
```

# SECTION 05 인덱스의 성능 비교

## 인덱스의 성능 비교

- 인덱스 없는 경우, 클러스터형 인덱스, 보조 인덱스를 설정하여 쿼리 속도 비교, 서버 부하 비교
  - 실습할 데이터베이스 만들기
    - `CREATE DATABASE IF NOT EXISTS indexdb;`
  - employees의 employees의 개수를 파악
    - `USE indexdb;`
    - `SELECT COUNT(*) FROM employees.employees;`
  - 테이블 3개로 복사
    - `CREATE TABLE emp SELECT * FROM employees.employees ORDER BY RAND();`
    - `CREATE TABLE emp_c SELECT * FROM employees.employees ORDER BY RAND();`
    - `CREATE TABLE emp_Se SELECT * FROM employees.employees ORDER BY RAND();`

# SECTION 05 인덱스의 성능 비교

## 인덱스의 성능 비교

- 인덱스 없는 경우, 클러스터형 인덱스, 보조 인덱스를 설정하여 쿼리 속도 비교, 서버 부하 비교

- 테이블 순서 확인

- SELECT \* FROM emp LIMIT 5;
- SELECT \* FROM emp\_c LIMIT 5;
- SELECT \* FROM emp\_Se LIMIT 5;

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	443622	1963-05-02	Elvis	Bage	M	1991-07-01
	293770	1959-08-23	Gianluca	Validov	M	1994-03-23
	463994	1962-01-11	Jouni	Borstler	F	1986-08-18
	448204	1953-05-31	Mani	Ebeling	M	1993-09-23
	269311	1953-09-09	Gino	Chepyzhov	F	1995-09-28

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	474999	1962-07-29	Leon	Staudhammer	M	1989-11-07
	495190	1958-04-07	Maia	Miara	F	1989-12-31
	264498	1955-12-07	Gaetan	Hertweck	M	1991-04-25
	406048	1956-02-03	Abdelghani	Alpay	F	1993-03-25
	259326	1955-06-30	Sachar	Perly	M	1988-08-25

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	459564	1960-07-06	Heping	Baek	M	1985-06-06
	61157	1954-02-25	Shim	McConalogue	F	1991-08-23
	71977	1964-05-08	Jordanka	Barbanera	M	1993-04-29
	497952	1953-01-31	Valeri	Plotkin	M	1997-05-16
	471926	1962-10-18	Hirochika	Chaudhury	F	1999-06-02

# SECTION 05 인덱스의 성능 비교

## 인덱스의 성능 비교

- 인덱스 없는 경우, 클러스터형 인덱스, 보조 인덱스를 설정하여 쿼리 속도 비교, 서버 부하 비교
  - SHOW TABLE STATUS문으로 테이블에 인덱스 있는지 확인
    - 세 테이블 모두 인덱스 없음

	Name	Engine	Version	Row_format	Rows	Avg_row_length	Data_length	Max_data_length	Index_length	Data_free	Auto_increment	Create_time
▶	emp	InnoDB	10	Dynamic	299088	57	17317888	0	0	4194304	<small>NULL</small>	2016-03-06 11:05:51
	emp_c	InnoDB	10	Dynamic	299841	57	17317888	0	0	4194304	<small>NULL</small>	2016-03-06 11:05:54
	emp_se	InnoDB	10	Dynamic	299389	57	17317888	0	0	4194304	<small>NULL</small>	2016-03-06 11:05:57

# SECTION 05 인덱스의 성능 비교

## 인덱스의 성능 비교

- 인덱스 없는 경우, 클러스터형 인덱스, 보조 인덱스를 설정하여 쿼리 속도 비교, 서버 부하 비교
  - emp\_c에는 클러스터형 인덱스(=Primary Key 인덱스)
  - emp\_Se에는 보조 인덱스를 생성
    - ALTER TABLE emp\_c ADD PRIMARY KEY(emp\_no);
    - ALTER TABLE emp\_Se ADD INDEX idx\_emp\_no (emp\_no);

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	443622	1963-05-02	Elvis	Bage	M	1991-07-01
	293770	1959-08-23	Gianluca	Validov	M	1994-03-23
	463994	1962-01-11	Jouni	Borstler	F	1986-08-18
	448204	1953-05-31	Mani	Ebeling	M	1993-09-23
	269311	1953-09-09	Gino	Chepyzhov	F	1995-09-28

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	10001	1953-09-02	Georgi	Facello	M	1986-06-26
	10002	1964-06-02	Bezalel	Simmel	F	1985-11-21
	10003	1959-12-03	Parto	Bamford	M	1986-08-28
	10004	1954-05-01	Chirstian	Koblick	M	1986-12-01
	10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12

	emp_no	birth_date	first_name	last_name	gender	hire_date
▶	459564	1960-07-06	Heping	Baek	M	1985-06-06
	61157	1954-02-25	Shim	McConalogue	F	1991-08-23
	71977	1964-05-08	Jordanka	Barbanera	M	1993-04-29
	497952	1953-01-31	Valeri	Plotkin	M	1997-05-16
	471926	1962-10-18	Hirochika	Chaudhury	F	1999-06-02



# SECTION 05 인덱스의 성능 비교

## 인덱스의 성능 비교

- 인덱스 없는 경우, 클러스터형 인덱스, 보조 인덱스를 설정하여 쿼리 속도 비교, 서버 부하 비교
  - 생성한 인덱스 적용
    - ANALYZE문 사용
      - ANALYZE TABLE emp, emp\_c, emp\_Se;

	Table	Op	Msg_type	Msg_text
▶	indexdb.emp	analyze	status	OK
	indexdb.emp_c	analyze	status	OK
	indexdb.emp_se	analyze	status	OK

# SECTION 05 인덱스의 성능 비교

## 인덱스의 성능 비교

- 인덱스 없는 경우, 클러스터형 인덱스, 보조 인덱스를 설정하여 쿼리 속도 비교, 서버 부하 비교
  - 인덱스 생성 후 테이블 인덱스 확인

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
--	-------	------------	----------	--------------	-------------	-----------	-------------	----------	--------	------	------------

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
▶	emp_c	0	PRIMARY	1	emp_no	A	299468	NULL	NULL		BTREE

	Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
▶	emp_se	1	idx_emp_no	1	emp_no	A	299339	NULL	NULL		BTREE

	Name	Engine	Version	Row_format	Rows	Avg_row_length	Data_length	Max_data_length	Index_length	Data_free
▶	emp	InnoDB	10	Dynamic	299238	57	17317888	0	0	4194304
	emp_c	InnoDB	10	Dynamic	299468	57	17317888	0	0	2097152
	emp_se	InnoDB	10	Dynamic	299339	57	17317888	0	5783552	2097152

## SECTION 05 인덱스의 성능 비교

### 인덱스의 성능 비교

- 인덱스 없는 경우, 클러스터형 인덱스, 보조 인덱스를 설정하여 쿼리 속도 비교, 서버 부하 비교
  - 인덱스 생성 후 테이블 인덱스 확인
    - emp 테이블은 인덱스 없음
    - emp\_c는 클러스터형(PRIMARY) 인덱스 생성되어 Data\_free 영역이 줄어듦
    - emp\_Se는 보조 인덱스 생성되어 데이터 변화 없음, 인덱스 페이지만 추가 생성됨.

# SECTION 05 인덱스의 성능 비교

## 인덱스의 성능 비교

- 인덱스 없는 경우, 클러스터형 인덱스, 보조 인덱스를 설정하여 쿼리 속도 비교, 서버 부하 비교
  - MySQL 전체의 시스템 상태 초기화
  - 인덱스 없는 emp 테이블 조회
    - SHOW GLOBAL STATUS LIKE 'Innodb\_pages\_read'; -- 쿼리 실행 전의 읽은 페이지 수
    - SELECT \* FROM emp WHERE emp\_no = 100000;
    - SHOW GLOBAL STATUS LIKE 'Innodb\_pages\_read'; -- 쿼리 실행 후에 읽은 페이지 수

The diagram illustrates the process of measuring database performance. It shows three tables. The top table is a status table with columns 'Variable\_name' and 'Value'. The middle table is the 'emp' table with columns 'emp\_no', 'birth\_date', 'first\_name', 'last\_name', 'gender', and 'hire\_date'. The bottom table is another status table with columns 'Variable\_name' and 'Value'. A red dotted arrow points from the 'Value' column of the top status table (containing 1032) to the 'emp' table, and another red dotted arrow points from the 'emp' table to the 'Value' column of the bottom status table (containing 2090).

Variable_name	Value
Innodb_pages_read	1032

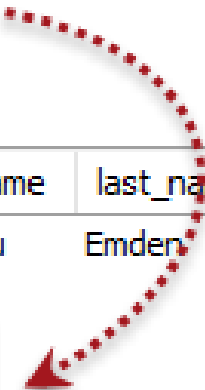
emp_no	birth_date	first_name	last_name	gender	hire_date
100000	1956-01-11	Hiroyasu	Emden	M	1991-07-02

Variable_name	Value
Innodb_pages_read	2090

## SECTION 05 인덱스의 성능 비교

### 인덱스의 성능 비교

- 인덱스 없는 경우, 클러스터형 인덱스, 보조 인덱스를 설정하여 쿼리 속도 비교, 서버 부하 비교
  - 클러스터형 인덱스가 있는 테이블 조회
    - SHOW GLOBAL STATUS LIKE 'Innodb\_pages\_read'; -- 쿼리 실행 전의 읽은 페이지 수
    - SELECT \* FROM emp\_c WHERE emp\_no = 100000;
    - SHOW GLOBAL STATUS LIKE 'Innodb\_pages\_read'; -- 쿼리 실행 후의 읽은 페이지



Variable_name	Value
Innodb_pages_read	3159

emp_no	birth_date	first_name	last_name	gender	hire_date
100000	1956-01-11	Hiroyasu	Emden	M	1991-07-02

Variable_name	Value
Innodb_pages_read	3162

## SECTION 05 인덱스의 성능 비교

### 인덱스의 성능 비교

- 인덱스 없는 경우, 클러스터형 인덱스, 보조 인덱스를 설정하여 쿼리 속도 비교, 서버 부하 비교
  - 보조 인덱스가 있는 테이블 조회
    - SHOW GLOBAL STATUS LIKE 'Innodb\_pages\_read'; -- 쿼리 실행 전의 읽은 페이지 수
    - SELECT \* FROM emp\_Se WHERE emp\_no = 100000;
    - SHOW GLOBAL STATUS LIKE 'Innodb\_pages\_read'; -- 쿼리 실행 후의 읽은 페이지 수

Variable_name	Value
Innodb_pages_read	3162

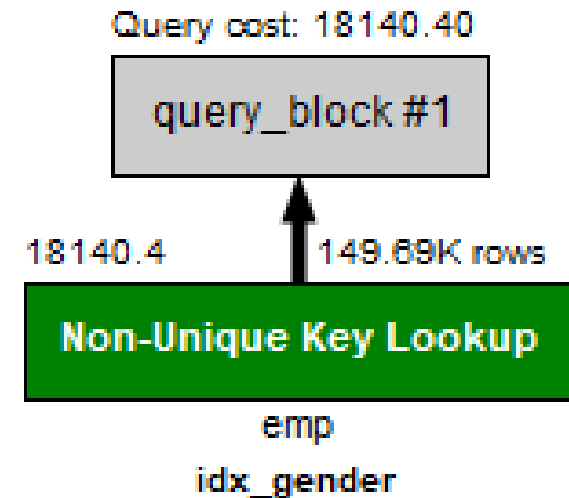
emp_no	birth_date	first_name	last_name	gender	hire_date
100000	1956-01-11	Hiroyasu	Emden	M	1991-07-02

Variable_name	Value
Innodb_pages_read	3167

# SECTION 05 인덱스의 성능 비교

## 인덱스의 성능 비교

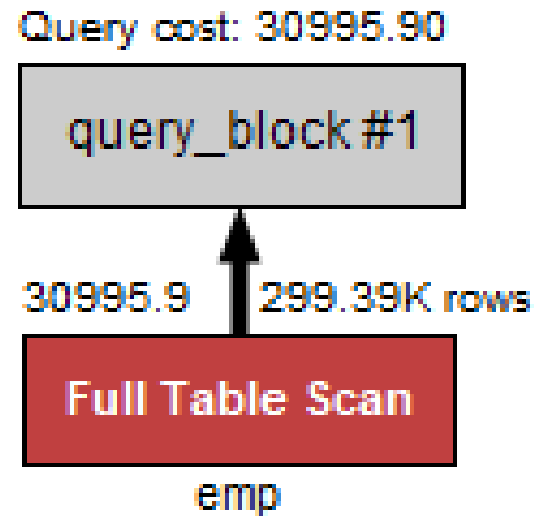
- 인덱스 없는 경우, 클러스터형 인덱스, 보조 인덱스를 설정하여 쿼리 속도 비교, 서버 부하 비교
  - 데이터 중복도에 따른 인덱스의 효용
  - 인덱스를 만들지 않은 emp테이블의 gender(성별)열에 인덱스 생성
    - ALTER TABLE emp ADD INDEX idx\_gender (gender);
    - ANALYZE TABLE Emp; -- 생성한 인덱스를 통계에 적용시킴
    - SHOW INDEX FROM Emp;
    - SELECT \* FROM emp WHERE gender = 'M' LIMIT 500000;
  - 쿼리 비용 약 2만 정도 나옴
  - Data Read 19MB 정도 읽음



## SECTION 05 인덱스의 성능 비교

### 인덱스의 성능 비교

- 인덱스 없는 경우, 클러스터형 인덱스, 보조 인덱스를 설정하여 쿼리 속도 비교, 서버 부하 비교
  - 강제로 인덱스를 사용하지 못하게 한 후에 실행 계획 비교
    - `SELECT * FROM emp IGNORE INDEX (idx_gender) WHERE gender = 'M' LIMIT 500000;`
  - 쿼리 비용 약 3만 정도 나옴





## SECTION 05 인덱스의 성능 비교

### 인덱스의 성능 비교

- 실습 결과

- 데이터의 중복도가 높은 경우에, 인덱스 사용하는 것이 효율이 있음
- 하지만 인덱스의 관리 비용과 INSERT 등의 구문에서는 오히려 성능이 저하될 수 있다는 점 등을 고려하면 **인덱스가 반드시 바람직하다고 보기는 어려움**

## SECTION 06 결론 : 인덱스를 생성해야 하는 경우와 그렇지 않은 경우

### 인덱스에 대한 결론

- 인덱스는 열 단위에 생성
  - 두 개 이상의 열을 조합해서 인덱스 생성 가능
- WHERE절에서 사용되는 열에 인덱스를 만들어야 함
  - 테이블 조회 시 WHERE절의 조건에 해당 열이 나오는 경우에만 인덱스 주로 사용
- WHERE절에 사용되더라도 자주 사용해야 가치가 있음
  - SELECT문이 자주 사용 되어야 효과적
  - INSERT문이 자주 사용되고 생성된 인덱스가 클러스터형이면 효율 감소
- 데이터의 중복도가 높은 열은 인덱스 만들어도 효과 없음
  - 인덱스의 관리 비용 때문에 인덱스가 없는 편이 나은 경우도 있음
- 외래 키 지정한 열에는 자동으로 외래 키 인덱스가 생성

## SECTION 06 결론 : 인덱스를 생성해야 하는 경우와 그렇지 않은 경우

### 인덱스에 대한 결론

- JOIN에 자주 사용되는 열에는 인덱스를 생성해 주는 것이 좋음
- INSERT/UPDATE/DELETE가 얼마나 자주 일어나는지 고려해야 함
  - 인덱스는 단지 읽기에서만 성능 향상
  - 데이터의 변경에서는 오히려 부담
- 클러스터형 인덱스는 테이블당 하나만 생성 가능
  - 클러스터형 인덱스를 생성할 열은 범위(BETWEEN, >, < 등의 조건)로 사용하거나 집계 함수를 사용하는 경우 아주 적절하게 사용
- 클러스터형 인덱스가 테이블에 아예 없는 것이 좋은 경우도 있음
- 사용하지 않는 인덱스는 제거
  - 공간 확보 및 데이터의 입력 시에 발생하는 부하 줄임

▶ 이것이 MySQL 이다

# Thank You!

