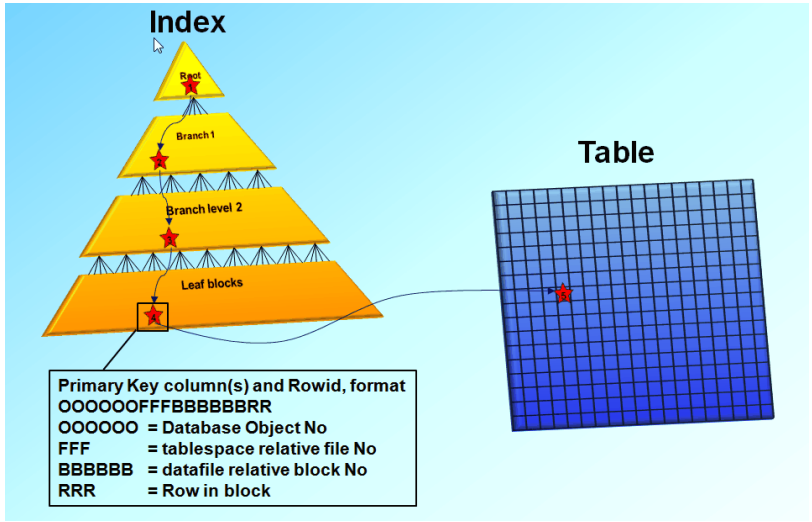
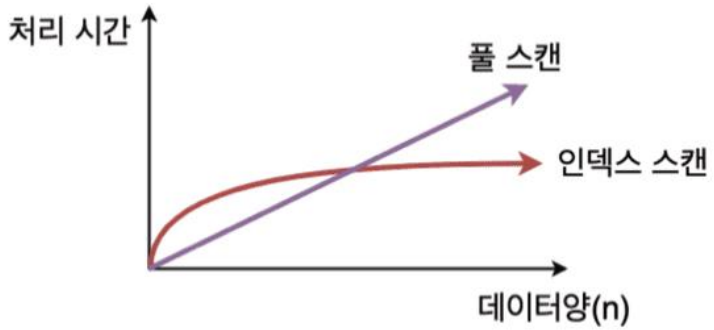
# MySQL의 클러스터링 인덱스

# 목차

# 1. 인덱스란?

# 인덱스



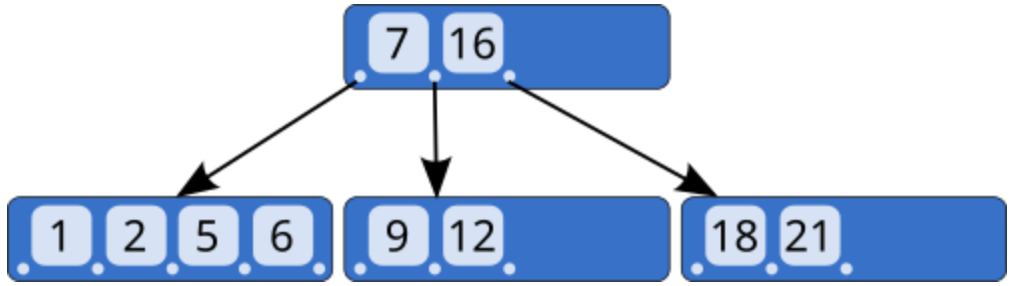



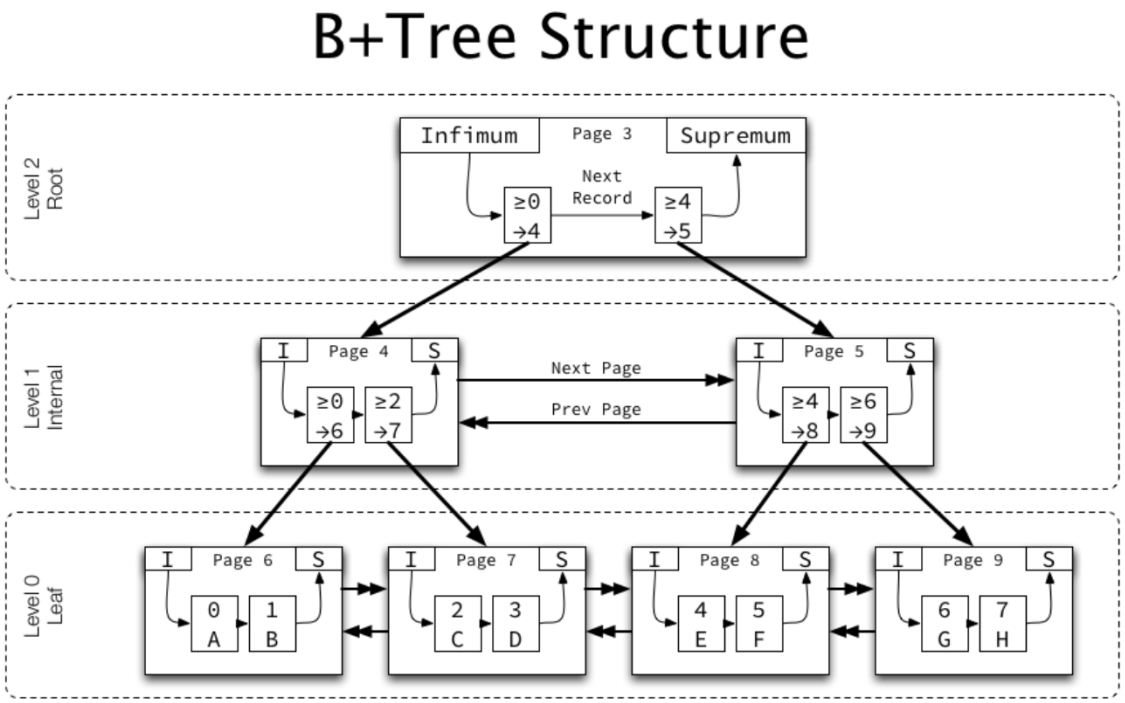

- 데이터를 더 빠르게 검색하고 접근할 수 있도록 도와주는 자료구조

- 읽기 성능 향상
- 쓰기 성능 저하

- CREATE INDEX 인덱스 이름 ON 테이블이름(컬럼1, 컬럼2, ... );

# (참고) MySQL 인덱스 세부 구조




- B-Tree
  - 하나의 노드에 여러 개의 키와 값 소유
  - 항상 완벽한 균형을 유지
  - 정렬된 구조를 유지

- B+Tree: B-Tree(Balanced Tree)에서 개선된 자료구조
  - 같은 깊이에서는 Linked List로 연결
  - 리프 노드에만 값 저장

- MySQL 및 대부분은 B+Tree로 인덱스를 저장
- PostgreSQL은 기본 인덱스를 B-Tree로 저장

# 2. 인덱스의 장단점

# 인덱스의 장점: 빠른 읽기

- 1. 읽기 속도 개선(B-Tree): O(N) -> O(log N)
- 2. 커버링 인덱스: 실제 레코드를 읽지 않고 인덱스만으로 데이터 조회하는 방식

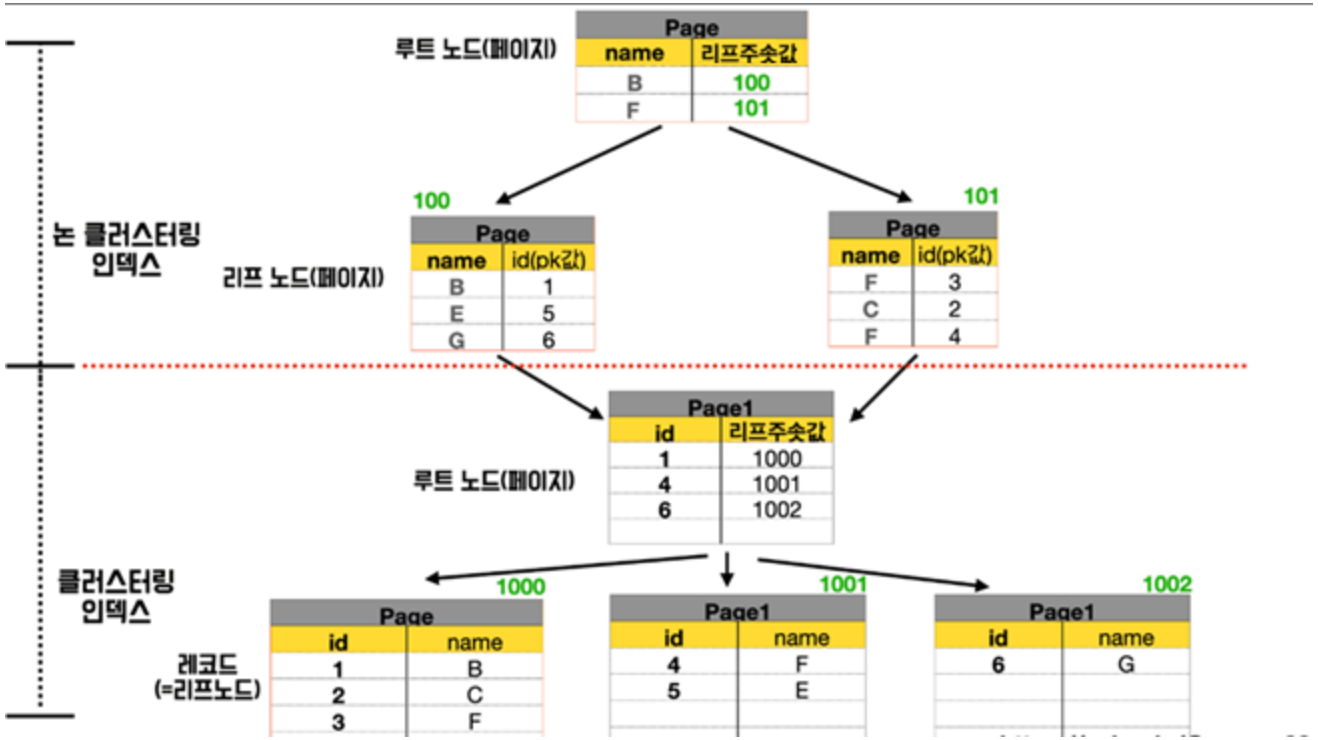
# 인덱스의 단점: 쓰기 성능의 부담

- **INSERT**: 새로운 데이터에 대한 인덱스 추가 -> 이중 작업
- **DELETE**: 인덱스 무효화 -> 공간 낭비
- **UPDATE**: 기존 인덱스 무효화 + 새로운 인덱스 추가 -> 공간 낭비

- **(참고) DBMS별 인덱스 삭제 방식 차이**
  - **MySQL**: DELETE 시 인덱스 항목을 삭제하지만, 즉시 공간을 반환하지 않고 나중에 재활용. DELETE가 잦을 수록 유리
  - **Oracle**: DELETE 시 해당 인덱스 항목에 삭제 플래그 표시. DELETE가 적을수록 빠른 처리에 유리하지만 빈번한 삭제 시 Fragmentation 문제 발생 가능

# 3. 클러스터링 인덱스

# 클러스터링 인덱스(Clustered Index)



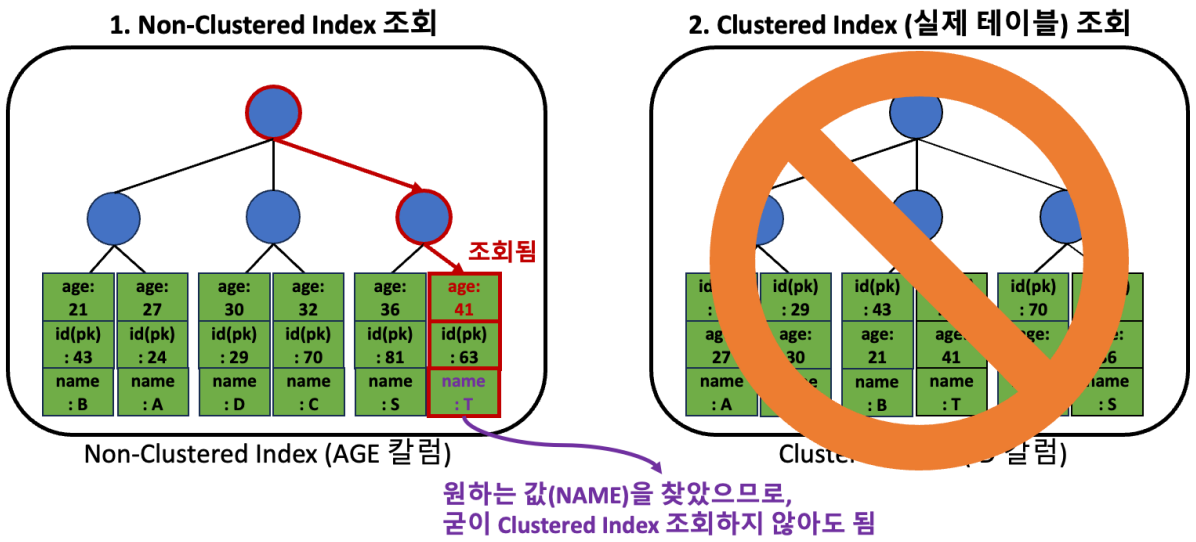

- **클러스터링 인덱스**
  - 리프노드 -> 실제 데이터 레코드
  - 레코드를 디스크에 Sequential하게 저장
  - 쓰기 오버헤드(데이터 크기)
  - 테이블 당 1개

- **논클러스터링 인덱스**
  - 리프노드 -> 인덱스 키 값 + PK 값
  - 읽기 오버헤드(이중 룩업)
  - 테이블 당 N개 가능

- **비클러스터링 DBMS 대비 장단점**
  - 읽기 관점에서 Sequential I/O 가능
  - 단, 쓰기 성능에서는 데이터 재정렬 부담

# 실습: 커버링 인덱스

# 커버링 인덱스



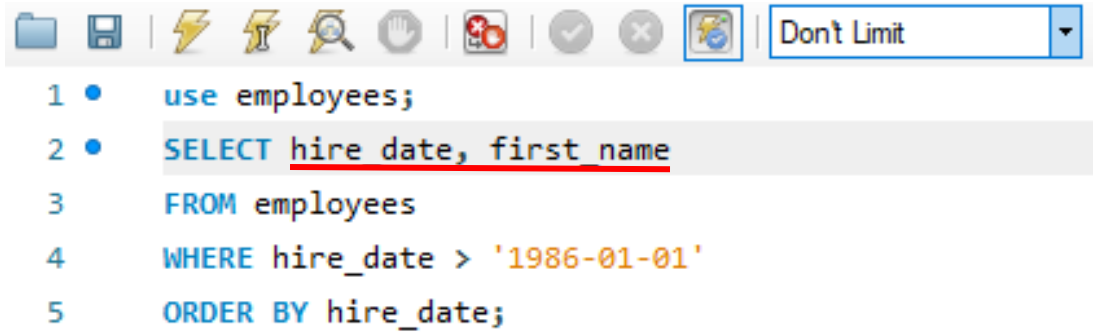

- 풀 테이블 스캔
  - 레코드 전체를 스캔
  - 실행 계획 -> type: all
- 인덱스 사용
  - 인덱스를 거쳐 일부 스캔
  - 실행 계획 -> Extra: Using index condition
- 커버링 인덱스 사용
  - 인덱스만으로 스캔
  - 실행 계획 -> Extra: Using index

# 커버링 인덱스 실습 1 – 기본 세팅

```
PS C:\SSAFY\git> git clone https://github.com/datacharmer/test_db.git
Cloning into 'test_db'...
remote: Enumerating objects: 121, done.
remote: Counting objects: 100% (53/53), done.
remote: Compressing objects: 100% (9/9), done.
remote: Total 121 (delta 44), reused 44 (delta 44), pack-reused 68 (from 1)
Receiving objects: 100% (121/121), 73.43 MiB | 19.95 MiB/s, done.
Resolving deltas: 100% (62/62), done.
```

```
C:\SSAFY\git\test_db>mysql -t -u root -p < employees.sql
Enter password: ****
+-----------------------------+
| INFO                        |
+-----------------------------+
| CREATING DATABASE STRUCTURE |
+-----------------------------+
+----------------------+
| INFO                 |
+----------------------+
| storage engine: InnoDB |
+----------------------+
+--------------------+
| INFO               |
+--------------------+
| LOADING departments |
+--------------------+
+------------------+
| INFO             |
+------------------+
```

# 커버링 인덱스 실습 2 – 인덱스 적용 전




*커버링 인덱스를 위해*
*(hire_date, first_name) 필요*

| type | possible_keys | key | key_len | ref | rows | filtered | Extra |
|---|---|---|---|---|---|---|---|
| ALL | NULL | NULL | NULL | NULL | 299246 | 33.33 | Using where; Using filesort |

264628 row(s) returned 0.156 sec / 0.047 sec

# 커버링 인덱스 실습 3 – 인덱스 적용 후

264628 row(s) returned

0.156 sec / 0.047 sec

| Extra |
| --- |
| Using where; Using filesort |

```
CREATE INDEX idx_hire_date_name
ON employees (hire_date, first_name);
```

264628 row(s) returned

0.015 sec / 0.172 sec

| Extra |
| --- |
| Using where; Using index |

# 추가 실습 – 이중 룩업의 함정

```
CREATE INDEX idx_hire_date_name
ON employees (hire_date);
```

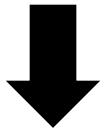
```
1  use employees;
2  explain SELECT hire_date, first_name
3  FROM employees
4  WHERE hire_date > '1986-01-01'
5  ORDER BY hire_date;
6
7  CREATE INDEX idx_hire_date_name
8  ON employees (hire_date);
9
0  DROP INDEX idx_hire_date_name ON employees;
1
```

sult Grid | Filter Rows: | Export: | Wrap Cell Content:

| e | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
|---|---|---|---|---|---|---|---|---|---|
| oyees | NULL | ALL | idx_hire_date_name | NULL | NULL | NULL | 298847 | 50.00 | Using where; Using filesort |

264628 row(s) returned

0.156 sec / 0.047 sec



264628 row(s) returned

0.157 sec / 0.031 sec

# 실습 코드

git clone https://github.com/datacharmer/test_db.git
mysql -t -u root -p <employees.sql

```
USE employees;
EXPLAIN SELECT hire_date, first_name
FROM employees
WHERE hire_date > '1986-01-01'
ORDER BY hire_date;

CREATE INDEX idx_hire_date_name
ON employees (hire_date, first_name);

DROP INDEX idx_hire_date_name ON employees;
```

# 감사합니다.