

Database

Subject 1. Disk

Subject 2. Index

Subject 3. B-Tree index scan

Subject 4. Clustering index

Subject 5. Clustering

Subject 6. Replication

Subject 7. Partitioning, Sharding

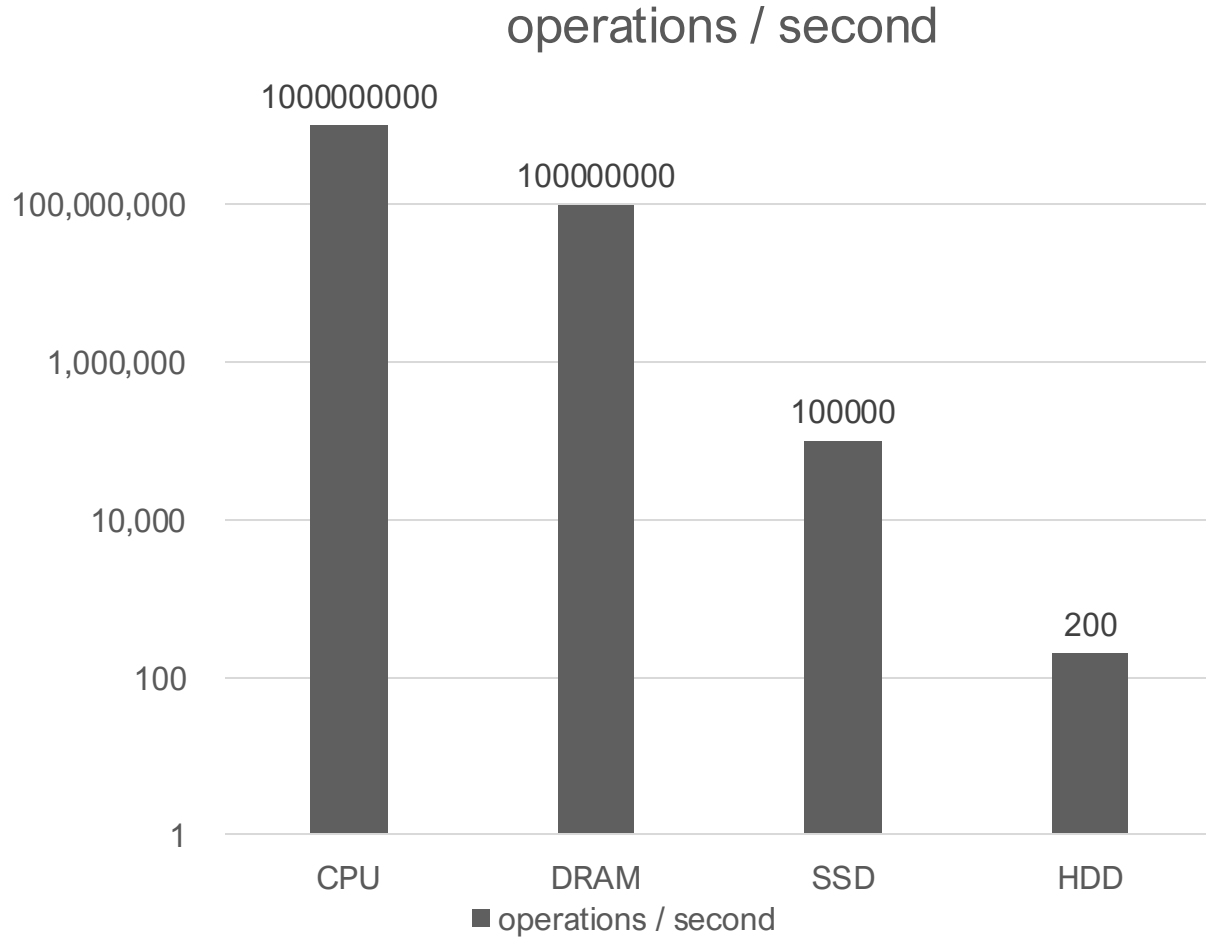
Subject 8. Deadlock

홍승택

subject 1.

Disk

HDD, SSD



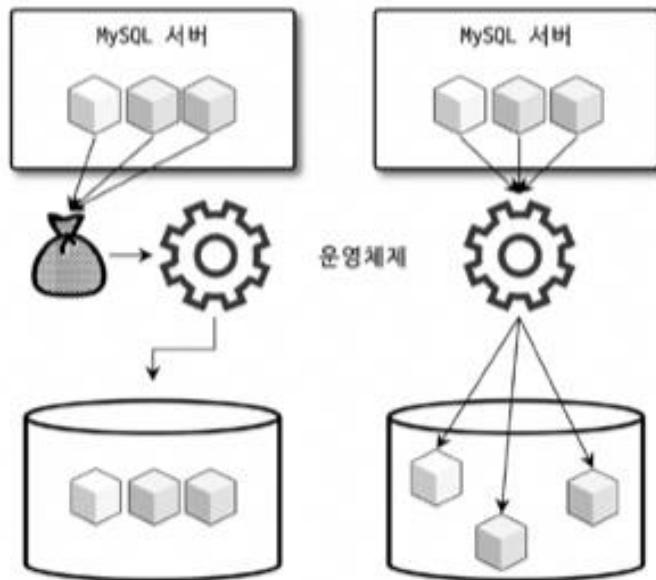
Random I/O



대부분의 비용은 디스크에서 읽는 시간이다.

Random I/O, Sequential I/O

쿼리 튜닝 = 랜덤 I/O 횟수를 줄이자



Samsung 990 PRO M.2 NVMe

Performance Sequential Read	Up to 7,450 MB/s * Performance may vary based on system hardware & configuration
Sequential Write	Up to 6,900 MB/s * Performance may vary based on system hardware & configuration
Random Read (4KB, QD32)	Up to 1,400,000 IOPS * Performance may vary based on system hardware & configuration
Random Write (4KB, QD32)	Up to 1,550,000 IOPS * Performance may vary based on system hardware & configuration

subject 2.

Index

Index metaphor

A - C		sorted
Access type		434
ACCOUNT LOCK		59
ACCOUNT UNLOCK		59
ACID		133
activate_all_roles_on_login		72
Active Redo Log		112
Adaptive flush		116
Adaptive Hash Index		137
ALL		446
ALTER INSTANCE	135, 198, 209	
ALTER USER		64
Anti Semi-join		336
Ascending index		244
Audit		8
AUTO_INCREMENT		169
AUTO-INCREMENT		276

key-value

Index = Sorted List

Data file = Array List

Sorted List

Insert, Delete, Update : **Slow**

Select : **Fast**

OLTP(On-Line Transaction Processing)

쓰기 : 읽기 = 2 : 8 (또는 1 : 9)

인덱스를 더 추가할까?

= 저장 속도를 어디까지 희생?

읽기 속도를 얼마나 빠르게?

Index classification

역할

- Primary key
- Secondary key

저장 방식(알고리즘)

- B-Tree
- Hash
- Fractal-Tree, Merge-Tree, ...

중복 허용 여부

- Unique
- Non-Unique

기능

- 전문 검색용
- 공간 검색용
- ...

B-Tree (Balanced Tree)

특수한 요건이 아닌 경우, 대부분 인덱스는 거의 B-Tree를 사용할 정도로 일반적인 용도에 적합

루트 노드

페이지 (1)	
인덱스 키	자식노드 주소
Aamer	2
Jaana	3
...	...

브랜치 노드

페이지 (2)	
인덱스 키	자식노드 주소
Aamer	4
Ebbe	5
Gad	6
...	...

페이지 (3)	
인덱스 키	자식노드 주소
Jaana	7
Lakshmi	8
Oddvar	9
...	...

리프 노드

페이지 (4)	
인덱스 키	프라이머리 키
Aamer	11800
Babette	10128
Candida	10418

페이지 (5)	
인덱스 키	프라이머리 키
Ebbe	10057
Fabrizio	11854
...	...

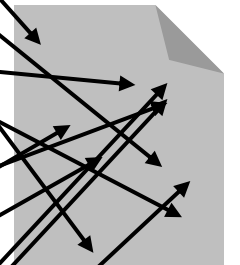
페이지 (6)	
인덱스 키	프라이머리 키
Gad	10799
Hailing	11085
Iara	11043

페이지 (7)	
인덱스 키	프라이머리 키
Jaana	11384
Kagan	12338
...	...

인덱스키 = 정렬 O
데이터 파일 = 정렬 X

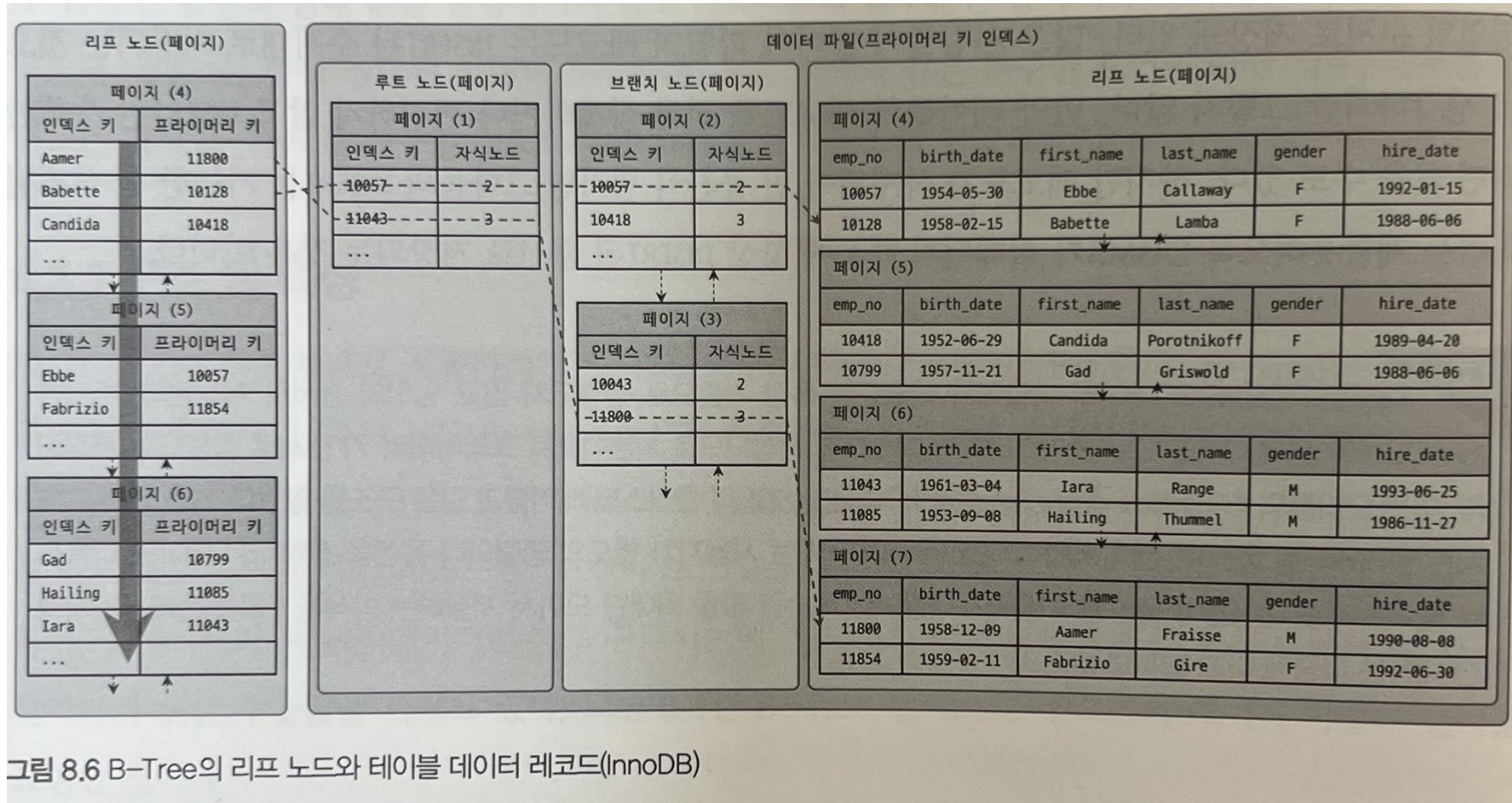
InnoDB..
PK 순서로 정렬(default)
(클러스터링)

데이터 파일



B-Tree Secondary index

2번의 인덱스 서칭 필요 (InnoDB)



B-Tree Secondary index

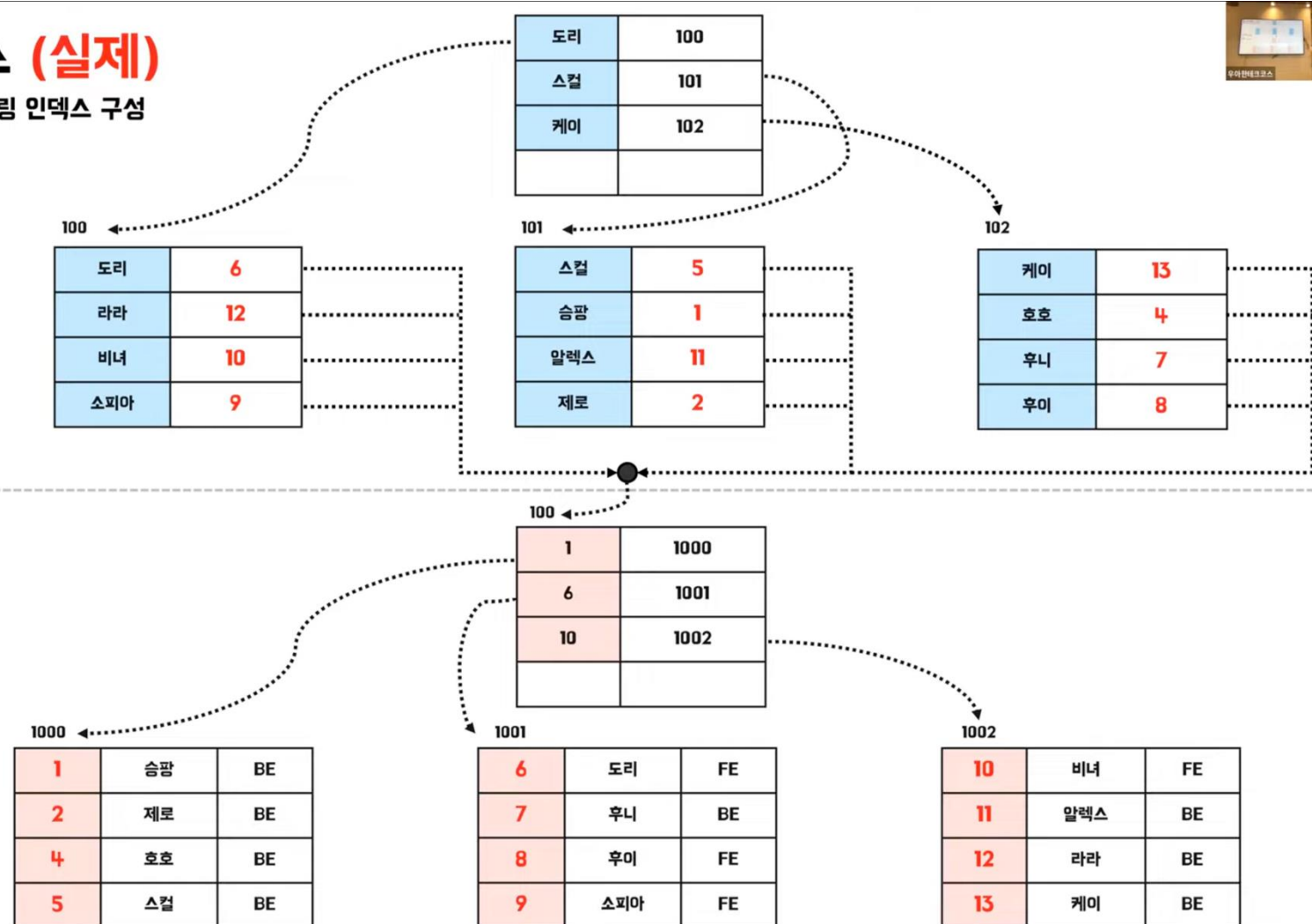
2번의 인덱스 서칭 필요 (InnoDB)

다수의 인덱스 (실제)

클러스터링 + 논-클러스터링 인덱스 구성

name 컬럼의
논-클러스터링 인덱스 페이지

id 컬럼의
클러스터링 인덱스 페이지



2022-08-18 14:22:26

B-Tree multi-column index

- 인덱스는 항상 정렬 되어 있음
- 정렬 기준 = 인덱스를 구성하는 컬럼 **순서대로**
- ex) index (dept_no, emp_no) 이라면
 1. dept_no 기준 정렬
 2. dept_no가 같다면 emp_no 기준 정렬
- 결론 : 인덱스 내에서 각 컬럼의 순서가 상당히 중요하다.

B-Tree index ordering (MySQL 8.0)

- 인덱스 생성 시점에 정렬 기준 설정 가능
 - `CREATE INDEX ix_firstname ON employees (first_name DESC);`
- 인덱스 스캔 시 **정순**과 **역순**으로 읽을 수 있다.
 - `SELECT * FROM employees ORDER BY first_name ASC LIMIT 10;`
 - `SELECT * FROM employees ORDER BY first_name DESC LIMIT 10;`
- 하지만 실제로는 정순 스캔이 더 빠르다(InnoDB)
 - 페이지 잠금이 정순 스캔에 적합한 구조다
 - 페이지 내의 인덱스 레코드는 단방향으로만 연결됨

MySQL 5.7까지는 항상 오름차순 정렬 인덱스만 생성 (문법상으로 제공)

B-Tree 인덱스 사용에 영향을 미치는 요소

인덱스 키 값의 크기

- Page 크기는 innodb_page_size 시스템 변수를 이용해 4KB ~ 64KB(default=16KB) 선택 가능 (MySQL 5.7~)
- 인덱스 키 값의 크기가 커진다 = Page에 저장할 수 있는 레코드 수가 줄어든다 = 디스크를 여러번 읽어야 된다 = 느
- InnoDB의 버퍼 풀의 크기도 제한적이기 때문에 캐시 레코드 수도 감소

B-Tree 깊이

- 직접 제어할 방법이 없음
- 인덱스 키 값의 크기를 가능하면 작게 만드는 것이 좋음
- 실제로는 아무리 대용량 DB라도 깊이가 5단계 이상까지 깊어지는 경우는 흔치 않다.

B-Tree 인덱스 사용에 영향을 미치는 요소(cont'd)

기수성(Cardinality)

- 모든 인덱스 키 값 가운데 유니크한 값의 수
- 인덱스는 기수성이 높을수록 검색 대상이 줄어든다 = 빠르다.

읽어야 하는 레코드의 건수

- 인덱스를 읽는 것 자체가 비용 발생
- 일반적인 DBMS의 옵티마이저에서는 인덱스를 통한 레코드 1건 읽는 것이 테이블에서 직접 읽는 것보다 4~5배 비용이 더 많이 드는 것으로 예측
- 인덱스를 통해 읽어야 할 레코드의 건수(옵티마이저 판단 예상)가 전체 테이블 레코드의 20~25%를 넘어서면 테이블을 모두 직접 읽고 필터링 방식으로 처리

Unique index

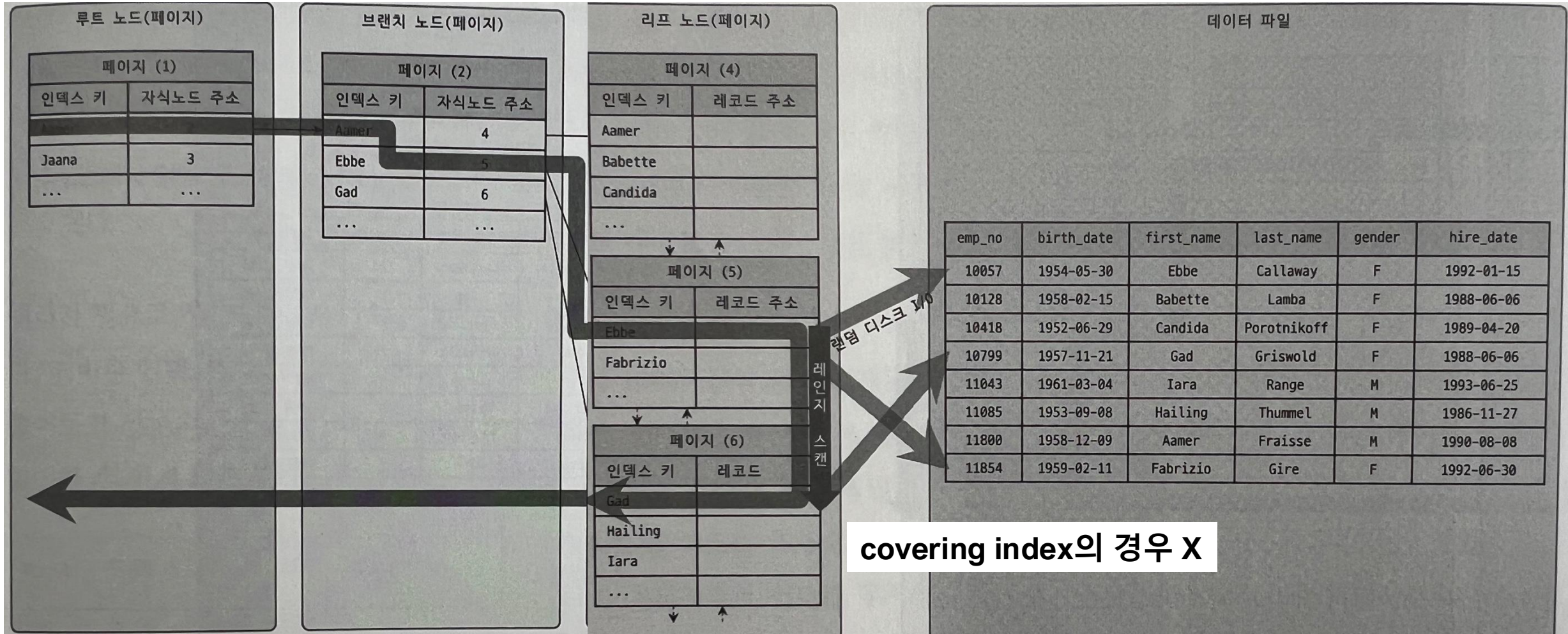
- MySQL에서는 인덱스 없이 유니크 제약만 설정할 방법이 없다.
- 유니크 인덱스는 NULL도 가능
- MySQL에서 PK는 기본적으로 NOT NULL + Unique 속성이 자동으로 부여
- 유니크 인덱스와 일반 세컨더리 인덱스는 사실상 거의 차이가 없다.
 - Read : 레코드는 1건을 더 읽느냐 마느냐 차이. CPU 작업이므로 거의 차이 없음
 - Write : 유니크 인덱스는 중복된 값이 있는지 체크하는 과정이 있어서 더 느림(특히 Lock 관련)
- 결론 : 유일성이 꼭 보장되어야 하는 컬럼에 대해서는 유니크 인덱스를 생성하되, 꼭 필요하지 않다면 세컨더리 인덱스를 생성하는 방법도 고려해보자

subject 3.

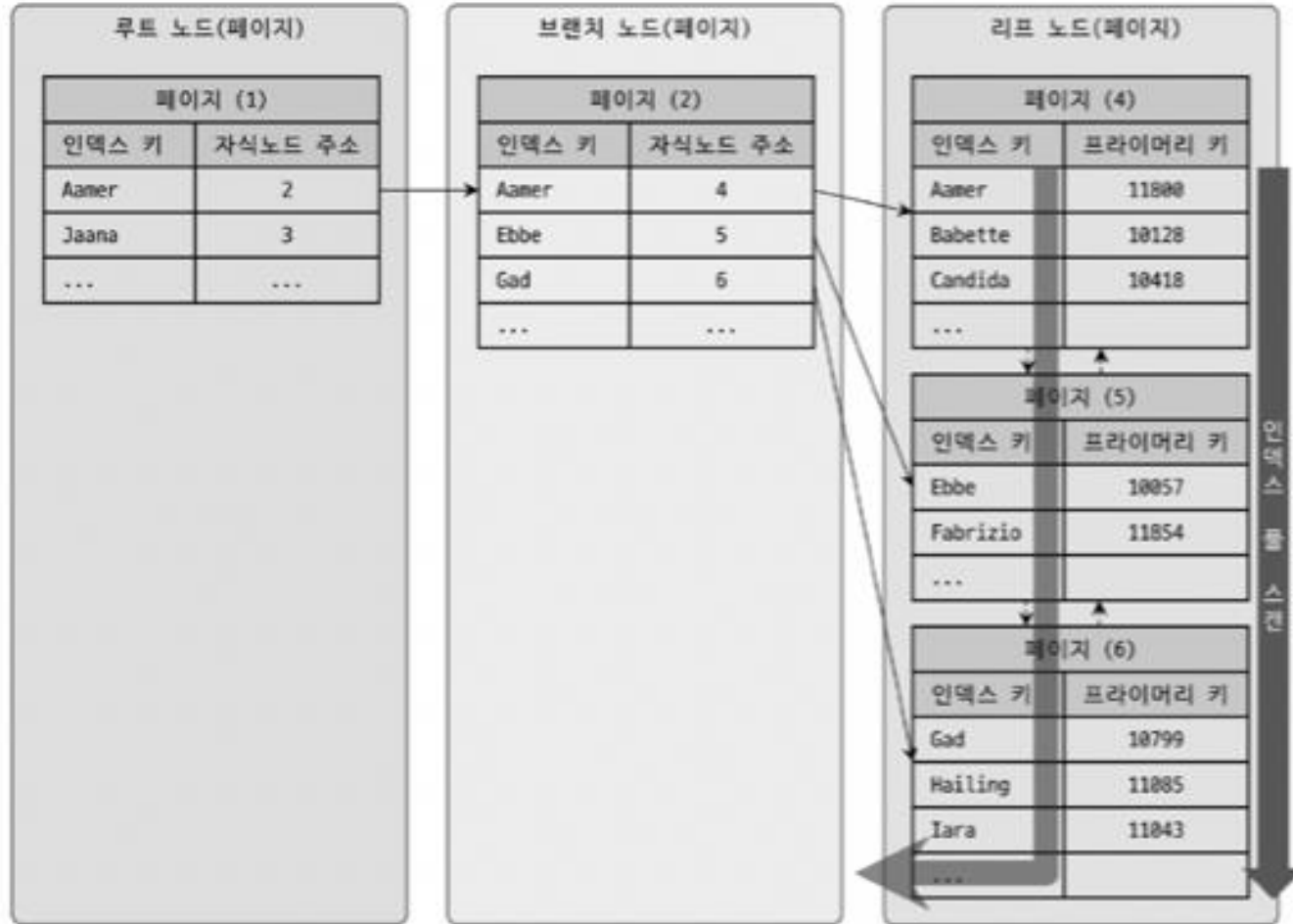
B-Tree index scan

B-Tree index scan – index range scan

검색해야 할 인덱스의 범위가 결정됐을 때



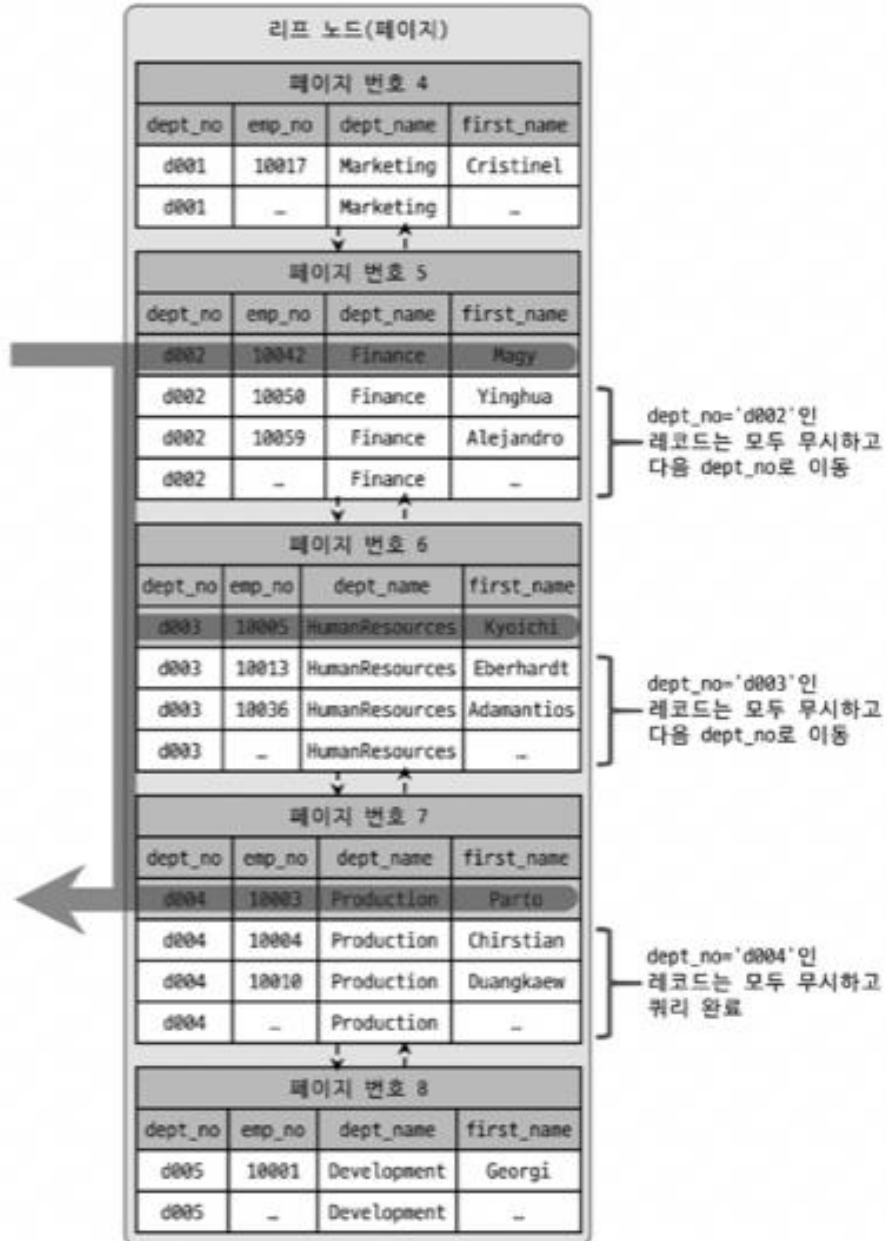
B-Tree index scan – index full scan



- 인덱스의 처음부터 끝까지 모두 읽기
- 쿼리가 인덱스에 명시된 칼럼만으로 조건을 처리할 수 있는 경우에 사용
- ex) 인덱스 (A, B, C) 일 때, 쿼리의 조건절이 B 또는 C로 검색하는 경우

인덱스를 사용한다 = range scan, loose scan
 인덱스를 사용안한다 = table full scan, index full scan

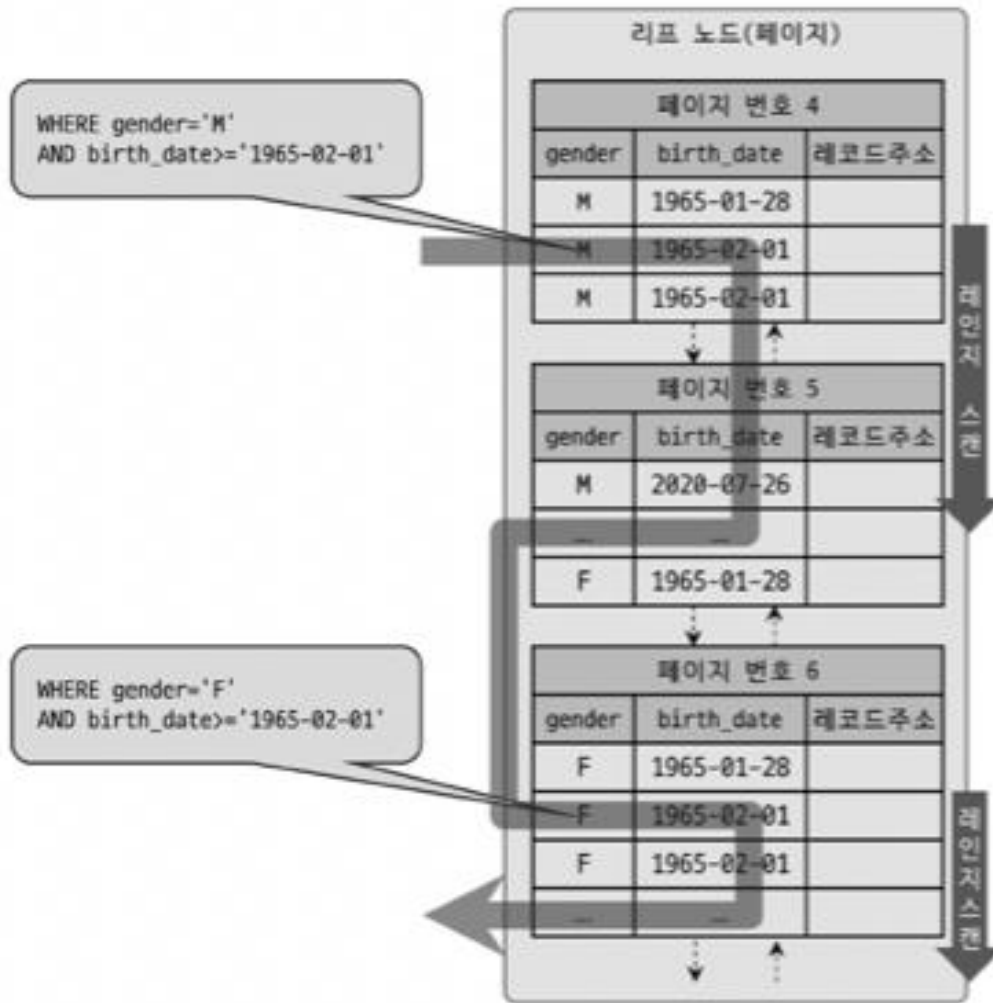
B-Tree index scan – index loose scan



- 등성등성 인덱스 읽기
- GROUP BY, MAX(), MIN() 가 사용될 때
- 여러 가지 조건을 만족해야 됨

```
SELECT dept_no, MIN(emp_no)
FROM dept_emp
WHERE dept_no BETWEEN 'd002' AND 'd004'
GROUP BY dept_no;
```

B-Tree index scan – index skip scan



- MySQL 8.0 버전부터 도입
- 가정) index (gender, birth_date)

```
SELECT gender, birth_date
FROM dept_employees
WHERE birth_date >= '1965-02-01';
```

Index full scan
Table full scan(not covering)

최적화

```
SELECT gender, birth_date
FROM dept_employees
WHERE gender='M' AND birth_date >= '1965-02-01';
```

```
SELECT gender, birth_date
FROM dept_employees
WHERE gender='F' AND birth_date >= '1965-02-01';
```

단점

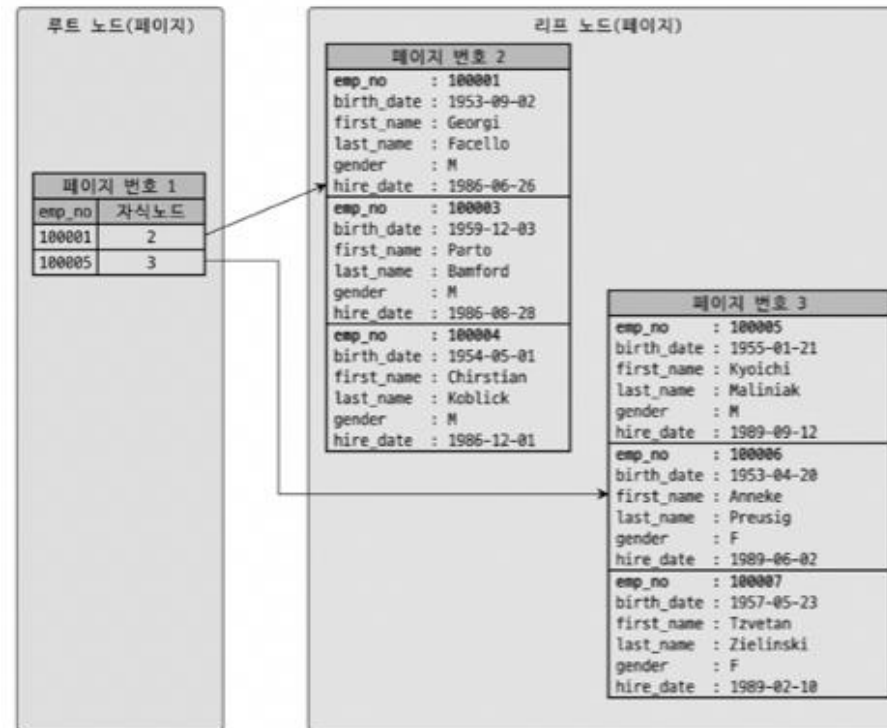
- 선행 칼럼의 유니크한 값의 개수가 적어야 함
- 커버링 인덱스인 경우만 가능

subject 4.

Clustering index

Clustering index

- PK 값이 비슷한 레코드끼리 묶어서 저장하는 것
- PK = 클러스터링 인덱스 = 클러스터링 키
- PK 값에 의해 레코드 저장 위치가 결정되며, 그 값이 변경된다면 레코드의 물리적인 저장 위치가 변경되어야 함
- 따라서 PK 값 자체에 대한 의존도가 상당히 크기 때문에 신중히 결정해야 한다.
- MySQL에서 클러스터링 인덱스는 InnoDB 스토리지 엔진에서만 지원



Clustering index

장점

- PK로 검색할 때 성능 좋음 (특히 PK 기준 범위 연산이 매우 빠름)
- 모든 세컨더리 인덱스가 PK를 가지고 있기 때문에 커버링 인덱스를 이용할 가능성 높음

단점

- 모든 세컨더리 인덱스가 PK를 가지고 있기 때문에 그 크기에 영향을 받음
- 세컨더리 인덱스를 통해 검색할 때 PK로 다시 검색해야 하므로 처리 성능 느림
- INSERT 할 때 PK에 의해 레코드 위치가 결정되므로 처리 성능 느림
- PK 변경할 때 레코드 위치를 변경해야하므로 느림

결론

빠른 읽기(SELECT)

느린 쓰기(INSERT, UPDATE, DELETE)

PK 선정

- When you define a **PRIMARY KEY** on a table, InnoDB **uses it as the clustered index**
- If you do not define a PRIMARY KEY for a table, InnoDB uses the **first UNIQUE index** with all key columns defined as **NOT NULL** as the clustered index.
- If a table has no PRIMARY KEY or suitable UNIQUE index, InnoDB generates a **hidden clustered index** named GEN_CLUST_INDEX on a synthetic column that contains row ID values. the rows ordered by the row ID are physically in order of insertion.
 - 사용자에게 노출되지 않으며, 쿼리 문장에 명시적으로 사용할 수 없음

Clustering table 사용 시 주의사항

클러스터링 인덱스 키의 크기

- 모든 세컨더리 인덱스는 PK 값을 포함한다. PK 크기가 커지면 세컨더리 인덱스도 자동으로 크기가 커짐
- 일반적으로 테이블에 세컨더리 인덱스가 4~5개 생성되므로, PK는 신중하게 선택하기

PK는 반드시 명시할 것

- 최소한 AUTO_INCREMENT 칼럼을 이용해서라도 생성하는 것을 권장

Multi-column PK 대신 AUTO_INCREMENT 칼럼 사용

- 세컨더리 인덱스가 필요하지 않다면 multi-column PK를 사용하는 것이 좋다
- 하지만 세컨더리 인덱스가 필요하다면 인조 식별자(Surrogate key)를 추가하는게 좋음
- 특히 INSERT 위주의 테이블에서는 더욱 성능 향상에 도움이 된다.

인덱스 조회 시 주의사항

- between, like, <, > 등 범위 조건은 해당 컬럼은 인덱스를 타지만, 그 뒤 인덱스 컬럼들은 인덱스가 사용되지 않습니다.
- or 연산자는 비교해야 할 ROW가 더 늘어나기 때문에 풀 테이블 스캔이 발생할 확률이 높습니다.
- 인덱스로 사용된 컬럼값 그대로 사용해야만 인덱스가 사용됩니다.
 - where salary * 10 > 150000;는 인덱스를 못타지만, where salary > 150000 / 10; 은 인덱스를 사용합니다.
 - 컬럼이 문자열인데 숫자로 조회하면 타입이 달라 인덱스가 사용되지 않습니다. 정확한 타입을 사용해야만 합니다.
- 최근엔 이전과 같이 꼭 인덱스 순서와 조회 순서를 지킬 필요는 없습니다

subject 5.

Clustering

Clustering

- 여러 개의 DB를 수평적인 구조로 구축하는 방식
- 클러스터링은 분산 환경을 구성하여 Single point of failure와 같은 문제를 해결할 수 있는 Fail Over 시스템을 구축하기 위해서 사용
- 클러스터링은 동기 방식으로 노드들 간의 데이터를 동기화

장점

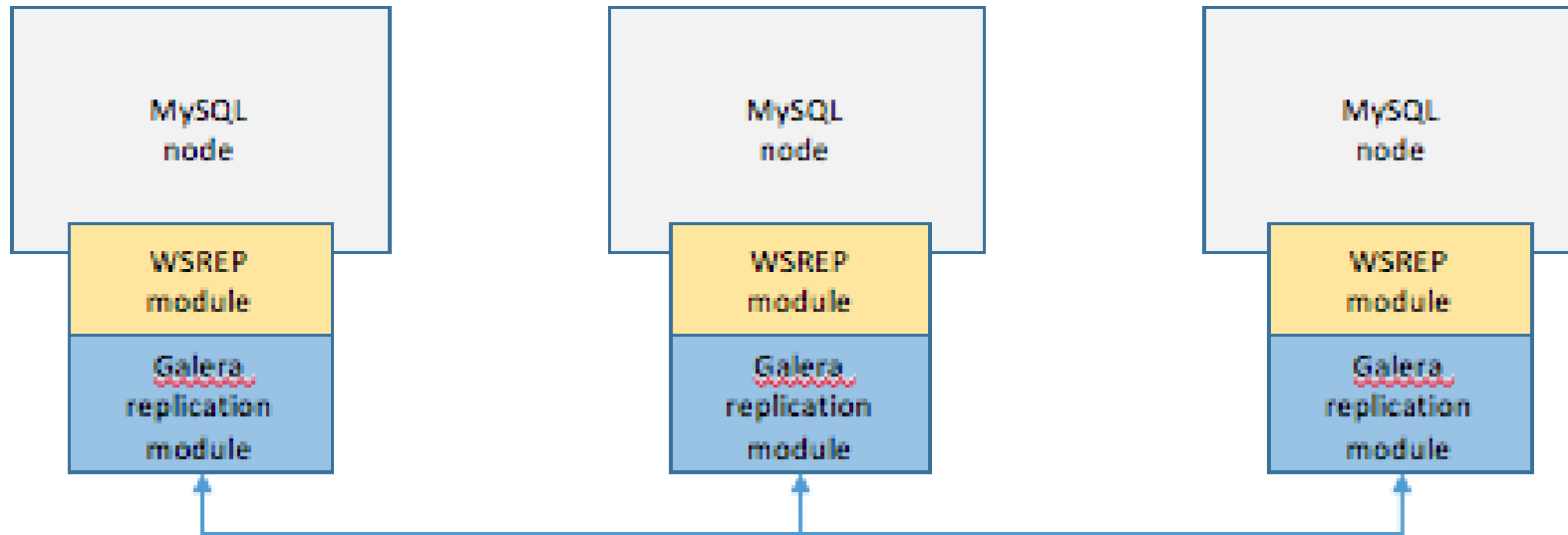
- 장점 노드들 간의 데이터를 동기화하여 항상 일관성있는 데이터를 얻을 수 있다.
- 1개의 노드가 죽어도 다른 노드가 살아 있어 시스템을 계속 장애없이 운영할 수 있다.

단점

- 여러 노드들 간의 데이터를 동기화하는 시간이 필요하므로 Replication에 비해 쓰기 성능이 떨어진다.
- 장애가 전파된 경우 처리가 까다로우며, 데이터 동기화에 의해 스케일링에 한계가 있다.

Clustering procedure

1. 1개의 노드에 쓰기 트랜잭션이 수행되고, COMMIT을 실행한다.
2. 실제 디스크에 내용을 쓰기 전에 다른 노드로 데이터의 복제를 요청한다.
3. 다른 노드에서 복제 요청을 수락했다는 신호(OK)를 보내고, 디스크에 쓰기를 시작한다.
4. 다른 노드로부터 신호(OK)를 받으면 실제 디스크에 데이터를 저장한다.



subject 6.

Replication

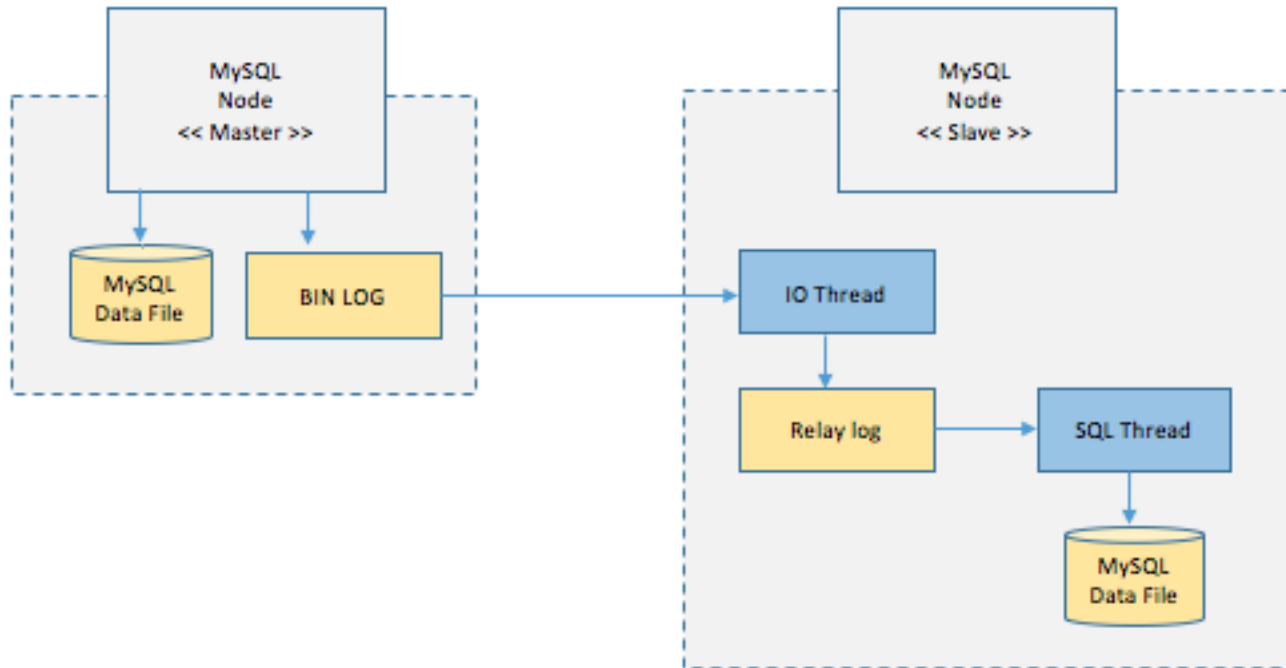
Replication

- 여러 개의 DB를 권한에 따라 수직적인 구조(Master-Slave)로 구축
- Master Node는 쓰기 작업 만을 처리하며, Slave Node는 읽기 작업 만을 처리
- 비동기 방식으로 노드들 간의 데이터를 동기화

- **장점**
 - DB 요청의 60~80% 정도가 읽기 작업이기 때문에 Replication만으로도 충분히 성능을 높일 수 있다.
 - 비동기 방식으로 운영되어 지연 시간이 거의 없다.
- **단점**
 - 노드들 간의 데이터 동기화가 보장되지 않아 일관성있는 데이터를 얻지 못할 수 있다.
 - Master 노드가 다운되면 복구 및 대처가 까다롭다.

Replication procedure

1. Master 노드에 쓰기 트랜잭션이 수행된다.
2. Master 노드는 데이터를 저장하고 트랜잭션에 대한 로그를 파일에 기록한다.(BIN LOG)
3. Slave 노드의 IO Thread는 Master 노드의 로그 파일(BIN LOG)를 파일(Replay Log)에 복사한다.
4. Slave 노드의 SQL Thread는 파일(Replay Log)를 한 줄씩 읽으며 데이터를 저장한다.



subject 7.

Partitioning, Sharding

Partitioning


table을 목적에 따라 작은 table들로 나누는 방식

vertical partitioning (column 기준)

- 게시글 table에는 content가 존재(데이터 큼)
- 게시판 목록만 보여줄 때는 content가 필요 없지만 db 상에서 메모리에 올릴 때는 content도 같이 올려야 한다
- 이런 경우 큰 데이터(content)를 별도의 테이블로 나눌 수 있다.
- 또는 보안상이나 자주 사용하는 테이블을 분리하는 등의 목적이 있을 수 있다
- 전체 데이터가 필요하다면 join을 이용

horizontal partitioning (row 기준)

- 테이블의 스키마는 유지하면서, 데이터를 분할 (**hash based, range based**)
- hash based : 꽤 균일하게 분산되나, 파티셔닝 개수를 바꾸기 어려움
- range based : 쉽게 증설할 수 있으나, 일부 DB에 데이터가 몰릴 수 있음
- directory based : 별도의 조회 테이블 사용(오버헤드 존재)

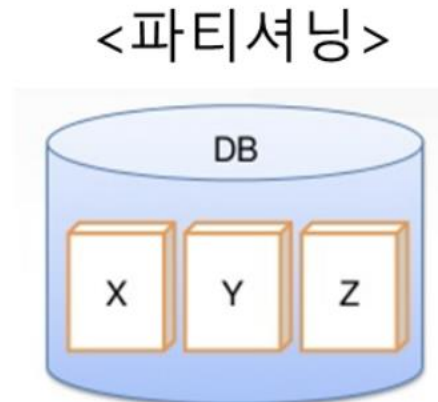
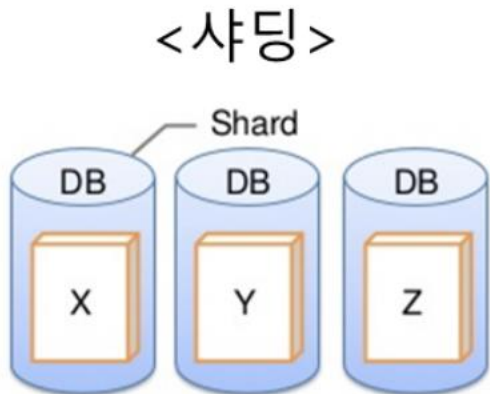


SUBSCRIPTION_0			
user_id	channel_id	alarm	membership
aaaaaaa	1
dingyo	395
aaaaaaa	28
aaaaaaa	42
dingyo	42

SUBSCRIPTION_1			
user_id	channel_id	alarm	membership
yeah	401
maryang	717
lolololol	1
ssony	22
maryang	101

Sharding

- 파티셔닝이랑 동일
- 다른점: 다른 DB 서버에 분할(물리적으로 분리) → 부하 분산
- 용어: partition key → shard key, 파티션 → shard
- 샤딩? 레플리케이션?



made by Seungtaek

The diagram shows two database servers, each with a table. The top server has a table labeled 'SUBSCRIPTION_0' and the bottom server has a table labeled 'SUBSCRIPTION_1'. Both tables have columns: user_id, channel_id, alarm, and membership.

user_id	channel_id	alarm	membership
aaaaaaa	1
dingyo	395
aaaaaaa	28
aaaaaaa	42
dingyo	42

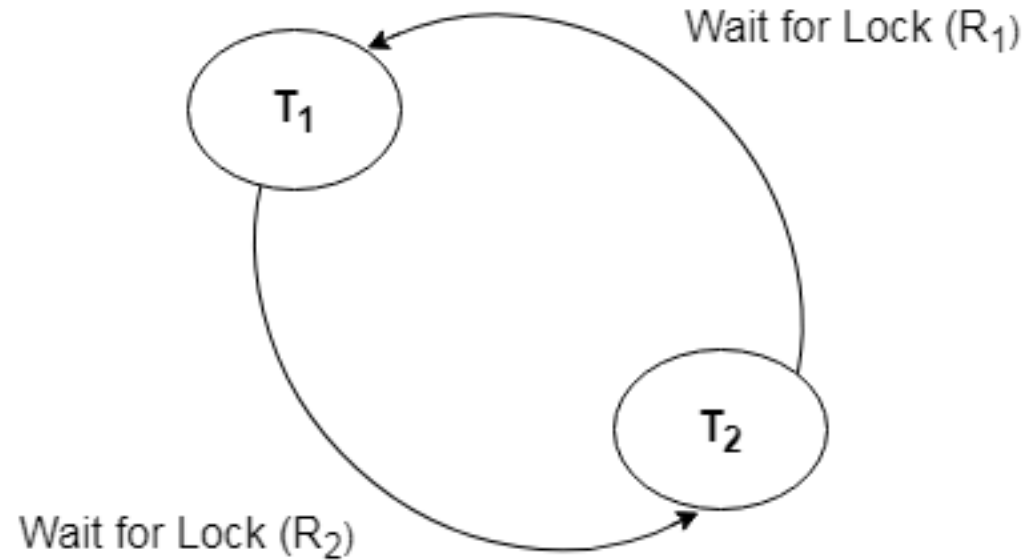
user_id	channel_id	alarm	membership
yeah	401
maryang	717
lolololol	1
ssony	22
maryang	101

subject 8.

Deadlock

Wait-for graph

- transaction을 노드로 갖는 그래프 방향 그래프
- $T_i \rightarrow T_j : T_i$ 은 필요한 리소스를 T_j 가 놓을 때까지 wait
- wait-for graph가 사이클을 갖는다 = 시스템이 deadlock 상태



Deadlock detection in InnoDB

- Deadlock 상태에 빠지지 않았는지 체크하기 위해 잠금 대기 목록을 Wait-for graph 형태로 관리
- 데드락 감지 스레드가 주기적으로 그래프 검사
- Deadlock에 빠진 트랜잭션들을 찾아서 그 중 하나를 강제 종료
 - 언두 로그 양이 적은 트랜잭션을 롤백
- innodb_deadlock_detect : 데드락 감지 스레드를 작동할거냐?
- innodb_lock_wait_timeout : 락 대기가 일정 시간이 지나면 자동으로 롤백

Deadlock prevention – large DB

- **Lock timeout**
- **Transaction 이전에 모두 Lock하기**
 - 어떤 item이 Lock될지 예측 어려움
 - 낮은 성능
- **Impose partial ordering**
 - Lock 순서를 정해 둠
- **Preemptive (wait-die, wound-wait)**
 - wait-die (non-preemptive)
 - 내가 오래된 트랜잭션이면 기다림. 아니면 내 자신을 rollback
 - wound-wait (preemptive)
 - 내가 새로운 트랜잭션이면 기다림. 아니면 상대방을 rollback

Reference

Real MySQL 8.0 (백은빈, 이성욱; 위키북스)

<https://jojoldu.tistory.com/243>

<https://mangkyu.tistory.com/97>

<https://www.youtube.com/watch?v=edpYzFgHbqs>

<https://dev.mysql.com/doc/refman/8.4/en/>

<https://jojoldu.tistory.com/476>

<https://jojoldu.tistory.com/481>