03장. 저장소와 검색DB란

들어가기에 앞서

- 관계형 DB와 NoSQL DB에 사용되는 저장소 엔진에 대해 공부한다.
- 로그 구조(log-structured) 계열 저장소 엔진과 (B-tree 같은) 페이지 지향(page-oriented) 계열 저장소 엔진을 검토한다.

DB란

DB작업

- data 저장 ,data 요청 시 제공

개발자가 DB 내 저장 및 검색 처리 방법을 주의해야하는 이유

- 처음부터 저장소 엔진을 구현하는 것이 아닌, 사용 가능한 여러 저장소 엔진 중 가장 적합한 엔진을 선택해야 하기 때문 (**작업 부하**에 맞춰 최적화 된 저장소 엔진/ **분석** 위해 최적화된 엔진 차이)

DB를 강력하게 만드는 데이터 구조

DB는 대부분 내부적으로 append-only 데이터 파일인 로그(`연속된 추가 전용 레코드`)를 사용한다.

DB에서는 일반적으로 특정 키의 값을 효율적으로 찾기 위해 Index를 사용한다.

- Index(색인)
    - 저장소 시스템에서 중요한 `trade off 관계`
    - 인덱스가 많이 있다면 여러 **WHERE 절 조건에 대응**할 수 있는 또는 **조회**에서 조금 더 효과 적인 인덱스를 사용할 수는 있겠지만, trade-off 관계로 Index 가 많아질수

록 데이터의 입력이나 갱신에서는 적은 인덱스를 사용하는 것에 비해 상대적으로 성능이 불리해 질 수 있다.

- 인덱스를 잘 선택하면 **읽기 속도가 향상** 되지만 모든 인덱스는 **쓰기(write) 속도를 떨어뜨린다**. (데이터를 쓸 때마다 인덱스도 갱신해야 하기 때문.)
- 따라서! DB는 일반적으로 모든 것을 인덱싱하지는 않고, 개발자가 수동으로 인덱스 설정을 하도록 해서 적은 오버헤드로 고효율을 주도록 한다.

색인 전략

	해시 색인	SS 테이블 & LSM 트리	B 트리
설명	- 가장 간단한 색인 전략 - 색인으로 key-value 저장소를 쓴다. - 키를 데이터 파일의 바이트 오프셋에 매핑해 인메모리 해시맵에 저장하는 것 (값을 조회하려면 해시 맵을 사용 해 `데이터 파일에서 오프셋을 찾아 위치를 구하고 값을 얻어`야한다.) - 시간 복잡도 O(n)	- 해시 색인과 다르게 제한이 없는 색인 방식 - 해시 색인에서 모든 `key-value 쌍을 키를 기준으로 정렬한 Log-structured 데이터 형식` : **Sorted String Table, SS 테이블** 칭함. -**LSM 트리**: SS 테이블의 형식으로 디스크에 key-value 데이터를 저장하는 색인 방식 (수 메가바이트 이상의 가변 크기를 가진 세그먼트 단위로 데이터를 기록하고 이는 항상 순차적)	- 가장 널리 사용되는 색인 구조 - 정련된 키-값 쌍이라 범위 질의에 효율적이라는 점 빼고는, LSM트리와 다 다르다. - **B 트리**는 보통 4KB 고정 크기의 `페이지 단위(page-oriented)로 읽기 또는 쓰기`를 수행한다. 이는 근복적으로 하드웨어와 조금 더 밀접한 관련이 있다. *분기 계수 500, 페이지 크기 4KB인 4단계 B 트리는 256TB 까지 저장할 수 있다.*
장단점	(단점) - 해시 맵을 전부 메모리에 유지하는 것을 조건으로 사용하기 때문에, **키가 너무 많으면 성능이 떨어진다.** - 디스크에 해시맵 유지는 가능하나, **무작위 접근 I/O**가 많이 필요해 디스크 상의 해시 **맵에 좋은 성능을 기대하기는 어렵**다. - 해시 맵은 `범위 질의(range query)에 굉장히 비효율적` → 모든 키를 개별적으로 조회해야하기 때문 - **로그 구조(log-structured)** 형식으로 데이터를 저장하면 파일에 항상 추가만 하기 때문	(장점) - 세그먼트 병합이 보다 효율적이다. 세그먼트가 정렬되어 있기 때문에 merge sort 가 가능하다. -파일에서 특정 키를 찾기 위해 더는 메모리에 모든 키의 색인을 유지할 필요가 없다. 아래 그럼처럼 일부 키 색인만 있으면 충분하다. (대략 수 킬로바이트당 키 하나)	

	에 디스크 공간이 부족해진다.		
적합한 작업	- 키 자체가 많지는 않지만 키당 write 수가 많은 작업 부하에 적합하다. -**로그 구조(log-structured)** 형식에서 특정 크기의 세그먼트로 나누고 **주기적으로 컴팩션**(compaction:중복된 키 버리고 각 키의 최신값만 유지)을 수행해서 디스크 공간 문제를 해결한다. → 해시 맵은 각 데이터의 오프셋과 키를 매핑하고 각 오프셋이 가리키는 값은 최신 값임을 보장한다.	**LSM 트리가 성능을 최적화한 방법** - **블룸 필터(Bloom Filter)** - 특정집합에 속하는지 여부 확률적으로 알아낼 수 있는 자료구조 - 추가적인 디스크 읽기 안할 수 있지만,블룸 필터는 데이터가 있다고 판단했으나 실제로 없는 경우 false positive 발생 - **레벨 컴팩션 (Leveled Compaction)** - SS테이블을 여러 단계로 구분, 단계가 높아질 수록 테이블이 커짐. 한 레벨의 SS 파일끼리는 키가 겹치지 않아 데이터 찾기 위해 level 수만큼만 쿼리하면 됨 - **Skip List** - 멤테이블을 skip list로 구현하기 (Rocks DB, level DB)	-