CP3. 저장소와 검색

Part 1. 데이터 시스템의 기초

3. 저장소와 검색

▼ 개요

- 데이터베이스가 데이터를 저장하는 방법과 데이터를 요청했을 때 다시 찾을 수 있는 방법을 설명한다.
- DB 가 저장과 검색을 내부적으로 처리하는 방법을 개발자가 주의해야하는 이유는 뭘까?
  - 대개 개발자가 처음부터 자신의 저장소 엔진을 구현하기보다는 사용 가능한 여러 저장소 엔진 중에 애플리케이션에 적합한 엔진을 선택하는 작업이 필요하다. (즉, 내부 엔진은 어떻게 동작하는지, 적합한 엔진을 사용하는 방법은 어떻게 되는지를 알아야 **선택할 수 있는 안목**을 기를 수 있다.)
- 트랜잭션 작업부하에 맞춰 최적화된 저장소 엔진과 분석을 위해 최적화된 엔진 간에는 큰차이가 있다. (트랜잭션 처리나 분석)
- 분석에 최적화된 저장소 엔진 (컬럼 지향 저장소)
- 이번장에, **RDB와 NoSQL 로 불리는 DB에 사용되는 저장소 엔진**을 설명하고 **로그 구조(Log-structured) 계열 저장소**와, **(B트리 같은) 페이지 지향(page-oriented) 계열 저장소 엔진**을 추가로 검토한다.

▼ 데이터베이스를 강력하게 만드는 데이터 구조

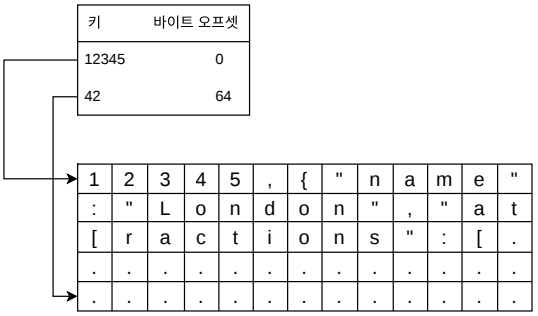
- 가장 간단한 DB

```
#!/bin/bash

db_set () {
    echo "$1, $2" >> database
}

db_get () {
    grep "^$1," database | sed -e "s/^$1,//" | tail -n 1
}
```

  - key value 저장소 구조에 db_get 시 가장 최근 값을 반환.

- 일반적으로, 파일 추가 작업은 매우 효율적이기 때문에, db_set 은 좋은 성능을 보여준다
  db set과 마찬가지로 많은 DB는 내부적으로 추가 적용 (append-only) 데이터 파일인 **로그(log)**[여기서는 연속된 추가 전용 레코드를 의미한다.] 를 사용한다.
- 반면에, db_get은 많은 레코드가 있을 경우 성능이 매우 좋지 않다. 매번 키를 찾을 때 마다 전체 DB 를 스캔한다.

- **색인 (index) - mysql index, PostgreSQL index**
  - DB에서 특정 키의 값을 효율적으로 찾기 위해서는 다른 데이터 구조가 필요하다. 바로 **색인**
  - 일반적인 개념은 어떤 부가적인 메타데이터를 유지 하는 것
  - 색인은 기본 데이터에서 파생된 추가적인 구조이다.
  - DB는 색인의 추가와 삭제를 허용한다.
    이 작업은 데이터베이스의 내용에는 영향을 미치지 않는다. 단지 질의 성능에만 영향을 준다.
    - 쓰기 과정에 오버 헤드가 발생한다. (인덱스 테이블에도 추가적인 적재가 필요함)
    - 색인을 잘 선택했다면 읽기 질의 속도가 향상한다.
  - 해쉬 색인
    - 키 값 저장소는 대부분 프로그래밍 언어에서 볼 수 있는 사전 타입과 유사하다. 보통 hashMap, hashTable 으로 구현한다.
    - 가장 간단한 색인 전략은 키를 데이터 파일의 바이트 오프셋에 매핑해 인메모리 해시 맵을 유지하는 전략이다.



    - 파일에 새로운 key-value 를 추가할 때마다 방금 기록한 데이터의 오프셋을 반영하기 위해 해쉬 맵도 갱신해야한다.
    - 값을 조회할 때는 해시 맵을 사용해 인메모리 오프셋을 찾아 해당 위치를 구하고 값을 읽는다.
    - 이 방법은 매우 단순해 보이지만, 실제로 많이 사용하는 접근법이다. ((비트캐스크)가 기본적으로 사용하는 방식)
      - 비트캐스크는 해시 맵을 전부 멤모리에 유지하기 때문에 사용 가능한 램(RAM)에 모든 키가 저장된다는 조건을 전제로 고성능 읽기, 쓰기를 보장한다.
    - 비트캐스크 같은 저장소 엔진은 각 키의 값이 자주 갱신되는 상황에 매우 적합하다.
      - 키당 쓰기 수가 많지만, 메모리에 모든 키를 보관할 수 있다.
    - 지금처럼 파일에 항상 추가만 한다면 결국 디스크 공간 부족이 된다.
      - 해결책은, 특정 크기의 세그먼트로 로그를 나누는 방식이 좋다.

- 특정 크기에 도달하면 세크먼트 파일을 닫고 새로운 세그먼트 파일에 이후 쓰기를 수행한다.
  세그먼트 파일들에 대해 컴팩을 수행할 수 있다. (컴팩션은 로그에서 중복된 키를 버리고 각 키의 최신 갱신 값만 유지)

▼ 트랜잭션 처리나 분석
▼ 컬럼 지향 저장소
▼ 정리