데이터 중심 애플리케이션 설계

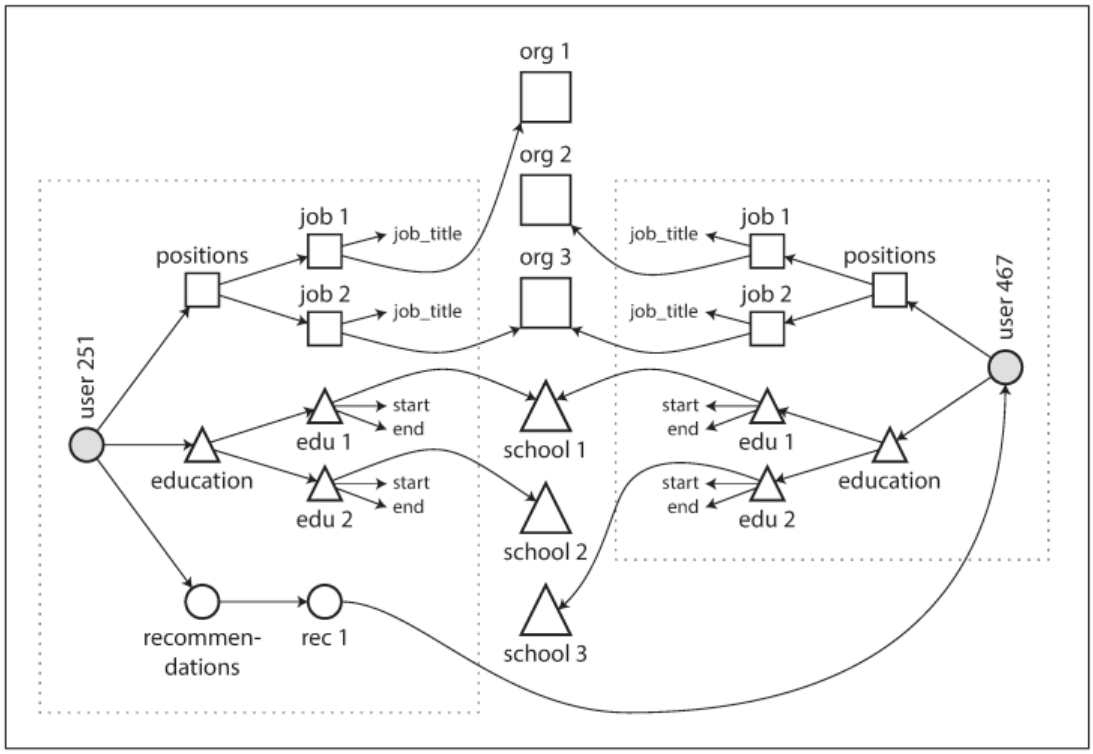
생성일	@2021년 11월 6일 오후 10:54
태그	

Part1

- 데이터 모델과 질의 언어
  - 저장하기를 원하는 데이터의 형식을 어떻게 할 것인가?
  - 애플리케이션의 요구사항에 따른 적합한 모델은 무엇인가?
  - 데이터 모델의 종류
    - 관계형 데이터 모델
    - 문서형 데이터 모델
    - 그래프형 데이터 모델
  - 관계형 모델
    - 특징
      - 각 데이터는 관계로 구성된다.
      - 관계는 순서가 없는 행으로 저장된다.
      - 데이터를 여러 테이블로 나누어 찢는 관계형 기법은 다루기 힘든 스키마와 복잡한 애플리케이션 코드 발생
    - 이용 사례
      - 트랜잭션 처리
      - 일괄 처리
    - 객체 관계형 불일치
      - 임피던스 불일치
        - 객체지향적 데이터를 SQL 데이터 모델로 변환할 경우 애플리케이션 코드와 데이터 베이스 모델 객체(테이블, 로우, 컬럼) 사이에 거추장 스런 전환 계층이 필요
        - 액티브레코드(ActiveRecord), 하이버네이트(Hibernate)와 같은 ORM이 간격을 매워주지만 두 모델간의 차이를 좁힐 수는 없음

        - 일대다(one-to-many) 관계
            - 예) 사용자 정보 중, 경력이 여러개 일 경우
            - JSON 모델을 통해 임피던스 불일치를 줄일 수 있다.
                - 몽고DB, 리싱크DB, 카우치DB, 에스프레소
    - 문서 모델
        - 특징
            - 상위 레코드 내에 중첩된 레코드를 저장
            - 조인이 필요하지 않음

    - 다대일(many-to-one) 관계
        - 중복된 데이터의 정규화를 통해 생성
        - 문서모델, nosql에 적합하지 않은 관계
    - 다대다(many-to-many) 관계
        - 관련 항목을 표현하기 위해 외래키 혹은 문서 참조로 표현
        - 다대다 관계에 해당 되는 실제 예제는 무엇이 있을까요?
            - JPA에서는 이걸 어떻게 풀어낼까요?



- 관계형 데이터베이스와 문서 데이터베이스
  - 애플리케이션이 일대다 관계 (트리구조) 이거나 레코드 간의 관계가 없다면 문서 데이터베이스
  - 다대다 관계가 일반적으로 활용되는 애플리케이션일 경우에는 관계형 데이터베이스

- 데이터를 위한 질의 언어
  - 선언형 질의 언어
    - 결과가 충족해야 하는 조건과 데이터를 어떻게 변환해야 할지 질의
    - 데이터베이스에게 자동으로 최적화 할 수 있는 여지를 제공
    - 병렬 실행에 적합
  - 명령형 질의 언어
    - 특정 순서로 특정 연산을 수행하게끔 컴퓨터에게 질의
    - 명령어를 특정 순서대로 수행하게끔 지정하기에 병렬 처리가 매우 어려움
  - 맵리듀스 질의

        - 맵리듀스 : 많은 컴퓨터에서 대량의 데이터를 처리하기 위한 프로그래밍 모델
        - 몽고DB, 카우치DB
        - 많은 문서를 대상으로 읽기 전용 질의를 수행할 때 사용
        - 선언형 질의와 명령형 질의의 중간 (하이브리드?)
        - map(collection)과 reduce(fold, inject) 함수 기반
            - 람다와 스트림에서 제공하는 기능을 말해 봅시다.
        - 입력으로 전달된 데이터만 사용하고 추가적인 데이터 베이스 질의를 수행할 수 없음

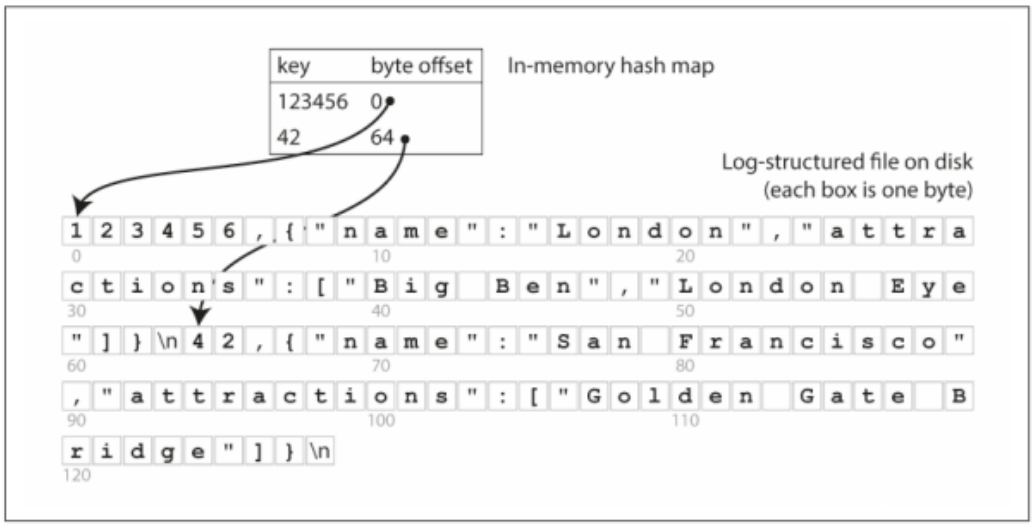
- 그래프형 데이터 모델
    - 다대다 관계, 특히 데이터간 연결이 복잡한 데이터를 표현하기 위한 모델
    - 정점과 간선으로 표현
    - 사례
        - 네비게이션, 도로나 철도 네트워크
        - 소셜 그래프 (페이스북 친구 관계)
        - 웹 그래프
    - 속성 그래프
        - Neo4j, 타이탄, 인피니티그래프
    - 트리플 저장소
        - 데이토믹, 알레그로그래프
- 사이퍼 질의 언어
- 트리플 저장소
    - 모든 정보를 주어, 서술어, 목적어 형식으로 저장
    - 예) 데기, 결혼하다, 여자친구
        - 데기와 여자친구는 정점 결혼하다라는 간선을 통해 이어짐

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/e4e09c77-bc05-45a2-a4f6-e9215a8db703/트리플스토어_기술서_v3.1-ajg-kms.hwp

    - 시맨틱 웹
        - WWW(World Wide Web)의 창시자 팀 버너스리가 제안한 차세대 지능형 웹 모델
        - Semactic Web (의미론적인 웹)이란 뜻으로 각 리소스 (정보와 자원) 사이의 관계-의미 정보를 기계가 처리할 수 있는 형태(온톨로지)로 표현하도록 하는 기술
        - RDF(Resource Description Framework)
            - 웹상의 자원의 정보를 표현하기 위한 규격
            - 서로 다른 메타데이터 간의 어의, 구문 및 구조에 대한 공통적 규격을 제공
            - 스파클(SPARQL) 질의 언어
        - 온톨로지

- 저장소와 검색
    - 가장 기본적인 데이터 베이스 구조 (append, key-value)
        - 연속된 추가전용 레코드인 log 데이터
        - 호출 할때마다 파일의 끝에 log를 추가
        - log 검색 비용은 O(n)이며, 데이터가 많을 경우 오래 걸림
        - 특정 키의 값을 효율적으로 찾기 위한 색인 구조 필요

    - 해시 색인
        - 기본적인 인메모리 데이터 구조
        - 해시맵 구조
        - key-value 저장소
            - key값에 해당하는 value로 데이터 파일의 오프셋을 맵핑해 둠



    - O(1) 검색
- append 형태로 데이터를 추가시 디스크 공간 부족 현상 발생
    - 세그먼트로 로그를 나눔
    - 세그먼트 파일을 컴팩션 수행
        - 컴팩션? 중복된 키를 버리가 각 키의 최신 갱신 값만 유지
        - 백그라운드 수행을 통해 새로운 데이터를 생성하여 해당 메모리 주소(오프셋)를 바라보게 함
- 한계
    - 키가 많아지면 해시 테이블은 확장을 하고 결국 메모리 상에 유지할 수 없다.
    - 디스크 상에 해시맵 유지가 가능하지만 성능상 이점이 없다.
    - 범위 질의에 효율적이지 않다. 모든 키는 해싱된 값을 통해 저장되므로 key1다음에 key2가 저장된다는 보장이 없다.

- SS테이블과 LSM트리
    - SS 테이블
        - key- value쌍을 키로 정렬한 Sorted String Table을 SS 테이블이라고 한다.
        - 각 키는 각 병합된 세그먼트 파일 내에 한번만 나타나야 한다
            - 세그 먼트 내에는 각 키별 오프셋이 저장
            - 각 세그먼트를 병합시 중복된 키 존재

    - 가장 최신의 키를 유지하여 최적화
- 정렬된 상태이므로 모든 키의 색인이 필요 없음 (일부 키의 색인만 존재)
    - 일부 색인으로 접근하여 특정 키를 탐색한다
    - 색인에 해당하는 키를 기준으로 레코드들을 그룹화하여 압축하여 디스크에 저장
    - 디스크 공간 절약 및 I/O 대역폭 사용 감소

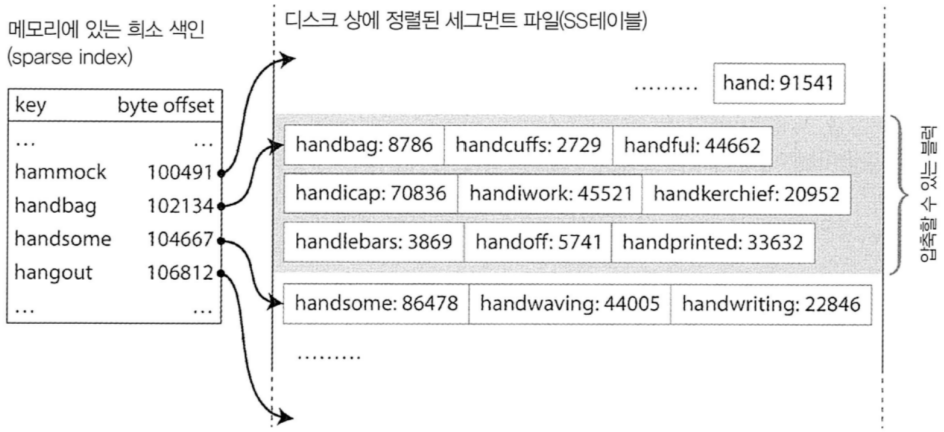


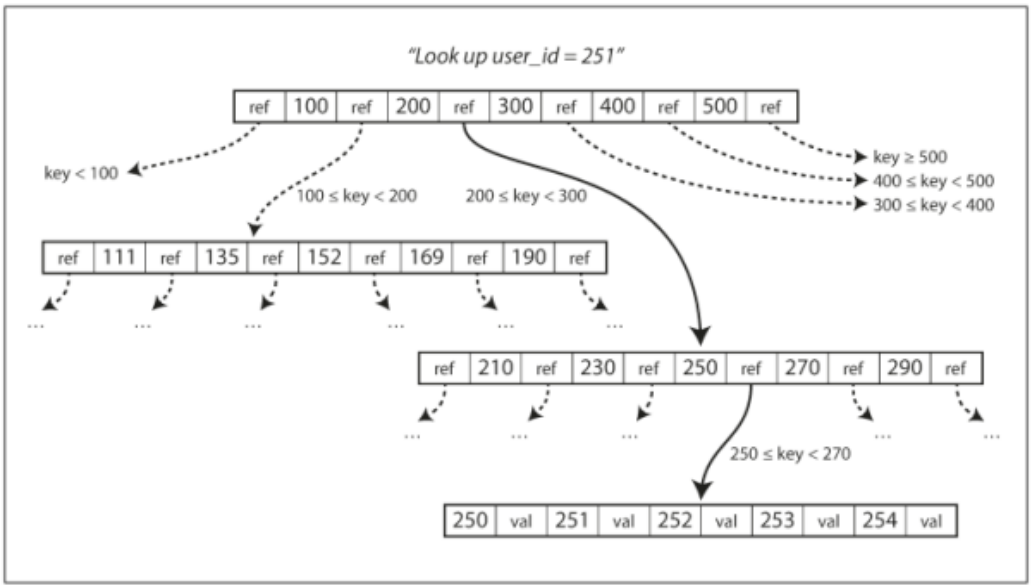
그림 3–5. 인메모리 색인을 가진 SS테이블

- 정렬 유지를 위해 레드블랙트리 혹은 AVL트리를 이용한다.
    - 쓰기 작업시 해당 균형 트리(balanced tree)데이터 구조에 추가하며 이를 멤테이블이라고 함
    - 멤테이블이 임곗값보다 커지면 (데이터 구조 확장시) SS테이블 파일로 디스크에 저장
        - 이미 정렬된 값이므로 효율적으로 저장 수행
        - SS 테이블 파일은 가장 최신 세그먼트
        - 쓰기 작업시 새 멤테이블에 수행
    - 읽기 요청시 멤테이블에서 키를 탐색, 최신 세그먼트 그다음 세그먼트를 계속해서 탐색
    - 세그먼트의 병합과 컴팩션은 백그라운드에서 수행
    - 아직 디스크에 기록되지 않은 멤테이블의 데이터를 위해 따로 디스크에 로그를 기록
- LSM 트리

- 로그 구조화 병합트리(Long-Structured Merge-Tree)
- 정열된 파일 병합과 컴팩션 원리를 기반

- 최적화
  - 블룸필터
    - 해당 값이 집합에 속해 있는가? (membership checking)
    - 해당 값을 여러 함수를 통해 해싱 값을 얻고 해당 해싱 값들에 대응 하는 테이블이 true이면 데이터 존재

- Btree
  - 데이터 베이스 색인(index)에서 사용되는 자료구조
  - 4KB의 고정 블록이나 페이지 단위로 수행
  - SS 테이블과 마찬가지로 정렬된 키-값 쌍을 유지
  - 루트부터 시작해 최종 리프 페이지까지 탐색



  - 하위 페이지 참조하는 ref의 수(분기 계수)

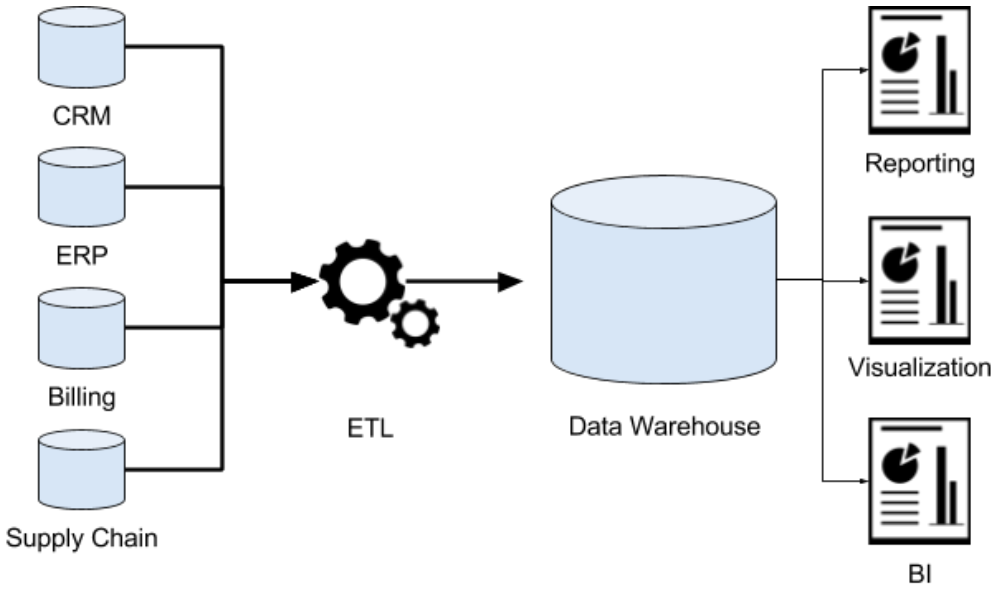
  - 새로운 키 추가
    - 새로운 키를 포함하는 페이지 탐색
    - 해당 범위의 페이지에 키 추가
      - 페이지 공간이 부족하면, 페이지를 쪼개고 상위 페이지에 링크를 갱신
    - 계속해서 균형 유지 n개의 키를 가질 경우 log(n)의 깊이 유지

        - 분기 계수를 늘림
    - 고장 상황
        - 쓰기중 고장이 발생할 경우 색인이 훼손되고 고아페이지가 발생
        - 디스크에 쓰기전 변경 내역을 기록하는 전용 파일 운영
    - 다중 제어
        - 다중 쓰레드 운영을 통해 동시성 제어 필요
        - 래치(latch)로 트리의 데이터 구조 보호
    - 최적화
        - 로그 유지대신 쓰기 복사 방식을 이용해 변경된 데이터는 다른곳에 쓰기 후 새로운 버전을 생성하여 그 위치를 가르키게 함, 동시성 제어에 효과 (굳이 페이지에 latch를 걸 필요가 없음)
        - 페이지에 전체 키가 아닌 키를 축약해서 씀
            - b+tree 모델 참조
        - 트리에 포인터를 추가하여 리프 페이지가 양쪽 형제 페이지에 대한 참조를 가져 연속적으로 스캔이 가능하도록 함 (같은 계층에서는 연속적 스캔이 가능)
    - LSM트리 vs Btree
        - LSM트리는 쓰기 증폭이 더 낮다.
            - 쓰기 증폭 : 쓰기 한번이 디스크에 여러번 쓰기를 야기하는 효과
        - LSM트리는 압축률이 좋다
            - btree보다 디스크에 더 적은 파일 생성
            - btree는 페이지 단위의 생성으로 데이터 파편화 문제
        - LSM의 컴팩션은 성능에 영향을 미친다.
        - btree의 키 색인은 한 곳에만 존재, LSM은 다른 세그먼트에 중복 키 존재
- 기타 색인
    - 보조 색인
        - 기본 색인이 있는 파일에 보조 색인을 통해 다양한 검색을 지원
    - 색인 안에 값 저장

            - 키에 해당하는 값이 데이터가 아닌 레퍼런스를 저장하는 경우 힙에 저장한다.
            - 힙 파일을 참조하는 것은 성능에 불이익이 크다.
            - 클러스터드 인덱스를 통해 인덱스 안에 데이터의 참조(ref)를 저장한다.
                - MySQL의 InnoDB : 기본키는 클러스터드 인덱스 이고 보조색인은 기본키
                - MSSQL : 테이블당 하나의 클러스터드 인덱스 지정가능
            - 클러스터드 인덱스와 논클러스터드 인덱스의 절충안 : 커버링 인덱스
                - 색인 안에 테이블 컬럼의 일부를 저장
    - 다중 컬럼 색인
        - 결합 색인
        - 하나의 키에 여러 필드를 단순히 결합
        - key : (성, 이름) value : 전화 번호인 구식 전화부와 유사
    - 인메모리 데이터 베이스
        - Memcached
        - 빠른 읽기
        - 휘발성
            - persistence 기능으로 회피
                - snapshot
                - AOF(append of file)
        - 인메모리 키-값 저장소
            - redis
                - set, sorted set, hash, list
            - couchbase
            - arcus
                - collection(set, list, map, b+tree), hash
            - 메모리 데이터 구조를 디스크에 기록하는 오버헤드 회피
    - 트랜잭션 처리 및 분석

Aa 특성	트랜잭션 처리 시스템(OLTP)	분석 시스템(OLAP)
주요 읽기 패턴	질의당 적은 수의 레코드, 키 기준으로 가져옴	많은 레코드에 대한 집계
주요 쓰기 패턴	임의 접근, 사용자 입력을 낮은 지연 시간으로 기록	대규모 불러오기, 이벤트 스트림
주요 사용처	웹 애플리케이션을 통한 최종 사용자	의사결정 지원을 위한 내부 분석가
데이터 표현	데이터의 최신 상태	시간이 지나며 일어난 이벤트 이력
데이터셋 크기	기가바이트에서 테라바이트	테라바이트에서 페타바이트
제목 없음	사업 운영에 중요	
제목 없음	높은 가용성, 낮은 지연 시간의 트랜잭션 처리 기대	
병목	디스크 탐색	디스크 대역폭

- 데이터 웨어 하우스
  - OLTP 질의는 질의 비용이 비쌈 (성능 저하)
  - 이를 마음껏 질의할 수 있는 개별 데이터 베이스
  - ETL을 통해 데이터 웨어하우스로 데이터를 가져옴



- 데이터 마켓, 데이터 레이크, 데이터 허브

- 컬럼 지향 저장소
  - 여러개의 컬럼을 통합한 한 로우로 데이터를 처리 하지 않음 (로우 지향 방식 탈피)
  - 컬럼 단위로 쪼개어 데이터를 관리
  - 질의에 사용되는 컬럼만 조회