SQL 레벨업 1장

DBMS 아키텍처

- 쿼리 평가 엔진
- 버퍼 매니저
- 디스크 용량 매니저
- 트랜잭션 매니저, 락 매니저
- 리커버리 매니저

성능이라는 관점에서 중요한 것은 쿼리평가 엔진(이 세우는 실행계획)

DBMS와 버퍼

기억 비용

데이터를 저장하는 데에 소모되는 비용

HDD는 메모리보다 기억 비용이 낮다 → 같은 비용으로 저장할 수 있는 데이터 용량이 많다.
하지만 데이터 접근 속도는 더 떨어짐

DBMS가 사용하는 대표적인 기억장치

- HDD
  - DBMS 데이터 저장은 대부분 대부분 HDD
  - 2차 기억장치
- 메모리
  - 디스크(HDD)에 비해 기억 비용이 굉장히 비쌈 → 하드웨어 1대에 탑재할 수 있는 양이 크지 않음
  - 규모 있는 상용 시스템의 내부 데이터를 모두 올리는 것은 불가능
  - 아무리 많다해도 100GB 넘는 경우 거의 없음
  - 성능 향상 때문에 사용
- 버퍼를 활용한 속도 향상
  - 디스크 접근을 줄일 수 있다면 큰 폭의 성능 향상이 가능

- 성능 향상을 목적으로 데이터를 저장하는 메모리 → 버퍼
- 물리적인 매체로 메모리가 사용되는 경우가 많음
- 버퍼에 데이터를 어떻게, 어느 정도 기간 동안 올릴지 관리하는 것이 버퍼 매니저

메모리 위에 있는 두 개의 버퍼

- 데이터 캐시
  - ex. MySQL → 버퍼 풀, PostgreSQL → 공유 버퍼
  - 디스크에 있는 데이터의 일부를 메모리에 유지하기 위해 사용
  - '디스크를 건드리는 자는 불행해진다' … 디스크 갔다오면 느려짐
- 로그 버퍼
  - ex. MySQL → 로그 버퍼, PostgreSQL → 트랜잭션 로그 버퍼
  - 로그 버퍼는 갱신처리와 관련이 있음
  - 갱신 처리는 비동기로 이루어짐(먼저 로그 버퍼로 감)

메모리의 성질이 초래하는 트레이드오프

- 휘발성
  - 장애 발생 시 데이터 모두 사라짐
  - 이를 회피하고자 DBMS는 커밋 시점에 반드시 갱신 정보를 로그 파일에 씀
  - 커밋 → 갱신 처리 확정
  - DBMS는 커밋된 데이터 영속화(⇒ 디스크에 동기 접근, 지연 발생)

워킹 메모리

- DBMS는 정렬, 해시 관련 처리에 사용되는 작업용 영역을 하나 더 가지고 있음
- PostgreSQL → 워크 버퍼, MySQL → 정렬 버퍼로 부름
- 이 영역이 성능적으로 중요한 이유는 만약 이 영역이 다루려는 데이터양보다 작아 부족해지는 경우가 생기면 대부분의 DBMS가 저장소를 사용 (swap)
- 워킹 메모리가 부족할 때 사용하는 일시 영역: PostgreSQL → pgsql_tmp, Oracle → TEMP Tablespace 등(저장소 위에 있어서 느림)

- 저장소가 부족해지면 전체적인 속도가 느림
- DBMS는 메모리가 부족하더라도 무언가를 처리하려고 계속 노력하는 미들웨어

DBMS와 실행 계획

데이터 접근 방법

- 파서
  - 사용자로부터 입력받은 SQL 구문을 검사
- 옵티마이저
  - 최적화의 대상은 실행 계획
  - 인덱스 유무, 데이터 분산/편향 정도, 내부 매개변수 등의 조건을 고려해서 가능한 많은 실행 계획을 작성하고 비용을 연산하여 실행 계획을 선택
- 카탈로그 매니저
  - DBMS 내부 정보를 모아놓은 테이블 → 테이블 또는 인덱스의 통계 정보가 저장되어 있음
    - 각 테이블의 레코드 수
    - 각 테이블의 필드 수와 필드의 크기
    - 필드의 카디널리티(값의 개수)
    - 필드값의 히스토그램(얼마나 분포되어 있는가)
    - 필드 내부에 있는 NULL 수
    - 인덱스 정보
- 플랜 평가
  - 최적의 실행 결과를 선택하는 것

옵티마이저와 통계 정보

- 카탈로그 매니저가 관리하는 통계 정보에 대해서는 데이터베이스 엔지니어가 신경써야 함
- 플랜 선택을 옵티마이저에게 맡기는 경우 실제로 최적의 플랜이 선택되지 않는 경우가 꽤 많음

- 카탈로그 정보가 갱신되지 않는다면 옵티마이저는 오래된 정보를 바탕으로 잘못된 실행 계획 세움

최적의 실행 계획

- 테이블의 데이터가 많이 바뀌면 카탈로그의 통계 정보도 함께 갱신해야 함
    - 쿼리 실행할 때 통계 정보를 실시간으로 수집하는 기능(JIT)을 가진 DBMS도 있음 → 지연이 일어날 가능성이 있음
- 배치 처리가 있을 때는 Job Net(각각의 작업의 실행 순서) 조합
- DBMS마다 통계 정보 갱신 시점이 다름
- 통계 정보 갱신은 실행 비용이 높은 작업

실행 계획이 SQL 구문의 성능을 결정

실행 계획 확인 방법

- EXPLAIN (DBMS마다 명령어 다름)

확인 가능한 내용

- 조작 대상 객체
- 객체에 대한 조작의 종류
- 조작 대상이 되는 레코드 수

인덱스 스캔

테이블 결합의 실행 계획

- DBMS는 결합을 할 때 세 가지 종류의 알고리즘 이용
    - 가장 간단한 알고리즘 → Nested Loops : 한 쪽 테이블을 읽으면서 레코드 하나마다 결합 조건에 맞는 레코드를 다른 쪽 테이블에서 찾는 방식(=중첩 반복)

    - Sort Merge: 결합 키로 레코드를 정렬하고 순차적으로 두 개의 테이블을 결합하는 방법
    - Hash: 결합 키값을 해시값으로 맵핑
- 결합의 경우 먼저 접근하는 테이블(드라이빙 테이블)이 중요

실행 계획의 중요성

- 옵티마이저가 실수를 할 경우 실행 계획을 수동으로 변경해야 함
- 실행 계획을 변경하려면 어떤 선택지가 있는지를 알아야 함

RECAP

- 데이터베이스는 다양한 트레이드오프의 균형을 잡으려는 미들웨어
- 특히 성능적 관점에서는 데이터를 저속의 저장소(디스크)와 고속의 메모리 중에 어디에 위치시킬지의 트레이드오프가 중요
- 데이터베이스는 갱신보다도 검색과 관련된 것에 비중을 두도록 기본 설정되어 있지만, 실제 시스템에서도 그럴지는 판단이 필요
- 데이터베이스는 SQL을 실행 가능한 절차로 변환하고자 실행 계획을 만듦
- 사실 사용자가 실행 계획을 읽는다는 것은 데이터베이스의 이상을 어기는 일이지만 세상 모든 것이 이상적으로 돌아가지는 않음

추가)

LRU(Least Recently Used)

- 불특정 다수의 사용자로부터 요구되는 데이터의 캐시 히트율을 올리고 싶을 때, 어떤 데이터를 캐시하는 것이 좋을지를 기계적으로 풀기 위한 기본적인 알고리즘
- 참조 빈도가 가장 적은 것을 캐시에서 버리는 알고리즘