소프트웨어를 해부하면 크게 동작과 데이터로 나눌 수 있다. DB모델링을 해야 하는 이유? 데이터를 잘 다루기 위해서.

모델링? 모델을 보고 닮게 만든 일.

모델링

- 만들고자 하는 머릿속 생각을 누구나 볼 수 있게 본을 뜨는 것.
- 객체나 데이터베이스를 그림으로 표현하는 것.

현업에선 보통 3개 씀.

- 시퀀스 다이어그램
- UML
- ERD

논리모델과 물리모델

IE 표기법

Barker 표기법

RDBMS는 데이터 무결성을 준수한다. (NoSQL는 데이터 무결성을 보장하지 않는다.)

RDBMS의 무결성

RDBMS 무결성는 아래 3가지 무결성을 충족한다.

- 개체 무결성
- 참조 무결성
- 범위 무결성 : 쉽게 얘기하면 같은 컬럼이면 동일한 데이터 타입만 들어간다.

알아야 할 용어

- Identifying Relationship : 식별 관계
- Mandatory : 필수
- Cascade : 연쇄

Identifying Relationship & Mandatory

Identifying Relationship : A 테이블의 PK가 B 테이블에서도 PK이다.

Mandatory : 내가 태어나기 위해서는 부모님이 계셔야 하지만 Optional : 부모님이 태어나기 위해서 내가 필수적인 건 아니다.

Mandatory : !Null

Foreign key option

RESTRICT : 참조값이 있으면 에러 CASCADE : 연쇄적으로 참조값이 있는 애들을 다 지움 SETNULL : 참조값을 NULL로 세팅 NOACTION : RESTRICT랑 같음

사용자 흐름대로 데이터베이스를 설계하는게 가장 편했음. ex) 글을 쓰려면? 회원가입을 먼저해야됨. 그래서 user 테이블 생성.

관행적으로 테이블명은 복수형으로 씀.

테이블 설계시엔 기술적인 차원 뿐만 아니라 기획적인 측면도 고민해야됨.

- ex) username을 PK로 하면 아이디 변경이 불가능함.

보통은 DATETIME보단 TIMESTAMP를 씀. 글로벌 서비스를 생각해서.

글로벌 서비스에서 보통 타임존을 고려하지 않음. ex) SNS 포스트의 10초 전. 10분 전. (TIMESTAMP 연산. 서버 입장에서 매우 효율적)

ERD(Entity-Relation-Diagram)

점선 : Non Identifying Relationship. 실선 : Identifying Relationship일 때. Composite key.

LINE, 카카오 등 메신저

- 친구 메시지도 내 id 데이터로 등록되어 있음. 방 인원수만큼 같은 메시지가 복사되어 있는 구조.
- 왜? JOIN을 없앨 수 있고, 극단적으로 성능을 높일 수 있음.
- 대신 수정에 대한 비용이 높음(다 수정해줘야 되기 때문). 트위터는 수정이 안 됐었음.

정규화 & 역정규화(성능 향상을 위해 데이터 중복을 허용)

- 정규화 : 중복을 최대한 줄임.
- 장바구니 화면을 그리려면 product_in_cart와 goods를 loop로 연산해야 된다.
- product_in_cart에 장바구니 화면을 그리기 위한 goods 필수 데이터(price, name 등)를 중복하여 넣어준다.
- 이렇게 하면 join하지 않고 select 1번으로 줄여서 성능 향상을 달성할 수 있음. 이게 역정규화.
- 역정규화란 개념이 먼저 있던게 아니라 성능 향상을 위해 조치를 취하다 보니 이러한 이름을 붙이게 된 것.

과제: 전체적으로 필요한 컬럼 추가. 배송 회사 테이블, 상품 이미지 테이블. 개인 프로젝트 ERD.

이미지 저장법 user_id.jpg fastbook.com/images/0001.jpg

url로 저장하면 나중에 서버 확장하기 쉬움 img1.fastbook.com img2.fastbook.com

브라우저에 캐싱되기 때문에 파일명이 같아서 이미지가 바뀌어도 변경이 안 될 수 있음. 그래서 보통 갱신을 위해 (user_id).jpg?20190629 과 같은 query string을 붙여서 저장한다.

상의 0001 티 0002 반팔 0009 어린이 0008 반팔 0010

gooods에 category_tag 개념으로 #0001 #0002 #0009 #0008 #0010 을 넣어서

복수의 카테고리에 해당된 상품 N:M 관계를 해결하며 중계 테이블도 없어도 됨.

검색은 0010을 해버리면 반팔인 상품 모두 검색 가능.

user = 1천만 post = 10억개

SELECT * FROM post WHERE LIMIT 100000, 100

LIMIT 연산은 100100개를 가져와서 10만개 버리고 100개 보여줌.

SELECT * FROM post WHERE id > 100000 LIMIT 100 절대값 10만으로 가져오면 중간에 삭제된 경우 정확한 처리가 안 됨.

상대값 을 따로 컬럼으로 관리하면 해결 가능. but, 매번 update를 해줘야 함.

과연 10만번째 페이지를 보는 경우가 많을 것인가? 글삭제가 많을 것인가? 는 고민해봐야 됨.

사실은 이렇게 규모가 커지면 페이징 처리를 안 해도 되도록 UX를 유도함.

INDEX? 데이터의 목차 생성

쿼리 튜닝 SELECT * FROM post WHERE user_id = 10 ORDER BY id DESC INDEX(user_id, id)

쿼리 튜닝은 검색의 범위를 좁히는 방향으로 해야한다. -> WHERE location=서울 AND sex=FEMALE (O) sex=FEMALE AND location=서울(X) sex의 모수가 크기 때문.

분포 비율을 생각하며 짜야 됨.

INDEX(location, sex) 를 해줘야 됨. 쿼리를 location부터 날리기 때문.

근데 사실 sex는 인덱스 걸어봤자 50%라 효용이 없음. EXPLAIN(실행계획 분석)을 통해 분석해보고 결정하는게 좋음.

컬럼 배치 순서도 속도에 관련이 있음. id, content, created_date, updated_date, user_id ... 면 3칸을 점프해야됨. id, user_id, content, created_date,...

index를 걸지 않은 컬럼의 경우 WHERE 절에서 컬럼끼리 거리가 가까울수록 빠름.

RDBMS 이해

인덱스

데이터 '목차' SELECT 성능에 매우 중요하다. UPDATE, DELETE를 할 때에도 도움이 된다.(결국엔 SELECT를 먼저하기 때문) 인덱스도 결국 I/O를 사용하는 데이터이므로 적절하게 사용하는게 좋다.(4개 컬럼 정도가 가장 성능이 좋다는 얘기가 있음.) MySQL 8.0부터 Descending Indexes 지원 (Ascending일 경우 ORDER BY DSC 하더라도 임시 파일(임시 테이블)에 기록해놓고 다시 뒤집기 때문에 오버헤드 발생했었음.)

로그성 테이블엔 Index를 안 걸어도 무관.(SELECT만 정말 가끔 하기 때문)

인덱스를 지정할 때에는 Cardinality 가 높은 것부터(분포도가 좋은) 자주 사용되는 칼럼 기본키, 외래키처럼 조인의 연결고리가 되는 칼럼

인덱스가 city, age, gender로 지정되어 있을 때 인덱스를 태우는 방법은?

- 인덱스 첫번째 항목은 꼭 포함되어야 한다.
- WHERE 에서는 인덱스 왼쪽부터 되도록 순서대로.
- ORDER BY, GROUP BY

WHERE age = "19" AND gender = "M"; 는 인덱스를 안 탐. age, gender로 인덱스 따로 또 걸어야 됨.

ANALYZE TABLE

테이블에 대한 통계 및 정보를 분석하여 저장하는 기능 쿼리 옵티마이저에게 큰 도움. 정기적으로 실행해 주면 좋다. 테이블에 READ LOCK이 걸리므로 주의.

OPTIMIZE TABLE

가변 데이터(VARCHAR, TEXT...)에 대한 조각모음 효과 READ LOCK이 걸리므로 역시 주의 시간이 오래 걸릴 수 있으므로 한가한 시간대에 실행 DELETE 데이터 정리

DELETE 한다고 실제 데이터 크기가 줄어들진 않음. OPTIMIZE를 하면 중간에 빈줄을 채우기 위해 밑에 줄부터 위로 땡겨서 새로 씀. 그래서 굉장히 무거울 수 있는 작업.

EXPLAIN

Type

- Const: PK 나 Unique 처럼 오직 하나의 데이터 (id = 1)
- Eq_ref: JOIN 에서 PK 처럼 유일한 값이 존재하는 경우 (ON a.id = b.id)
- Ref: JOIN 에서 몇 개의 값이 존재하는 경우 (WHERE a.post_code = 'a')
- Range: 주어진 범위 내의 값을 가져오는 경우 (BETWEEN)
- Index: 인덱스의 모든 값을 검색
- All: 테이블 전체 데이터 검색. 가장 좋지 않음.

JOIN

나쁜 게 아님. 잘 쓰면 좋음. INNER: 교집합. OUTER: 합집합. (LEFT OUTER, RIGHT OUTER) (기준을 어디에 두느냐)
INNER JOIN: 회원 중에 글쓴 사람. (user, post)

JOIN vs SubQuery

JOIN 이 SubQuery 보다 빠르다. (SubQuery를 실행하면 그 결과를 임시 테이블에 저장하기 때문에 느리다. JOIN은 인덱스만 잘 태우면 임시 테이블 필요 X) 그럼에도 불구하고 많은 사람들이 SubQuery를 사용하는 이유? (가독성 때문이라는 견해들이 많음.)

성능에 영향을 미치는 쿼리

ORDER BY

- 정렬 기준이 너무 많을 때 (가상 테이블을 생성해야 될 수도)
- GROUP BY 또는 DISTINCT 결과를 정렬해야 할 때
- UNION 등 임시 테이블 결과를 다시 정렬할 때
- 랜덤하게 결과를 가져와야 할 때

임시 테이블 사용

- 두 개 이상의 테이블 JOIN(예외: 첫 테이블만 정렬할 때)
- ORDER BY 와 GROUP BY 명시된 칼럼이 다를 때(동시에 쓸 때) (GROUP BY도 정렬 후 카운팅을 하기 때문)
- 인덱스를 사용하지 못하는 GROUP BY
- 인덱스를 사용하지 못하는 DISTINCT(중복 제거)

Stored Procedure

DBMS용 프로그래밍

장점

- 쿼리 실행 성능 향상: 캐시
- 네트워크 트래픽 감소
- 보안성(패킷에서 쿼리가 노출되지 않음)
- DB 프로그램이기 때문에 어플리케이션 독립적

그러나, Stored Procedure는 프로그래밍 언어(Java, PHP 등)로 작성하는 것보다 훨씬 느리다. 디버깅도 어렵고, 러닝커브도 있고...

SERVICE ARCHITECTURE

Monolithic vs MSA

아키텍쳐에서 집중하는 주요 포인트

- Scaling
- Single point of failure

서비스 개발에 필요한 요소

사람 + 소프트웨어 + 시스템

서비스 시작 단계 프로그래밍 언어, DBMS 등은 자유롭게 개발자 1명, UI디자이너 1명, 기획자 1명 서버는 개발을 위한 1대

소프트웨어 개발에 따라 아키텍쳐 전략이 달라질 수 있다.

단계별 기준을 정한다

- 예) 우리 서비스는 5초 이내에 화면이 보여져야 한다.

기본 전략

- 용병은 수를 늘리고 포는 화력을 높인다

1단계

- 서버 1대 = 웹 + DB
- 동접 20명 정도...
- DBMS가 I/O 등 자원을 많이 쓰면서 웹서버가 일을 못함.
- 임계치를 넘어서면 멈춰있따는 느낌이 들 수 있다.

2단계

- 웹서버 1대, DBMS서버 1대
- 동접 100~200명

3단계

- 서버 3대 = (로드밸런서) + 웹 서버 2대 + DBMS서버 1대
- 이용자가 증가하면서 느려지는 현상이 나타날 수 있다.

- 로드밸런스(L4): OSI 7계층 Layer 4

4단계

- 정적 파일 분리
- 웹브라우저는 서버당 자원을 순차적으로 가져온다.
- 로드밸런서 + 웹서버 2대 + DBMS서버 1대, 정적 웹서버 1대

5단계

- 로드밸런서 + 웹서버 4대 + DBMS서버 2대
- 횡으로 확장. Scale Out. DBMS서버는 Scale Out이 쉽지 않음. Scale Up 전략을 주로 함. 처음부터 비싼 거 쓰는게 편함.
- 정적 캐시 적용.
- 지속적으로 변하는 데이터와 그렇지 않은 데이터를 구분한다.

6단계

- DBMS 규모 확장
- 캐시 서버 적용(Redis)

모바일 서비스의 특징

- 이동시간대에 이용률이 증가
- 가벼운 컨텐츠 위주로 소비
- 웹 보다는 앱의 형태를 선호

모바일 서비스의 이용시간대

- 출퇴근시간
- 점심시간
- 잠들기 전

모바일 커머스의 특징

- 오픈마켓, 소셜커머스를 주로 이용
- 구입보다는 구경
- 결제보다는 장바구니

온라인 쇼핑몰 서비스

- 결제, 재고관리가 가장 중요
- DBMS를 Scale out 하기 쉽지 않다.
- 결제DB부터 분리하는게 일반적

DNS에서 Round robin방식으로 로드밸런싱 가능.

실시간 서비스

실시간으로 데이터를 가져가는 방법

- Polling: 주기적으로 웹서버에 접속하여 요청(HTTP, 개발비용 최소화 할 때, 간단한 상담 채팅)

- Long Polling:
- Streaming: 서버가 전체 클라이언트한테 데이터를 밀어줌
- WebSocket: Socket의 단점을 그대로 가져감을 잊으면 안됨. 그래서 웹소켓 서버와 웹서버는 분리를 해야 효용이 좋음. 소켓 자체가 자원을 많이 먹기 때문.

모니터링 툴이 필요한 이유

- 안정적인 서비스와 빠른 대응 + 개발을 위해
- 더욱 전문적인 모니터링 툴이 필요하다

로그는 수집 + 분석

- Logstash, fluentd

통계 Nagios, Zabbix, Icinga

모니터링 툴을 사용할 때 주의사항!!!

- 어쨌든 얘네도 컴퓨터 자원을 사용한다. 배보다 배꼽이 더 커질 수도.

그래서 같은 군집에서 웹서버 1대에만 설치하는 경우가 많음. 한대만 보면 나머지도 거의 같다고 볼 수 있기 때문.

HAProxy : L4, L7 스위치 L7은 어플리케이션 레벨에서 로드 밸런싱. 세션이나 쿠키에 따라서 분기할 수 있음. user_id 따라. 채팅 서버 분배 같은 것도 L7에서.

웹서버랑 DBMS 사이에도 HAProxy를 둘 수도 있음.

DB를 이해해야 결국 서비스 아키텍쳐도 이해할 수 있음.