

Database Architecture

BAF675 금융 빅데이터 분석

이재훈, Week 2

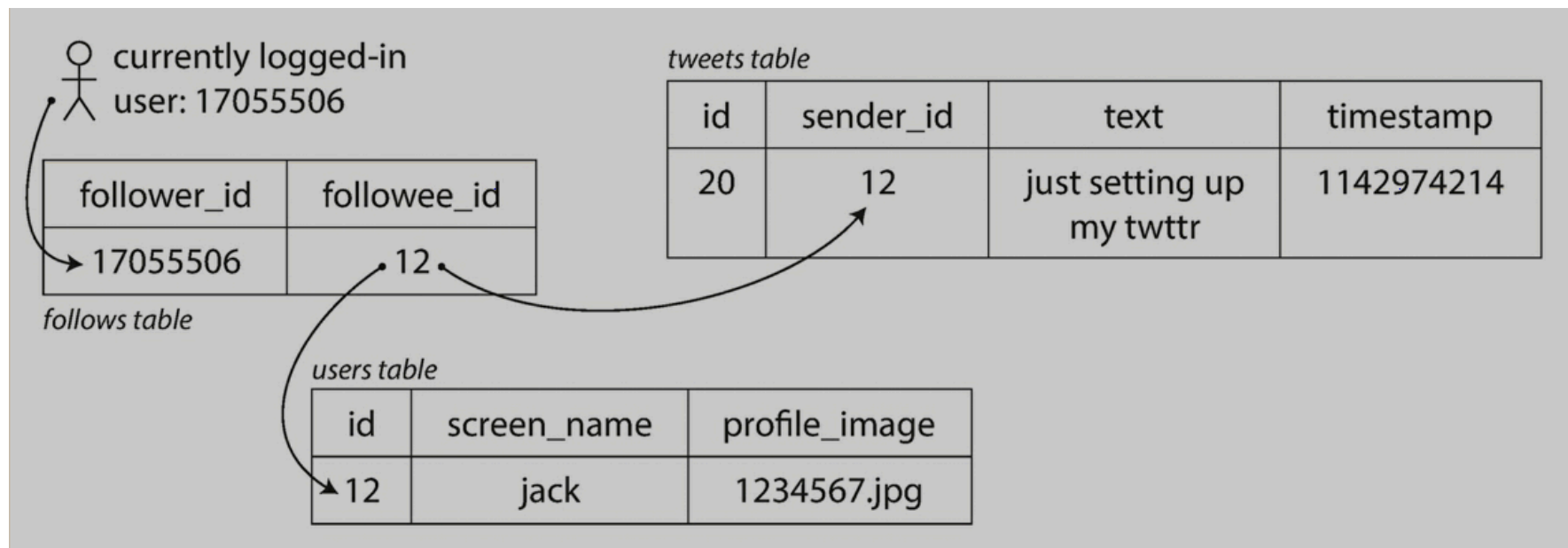
Database Overview

- 교과서 무시하기 (특히 정규화)
- 데이터 특성에 따른 DB 선택
- 텍스트 검색의 특징

교과서 무시하기

- 정규화 (**normalization**): 학교에서 DB 수업을 들으면 가장 먼저 가르쳐주는 개념
 - 단계: 1NF, 2NF, 3NF, Boyce and Codd NF, ...
 - 데이터 중복을 방지해서 저장 장치의 효율적인 사용
 - 데이터 사이의 관계 및 일관성 유지에 도움
 - RDBMS 또는 SQL 의 가장 기본적인 데이터 형태

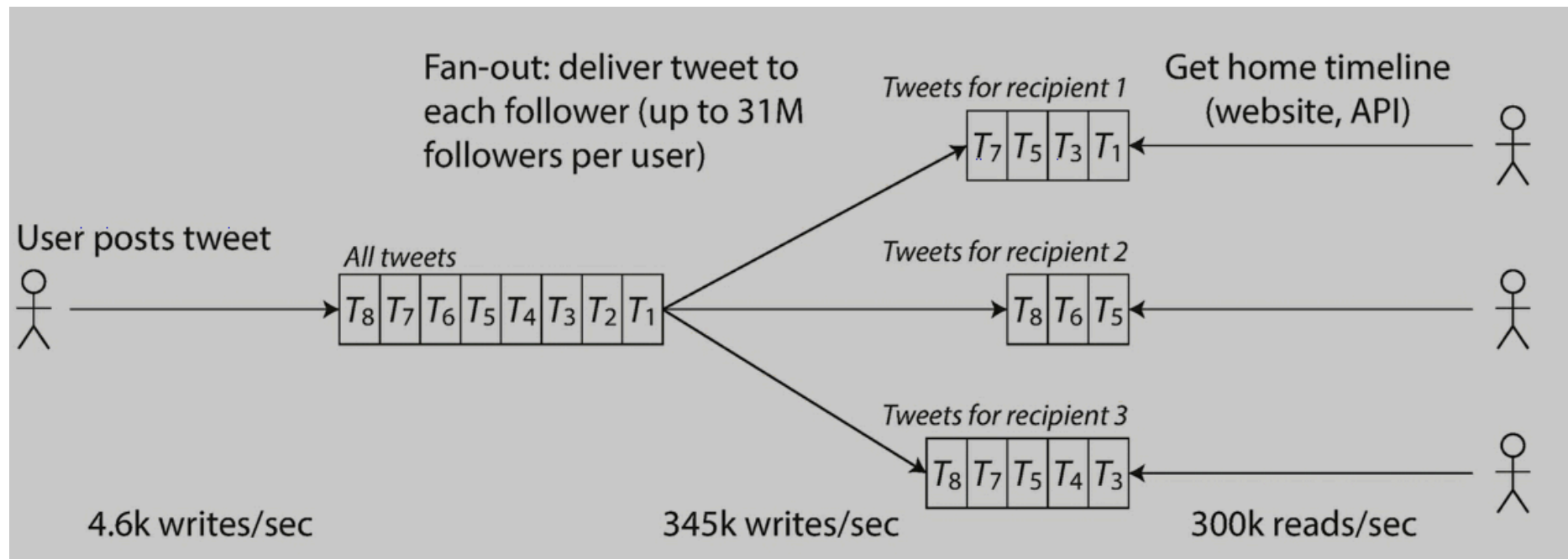
- 한번 트위터를 예로 들어 생각해보죠.
- 교과서에서 가르쳐준대로 정규화를 해서 DB 를 구축하면 이렇게 됩니다.



- 데이터가 중복된거 없이 예쁘게 잘 정리되어있죠.
- 다만 한가지 문제가 있습니다.. 저렇게 하면 시스템 죽습니다.
- 사용자가 몇만명 안되면 저렇게 해도 버틸 수 있어요.
- 하지만 수십만명이 넘어가면 세상에서 제일 비싼 슈퍼 컴퓨터를 써도 감당이 안됩니다.
- 왜 그럴까요??

- 정규화의 가장 큰 문제
 - Merge/join 이 빈번하게 일어남
 - 그 후에 정렬도 해줘야됨
 - Data analytics 에서 가장 비싼 transaction 두 개임
 - 대규모 사용자 처리가 불가능

- 해결책
 - Join 을 안하면 됩니다.
 - 그런데 Join 을 안하려면 정규화 하면 안됩니다.
 - 따라서 정규화도 버리고, 데이터를 중복해서 여러 군데 중복해서 저장합니다.
 - 왜? 디스크는 완전 싸니까요. 👍



- 요런 식으로 누가 트윗을 올리면, follower 들에게 자동으로 미리 넣어줍니다.
- 이렇게 하면 join 이 필요 없죠. 따라서 문제 해결



- 트럼프는?? 팔로워가 6,700만명인데??
- 괜찮습니다. 그래봤자 280글자 인걸요.
- 그리고 디스크는 싸니까요. 아주 싸요. Dirt cheap.

- 이런 식으로 스케일을 위해서 효율성을 버려야할 때가 있습니다.
- 그러다보니 교과서와 반대로 가는, 비상식적인 방법이 해결책이 될 때가 있죠.
- 그래서 요즘엔 시스템 구축에 정해진 정답 같은게 없어졌습니다.
- 대신 개발자의 노하우가 훨씬 더 중요해지게 되었습니다.

데이터 특성에 따른 DB 선택

- 가장 원시적인 database
 - Log files
 - File system
 - Key-value pairs (e.g.: Redis)

Storage Access Speed

1. CPU register : fastest but most expensive

- L1 / L2 cache

2. Memory

- Apple Silicon: CPU 와 메모리를 칩 하나로 통합 (along with graphic processor and neural engine)

3. HDD / SSD : cheapest but slowest

- 물리적인 한계 존재. 일정 시간 이상 사용하면 디스크 마모

데이터 특성에 따른 DB 선택

- SQL: 가장 일반적인 DB
 - RDBMS 기반 테이블 구조 형태
 - 1974년부터 DB 세상을 정복
- NoSQL: SQL 이 아닌 온갖 종류의 DB들
 - MongoDB: JSON 형태의 document
 - ElasticSearch: 텍스트 저장/검색에 최적화

정형 데이터 => SQL

- 데이터 구조가 엑셀 파일처럼 테이블 형태로 생겨 있습니다.
- 숫자 데이터, 또는 분류코드를 다루기에 최적화되어 있습니다.
- 관계형 DB (RDB: Relational DB) 라고 많이 부릅니다.
- MySQL, PostgreSQL, MariaDB, Amazon Aurora 등 여러 종류가 있는데, 저는 PostgreSQL 을 가장 좋아합니다.
- Oracle 은 구시대의 유물 (Legacy) 로 전락... 시스템 업그레이드 할 여력이 없는 금융권에서나 쓸까, 지금은 망했다고 봐야죠.

OLTP vs OLAP

- OLTP : Online Transaction Processing
 - 예: 지난 한달 동안 SSG 에서 주문한 내역 조회
 - Row-based storage: MySQL, PostgreSQL
- OLAP : Online Analytical Processing
 - 예: 지난달 상품 분류별 전체 매출 합계
 - Column-based storage: Google BigQuery, PostgreSQL with cstore_fdw plugin

Data Warehouse vs. Data Lake

- **Data Warehouse**

- Requires a well-defined table schema
- Example: Google BigQuery

- **Data Lake**

- Vast pool of raw data including objects
- Open-source: Apache Kafka / Spark / parquet
- Cloud-native: AWS Kinesis / Glue / Athena / Redshift

RDB 와 JSON doc 사이의 trade-off

- RDBMS (e.g.: MySQL)
 - 불필요한 redundancy 를 없애줌
 - 데이터 **일관성 (consistency)** 유지하는데 최적화!
- JSON doc (e.g.: MongoDB, Elasticsearch)
 - Join/merge 를 하지 않아도 되어서 속도가 빠름
 - 따라서 **확장성 (scalability)** 에 최적화!

SQL's Trade-Off

- SQL 이 유용한 경우
 - 데이터의 일관성이 무엇보다 중요한 경우
 - 금융 거래, 또는 상품 거래 데이터
- SQL 이 필요 없는 경우
 - 데이터 일관성이 별로 필요 없는 경우
 - 트위터, 페이스북, 인스타그램 같은 SNS

Eventual Consistency

- 데이터가 반드시 모든 사람에게 동일하게 보여야만 할까??
- 예제
 - 트럼프 트윗 => 일부 follower 는 즉시 받고, 나머지 follower 는 20-30초 뒤에 받아도 큰 문제가 안됨
 - 구글 검색 결과 => 새로운 데이터가 미국 서버에 먼저 들어가고 유럽엔 두어시간 뒤에 들어가도 별 문제 없음

Eventual Consistency

- 데이터 일관성을 eventual consistency 형태로 느슨하게 해주면 대신 확장성을 얻는게 가능!
- 이처럼 일관성보다 확장성이 더 중요할 때에는 SQL 보다 NoSQL 이 올바른 선택

NoSQL: MongoDB / DynamoDB

- JSON 형태의 document 를 저장하기 위한 DB
- Pros
 - 서버 분산을 통한 확장성!!
 - Map-reduce 를 이용한 분산 처리
- Cons
 - 데이터 일관성 유지하기 어려움
 - Merge/join 불가능. Foreign key 도 없음

GraphQL

- NoSQL 이 자체적으로 DB Schema 관리가 안되니까 대신 application layer 에서 schema 관리하는 기능 제공
- 쿼리 하나 작성하는데도 귀찮은게 너무 많음

NoSQL : 기술 부채


- 기술 부채 (Technical Debt)
 - 서비스가 오래될수록 DB 가 점점 쓰레기 더미가 되어감...
- MapReduce
 - MapReduce 를 이용한 분산처리가 20년 전에는 획기적이었지만, 지금은 별로 쓸데가 없어요.
 - MapReduce 를 처음 개발한 구글도 이제는 MapReduce 안 씁니다.
 - 구글 : SQL 기반 Spanner 라는 분산형 데이터베이스 구축

NoSQL : 잘가, 굿바이..

- 아무리 생각해도 MongoDB 같은 NoSQL 은 답이 없단 말이죠.. 🤔
 - 트위터 처럼 잠깐 쓰고 버려지는 데이터 아니면 안 쓰는걸 추천

Michael Stonebraker : DB 의 신

- <https://www.youtube.com/watch?v=KRcecxvQ>



The Meaning of Big Data - 3 V's

- Big **V**olume
 - Business stuff with simple (SQL) analytics
 - Business stuff with complex (non-SQL) analytics
- Big **V**elocity
 - Drink from a fire hose
- Big **V**ariety
 - Large number of diverse data sources to integrate

DBg Database Group
MIT Computer Science and Artificial Intelligence Lab

2

0:54 / 55:52 · Three Big Data Problems >

텍스트 데이터를 처리하기 위해선 뭐가 더 필요할까?

- NoSQL DB 는 JSON doc 형태로 데이터를 저장한다는 것을 배웠습니다.
- 하지만 이것만으로는 부족합니다. 뭐가 문제일까요?

텍스트 처리 예제

- 여기, 우리나라에서 열리는 모든 이벤트를 관리하는 DB 가 있다고 가정합니다.
- MySQL 에 테이블 형태로 입력하면 이런 형태가 될거예요.

Table: events					
event_id	date_from	date_to	title	venue	email
...			...		
101	2020/07/01	2020/07/31	강아지와 함께 하는 즐거운 반려견 생활	서울 코엑스	dog@gmail.com
102	2020/07/15	2020/09/30	고양이 장난감 & 간식 전시회	롯데월드타워	cat@gmail.com
103	2020/08/01	2020/08/12	반려동물 시대, 고양이 미용의 모든 것	여의도 IFC몰	cat_beauty@naver.com
...			...		

텍스트 처리 예제: 문제

- 이제 여기서 고양이와 관련된 이벤트를 모두 찾겠다고 하면 어떻게 하면 될까요?
- Case 1: 제목에 "고양이"가 들어가는 모든 이벤트 검색
 - Full scan 을 돌면서 모든 레코드를 다 뒤져야함. 속도가 너무 느려서 쓸 수 없음

```
select * from events where title like '%고양이%'
```

- Case 2: 제목이 "고양이"로 시작하는 모든 이벤트 검색
 - Index 를 탈 수 있어서 속도는 빠르지만, "고양이"가 중간에 나오는 이벤트는 누락

```
select * from events where title like '고양이%'
```

텍스트 처리 예제: 해결책

- 그렇다면 이 문제를 어떻게 해결할 수 있을까요?
- 이벤트 제목의 키워드를 관리하는 인덱스 테이블을 하나 더 생성하면 됩니다!!

Table: index	
keyword	event_id
간식	102
강아지	101
고양이	102
고양이	103
미용	103
반려동물	101
반려동물	103
장난감	102

Table: events					
event_id	date_from	date_to	title	venue	email
...			...		
101	2020/07/01	2020/07/31	강아지와 함께 하는 즐거운 반려견 생활	서울 코엑스	dog@gmail.com
102	2020/07/15	2020/09/30	고양이 장난감 & 간식 전시회	롯데월드타워	cat@gmail.com
103	2020/08/01	2020/08/12	반려동물 시대, 고양이 미용의 모든 것	여의도 IFC몰	cat_beauty@naver.com
...			...		

텍스트 처리 예제: 해결책

- 그리고 쿼리는 아래처럼 실행할 수 있습니다!

```
select events.*  
from events  
  inner join index  
    on events.event_id = index.event_id  
where index.keyword = '고양이'
```

- 장점
 - 인덱스를 타기 때문에 빠른 속도
 - 고양이가 제목 중간에 나와도 검색 가능
- 단점
 - 인덱스 테이블을 따로 만들어주어야 해서 번거로움 (수동 or 자동?)

텍스트 처리 예제: 대안

- 앞에서 배운대로 인덱스 테이블을 따로 만들어주면 해결이 가능합니다.
- 하지만 이벤트 레코드를 추가할 때마다 매번 인덱스도 함께 추가해주려면 너무 번거롭죠.
- 게다가 이벤트 제목은 길이가 문장 하나 밖에 안되서 그러려니 하겠는데, 신문 기사 처럼 길이가 긴 텍스트 데이터에 대해 일일이 인덱스를 만들어주려면... 🤖
- 이 작업을 대신해주는게 **ElasticSearch** 입니다!!
- ElasticSearch 가 왜 텍스트 데이터 처리에 최적화되어 있다고 하는지 이제 이해 하시겠죠?

NoSQL: ElasticSearch

- Pros
 - 텍스트 검색에 최적화
 - 분산처리 가능, 확장성 좋음
- Cons
 - 검색 엔진용으로 개발되었기 때문에, 첫 결과 1,000개만 찾을 수 있는 식으로 결과 갯수 제한
 - "전주에 위치한 모든 기업 목록 찾아줘" 는 불가능
 - 따라서 기업 이름이나 주소는 텍스트 이지만 SQL 이 더 유리

Database 분류 요약

- **SQL** : 정형 데이터, 테이블 포맷
 - MySQL, PostgreSQL 이 가장 유명
- **NoSQL** : 정형 및 비정형 데이터, **JSON doc** 포맷
 - 범용적인 NoSQL 은 MongoDB 가 가장 유명
 - 텍스트 검색에 특화된 목적으로는 Elasticsearch 가 가장 유명

Database 분류 요약

- SQL 과 NoSQL 둘 중 하나만 exclusive 하게 선택해야 한다는 의미는 절대 아닙니다!!!
- 보통 meta-data 는 SQL 에 넣고, 텍스트 본문은 NoSQL 에 넣는 식으로 둘을 조합해서 사용합니다.

참고 도서

- Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems
- https://www.amazon.com/Designing-Data-Intensive-Applications-Reliable-Maintainable-ebook/dp/B06XPJML5D/ref=sr_1_1?dchild=1&keywords=data+intensive+application&qid=1618052096&sr=8-1

