CP2. 데이터 모델과 질의 언어

Part 1. 데이터 시스템의 기초

2. 데이터 모델과 질의 언어

▼ 개요

- 데이터 모델은 아마도 소프트웨어 개발에서 제일 중요한 부분일 것이다.
  왜냐면 데이터 모델은 소프트웨어가 어떻게 작성됐는지 뿐만 아니라 해결하려는
  문제를 어떻게 생각해야 하는지에 대해서도 지대한 영향을 미치기 때문이다.
- 대부분의 애플리케이션은 하나의 데이터 모델을 다른 데이터 모델 위에 계층을 둬서 만든다. 각 계층의 핵심적인 문제는 다음 하위 계층 관점에서 **데이터 모델을 표현**하는 방법이다.
  - 애플리케이션 개발자는 현실(사람, 조직, 상품 등)을 보고 객체나 데이터 구조, 그리고 이러한 데이터 구조를 다른 API를 모델링한다. (주로 애플리케이션에 특화됨)
  - 데이터 구조를 저장할 때는 JSON 이나 XML 문서, 관계형 데이터베이스 테이블이나 그래프 모델 같은 범용 데이터 모델로 표현한다.
  - 데이터베이스 소프트웨어를 개발하는 엔지니어는 JSON/XML/관계형/그래프 데이터를 메모리나 디스크 또는 네트워크 상의 바이트 단위로 표현하는 방법을 결정한다. 이 표현은 다양한 방법으로 데이터를 질의, 탐색, 조작, 처리할 수 있게 한다.
  - 더 낮은 수준에서 하드웨어 엔지니어는 전류, 빛의 파동, 자기장 등의 관점에서 바이트를 표현하는 방법을 알아냈다.
- 복잡한 애플리케이션에서는 여러 API를 기반으로 만든 API처럼 중간 단계를 더 둘 수 있지만 기본 개념은 여전히 동일하다. 각 계층은 명확한 데이터 모델을 제공해 하위 계층의 복잡성을 숨긴다. 추상화는 다른 그룹의 사람들(DBA, 애플리케이션 개발자 등)이 효율적으로 함께 일 할 수 있게 한다.
- 다양한 유형의 데이터 모델이 있고 각 데이터 모델은 사용 방법에 대한 가정을 나타낸다. 어떤 종류의 사용법은 쉽고 어떤 동작은 지원하지 않는다. 어떤 연산은 빠르고, 어떤건 느리다.어떤 데이터 변환은 자연스럽지만 다른 어떤 데이터는 부자연스럽다.
- 하나의 데이터 모델만을 완전히 익히는 데도 많은 노력이 필요하다. 데이터 모델을 하나만 사용하고 내부 동작에 대한 걱정이 없을지라도 소프트웨어 작성은 그 자체로 충분히 어렵다. 그러나 **데이터 모델은 그 위에서 소프트웨어가 할 수 있는 일과 할 수 없는 일에 지대한 영향을 주므로 애플리케이션에 적합한 데이터 모델을 선택하는 작업이 상당히 중요하다.**

▼ 관계형 모델과 문서 모델

- 개요 & 정의
  - 가장 잘 알려진 데이터 모델은 관계형 모델을 기반으로 한 SQL 이다.
    데이터는
    (SQL - table) 관계로 구성되고 각 관계는 순서 없는 **튜플 (SQL - row)** 모음이다.
  - 관계형 데이터베이스 관리 시스템(RDBMS)과 SQL은 정규화된 구조로 데이터를 저장하고 질의할 필요가 있는 사람들 대부분이 선택하는 도구가 되었다.
  - 관계형 데이터베이스의 근원은 **비즈니스 데이터 처리**에 있다. 이 사용 사례는 보통 **트랜잭션**과 **일괄 처리**로 오늘날의 관점에서는 일상적으로 수행되는 일이다.
  - 수년 동안 데이터 저장과 질의를 위해 많은 접근 방식이 경쟁했다. 하지만 이긴건 관계형 SQL
    (70, 80 초반 - 네트워크 모델과 계층 모델)
    (80후반, 90초반 - 객체 데이터 베이스)
    (00 초반 - XML 데이터베이스)
  - 컴퓨터가 훨씬 더 강력해지고 네트워크화됨에 따라 컴퓨터 점점 더 다양한 목적으로 사용하기 시작했다. (사용사례가 비즈니스 데이터 처리를 넘어 폭 넓은 다양한 사용 사례에도 보편화됨)
- NoSQL (Not Only SQL) 등장
  - NoSQL 데이터베이스를 채택하게 된 원동력
    - 대규모 데이터셋이나 매우 높은 쓰기 처리량 달성을 관계형 데이터베이스보다 쉽게 할 수있는 뛰어난 확장성 필요
    - 상용 데이터베이스 제품보다 무료 오픈소스 소프트웨어에 대한 선호도 확산
    - 관계형 모델에서 지원하지 않는 특수 질의 동작
    - 관계형 스키마의 제한에 대한 불만과 더욱 동적이고 표현력이 풍부한 데이터 모델에 대한 바람
  - 관계형 데이터베이스가 폭넓은 다양함을 가진 비관계형 데이터 스토어와 함께 사용될 것이다. 이런 개념을 **다중 저장소 지속성** 이라 한다.
- 객체 관계형 불일치
  - 대부분의 애플리케이션은 객체지향 프로그래밍 언어로 개발한다. 이는 SQL 데이터 모델을 향한 공통된 비판을 불러온다. 데이터를 관계형 테이블에 저장하려면 애플리케이션 코드와 데이터 베이스 모델 객체 사이에 거추장스러운 전환 계층이 필요하다. 이런 모델 사이의 분리를 **임피던스 불일치**라 부른다.
    - ORM (객체 관계형 매핑) 프레임워크로 양을 줄이지만, 두 모델 간의 차이를 완벽히 숨길 수 없다.
  - 1 대 N 관계를 처리하는 다양한 방법
    - SQL 방식, 테이블 여러개와 Join
    - SQL 에 구조화된 데이터 타입과 XML, JSON 지원으로 단일 로우에 다중 값 저장, 문서 내 질의와 색인 가능해짐.
    - N 이 될 수 있는 정보를 JSON, XML 문서로 부호화해 애플리케이션 단에서 해당 구조와 내용을 해석하게 하는 방식.
  - JSON 표현은 다중 테이블 스키마보다 더 나은 **지역성**을 갖는다. 관계형 테이블 이 여러개였다면 난잡한 다중 조인을 수행해야한다.
- 1 대 N과 N 대 M 관계
  - ID, Key를 사용하는 이유

        - ID 사용시 사람에게 의미 있는 정보는 한 곳에만 저장하고 그것을 참조하는 모든 것을 ID를 사용한다. (DB 내에서만 의미 있음)
        - 텍스트를 직접 저장한다면 그것을 사용하는 모든 레코드에서 사람을 의미하는 정보를 중복해서 저장하게 된다.
        - ID 자체는 아무런 의미가 없음으로 변경할 필요가 없다. 즉 식별 정보를 변경해도 ID는 동일하게 유지할 수 있다. 하지만 의미를 갖는 경우 미래에는 언젠간 변경해야 할 수도 있다. 정보가 중복되어 있으면 쓰기 오버헤드와 불일치 위험이 있다. 이런 중복을 제거하는 일이 **데이터베이스의 정규화** 핵심 개념이다. (중복된 데이터를 정규화하려면 다대일 관계가 필요)
    - 1 대 N 관계 - ex) 계정, 캐릭터
        - 계정 기준으로 여러개의 캐릭터를 둘 수 있다.
        - 캐릭터 기준으로 계정은 하나의 계정이다.
    - N 대 M 관계 - ex) 강의, 학생
        - 학생 기준으로 강의를 여러개 들을 수 있다.
        - 강의 기준으로 여러 학생을 수강 할 수 있다.
- 문서 데이터베이스(JSON 이라 생각하자)는 역사를 반복하고 있나?
    - IMS (70년도 비즈니스 데이터 처리 유망주 데이터베이스) - **계층 모델**
        - 문서 데이터베이스에서 사용하는 JSON 모델과 비슷하다.
        - 문서 데이터베이스 처럼 1 대 N 관계에서는 잘 동작한다.
          하지만 N 대 M 관계 표현은 어렵고 조인은 지원하지 않았다.
        - 데이터를 중복할지 한 레코드와 또 다른 레코드의 참조를 수동으로 해결할지 결정해야 했다.
        - 해당 한계를 해결하기 위한 해결책
            - **관계형 모델 (SQL)**
            - **네트워크 모델**
    - 네트워크 모델 (코다실 - CODASYL)
        - 계층 모델을 일반화한다.
          계층 모델의 트리 구조에서 모든 레코드는 정확하게 하나의 부모가 있다.
        - 네트워크 모델에서 레코드는 다중 부모가 있을 수 있다. (N 대 M 관계 모델링)
        - 네트워크 모델에서 레코드 간 연결은 외래 키보다는 프로그래밍 언어의 포인터와 비슷하다. 레코드에 접근하는 유일한 방법은 최상위 레코드에서부터 연속된 연결 경로를 따르는 방법이다. **(접근 경로)**
            - 가장 간단한 경우에는 접근 경로가 연결 목록의 순회와 같은 경우
              (목록의 맨 앞에서 시작해 원하는 레코드를 찾을 때까지 한 번에 하나의 레코드는 보는 방식)
            - 하지만, N 대 M 관계에는 다양한 경로가 같은 레코드로 이어질 수 있어서, 비효율 적이다. (코다실 오피셜로 n 차원 데이터 공간을 항해하는 것 같다고 인정)
    - 관계형 모델
        - 모든 데이터를 배치하고, 모든 접근 경로를 따지지 않고 조건에 일치하는 테이블 일부 또는 모든 로우를 선택해 읽을 수 있다.

        - 질의 최적화기는 질의의 어느 부분을 어떤 순서로 실행할지를 결정하고 사용할 색인을 자동으로 결정한다. 질의 최적화기가 자동으로 만들어줌으로 네트워크 모델의 접근 경로를 생각하지 않아도 된다. (PostgreSQL 에서 ANALYZE Vacuum)
    - 문서 데이터베이스와의 비교
        - 1 대 N, N 대 M 관계를 표현할 때 관계형 DB 와 문서 DB는 근본적으로 다르지 않다.
          둘 다 관련된 항목은
          고유한 식별자로 참조한다. 관계형 모델에서는 외래 키, 문서형 모델에는 문서 참조라 부른다.
- 관계형 데이터베이스와 오늘날의 문서 데이터베이스
    - 문서형 모델
        - 스키마 유연성, 지역성에 기인한 더 나은 성능
          + 일부 애플리케이션의 경우 문서형 모델과 데이터 구조가 더 가까움
        - **읽기 스키마 (데이터 구조는 암묵적이고 데이터를 읽을 때만 해석)**
          =
          스키마 유연성
            - 유효성 검사를 하지 않음으로 임의의 키 값을 추가 할 수 있고, 문서에 포함된 필드이 존재 여부를 보장하지 않는다
        - 데이터 지역성
            - 웹 페이지에서 구현시 local storage 를 활용하면 구조적 성능 이점이 있다.
              관계형인 경우 다중 테이블에 전체를 검색하기 위해 복잡성, 시간이 더 필요하다.
        - 애플리케이션에서 데이터가 문서와 비슷한 구조라면 문서형이 좋다,
          만약 관계형이라면 여러 테이블을 나눠서 불필요한 코드 작성이 추가된다.
        - 문서 모델에는 제한이 있다.
          원하는 데이터가 있어도 문서의 깊이가 깊어질 수록 참조 포인트가 늘어난다.
        - 미흡한 조인 지원은 애플리케이션에 따라 문제일 수도 아닐 수도 있다.

          하지만, 애플리케이션을 개발, 유지보수 하다 보면 발생할 수 밖에 없지 않나?
    - 관계형 모델
        - 조인, 1 대 N, N 대 M 을 더 잘 지원함
        - **쓰기 스키마 (스키마는 명시적이고 DB는 모든 데이터가 스키마를 따른다)**
            - 스키마 변경은 느리고 중단시간을 요구함으로 좋지 않다.
    - 통합 (PostgreSQL)
        - 대부분의 RDMS 는 00년도 이후로 XML, JSON 을 지원한다.
          해당 문서의 내부를 색인하고 질의하는 기능을 포함한다.
        - 각 데이터 모델이 서로 부족한 부분을 보완해나가고 있다.

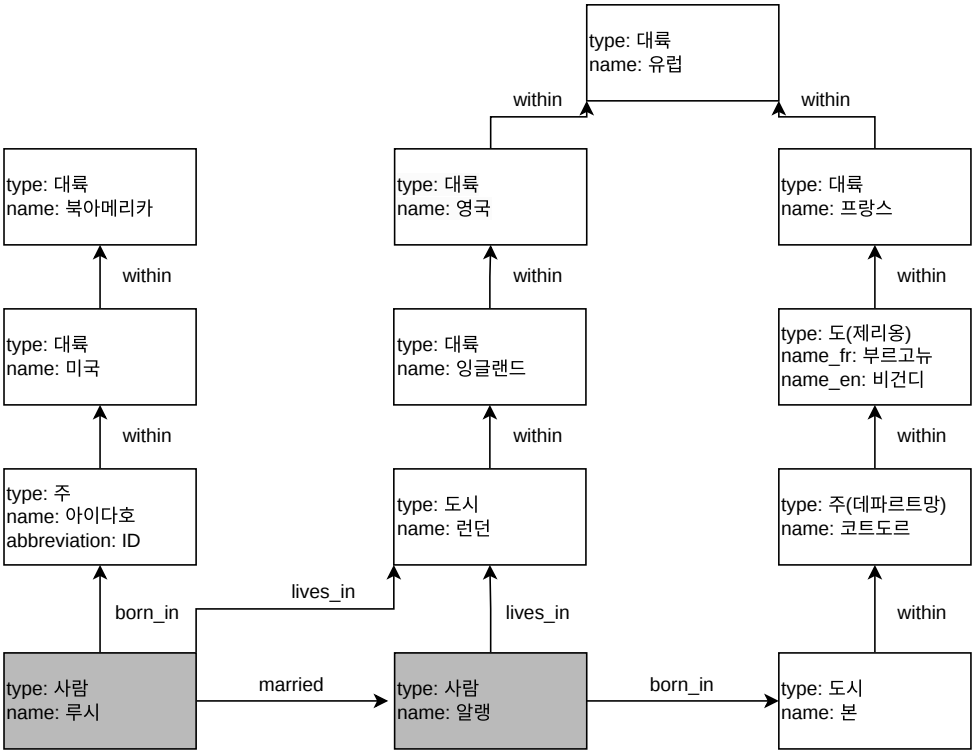
▼ 데이터를 위한 질의 언어

- **선언형** 질의(코드)
    - SQL
    - 관계 대수 구조를 유사하게 따랐음
    - 방법이 아닌 알고자하는 **데이터 패턴**, (결과가 충족해야하는 조건, 데이터의 변환, 집계)

    - 명령형 질의 보다 간결, 쉽게 작성 가능, **엔진의 구현이 숨겨져 있어 질의를 변경하지 않아도 해당 질의 성능 향상 가능**
    - 병렬 실행에 적합하다.
    - 선언형 질의는 DB 에만 국한되지 않고 웹의 CSS 선택자 로도 사용 가능하다.
        - CSS 선택시 패턴이 존재하기 때문
- **명령형** 질의(코드) (일반적인 프로그래밍 언어)
    - 코다실, IMS (계층, 네트워크 구조)
    - 특정 순서로 특정 연산을 수행하게끔 컴퓨터에게 지시한다.
    - 명령어를 특정 순서로 수행하게 끔 지정하기 때문에, 병렬처리가 어려움
- **맵리듀스(MapReduce)** 질의(코드)
    - 많은 컴퓨터에서 대량의 데이터를 처리하기 위한 프로그래밍 모델.
    - 몽고DB, 카우치DB를 포함한 일부 NoSQL 데이터 저장소는 제한 형태의 맵리듀스를 지원함.
        - 많은 문서를 대상으로 읽기 전용(read-only) 질의를 수행할 때 사용
    - 선언형 질의도 아닌 명령어 질의도 아닌 그 중간 정도임.
    - 여러 함수형 프로그래밍 언어에 있는 map(collect)과 reduce(fold, inject) 함수를 기반으로 함.
        - 언어마다 다르겠지만, 일반적으로 두 함수는 순수 함수[이펙티브 자바 CP. 스트림] 여야 한다.
            - 입력으로 전달된 데이터만 사용하고 추가적인 데이터베이스 질의를 수행할 수 없어야 하며 사이드 이펙트가 없어야 한다.
        - 이 두 함수를 신중하게 작성해야 하는 단점

▼ 그래프형 데이터 모델

- **그래프 데이터 모델링**
    - N 대 M 관계가 매우 일반적인 경우, 관계형, 문서형 보다 더 자연스러움
    - 구성
        - **정점** (vertex)(노드, 엔티티)
        - **간선** (edge)(관계, 호)
        - ex) sns - 정점 사람, 간선 사람 간에 서로 알고 있음
    - 그래프 알고리즘
        - 두 지점 간 최단 경로 검색
        - 노드의 가치 결정
        - 등등
    - 그림) 그래프 구조 데이터 예제



- 데이터를 구조화하고 질의하는 방법
  - 속성 그래프
    - 정점 구성 요소
      - 고유한 식별자
      - 유출 간선 집합
      - 유입 간선 집합
      - 속성 컬렉션 (key-value)
    - 간선 구성 요소
      - 고유한 식별자
      - 간선이 시작하는 정점(꼬리 정점)
      - 간선이 끝나는 정점(머리 정점)
      - 두 정점 간 관계 유형을 설명하는 레이블
      - 속성 컬렉션 (key-value)
    - 중요한 점
      - 정점은 다른 정점과 간선으로 연결된다. 특정 유형과 관련 여부를 제한하는 스키마는 없다.
      - 정점이 주어지면 정점의 유입과 유출 간선을 효율적으로 찾을 수 있고, 그래프를 순회할 수 있다.
        즉 일련의 정점을 따라 앞 뒤 방향으로 순회한다. (예시에서 index 를 생성한 이유)
      - 다른 유형의 관계에 서로 다른 레이블을 사용하면 단일 그래프에 다른 유형의 정보를 저장하면서도 데이터 모델을 깔끔하게 유지할 수 있다.
      - 간선 표현에 유연함을 제공한다.

- ex) 관계형 스키마를 사용해 속성 그래프 표현

```
CREATE TABLE vertices(
    vertex_id integer primary key,
    properties json
);

CREATE TABLE edge (
    edge_id integer primary key,
    tail_vertex integer references vertices (vertex_id),
    head_vertex integer references vertices (vertex_id),
    label text,
    properties json
);

CREATE INDEX edges_tails ON edges(tail_vertex);
CREATE INDEX edges_heads ON edges(head_vertex);
```

- **사이퍼(Cyper)**
  - 속성 그래프를 위한 선언형 질의 언어
  - ex) 위 그림 의 일부 그래프 DB로 삽입하는 사이퍼 질의

```
CREATE
    (NAmerica:Location {name:'North America', type:'contient'}),
    (USA:Locaion {name:'United States', type:'country'}),
    (Idaho:Location {name:'Idaho', type:'state'}),
    (Lucy:Person {name:'Lucy'}),
    (Idaho) -[:WITHIN]->(USA) -[:WITHIN]->(NAmerica),
    (Lucy) -[:BORN_IN]->(Idaho)
```

  - ex) 미국에서 유럽으로 이민 온 사람을 찾는 사이퍼 질의

```
MATCH
    (person) -]:BORN_IN]-> () -[:WITHIN*0..]-> (us:Location {na
    (person) -]:LIVES_IN]-> () -[:WITHIN*0..]-> (us:Location {n
RETURN person.name
```

    - person은 어떤 정점을 향하는 BORN_IN 유출 간선을 가진다.
      이 정점에서 name 속성이 ‘United States’인 Location 유형의 정점에 도달할 때까지 일련의 WITHIN 유출 간선을 따라간다.
    - 같은 person 정점은 LIVES_IN 유출 간선도 가진다.
      이 간선과 WITHIN 유출 간선을 따라가면 결국 name 속성이 ‘Europe’ 인 Location 유형의 정점에 도달하게 된다.
    - :WITHIN*0.. 은 0회 이상 WITHIN 간선을 따라가는 의미

- SQL의 그래프 질의

    - 관계형 DB 에서 그래프 데이터를 표현하는 방법
      그래프 질의에서는 찾고자 하는 정점을 찾기 전에 가변적인 여러 간선을 순회해야 함으로, 미리 조인 수를 고정할 수 없음.
    - 재귀 공통 테이블 식
        - SQL 가변 순회 경로에 대한 질의 개념
        - 사이퍼 질의에 비해 문법이 매우 어려움
- 트리플 저장소
    - 트리플 저장소 모델은 속성 그래프 모델과 거의 동등하다.
    - 구성 요소
        - 주어(subject) - 그래프의 정점
        - 서술어(predicate) - 그래프의 간선
        - 목적어(object) - 원시 데이터 타입의 값 or 그래프의 다른 정점
    - ex) 위 그림 의 일부 그래프 DB로 삽입하는 Turtle 트리플로 표현

```
@prefix : <urn:example:>

_:lucy    a :Person;   :name "Lucy";     :bornIn _:idaho.
_:idaho   a :Location; :name "Idaho";    :type "state";    :within _:
...
```

- 스파클
    - RDF 데이터 모델(XML 가 유사하다)을 사용한 트리플 저장소 질의 언어
- 초석: 데이터로그
    - 데이터로그는 서술어(주어, 목적어로만 구성)
    - rule 을 정의한 후 rule 을 실행하여 해당 데이터를 보여줌 (함수 정의 유사해보임)

▼ 정리

- 데이터 모델의 역사
    - 데이터를 하나의 큰 트리(계층 모델)로 표현하려했으나, N 대 M 관계 표현에는 트리 구조가 적합하지 않았다.
    - 이 문제를 해결하기 위해 관계형 모델이 고안되었다.
    - 최근, 관계형 모델에도 적합하지 않은 애플리케이션이 있다는 것을 발견했다.
        - 새롭게 등장한 비관계형 데이터저장소 NOSQL
            - 문서 데이터베이스
            - 그래프 데이터베이스
- 문서, 관계, 그래프 모델 모두 현재 널리 사용되고 있으며 각자의 역할에서 훌륭하다.
    - 한 모델이 다른 모델을 흉내 낼 수는 있지만, 결과는 대부분 엉망이다.
- 문서, 그래프 DB의 공통점, 일반적으로 저장할 데이터를 위한 스키마를 강제하지 않아 변화하는 요구사항에 맞춰 (일기 스키마) 애플리케이션을 쉽게 변경할 수있다.

하지만, 애플리케이션은 데이터가 특정 구조를 갖는다고 가정할 가능성이 높다. (쓰기 스키마)

- 각 데이터 모델은, 고유한 질의 언어나 프레임워크를 지원한다.
  - SQL, 맵리듀스, 몽고 DB의 집계 파이프라인, 사이퍼, 스파클 등
  - 추가적… 게놈 (DNA 분자관련) (염기 서열 유사도 검색), 전문 검색(full-text) 검색 등