EBS 기술인협회 스터디

『넷플릭스 기술 연구회』 5부

#GraphQL #Elasticsearch #SoftwareDevelopment
#FrontEndDevelopment

EBS 기술인협회 스터디
『넷플릭스 기술 연구회』

넷플릭스에서 검색 기능을 사용해 보신 분들이라면 누구나 검색 결과에 내가 보고 싶은 작품과 더불어 그와 상당히 유사한 작품들이 함께 추천되어, 끝없이 무한 스크롤 되는 것을 경험해 보신 적이 있을 텐데요. 상당히 정확한 추천·검색 알고리즘은 넷플릭스의 가장 큰 매력 중 하나이기도 합니다. 제목이나 작품 설명에도 겹치는 단어가 없는 것 같은데, 비슷한 작품들을 어떻게 이렇게 잘 찾아내는 걸까요? 넷플릭스 마케팅 기술팀에서는 2019년에 GraphQL을 이용한 검색 인덱싱 기술을 도입하여 이미 선진적인 검색 시스템을 구현했습니다.
이번 호에서는 GraphQL 내에 정의된 관계 및 스키마를 활용하여 검색 데이터베이스를 자동으로 구축하고 유지하기 위한 넷플릭스의 인덱싱 기술에 대하여 자세히 들여다보겠습니다.

GraphQL 검색 인덱싱

원본 정보

제목	GraphQL Search Indexing (GraphQL 검색 인덱싱)
URL	https://netflixtechblog.com/graphql-search-indexing-334c92e0d8d5

그림 1. 원문 QR-Code

Marketing Tech at Netflix - 넷플릭스의 마케팅 기술

넷플릭스에서는 수천 개의 쇼가 서비스되고 있고, 190개 이상의 국가에서 이용 가능하며 약 30개 언어를 지원하고 있습니다. 넷플릭스 마케팅팀의 목표는 이 모든 쇼 · 국가 · 언어에 대해 전 세계 각각의 잠재적 이용자들의 취향에 적합한 크리에이티브[1]를 제작하고 노출시켜 최고의 홍보 효과를 이끌어내는 것이었습니다. 마케팅팀은 이 프로젝트를 통해 새로운 인덱싱 기술을 도입하여 마케팅에 활용하였으며, 매달 수십억 건의 광고 노출 효과를 얻을 수 있었습니다.

그림 2. 검색 결과로 보여지는 크리에이티브들

1. 넷플릭스에서 제작한 홍보용 콘텐츠를 지칭하는 단어로 아트워크, 포스터, 예고편 등을 의미합니다.

전체 콘텐츠에 대해 새롭게 인덱싱을 수행하기 위해선 여러 서비스에 분산된 넷플릭스의 거대한 데이터들을 모으는 작업이 첫 번째입니다. GraphQL을 활용하면 이러한 집계 과정을 더 쉽게 할 수 있는데, 후술할 챕터에서 더 자세히 알아보겠습니다.

Challenges of Searching Decentralized Data - 분산된 데이터 검색에 대한 어려움

검색 결과로 단 하나의 크리에이티브를 도출해 내는 것이 가장 명확하겠지만, 문제는 넷플릭스에는 검색 대상이 될 크리에이티브가 많아도 너무 많다는 겁니다. 만약 넷플릭스가 지원하는 쇼, 언어, 국가 중 몇 가지만 한정 지어 검색 결괏값을 도출한다고 해도, 검색 대상이 5천만 개가 넘기도 합니다. 검색 성능 향상을 위해선 적절한 검색 솔루션 도입이 꼭 필요했습니다.
솔루션 도입 과정을 논하기에 앞서 먼저 넷플릭스의 서비스 구조에 대해 살펴보겠습니다. 넷플릭스 내에는 여러 가지 독립적인 서비스들이 다수 존재하며, 이 중 전체 시스템에 대한 완전한 호출 정보를 가지고 있는 서비스는 하나도 없기 때문에 통합 검색을 위해선 분산된 여러 서비스들로부터 데이터를 집계하는 과정이 필요합니다. 만약 각 서비스에서 자체 검색 데이터베이스를 운영하더라도 결국엔 전체 데이터 검색을 위한 집계 장치가 있어야 합니다.

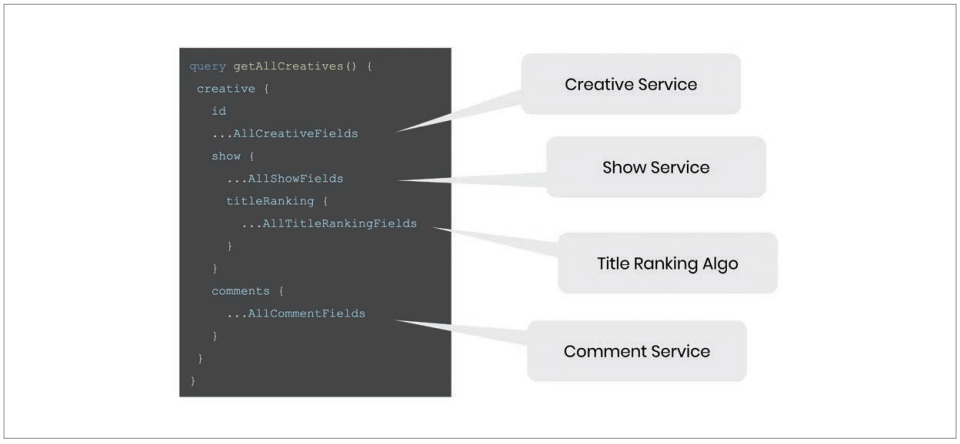

그림 3. 인덱싱하기 전의 GraphQL 쿼리로 여러 서비스가 포함됨

만약 이 서비스들에 대해 모든 데이터를 가지고 있는 하나의 거대한 데이터베이스를 운영한다면 어떨까요? 단일 DB 환경에서는 몇 개의 Join문과 where절의 조합만으로도 금방 검색 결과를 명확하게 얻어낼 수 있을 것입니다. 하지만 같은 DB를 공유하기 때문에 개별 팀들이 독립적으로 작업할 수 있는 유연성이 제한되고, 규모에 대한 성능 제한도 있습니다. 특히 넷플릭스 같이 엄청나게 방대한 데이터를 가지고 있는 시스템엔 적합하지 않죠.
다른 방법은 분산된 여러 데이터들에 대해 커스텀 집계 서비스를 운영하여 자체 인덱스를 구축하는 것입니다. 이 집계 서비스는 각 데이터가 어디에서 왔는지, 또 데이터 간에 어떻게 연결되어 있는지를 완전히 이해하고 있으며, 다양한 방식으로 데이터를 결합할 수도 있어야 합니다. 넷플릭스가 GraphQL을 이용한 검색 인덱싱 기술을 선택한 이유는 GraphQL이 앞서 설명한 특징들과 완벽하게 일치하는 특성들을 가지고 있다는 점 때문입니다. GraphQL은 DB의 종류에 상관없이 사용 가능하며, 서로 다른 데이터 소스로부터 효율적으로 필요한 데이터를 가져올 수 있게 합니다.

Indexing the Data - 전체 데이터 인덱싱하기

넷플릭스는 1년 전 이미 DB용 쿼리 언어로 GraphQL을 채택하여 성공적으로 도입을 마쳤습니다. 본 프로젝트에서는 기존에 사용하던 GraphQL 쿼리의 형식을 약간 수정하여 추가로 인덱싱에 활용하는 것이 최종 목표입니다.

```
query getCreativesById($creativeIds: [UUID]) {
 creative(creativeIds: $creativeIds) {
   id
   ...AllCreativeFields
   show {
     ...AllShowFields
     titleRanking {
       ...AllTitleRankingFields
     }
   }
   comments {
     ...AllCommentFields
   }
 }
}
```

그림 4. 인덱싱을 위한 GraphQL 쿼리

GraphQL 쿼리로 Elasticsearch에 인덱싱하는 과정은 다음과 같습니다. 수정된 GraphQL 쿼리[그림 4]를 이용해 특정 크리에이티브와 관련된 모든 데이터를 가져올 수 있도록 한 후, DB 내에 있는 전체 크리에이티브에 대하여 각각 한 번씩 단일 쿼리를 수행하였습니다. 그리고 최종적으로 이 쿼리의 결괏값들을 Elasticsearch의 스키마[2]에 매핑시켜서 인덱싱 작업을 수행하였습니다. 데이터가 인덱스 되고 난 이후에는 이 데이터에 대한 정렬·그룹화·필터링이 가능하며, 자동완성 기능을 제공할 수 있습니다. 뿐만 아니라 사용자의 디바이스에서 데이터를 점진적으로 로드시키며 무한 스크롤 경험을 제공하는 것이 가능해집니다. Elasticsearch는 많은 커스텀 옵션을 제공하지만, 대부분은 기본 설정만으로도 좋은 결괏값을 얻을 수 있었습니다. 모든 정보가 Elasticsearch에 캐싱되므로 페이지 로드 속도가 훨씬 빨라지는 것은 덤입니다.

Keeping Everything Up To Date - 데이터베이스의 인덱스를 최신 상태로 유지하는 법

위에서 기술한 대로 전체 데이터에 대해 하나하나 쿼리를 수행하여 일차적으로 인덱싱을 마쳤지만, 데이터를 한 번 인덱싱하는 것으로는 충분하지 않습니다. 실시간으로 변화하는 데이터에 따라서 인덱스 또한 항상 최신 상태를 유지해야 합니다.

> **"예를 들어 추천 알고리즘이 어떤 시점에 최신 인기 시리즈 순위를 새로고침했다면, 그에 해당하는 DB가 변경된 것이므로 그에 따라 인덱스도 변경되어야 합니다."**

넷플릭스 검색 시스템은 데이터가 변경될 때마다 즉각적으로 Kafka 이벤트[3]가 발생하도록 구성되어 있으며, 이벤트를 수신한 후 그에 맞는 대응을 수행함으로써 인덱스를 최신 상태로 유지할 수 있습니다. 인덱서가 변경 이벤트(kafka)를 감지하면 영향을 받는 모든 크리에이티브들을 찾아서 다시 인덱싱합니다. 예를 들어, 타이틀(회차)에 대한 랭킹이 변경되면 이와 관련된 쇼(시리즈)를 찾고, 해당 쇼와 연관이 있는 크리에이티브까지 전부 다시 인덱싱을 거칩니다. 하지만 특정 Kafka 이벤트와 연관이 있는 데이터들을 어떻게 발라낼 수 있을까요? 나름의 규칙을 만들어 하드코딩 해서 심는다고 해도, 데이터가 변화하고 새로운 인덱스를 구축할 때마다 이러한 규칙을 최신 상태로 유지하기란 몹시 어려울 겁니다.

2. Elasticsearch의 스키마는 데이터 타입의 집합으로, API 문서의 역할과 유사합니다.
3. Apache Kafka를 사용하여 전달되는 메시징 이벤트를 의미하며 일반적으로 데이터의 상태 변화를 추적하고 이를 다른 시스템에 알리는 데 사용됩니다.

다행히도, GraphQL의 객체-관계 모델(Entity Relationships Model/ERD)을 적용한다면 어떤 데이터를 다시 인덱싱해야 하는지 정확하게 찾을 수 있습니다. 넷플릭스의 검색 인덱서는 공유된 GraphQL 스키마에 접근함으로써 이러한 관계들을 이해하고 데이터끼리의 연관성을 파악합니다.

앞의 예시를 다시 살펴보겠습니다. 인덱서는 '인기 title 순위 변경'이라는 Kafka 이벤트가 발생하자마자 타이틀 순위와 연관되어 있는 쇼를 찾기 위해 GraphQL 쿼리를 자동으로 생성합니다. 뒤이어 타이틀과 쇼의 랭킹 데이터를 기반으로 Elasticsearch에 쿼리를 수행함으로써 해당 값들을 참조하는 크리에이티브들을 찾아냅니다. 이렇게 특정 Kafka 이벤트와 연관이 있는 데이터들을 찾고 나면 앞에서 설명한 방법을 똑같이 수행하여 다시 인덱싱합니다. 이 방법의 장점은 모든 과정은 자동으로 이루어지며, 처음에 한번 GraphQL 스키마와 리솔버 함수를 정의해주고 나면 더 이상의 추가 작업이 필요하지 않다는 겁니다.

Inverted Graph Index - 역전된 그래프 인덱스

이 챕터의 소제목에서 그래프가 등장한 이유는 GraphQL이 그래프 데이터 모델을 기반으로 하는 쿼리 언어이기 때문입니다. 우리에게 익숙한 그래프 자료구조로 설명하자면 GraphQL에서의 정점은 데이터를, 엣지는 데이터끼리의 관계를 의미합니다.

지금부터는 검색 인덱서가 수행하는 세 가지 단계인 검색, 인덱스, fan-out에 대해 예시를 통해 자세히 살펴보겠습니다. 갑자기 추천 알고리즘이 핀란드에서 쇼 <80186799>를 추천하기 시작했다면, 인덱서는 GraphQL 쿼리를 생성하여 즉시 알고리즘 데이터가 참조하는 부모 격의 쇼를 찾습니다. 해당 쇼의 부모가 <기묘한 이야기> 시리즈임을 찾아내면 Elasticsearch의 역전된 인덱스[4]를 사용하여 <기묘한 이야기>가 참조하고 있는 모든 크리에이티브를 찾아냅니다. 또한 최초의 변경 이벤트였던 추천 알고리즘이 실시간으로 참조하고 있는 모든 크리에이티브들도 함께 찾습니다. 이 크리에이티브들은 또 다른 GraphQL 쿼리에 의해 업데이트되고 Elasticsearch에도 다시 인덱싱됩니다. 이 과정이 검색 및 인덱스 단계입니다.

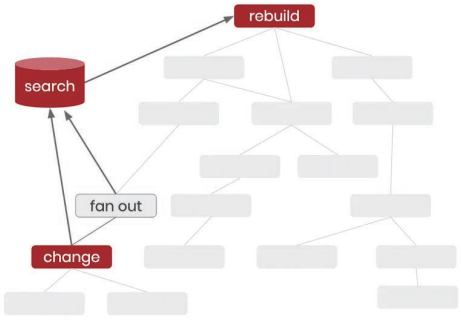

그림 5. 복잡한 그래프에서의 fan out을 이용한 검색 과정

앞의 예시로 다시 돌아가 보겠습니다. 만약 알고리즘이 핀란드에서의 <기묘한 이야기> 랭킹을 결정해야 하는데, 순위를 매길 수 있는 데이터가 충분하지 않다면 어떻게 순위를 결정해야 할까요? 이 경우엔 앞에서 설명한 검색 단계만으로는 인덱스 내에서 적절한 데이터를 찾아내는 것은 불가능합니다. 이때 fan out 단계를 수행하여 다른 데이터 소스로부터 데이터를 가져올 수 있습니다.

[그림 5]처럼 <기묘한 이야기>에 대한 데이터는 없어도 해당 정점과 연결된 인접 정점으로 이동하여 그 데이터를 순위 매길 때 활용할 수 있습니다. fan out 단계는 전체 그래프 탐색이 아닌 현재 정점의 엣지 즉 인접 접점에 대해

4. 역전된 인덱스는 특정 데이터 필드의 값을 키로 사용하여 해당 값을 포함하는 데이터를 빠르게 찾아주는 특징을 가지고 있습니다.

서만 수행하는데, 그 이유는 GraphQL 리솔버가 직계 부모의 데이터에만 의존하기 때문입니다.

인덱서는 초기 인덱스를 구축할 때 사용한 GraphQL 쿼리와 동일한 쿼리를 다시 실행하긴 하지만, 변경된 정점의 부모와 그 하위의 변경된 사항만 다시 검색하기 때문에 다시 인덱싱하는 과정에 많은 시간이 소요되지는 않습니다. 실제로 최소한의 성능 최적화로도 중간 지연 시간이 500ms 미만으로 확인되었으며, 데이터양의 증가에 따른 속도 저하도 거의 없었습니다.

Initial Setup - 초기 설정

넷플릭스는 인덱싱, GraphQL과의 통신, 그리고 변경 사항 처리와 관련된 모든 로직을 하나의 검색 인덱서 서비스에 구축했습니다. 만약 이러한 환경을 구축하고 싶다면 다음과 같은 초기 환경 셋업이 필요합니다.

변경 이벤트 알림 시스템(Kafka) : 데이터에 변경 사항이 발생하면 인덱서에 이 정보를 알려줄 수 있는 시스템이 있어야 합니다. 넷플릭스는 Kafka를 사용하여 변경 이벤트를 처리하지만, 데이터의 변경 사항을 인덱서에 알릴 수 있는 시스템이라면 어떤 것이라도 괜찮습니다.
GraphQL : 변경 이벤트에 대응하기 위해선 introspection을 지원하는 GraphQL 서버가 필요합니다. 이때 그래프에서 각 정점은 고유한 ID를 가져야 하며, 이 고유 ID를 이용하여 검색 단계에서 각 데이터를 식별합니다. 또한 그래프의 엣지가 양방향이어야 fan out이 작동할 수 있습니다.
검색 데이터베이스(Elasticsearch) : 빠른 검색을 위해 데이터는 검색 데이터베이스에 저장되어야 합니다. 넷플릭스에서는 Elasticsearch를 사용하지만, 다른 시스템을 사용해도 됩니다.
검색 인덱서 : 인덱서는 위의 세 가지 항목을 결합하는 역할을 합니다. 그래프 서버 내의 정점에 엑세스하거나 검색 데이터베이스로의 연결, 그리고 Kafka 이벤트를 그래프의 각 정점에 매핑하는 기능을 가지고 있습니다.

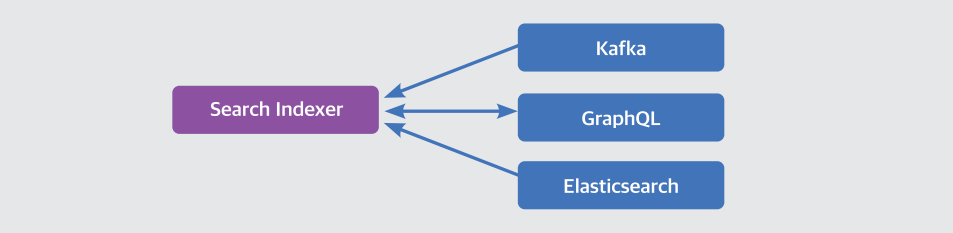

그림 6. 검색 인덱서와 시스템 간 연결

Building a New Index - 새로운 인덱스 구축

초기 환경 세팅 이후, 새로운 인덱스를 정의하고 최신 상태를 유지하는 것은 아주 쉽습니다. 그냥 인덱싱하고 싶은 데이터에 대하여 GraphQL 쿼리만 정의하면 됩니다.

<원 문>

1. GraphQL Query. We need to define the GraphQL query that retrieves the data we want to index.
2. That's it.

한번 초기 설정을 완료한 후에는 GraphQL 쿼리를 정의하기만 하면 DB 전체에 대해서 새로운 인덱스를 구축할 수 있습니다. 사용자가 필요한 만큼 많은 인덱스를 정의할 수 있으며, GraphQL 쿼리로 인덱싱하고자 하는 필드를 정의하고 나면 인덱서는 해당 필드의 데이터를 검색할 수 있습니다.

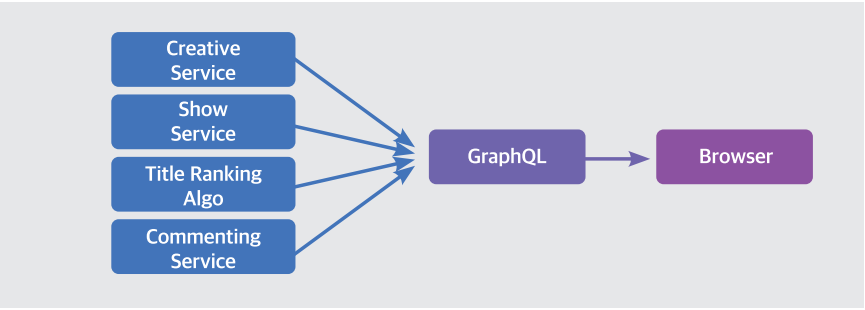

그림 7. 인덱싱하기 전 데이터 흐름

Where the Index Fits - 인덱스의 위치

인덱스 적용 전후로 데이터의 흐름과 데이터를 호출하는 주체도 달라졌습니다. 데이터를 인덱싱하기 전에는 브라우저 애플리케이션이 GraphQL 서버에 요청하여 전체 데이터를 집계하고, 클라이언트 측에서 필요한 데이터만을 필터링하는 방식이었습니다.

인덱싱 이후에는 브라우저가 Elasticsearch를 직접 호출할 수 있습니다. GraphQL 검색 인덱서의 출력데이터가 Elasticsearch 데이터베이스에 전달되므로, 클라이언트에서 검색 및 필터링을 수행할 필요가 없어지고 대신 브라우저에서 Elasticsearch가 제공하는 필터링과 같은 검색 기능을 얼마든지 활용하여 필요한 데이터만 처리 후 클라이언트에게 제공할 수 있습니다. 이 모든 과정에서 데이터는 원본 GraphQL 쿼리와 동일한 형태이므로 주요한 코드 변경 또한 필요치 않습니다.

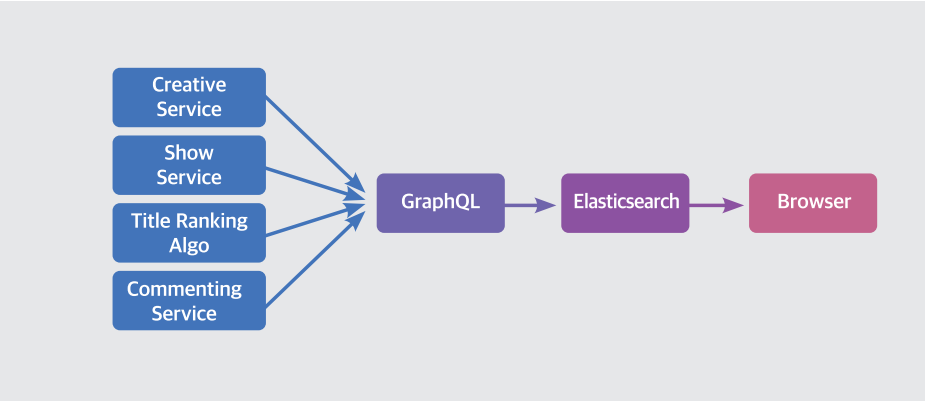

그림 8. 인덱싱 이후의 데이터 흐름

추가로, 넷플릭스는 GraphQL을 통해 Elasticsearch에 접근할 수 있도록 하는 추상화 계층을 도입하는 것도 고려하고 있습니다. 브라우저가 바로 GraphQL 서버를 호출하는 것은 기존 방식과 같지만, 이후 GraphQL의 리졸버는 검색 조건이 전달되면 Elasticsearch를 직접 호출하여 필요한 정보만 추출할 수 있습니다.

아예 검색 인덱서를 GraphQL 서버 내에 존재하는 미들웨어로 구현할 수도 있습니다. 이 방법을 이용하면 인덱싱된 데이터의 스키마를 더 좋게 향상할 수 있고, 클라이언트에서 검색 시도 시 인터럽트를 호출할 수 있습니다. 이 접근방식은 최소한의 구성만으로도 검색 기능을 일종의 '플러그인'으로 만들어 모든 GraphQL 서버에 사용할 수 있다는 장점이 있습니다.

Caveats · GraphQL 인덱싱 시스템 도입 시 주의사항

그래프 형식의 데이터베이스에 자동 인덱싱 기능을 도입함으로써 엄청나게 많은 이점들이 있지만, 다음과 같은 주의 사항을 함께 고려해야 합니다.

- 그래프 구조에서는 항상 슈퍼노드(Supernodes)[5]가 문제의 원인이 될 수 있습니다. 만약 Kafka 이벤트로 인한 인덱스 대상이 슈퍼노드를 포함할 경우, 이 슈퍼노드와 연결된 수많은 정점 또한 재인덱싱의 대상입니다. 이 경우 이 하나의 이벤트를 처리하느라 다른 Kafka 이벤트에 대한 재인덱싱 작업은 후순으로 미뤄져 정상적으로 반영되지 않는 문제가 발생합니다. 그렇기 때문에 슈퍼노드의 인덱스를 변경하는 Kafka 이벤트에 대해선 제한할 필요성이 있으며, 단일 정점에만 영향을 미치는 작은 변경 사항을 우선 처리하게끔 만들어 주어야 합니다.
- GraphQL에서 정의된 관계성, 즉 엣지는 Kafka 이벤트가 발생할 경우 재인덱싱해야 할 대상을 결정하는 데 가장 중요한 정보입니다. 그러나 감춰진 엣지(Hidden edge)[6]가 있다면 재인덱싱 대상에서 특정 정점이 누락될 수 있습니다.
- 또한, GraphQL을 사전에 도입하지 않은 상황이거나 데이터가 여러 데이터베이스에 분산되어 있지 않은 경우라면 이 시스템 말고도 빠른 검색을 수행할 수 있는 다양한 방법이 있습니다. 예를 들면 관계형 데이터베이스에서는 몇 개의 테이블 조인으로도 꽤 좋은 결과를 얻을 수 있습니다.

데이터 검색 방법은 아주 다양하며 각각은 저마다의 장단점을 가지고 있습니다. 그중에서도 넷플릭스가 검색 인덱스를 구축하고 유지하기 위해 GraphQL을 사용하는 가장 큰 이유는 유연성, 구현의 용이성 및 저렴한 유지보수 비용 때문입니다. GraphQL의 형식으로 데이터를 표현하는 것은 검색에 있어서 매우 강력한 효력을 발휘하며, 넷플릭스에서 지금까지는 상상도 하지 못했던 사례에도 적용이 가능해 엄청난 확장성을 제공하고 있습니다.

마치며

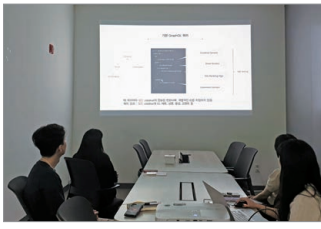

그림 9. PT 발표

최근 모든 콘텐츠 사업자들이 그러하듯 EBS 내부에서도 우리가 보유하고 있는 수십만 개의 방대한 방송 콘텐츠 중에서도 사용자가 원하는 콘텐츠를 어떻게 정확하게 찾아줄 수 있을지에 대한 논의가 끊임없이 이루어지고 있습니다. 저 또한 온라인 및 모바일 서비스 담당자로서 앞으로 방송사의 스트리밍 플랫폼이 나아가야 할 방향을 가늠코자 본 주제를 선택하기도 했습니다. 번역 작업을 진행하며 EBS의 방송 콘텐츠 DB 구조가 GraphQL 기반이 아니기 때문에 동일한 구성은 어려울 것이라는 결론에 도달했음에도 불구하고, 변경 이벤트를 이용한 자동 최신화 시스템 구축법 등 추후 시스템 개선 시 고려해 볼 만한 여러 시사점을 발견한 것은 가치 있는 경험이었습니다.
한편으로는 검색 분야의 절대강자인 넷플릭스의 인덱싱 기술 도입 과정을 자세하게 살펴보며 기술력뿐만 아니라 독립적이고 분산화된 조직 구성, 단계적 서비스 확장, 긴밀한 업무 체계 등에 놀랐습니다. 넷플릭스가 DVD 렌탈 서비스 사업자에서 선도하는 IT 기업으로 디지털 전환에 성공할 수 있었던 것은 뛰어난 기술력을 뒷받침해 주는 전사적인 IT 조직 체계가 있었기 때문입니다. 최근 방송사의 IT 역량의 중요성이 커지고 있는 만큼, 기술력 강화뿐만 아니라 조직·체계 또한 이제는 넷플릭스가 추구하는 방향성인 **Highly Aligned, Loosely Coupled(높은 일치도, 느슨한 결합도)**를 지향할 때라는 생각이 듭니다.

5. 그래프에서 아주 많은 수의 엣지를 가진 특정 정점을 의미합니다.
6. 두 정점 사이의 연관성을 완전하게 정의하지 않는 엣지를 의미합니다.