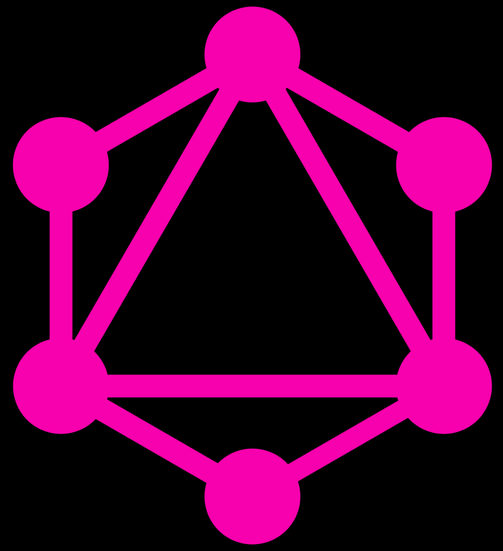
GraphQL 잘 쓰고 계신가요?

권기범, 오제관
NAVER Glace

NAVER DEVIEW 2023

GraphQL이란

NAVER
DEVIEW
2023



Query Language for API

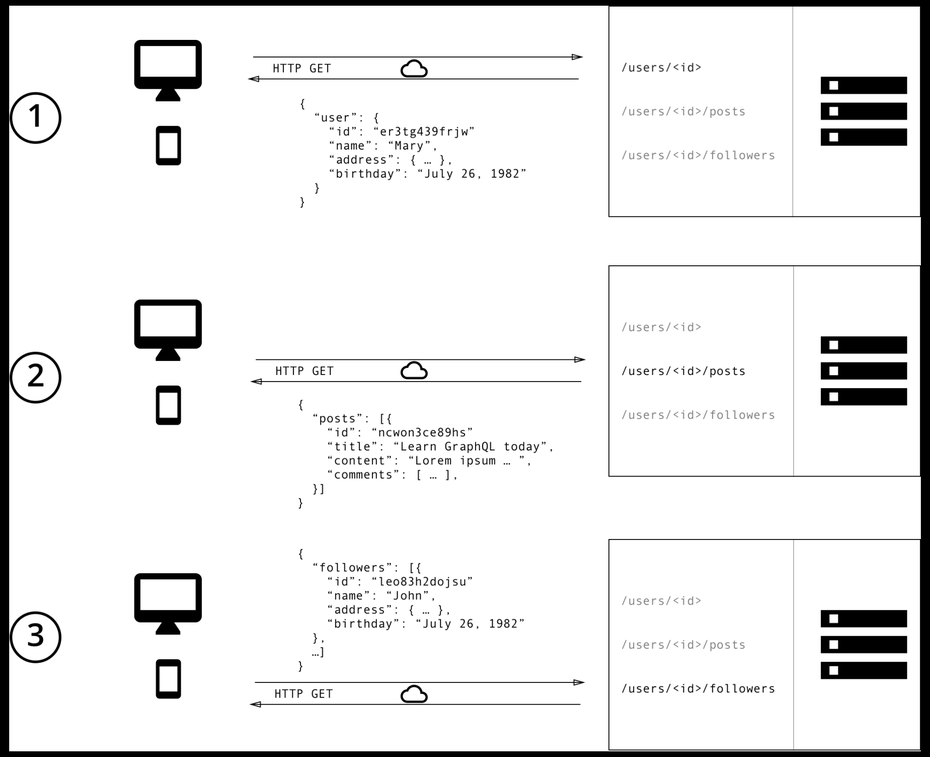
필요한 것만 요청하고 받아오기

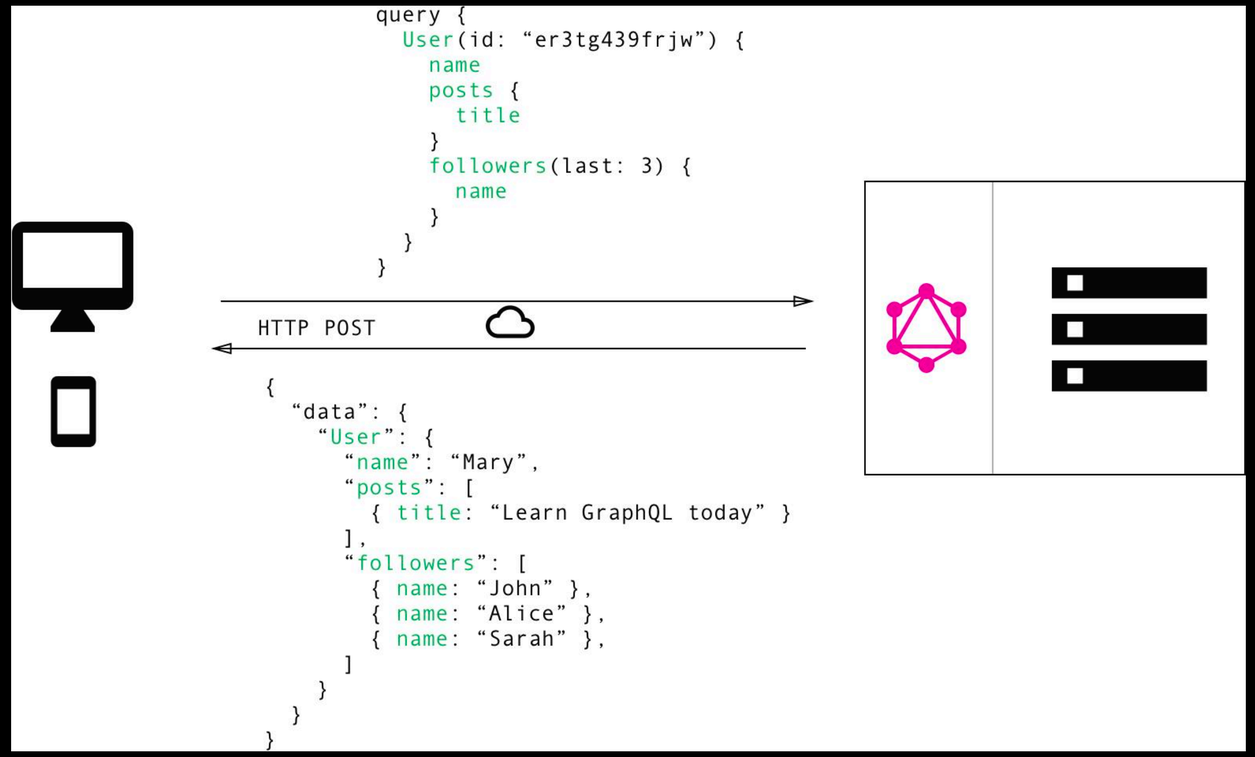
단일 요청으로 많은 데이터 가져오기

가능한 케이스를 타입 시스템으로 표현하기

GraphQL을 사용하는 목적?

NAVER
DEVIEW
2023





단순히 Payload를 줄이기 위해?

GraphQL을 이렇게 사용하고 있지 않나요?

NAVER
DEVIEW
2023

```
{
  "data": {
    "_a": 10,
    "bs": "name",
    “cd_name”: 20,
    "asd": 10,
    "FooHoo": null,
    "bob": "undefined",
    "HELL": 1000,
    "log": "no"
  }
}
```

?

```
type Response {
  _a: Int
  bs: String
  cd_name: Int
  asd: Int
  FooHoo: String
  bob: String
  HELL: Int
  log: String
}
```

REST API

GraphQL Schema ?

NAVER
DEVIEW
2023

GraphQL 잘 쓰고 계신가요?
Production-ready GraphQL

이런 분들이 들으면 좋아요

NAVER
DEVIEW
2023

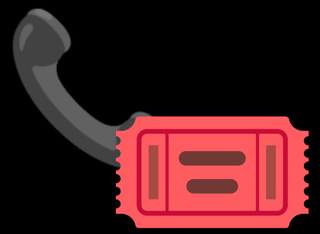
GraphQL User

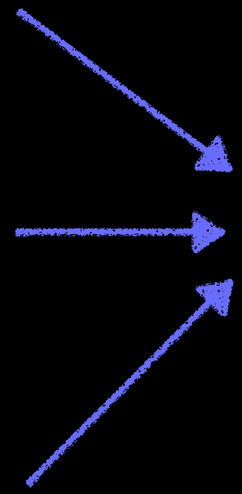
GraphQL BFF* User
(Back-End For Front-End)

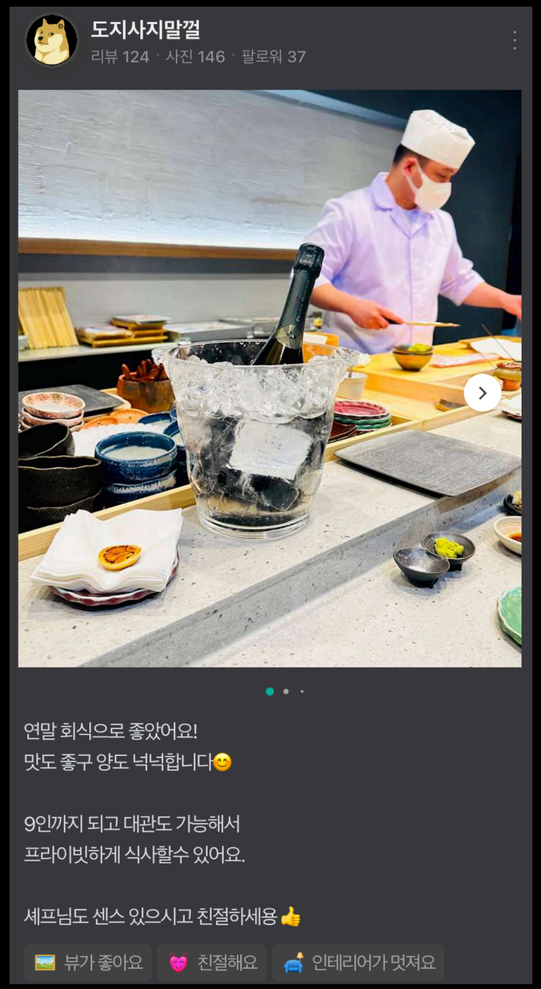
Rest API랑 비슷하던데?

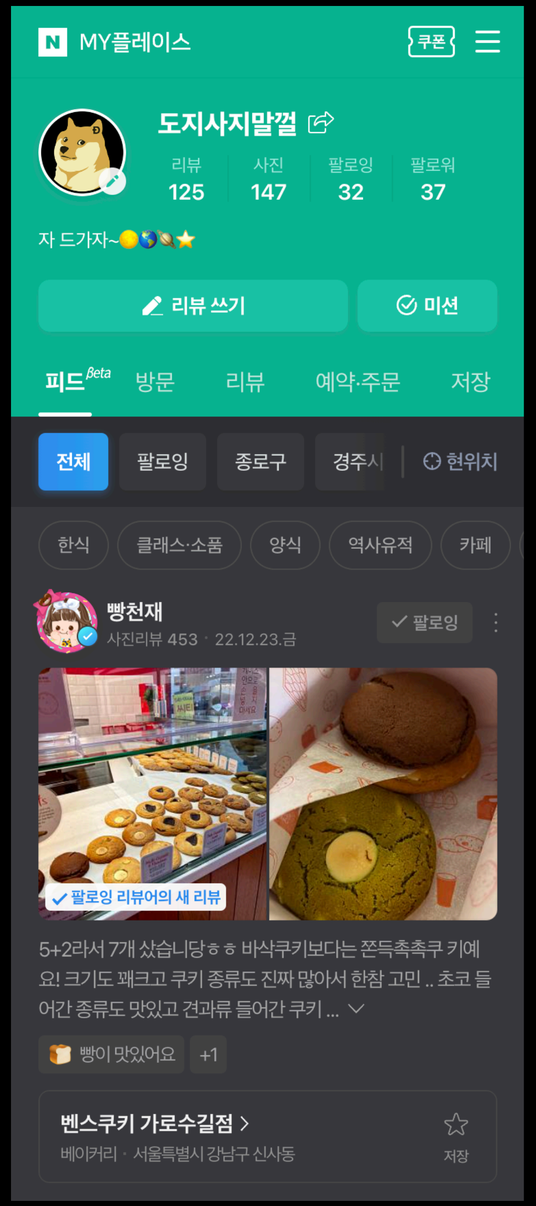
MY플레이스 소개

NAVER
DEVIEW
2023

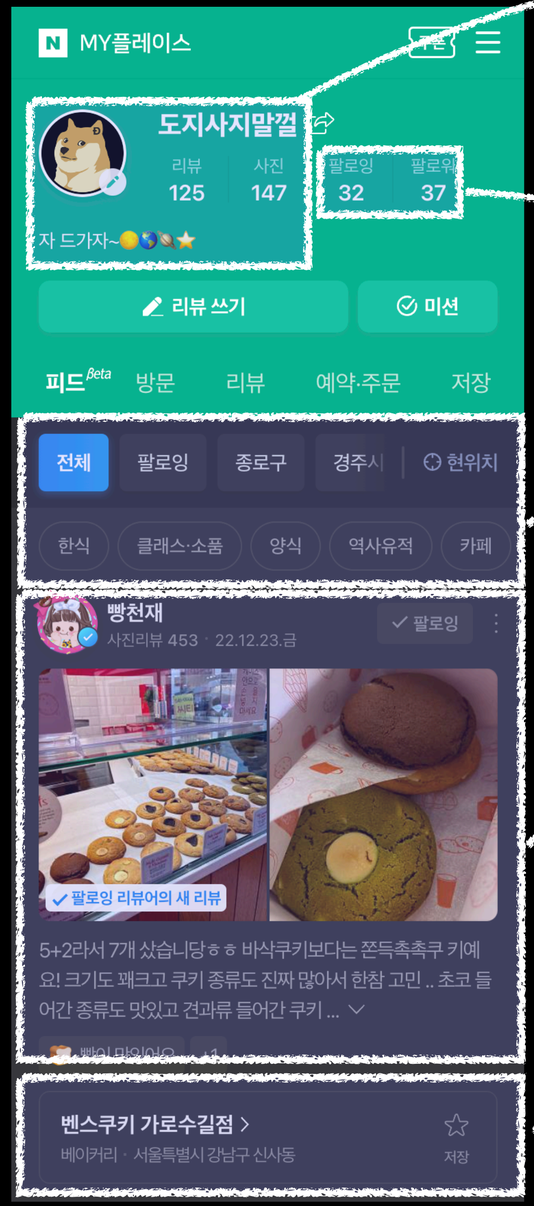


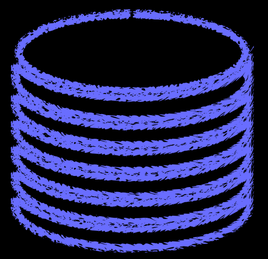






GraphQL을 사용하게 된 이유





Internal DBs

External APIs

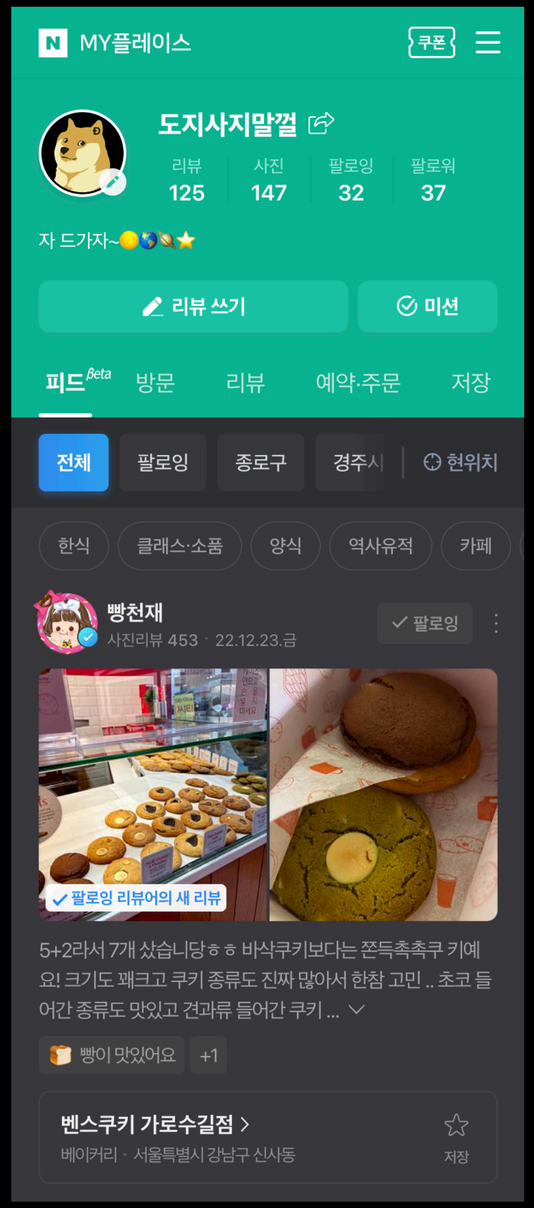
팔로잉, 피드 API

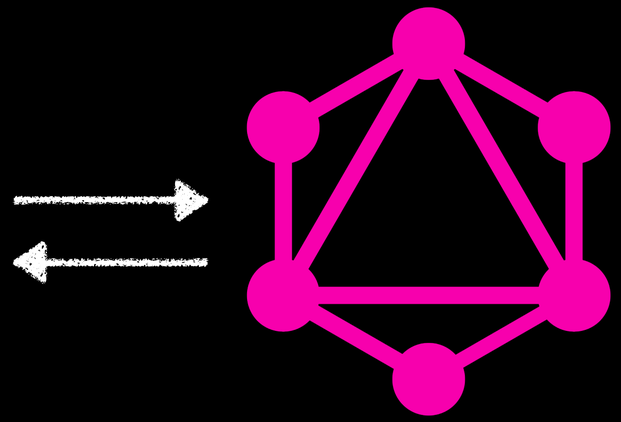
리뷰 API

장소, 업체 API

NPay (페이포인트)

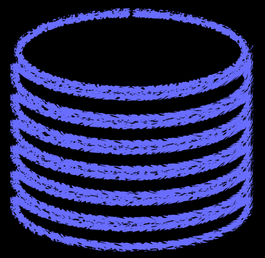
GraphQL을 사용하게 된 이유





GraphQL BFF

Internal DBs



External APIs

- 팔로잉, 피드 API
- 리뷰 API
- 장소, 업체 API
- NPay (페이포인트)

NAVER DEVIEW 2023

Content

NAVER
DEVIEW
2023

1. Schema
2. Field Argument
3. Enum
4. Error Handling
5. Custom Scalar
6. Field Resolver
7. Normalization
8. Fragment

권기범
Naver Glace, FE Developer

오제관
Naver Glace, FE Developer

What is Schema?

1. Schema - 어원

NAVER
DEVIEW
2023

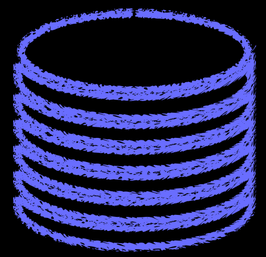
ChatGPT

The word "schema" comes from the Greek word "skhēma",
which means "shape", "form" or "figure".

우리가 다룰 데이터의 모양

1. Schema - 목적

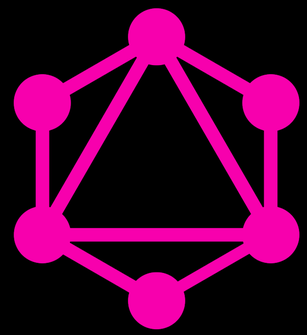
NAVER
DEVIEW
2023



Database Schema

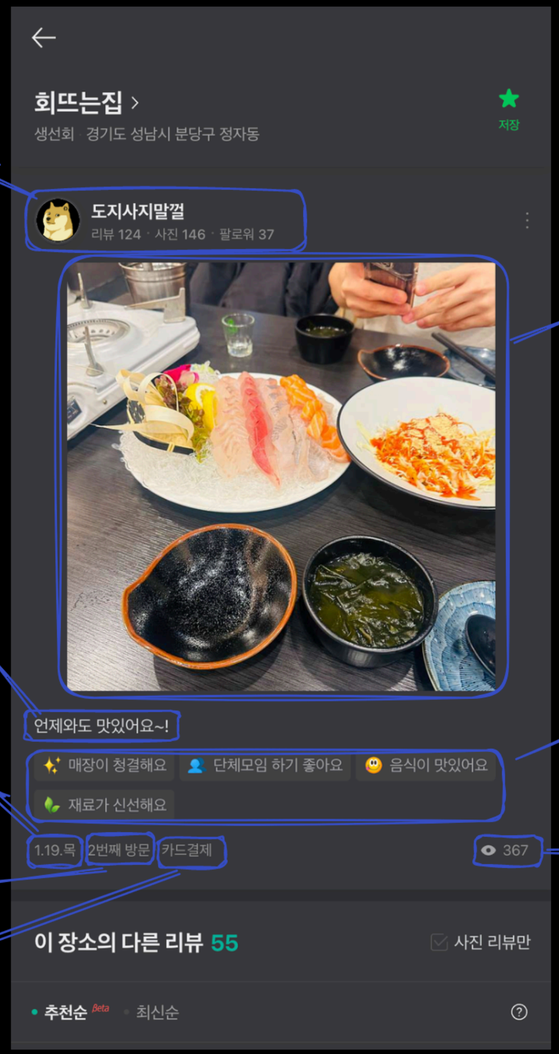
데이터를 R/W 하는데 최적화
관계와 참조로 중복 해소 (정규화)

GraphQL Schema?



1. Schema - GraphQL 스키마의 목적

NAVER
DEVIEW
2023



review.author

review.content.mediaItems

review.content.text

review.content.keywords

review.visit.createdDateTime

review.viewCount

review.visit.count

review.visit.proof

Client-Centric

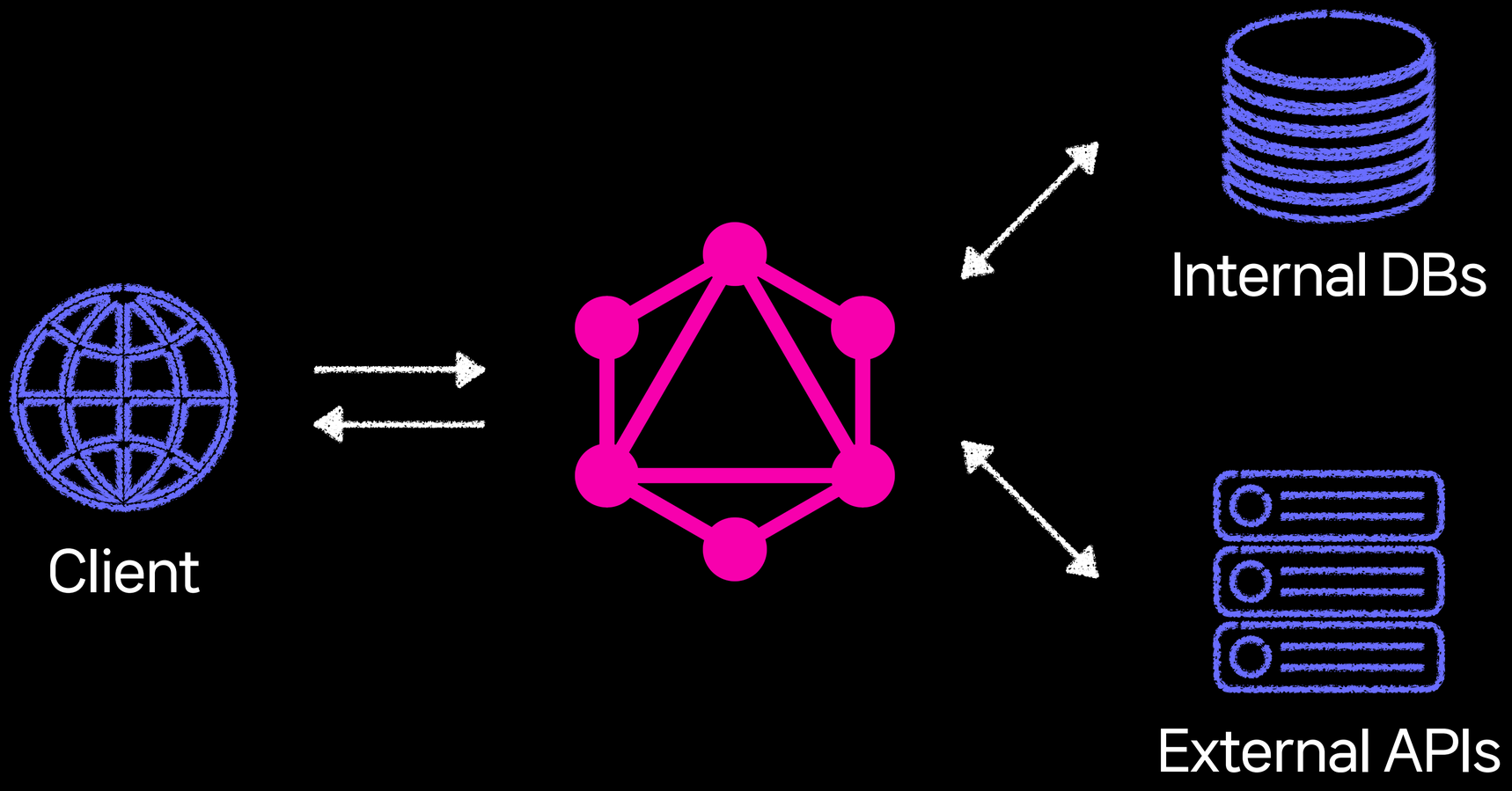
```
type ReviewContent {
  text: String!
  mediaItems: [Media!]!
  keywords: [Keyword!]!
}
```

```
type Review {
  author: User
  content: ReviewContent!
  visit: Visit!
  viewCount: Int!
}
```

```
type Visit {
  id: ID!
  createdDateTime: DateTime!
  count: Int!
  proof: Proof!
}
```

1. Schema - Datasource agnostic

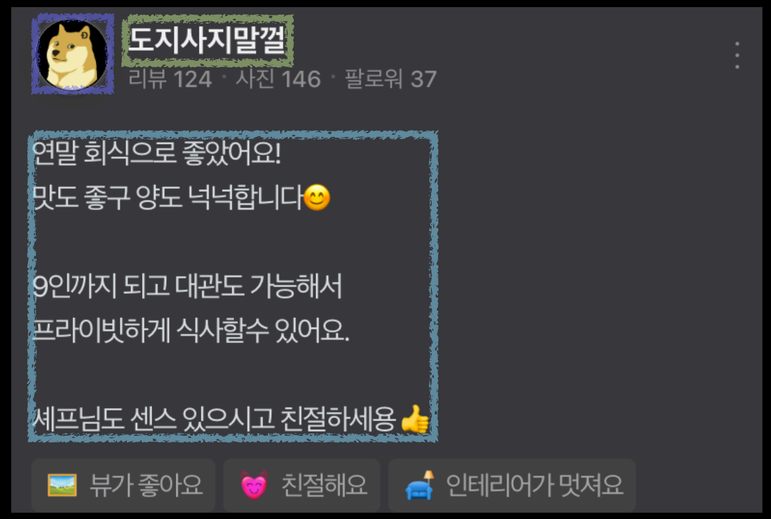
NAVER
DEVIEW
2023



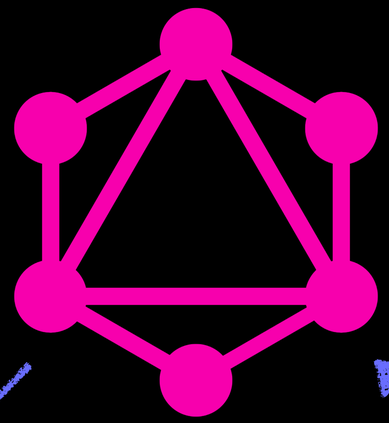
출처가 다른 데이터들의 통합 타입 시스템

1. Schema - Example

NAVER
DEVIEW
2023




```
{
  "id": "63abc6fab1d4c678e5b603e8",
  "author": {
    "id": "6358ca7805fc39ef29b7521b",
    "nickname": "도지사지말껄",
    "thumbnailUrl": "https://kiboem.kwon/1.png"
  },
  "text": "연말 회식으로 좋았어요!"
}
```

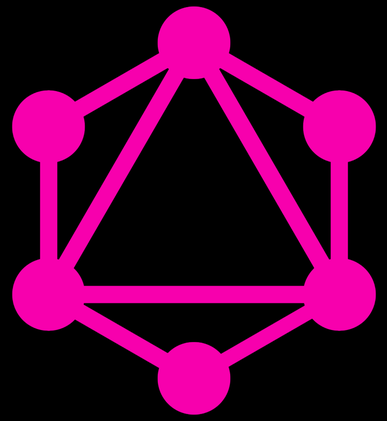


```
{
  "_id": "6358ca7805fc39ef29b7521b",
  "nickname": "도지사지말껄",
  "thumbnailUrl": "https://kiboem.kwon/1.png",
  "__v": 1
}
```

User DB

```
{
  "id": "63abc6fab1d4c678e5b603e8",
  "user_id": "6358ca7805fc39ef29b7521b",
  "text": "연말 회식으로 좋았어요!",
  "visitDate": "2023-02-27"
}
```

Review API



🥰 Client-Centric Schema Layer

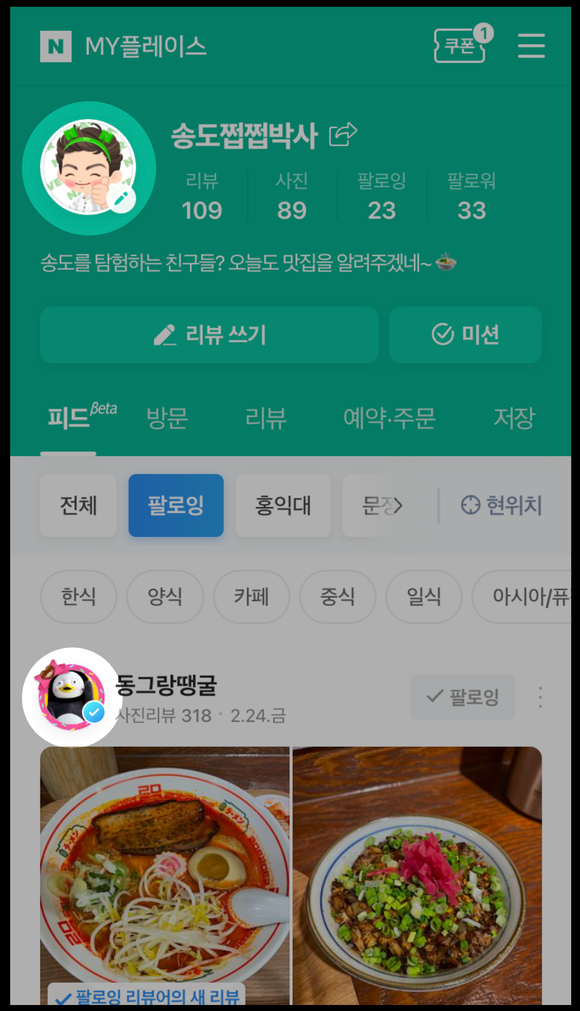
NAVER
DEVIEW
2023

Case.
컴포넌트별 다른 이미지 크기

컴포넌트별 다른 이미지 크기

NAVER
DEVIEW
2023

60px



42px

https://.../profile.png?width=100

100px

https://.../profile.png?width=50

50px

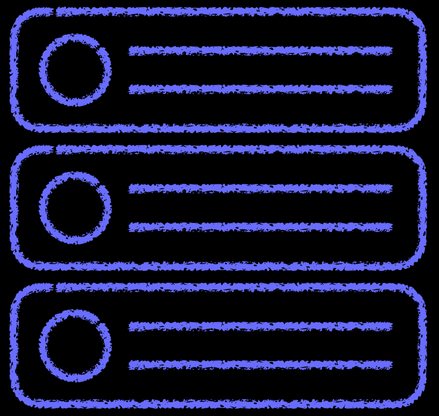
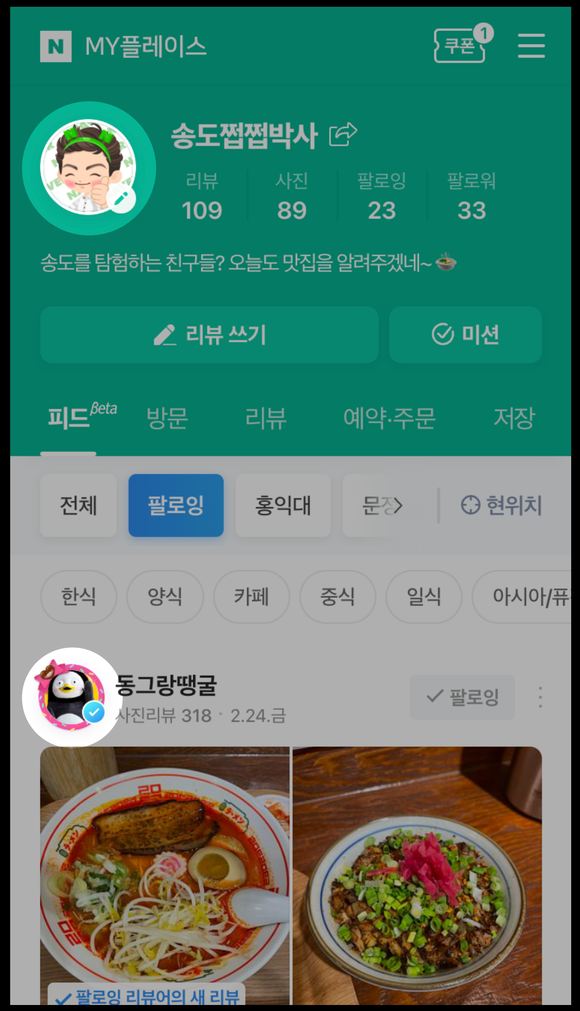


Image Server

스키마 표현한다면

NAVER
DEVIEW
2023

60px



42px

```
type Profile {
  name: String
  thumbnailUrl: String
  bigThumbnailUrl: String
  bigThumbnailUrl_100px: String
  reviewThumbnailUrl: String
  bigReviewThumbnailUrl: String
  bigReviewThumbnailUrl_200px: String
}
```

```
type Review {
  id: ID!
  nickname: String!
  images(sort: ReviewImageOrder): [Image!]!
}
```

Field Argument

2. Field Argument

NAVER
DEVIEW
2023

Client에서 필요한 사이즈를 선언할 수 있다면?

"width가 200px 인 이미지가 필요해"

```
type User {
  name: String
  image(width: Int): String
}
```

2. Field Argument

NAVER
DEVIEW
2023

Client에서 필요한 사이즈를 선언할 수 있다면?

"width가 200px 인 이미지가 필요해"

```
query {
  user(id: "1234") {
    name
    image(width: 200)
  }
}
```

NAVER
DEVIEW
2023

Case.
지원하는 이미지 사이즈를 모를 때

지원하는 이미지 사이즈를 모를 때

NAVER
DEVIEW
2023

```
type User {
  name: String
  image(width: Int): String
}
```

```
query {
  user(id: "1234") {
    image(width: 40)
  }
}
```

```
{
  "data": {
    "user": {
      "image": null
    }
  }
}
```

40px 지원 ✗

```
query {
  user(id: "1234") {
    image(width: 64)
  }
}
```

```
{
  "data": {
    "user": {
      “image”:"https://kiboem.kwon/1.png"
    }
  }
}
```

64px 지원 ✓

지원하는 사이즈를 스키마에 표현할 수는 없을까

```
enum Fruit {
  Banana
  Apple
}
```

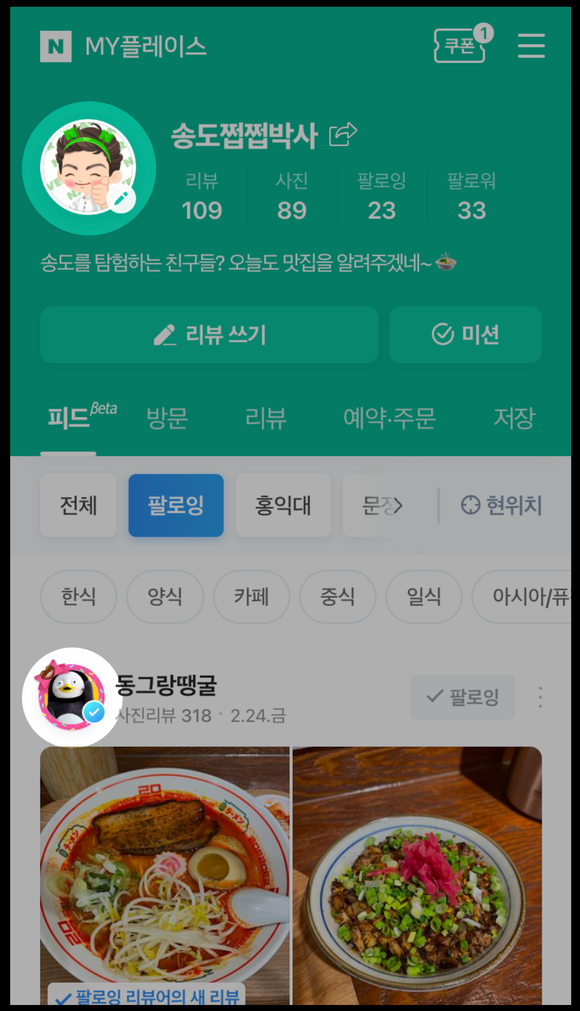
Enum

3. Enum - 지원하는 이미지 사이즈를 타입으로

NAVER
DEVIEW
2023

60px

42px




```
enum ImageWidth {
  W_42
  W_60
  W_100
}
```

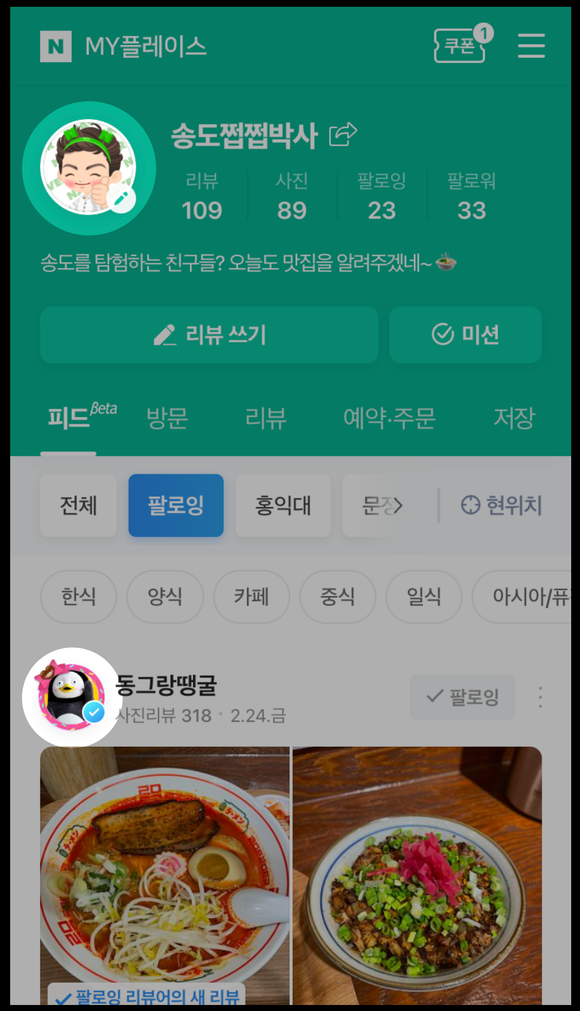
```
type User {
  name: String
  image(width: ImageWidth = W_42): String!
}
```

3. Enum - 지원하는 이미지 사이즈를 타입으로

NAVER
DEVIEW
2023

60px

42px




```
query {
  user(id: "1234") {
    name
    image(width: W_42)
  }
}
```

ImageWidth

리사이즈 지원하는 이미지 크기

Enum Values

- W_42
- 42px
- W_60
- 60px
- W_100
- 100px

3. Enum - 기존 스키마와 비교

NAVER
DEVIEW
2023

```
type Profile {
  name: String
  thumbnail42: String
  thumbnail60: String
  thumbnail100: String
  thumbnail150: String
  thumbnail200: String
  backgroundImage42: String
  backgroundImage60: String
  backgroundImage100: String
  backgroundImage150: String
  backgroundImage200: String
}
```

```
type Profile {
  name: String
  thumbnailImage(width: ImageWidth = W_42): String!
  backgroundImage(width: ImageWidth = W_100): String!
}
```

3. Enum - 기존 스키마와 비교

NAVER
DEVIEW
2023

```
type Profile {
  name: String
  thumbnail42: String
  thumbnail60: String
  thumbnail100: String
  thumbnail150: String
  thumbnail200: String
  backgroundImage42: String
  backgroundImage60: String
  backgroundImage100: String
  backgroundImage150: String
  backgroundImage200: String
}
```

```
type Profile {
  name: String
  thumbnailImage(width: ImageWidth = W_42): String!
  backgroundImage(width: ImageWidth = W_100): String!
}
```

3. Enum - 기존 스키마와 비교

NAVER
DEVIEW
2023

```
type Profile {
  name: String
  thumbnailImage(width: ImageWidth = W_42): String!
  backgroundImage(width: ImageWidth = W_100): String!
}
```

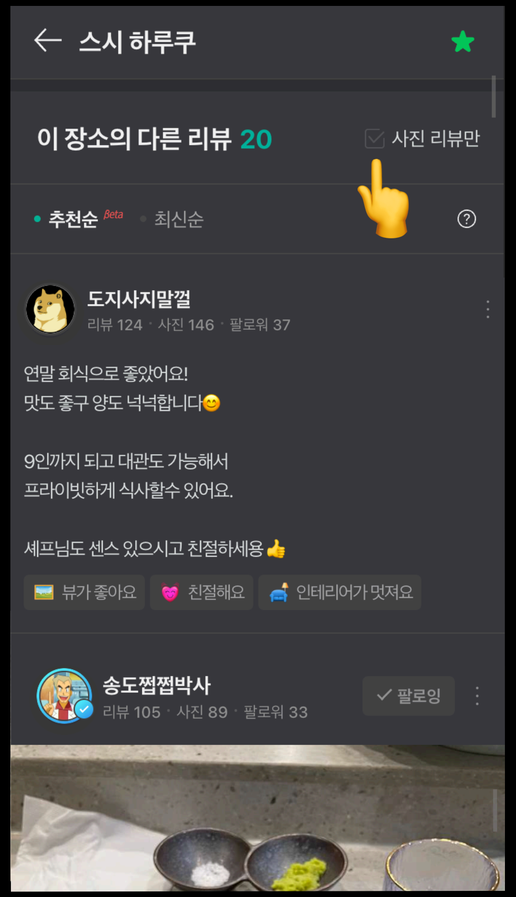
명확하게 표현함으로써 실수를 줄일 수 있습니다

NAVER
DEVIEW
2023

Case.
암시적인 API Parameter

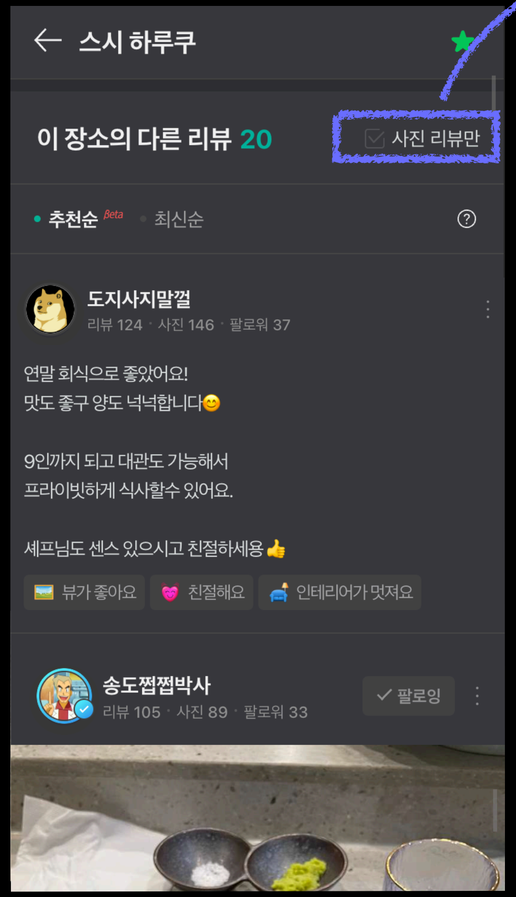
사진 리뷰만 요청할 때

NAVER
DEVIEW
2023



hasMedia 필터

NAVER DEVIEW 2023

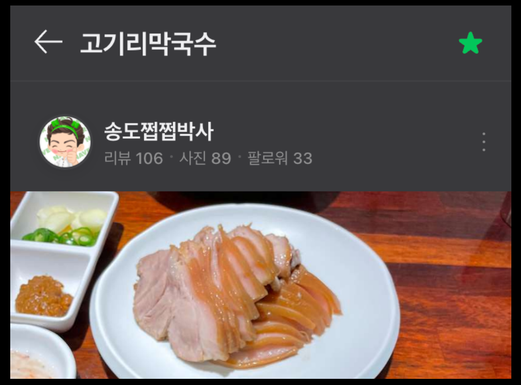


`hasMedia`	`boolean`	미디어가 포함되어 있는지 여부
`hasVideo`	`boolean`	비디오가 포함되어 있는지 여부
`hasText`	`boolean`	본문이 포함되어 있는지 여부

hasMedia

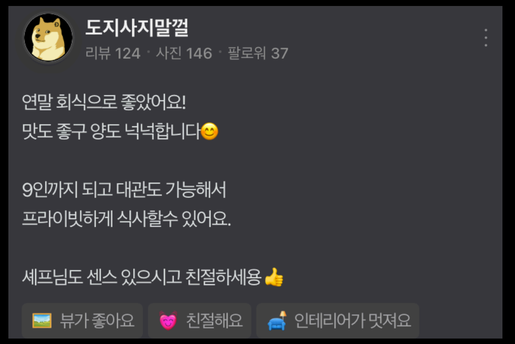
true

사진이 포함된 리뷰



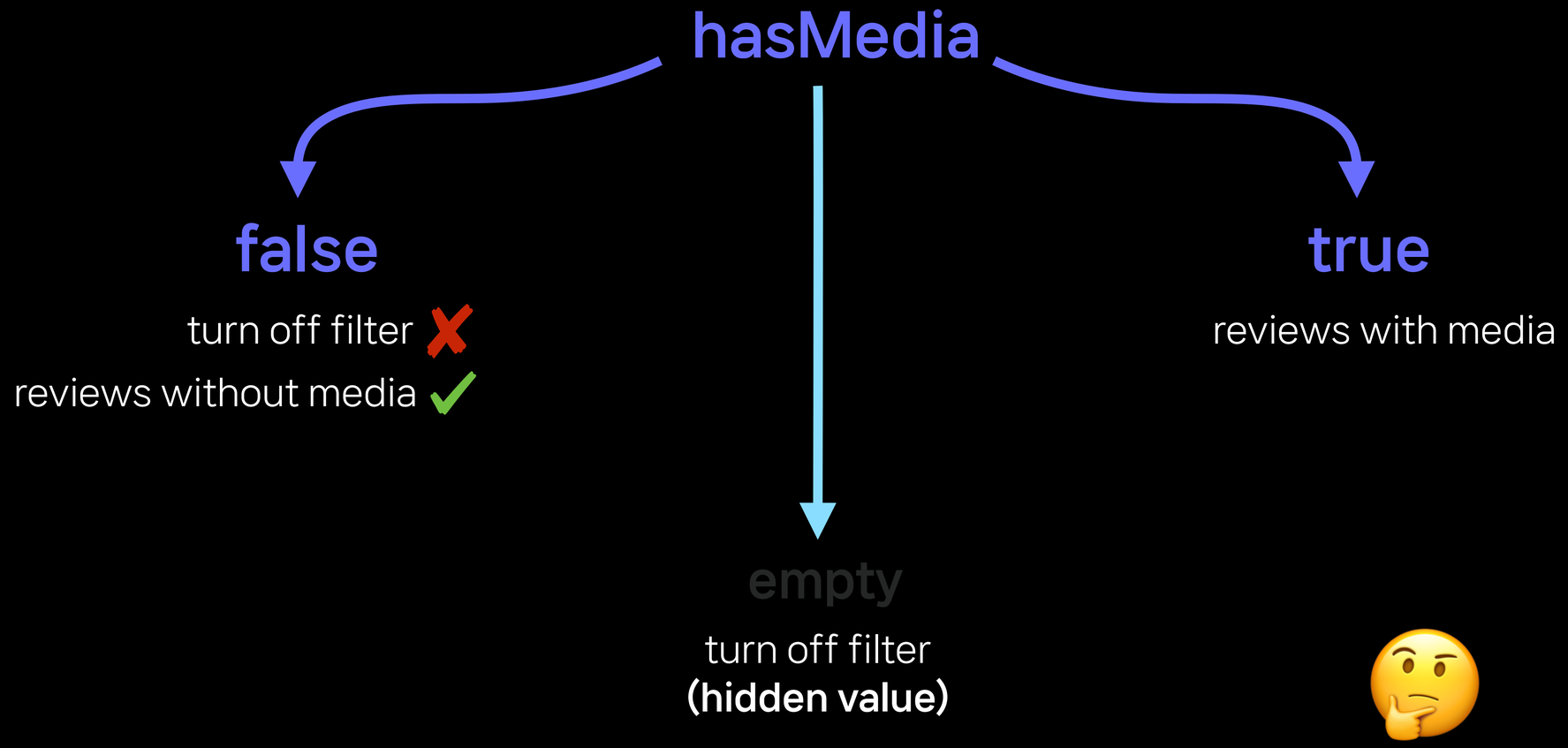
false

hasMedia 필터 제거



hasMedia 필터의 암시적 의미

NAVER
DEVIEW
2023



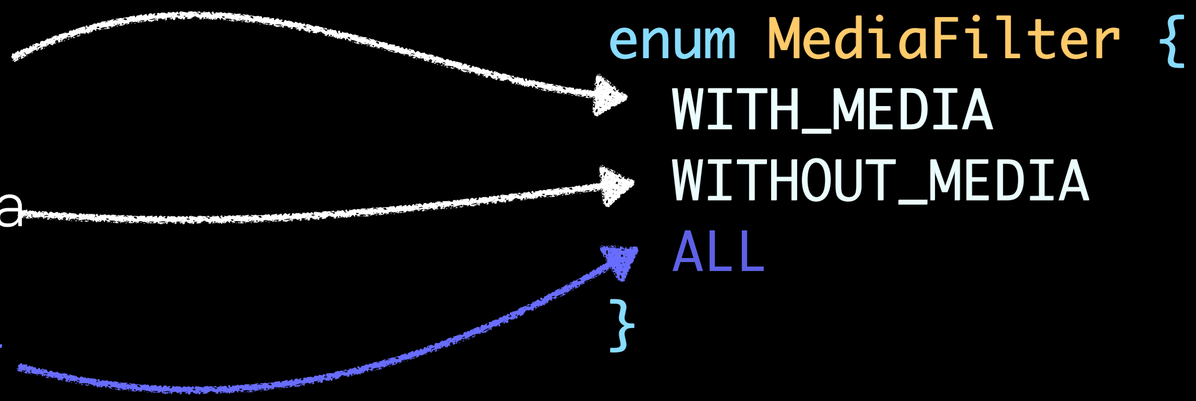
🤔

조금 더 명시적으로 드러낼 수는 없을까?

3. Enum - MediaFilter

NAVER
DEVIEW
2023

hasMedia	filter
true	with media
false	without media
empty	turn off filter



```
enum MediaFilter {
  WITH_MEDIA
  WITHOUT_MEDIA
  ALL
}
```

empty가 의미를 가질 때 Enum을 사용할 수 있습니다

3. Enum - MediaFilter

NAVER
DEVIEW
2023

```
enum MediaFilter {
  WITH_MEDIA
  WITHOUT_MEDIA
  ALL
}
```

MediaFilter

미디어 기준 리뷰 필터

(·) Enum Values

WITH_MEDIA

미디어 있는 리뷰만 필터링

WITHOUT_MEDIA

미디어 없는 리뷰만 필터링

ALL

필터링 적용하지 않음

```
type Query {
  placeReviews(placeId: ID!, mediaFilter: MediaFilter = ALL): [Review!]!
}
```

3. Enum - 사진이 있는 리뷰

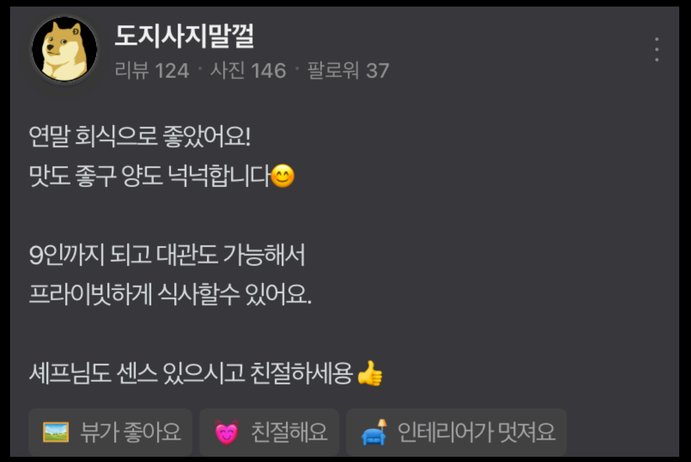
NAVER
DEVIEW
2023

```
query {
  placeReviews( placeId: "123123", mediaFilter: WITH_MEDIA ) {
    id
    mediaItems {
      id
      type
      url
    }
  }
}
```

3. Enum - 사진이 없는 리뷰

NAVER
DEVIEW
2023

```
query {
  placeReviews( placeId: "123123", mediaFilter: WITHOUT_MEDIA ) {
    id
    mediaItems {
      id
      type
      url
    }
  }
}
```

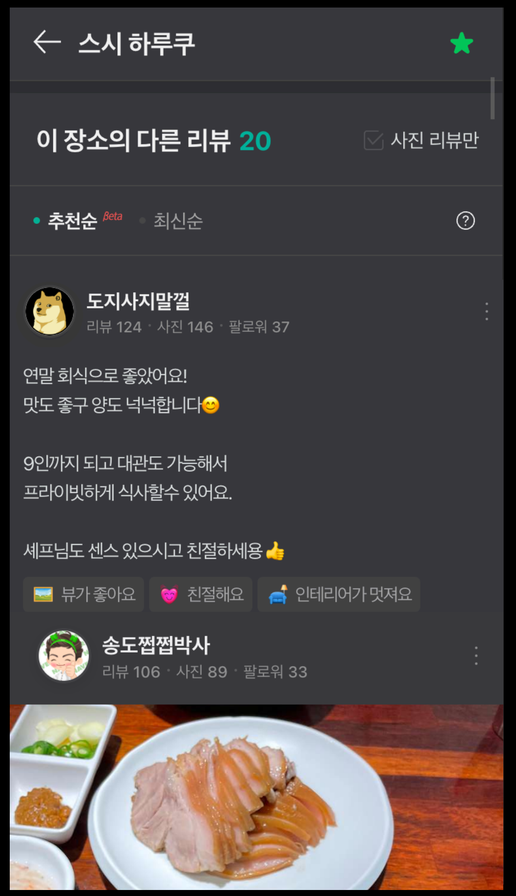


3. Enum - 필터링이 없는 모든 리뷰

NAVER
DEVIEW
2023

```
query {
  placeReviews(placeId: "123123") {
    id
    mediaItems {
      id
      type
      url
    }
  }
}
```

```
mediaFilter: MediaFilter = ALL
```



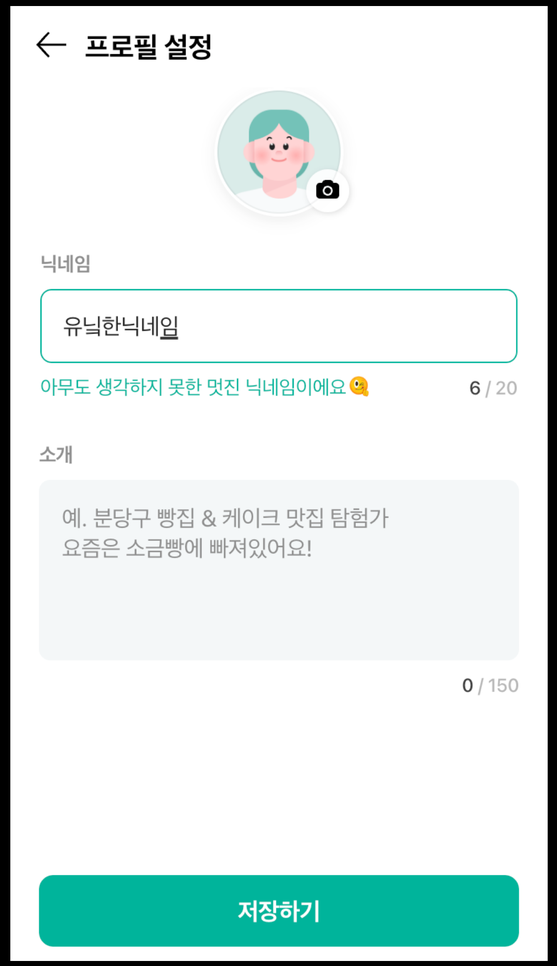
API 사용자 관점으로 명확하게 스키마 표현

NAVER
DEVIEW
2023

Case.
프로필 설정화면의 Error Handling

프로필 설정 화면

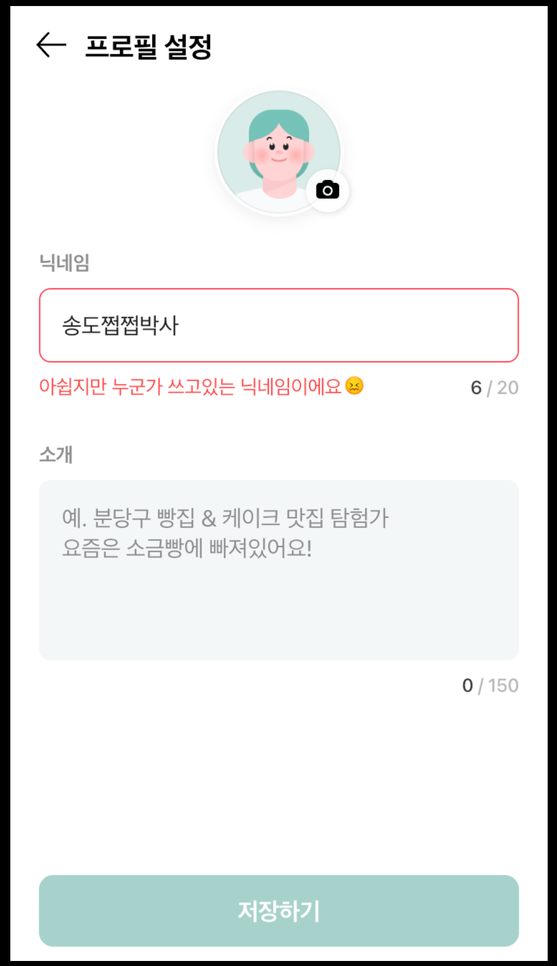
NAVER
DEVIEW
2023



닉네임 사용 가능 ✓

프로필 설정 화면 - 닉네임 중복 검사

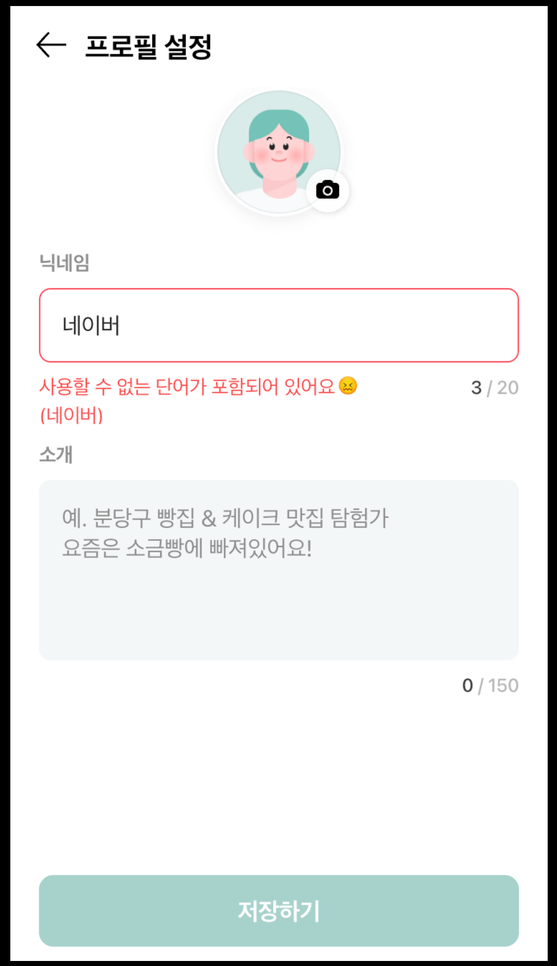
NAVER
DEVIEW
2023



닉네임 중복 ✘

프로필 설정 화면 - 금칙어 검사

NAVER
DEVIEW
2023



금칙어 사용 ✘

4. Error Handling - 상태 코드

NAVER
DEVIEW
2023

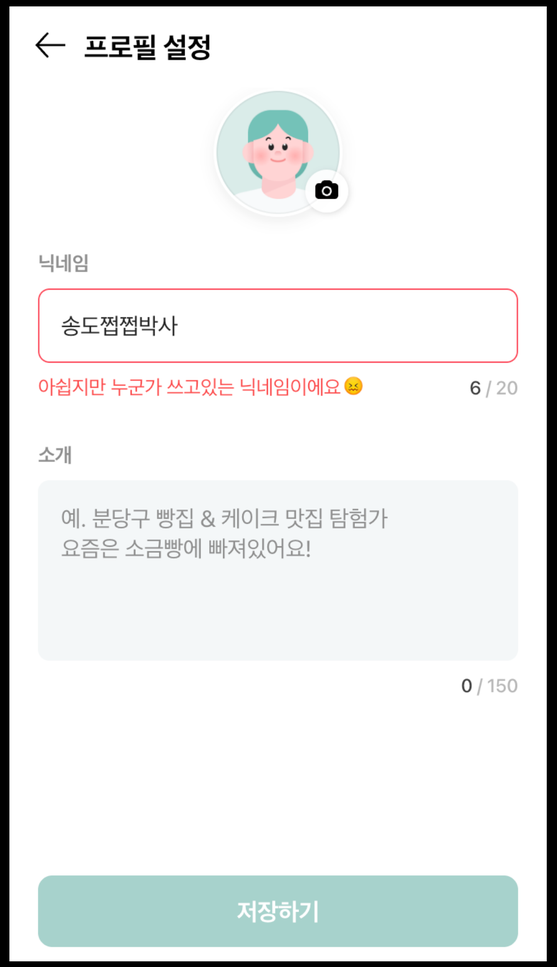
닉네임 중복 ✘

금칙어 사용 ✘

GraphQL의 모든 상태 코드는 200

4. Error Handling - Error Object

NAVER
DEVIEW
2023

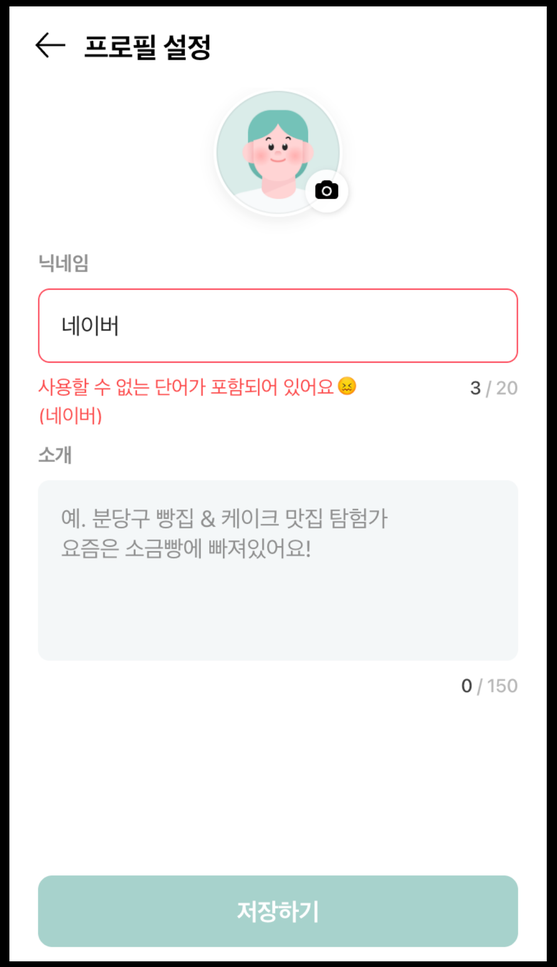



```
{
  "errors": [
    {
      "message": "닉네임이 중복입니다.",
      "extensions": {
        "code":"DuplicatedNicknameError"
      }
    },
  ]
}
```

에러 코드 관리는 어떻게 해야할까요

4. Error Handling - Error Object

NAVER
DEVIEW
2023




```
{
  "errors": [
    {
      "message": "금칙어를 포함하고 있습니다.",
      "extensions": {
        "code":"PwordError",
        "exception": {
          "words": ["네이버"]
        }
      },
    },
  ]
}
```

메타데이터가 필요할 경우 어떻게 관리해야 할까요

4. Error Handling

NAVER
DEVIEW
2023

이런 에러들을
Schema로 가져올 수는 없을까?

4. Error Handling - GraphQL Schema

NAVER
DEVIEW
2023

닉네임 중복 ✘

금칙어 사용 ✘

1일 5회 제한 ✘

```
type ValidateNicknameResult {
  success: Boolean!
  isDuplicate: Boolean!
  words: [String!]!
} isCountOver: Boolean!
  todayCount: Int!
}
```

4. Error Handling - 불가능한 상태

NAVER
DEVIEW
2023

닉네임 중복 ✘

금칙어 사용 ✘

1일 5회 제한 ✘

```
{
  "data": {
    "success": true,
    "isDuplicate": true,
    "words": ["네이버"],
    "isCountOver": true,
    "todayCount": 5
  }
}
```

분명 성공은 true 인데, 다른 데이터가 있네?

```
union Result = Succeed | Error
```

Union

에러 케이스를 유니언 타입으로

NAVER
DEVIEW
2023

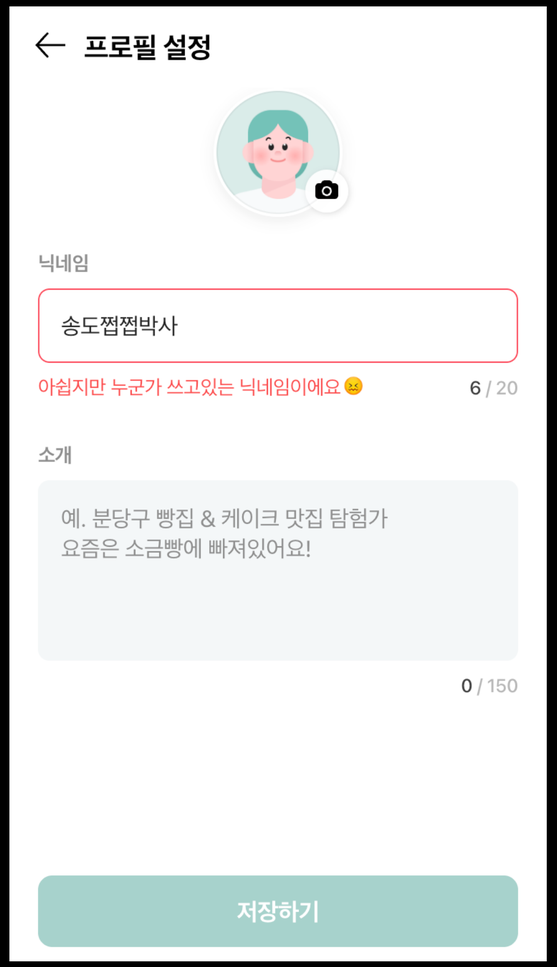
```
type DuplicatedNicknameError {
  message: String!
}

type PwordError {
  words: [String!]!
  message: String!
}
```

```
union CheckNicknameOutput = NicknameSucceed | DuplicatedNicknameError | PwordError
```

4. Error Handling - 클라이언트 쿼리

NAVER
DEVIEW
2023




```
query {
  checkNickname(nickname: "도지사지말껄") {
    __typename

    ... on NicknameSucceed {
      nickanme
    }

    ... on DuplicatedNickname {
      message
    }

    ... on PwordError {
      words
    }
  }
}
```

4. Error Handling - 데이터는 어떻게 올까요

NAVER
DEVIEW
2023

```
query {
  checkNickname(nickname: "도지사지말껄") {
    __typename

    ... on NicknameSucceed {
      nickname
    }

    ... on DuplicatedNickname {
      message
    }

    ... on PwordError {
      words
    }
  }
}
```

?

4. Error Handling - 닉네임 사용 가능할 때

NAVER
DEVIEW
2023

```
query {
  checkNickname(nickname: "도지사지말껄") {
    __typename

    ... on NicknameSucceed {
      nickname
    }

    ... on DuplicatedNickname {
      message
    }

    ... on PwordError {
      words
    }
  }
}
```

```
{
  "data": {
    "__typename": "NicknameSucceed",
    "nickname": "도지사지말껄"
  }
}
```

4. Error Handling - 닉네임이 중복일 때

NAVER
DEVIEW
2023

```
query {
  checkNickname(nickname: "도지사지말껄") {
    __typename

    ... on NicknameSucceed {
      nickanme
    }

    ... on DuplicatedNickname {
      message
    }

    ... on PwordError {
      words
    }
  }
}
```

```
{
  "data": {
    "__typename": "DuplicatedNickname",
    "message": "중복된 닉네임입니다."
  }
}
```

4. Error Handling - 금칙어가 존재할 때

NAVER
DEVIEW
2023

```
query {
  checkNickname(nickname: "도지사지말껄") {
    __typename

    ... on NicknameSucceed {
      nickanme
    }

    ... on DuplicatedNickname {
      message
    }

    ... on PwordError {
      words
    }
  }
}
```

```
{
  "data": {
    "__typename": "PwordError",
    "words": [
      "네이버"
    ]
  }
}
```

유니언 타입으로 다 해결 되었을까요?

새로운 스펙과 에러가 추가된다면

NAVER
DEVIEW
2023

```
union CheckNicknameOutput = NicknameSucceed | DuplicatedNicknameError | PwordError
```

```
type CountOverError {
  count: Int!
  message: String!
}
```

클라이언트는 알 수 있을까요?

NAVER
DEVIEW
2023

```
query {
  checkNickname(nickname: "도지사지말껄") {
    __typename

    ... on NicknameSucceed {
      nickname
    }

    ... on DuplicatedNickname {
      message
    }

    ... on PwordError {
      words
    }
  }
}
```

```
{
  "data": {
    "__typename": "CountOverError",
  }
}
```

확장에 닫혀 있네?

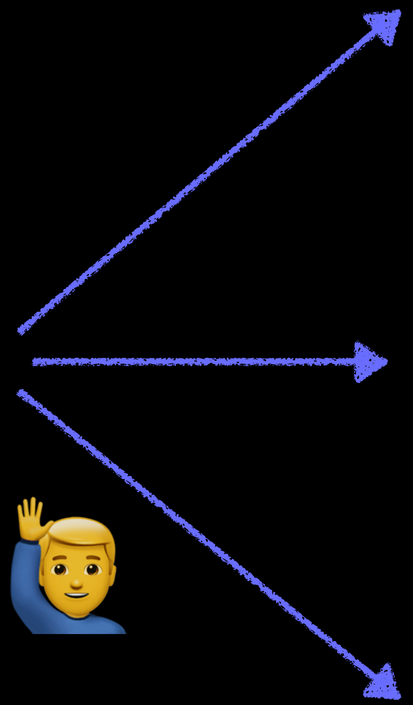
```
interface BaseError {
  message: String!
}
```

Interface

4. Error Handling - Interface 적용

NAVER
DEVIEW
2023

```
interface BaseError {
  message: String!
}
```



```
type DuplicatedNickname implements BaseError {
  message: String!
}
```

```
type PwordError implements BaseError {
  words: [String!]!
  message: String!
}
```

```
type CountOverError implements BaseError {
  count: Int!
  message: String!
}
```

4. Error Handling

NAVER
DEVIEW
2023

```
query {
  checkNickname(nickname: "도지사지말껄") {
    __typename

    ... on NicknameSucceed {
      nickanme
    }

    ... on DuplicatedNickname {
      isDuplicated
    }

    ... on PwordError {
      words
    }

    ... on BaseError {
      message
    }
  }
}
```

```
{
  "data": {
    "__typename": "CountOverError",
    "message": "일일 변경을 초과했습니다."
  }
}
```

명시적이고 유연하게 에러 핸들링

Quiz. 무슨 Type이 들어갈까요?

NAVER
DEVIEW
2023

email. jegwan.oh@navercorp.com

phone. 010-XXXX-XXXX

Hello world

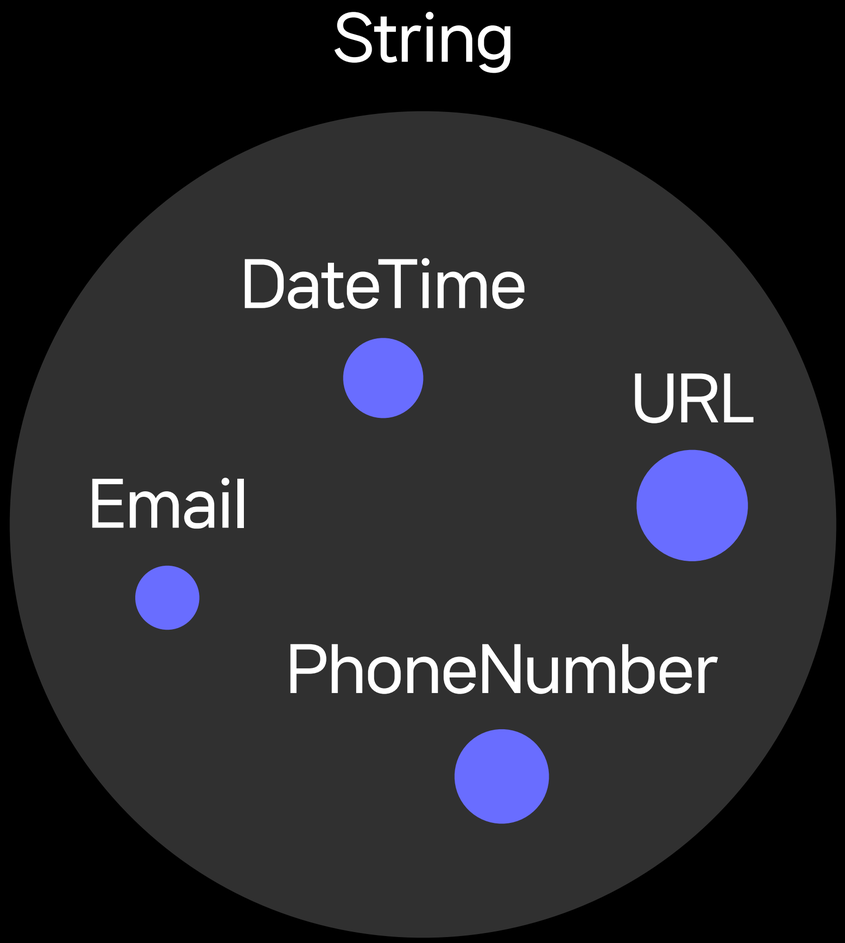
TypeScript & GraphQL Enthusiast!

```
type User {
  email: String!
  phone: String!
  thumbnailUrl: String!
  description: String!
}
```

최선입니까 휴먼?

타입을 집합으로 나타내면...

NAVER
DEVIEW
2023



Venn diagram

```
type User {
  email: Email!
  phone: String!
  thumbnailUrl: String!
  description: String!
}
```

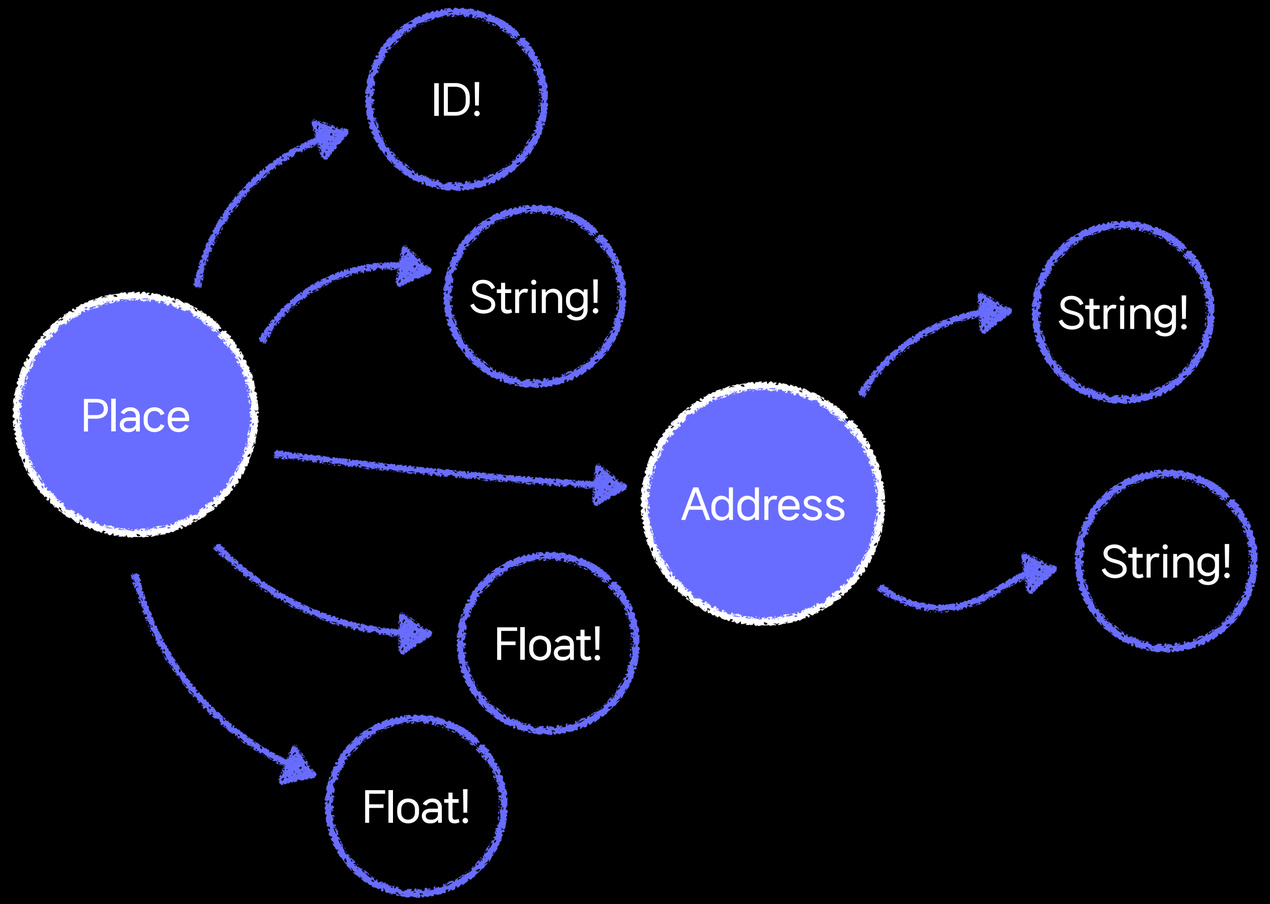
Scalar

5. Custom Scalar

NAVER
DEVIEW
2023

- 3.5 Scalars
  - 3.5.1 Int
  - 3.5.2 Float
  - 3.5.3 String
  - 3.5.4 Boolean
  - 3.5.5 ID
  - 3.5.6 Scalar Extensions

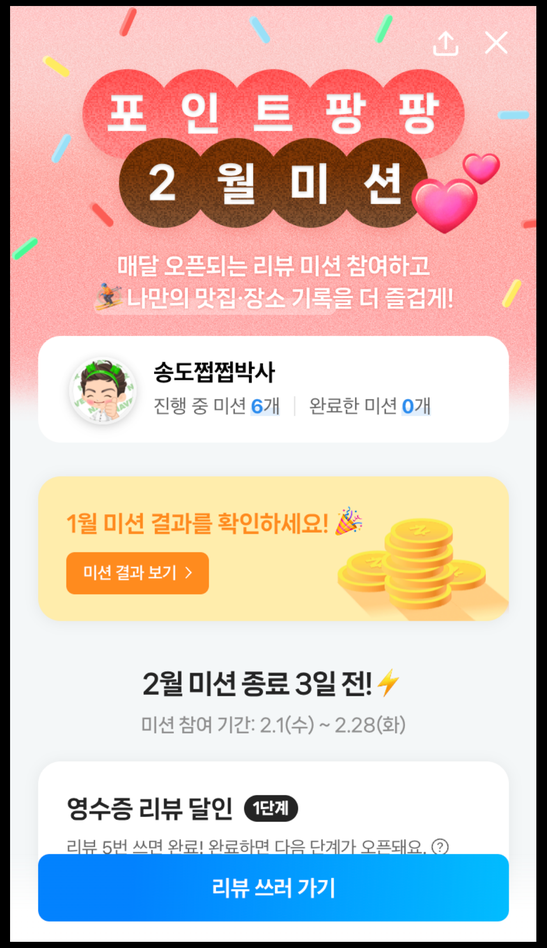
GraphQL built-in scalar types



Scalar is leaf of GraphQL type

5. Custom Scalar

NAVER
DEVIEW
2023




```graphql
query {
  missions(yearMonth:"2022-02"){
    id
    name
    # ...
  }
}
```

"YYYY-MM"
년월 형식의 문자열

5. Custom Scalar

NAVER
DEVIEW
2023

```
scalar DateTime
scalar ObjectID
```

Custom Scalar(SDL)

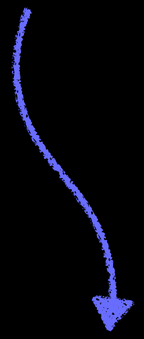
실제 구현은? (serialize, parse)

입맛에 맞게

5. Custom Scalar

NAVER
DEVIEW
2023

```
const GraphQLYearMonth = new GraphQLScalarType({
  name: 'YearMonth',
  description: '년월을 `YYYY-MM` 포맷의 스트링으로 받고 유효한 날짜인지 검증합니다.',
  serialize,
  parseValue,
})
```



```
type Query {
  missions(yearMonth: YearMonth!): [Mission!]!
}
```

5. Custom Scalar

NAVER DEVIEW 2023

```
query ($after: DateTime!) {
  …
}
```

"2022-10-01T15:00:00.000Z"
type String

method: POST
content-type: application/json

parseValue

new Date("2022-10-01T15:00:00.000Z")
type Date

serialize

"2022-10-01T15:00:00.000Z"
type String

Client

Server

5. Custom Scalar

NAVER
DEVIEW
2023

구현 : Server에서는 Dayjs, Client에서는 String으로 전달

```
const GraphQLYearMonth = new GraphQLScalarType({
  name: 'YearMonth',
  description: '년월을 `YYYY-MM` 포맷의 스트링으로 받고 유효한 날짜인지 검증합니다.',
  serialize: (value: Dayjs): string => {
    return value.toISOString()
  },
  parseValue: (value: unknown): Dayjs => {
    if (typeof value !== 'string') throw new GraphQLError('Provided YearMonth is not an string')

    if (!YEAR_MONTH_REGEX.test(value))
      throw new GraphQLError('Provided YearMonth is not "YYYY-MM" format string')

    const parsedValue = dayjs(value)
    if (!parsedValue.isValid()) throw new GraphQLError('Provided YearMonth is invalid date format')

    return parsedValue
  },
})
```

5. Custom Scalar

NAVER
DEVIEW
2023

유효하지 않은 값을 넣으면?

```
1 ▾ {
2 ▾   missionGroups(yearMonth: "202-10") {
3 ▾     ... on MissionGroupsSucceed {
4         __typename
5 ▾       missionGroups {
6           id
7           name
```

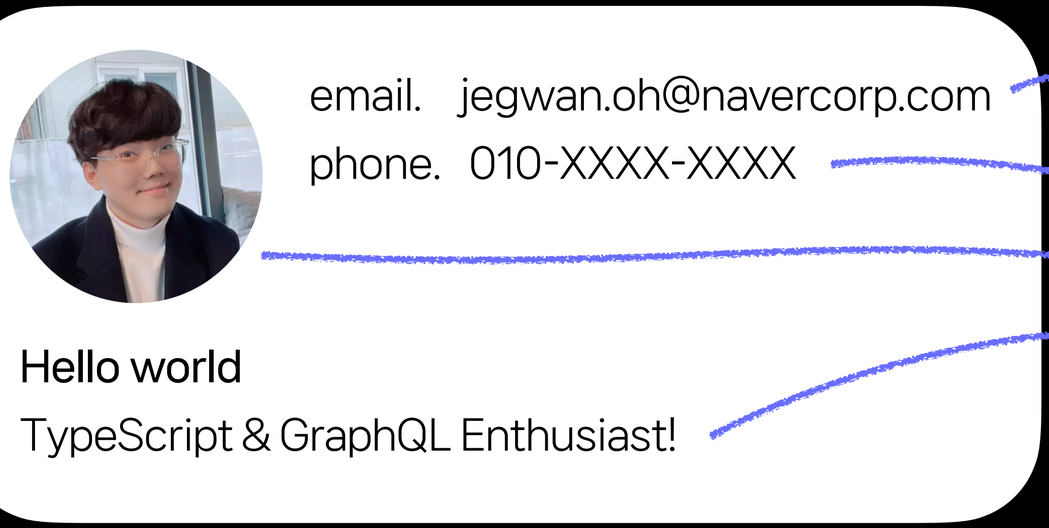
parseValue 과정에서 에러 리턴!


```
▾ {
▾   "errors": [
▾     {
        "message": "Provided YearMonth is not \"YYYY-MM\" format string",
▾       "extensions": {
          "code": "GRAPHQL_VALIDATION_FAILED",
▾         "exception": {
▾           "stacktrace": [
```


Scalar ensures type-safety by validation!

5. Custom Scalar

NAVER
DEVIEW
2023




```graphql
type User {
  email: Email!
  phone: PhoneNumber!
  thumbnailUrl: URL!
  description: Markdown!
}
```

5. Custom Scalar

NAVER
DEVIEW
2023

graphql-scalar

support URL, JWT, UUID, ObjectID...

NAVER
DEVIEW
2023

Case.
클라이언트에서 복잡한 조건을 다룰 때

리뷰 노출 조건

NAVER
DEVIEW
2023


```
const isShowable = !review.isPrivate
  && review.isQualified
  && !review.isReportedByOthers
```

클라이언트의 관심사일까?

리뷰 노출 조건

NAVER
DEVIEW
2023


```
const isShowable = !review.isPrivate
  && review.isQualified
  && !review.isReportedByOthers
```

iOS

```
let isShowable = !review.isPrivate
  && review.isQualified
  && !review.isReportedByOthers
```

android

```
val isShowable = !review.isPrivate
  && review.isQualified
  && !review.isReportedByOthers
```

NAVER
DEVIEW
2023

클라이언트 관심사?

```
const isShowable = !review.isPrivate
  && review.isQualified
  && !review.isReportedByOthers
```

What if...

```
review.isShowable
```

Field Resolver

6. Field Resolver

NAVER
DEVIEW
2023

Field resolver = Getter Function

6. Field Resolver

NAVER
DEVIEW
2023

```
@ObjectType()
class TestUser {
  @Field(() => ID)
  id: string

  @Field()
  firstName: string

  @Field()
  lastName: string
}
```

nestjs

type-graphql

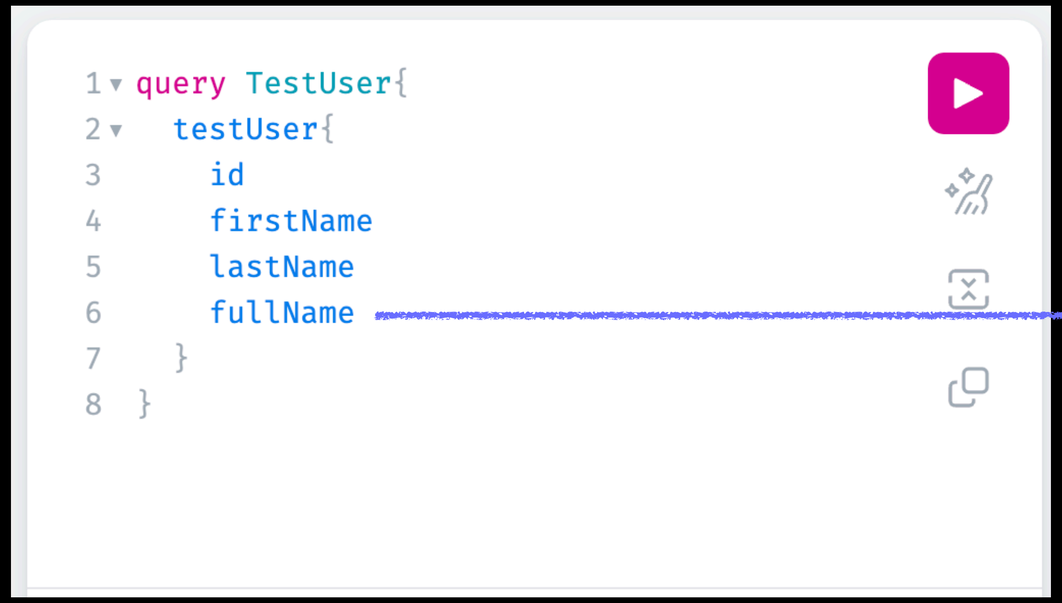
```
@Resolver(() => TestUser)
export class TestUserResolver {
  @Query(returns => TestUser)
  testUser(): TestUser {
    return {
      id: '1',
      firstName: '오',
      lastName: '제관',
    }
  }

  @ResolveField(() => String)
  async fullName(parent: TestUser) {
    return parent.firstName + parent.lastName
  }
}
```

Field resolver

6. Field Resolver

NAVER
DEVIEW
2023



6. Field Resolver

NAVER
DEVIEW
2023

그건 REST에서도 되지 않음???

```
@Get('user/:id')
async user(@Param('id') id: string): Promise<TestUser> {
  const user = this.userModel.findOne({ _id: id })

  return {
    ...user,
    fullName: user.firstName + user.lastName,
  }
}
```

문제점

- 필요에 상관 없이 항상 계산 (무거운 연산이라면?)
- 필요시 받을 수 있게 하려면 파라미터 또는 엔드포인트를 추가 해야함

6. Field Resolver

NAVER
DEVIEW
2023

```
query Review($id: ID!) {
  review(id: $id) {
    id
    text
  }
}
```

field 를 요청하지 않을 때

isShowable() 실행안함

```
query Review($id: ID!) {
  review(id: $id) {
    id
    text
    isShowable
  }
}
```

field 가 요청될 때

isShowable() 실행!

On Demand Execution!

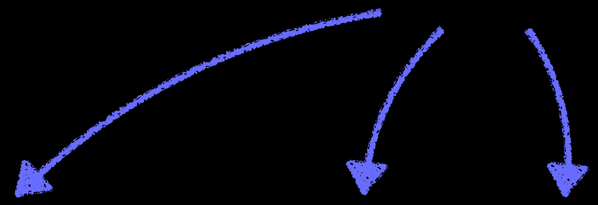
6. Field Resolver

NAVER
DEVIEW
2023

```
@Resolver(of => Review) {
class ReviewResolver {

  // ...

  @ResolveField()
  isShowable(parent: Review, args: unknown, context: AppContext) {
    return !parent.isPrivate && parent.isQualified && !parent.isReportedByOthers
  }
}
```

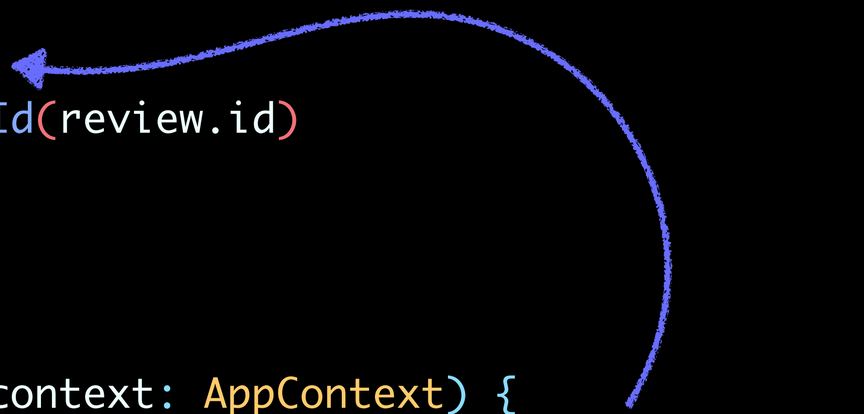


6. Field Resolver

NAVER
DEVIEW
2023

```
@Resolver(of => Review) {
class ReviewResolver {
  private async isReported(review: Review) {
    const { data } = await getReportByReviewId(review.id)
    return !!data
  }

  @ResolveField()
  isShowable(parent: Review, args: unknown, context: AppContext) {
    return !parent.isPrivate && parent.isQualified && !this.isReported(parent)
  }
}
```



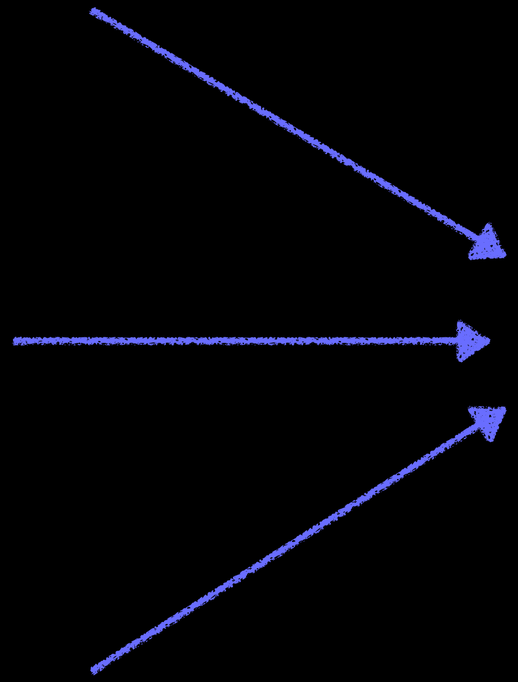
6. Field Resolver

NAVER
DEVIEW
2023

```
const isShowable = !review.isPrivate
  && review.isQualified
  && !review.isReportedByOthers
```

```
let isShowable = !review.isPrivate
  && review.isQualified
  && !review.isReportedByOthers
```

```
val isShowable = !review.isPrivate
  && review.isQualified
  && !review.isReportedByOthers
```



```
@ResolveField()
async isShowable(parent: Review) {
  return !parent.isPrivate …
}
```

Server-Side Hoisting

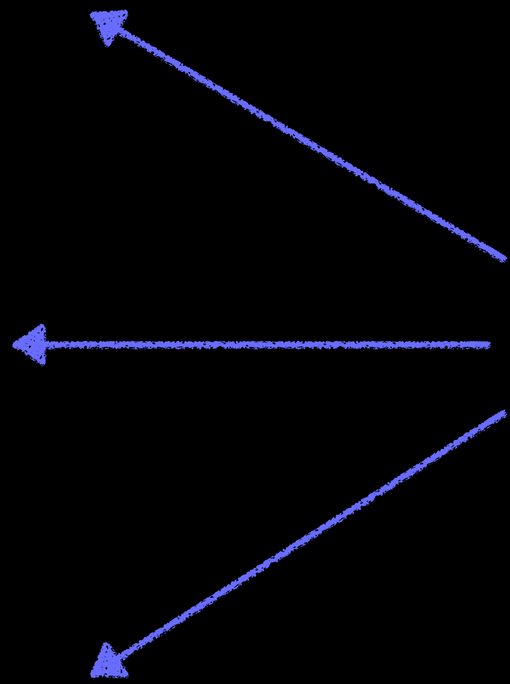
6. Field Resolver

NAVER
DEVIEW
2023

`review.isShowable`

`review.isShowable`

`review.isShowable`



```
@ResolveField()
async isShowable(parent: Review) {
  return !parent.isPrivate …
}
```

Server-Side Hoisting

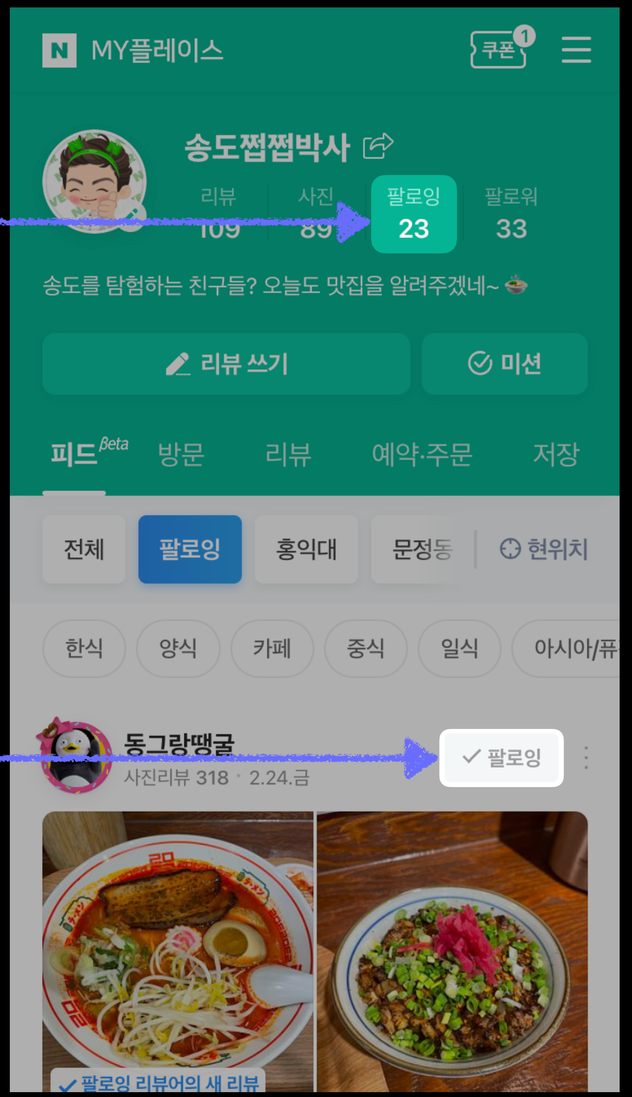
NAVER
DEVIEW
2023

Case.
팔로잉 후 멀리 떨어진 UI 업데이트 문제

문제 상황. 팔로잉 후 UI 업데이트

NAVER DEVIEW 2023


```
type Mutation {
  following(id: ID!): Boolean!
}
```



문제 상황. 팔로잉 후 UI 업데이트

NAVER
DEVIEW
2023


```
query {
  me {
    id
    nickname
    stats {
      followingCount
      followerCount
    }
  }
}
```

```
query ($input: PaginationInput!) {
  reviews(input: $input) {
    author {
      id
      nickname
      followState
    }
  }
}
```

문제 상황. 팔로잉 후 UI 업데이트

NAVER
DEVIEW
2023

문제 상황. 팔로잉 후 UI 업데이트

NAVER
DEVIEW
2023

방법 1. Follow mutation → Update Global Store?

store.reviews[3].author.followState...

집중력 총동원...!

상태가 조금 더 복잡해진다면?

```
{
  "store": {
    "reviews": [
      {
        "id": "...",
        "content": "...",
        "images": [
          // ...
        ],
        "author": {
          "id": "..."
        }
      },
      // ...
      {
        "id": "...",
        "content": "...",
        "images": [
          // ...
        ],
        "author": {
          "id": "...",
          "followState": "REQUESTED"
        }
      }
    ]
  }
}
```

문제 상황. 팔로잉 후 UI 업데이트

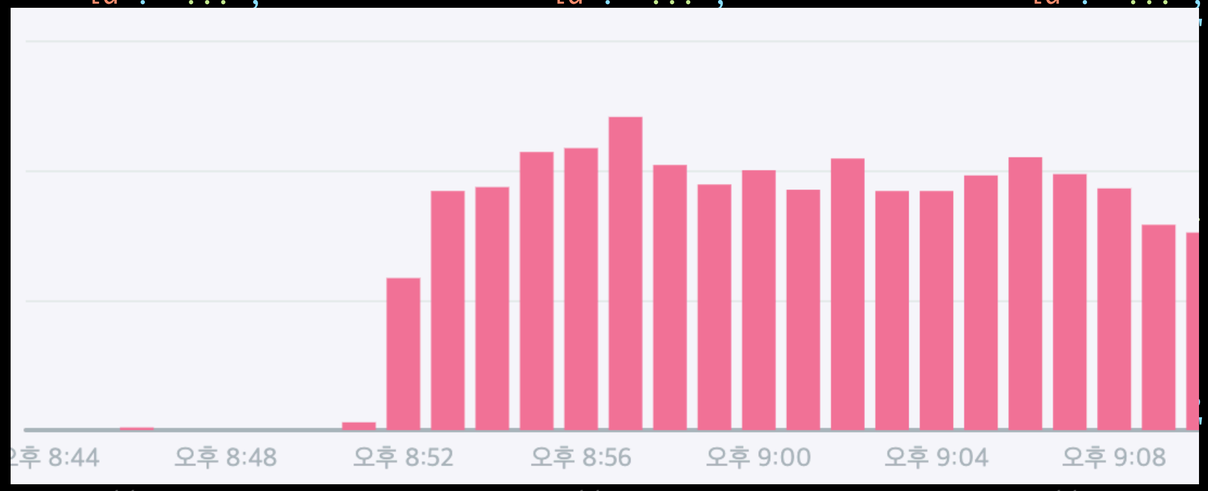
방법 2. Follow mutation → Refetch?

NAVER DEVIEW 2023

- followUser
- reviews
- followUser
- reviews
- followUser
- reviews

누구냐?!!!

백엔드 개발자

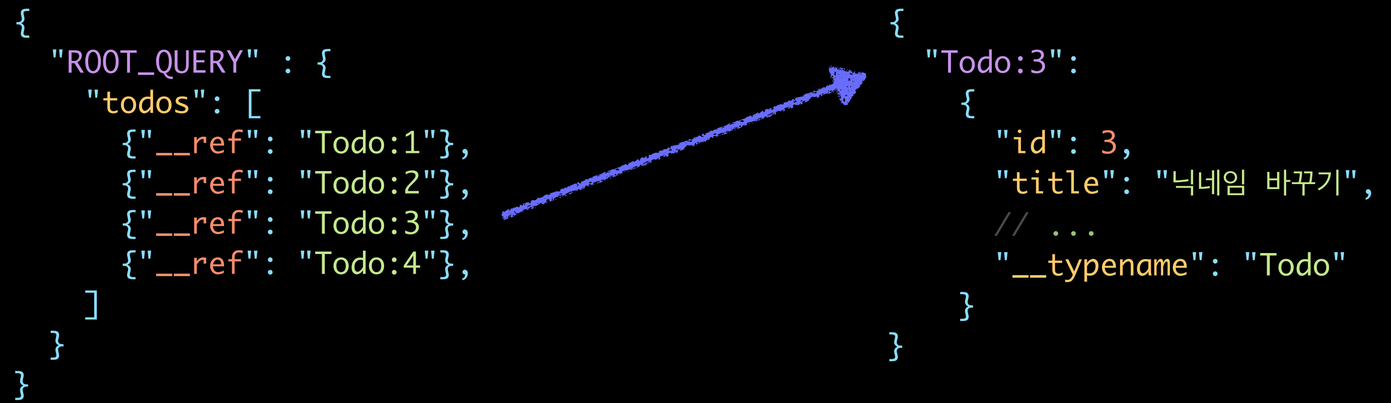



```
{
  “data”: {
    "reviews": [
      {
        "id": "...",
        // ...
      ],
      "author": {
        "id": "...",
        "followState": "REQUESTED"
      }
    }
  ]
  }
}
```

```
{
  “data”: {
    "reviews": [
      {
        "id": "...",
        // ...
      ],
      "author": {
        "id": "...",
        "followState": "REQUESTED"
      }
    }
  ]
  }
}
```

```
{
  “data”: {
    "reviews": [
      {
        "id": "...",
        // ...
      ],
      "author": {
        "id": "...",
        "followState": "REQUESTED"
      }
    }
  ]
  }
}
```

NAVER
DEVIEW
2023

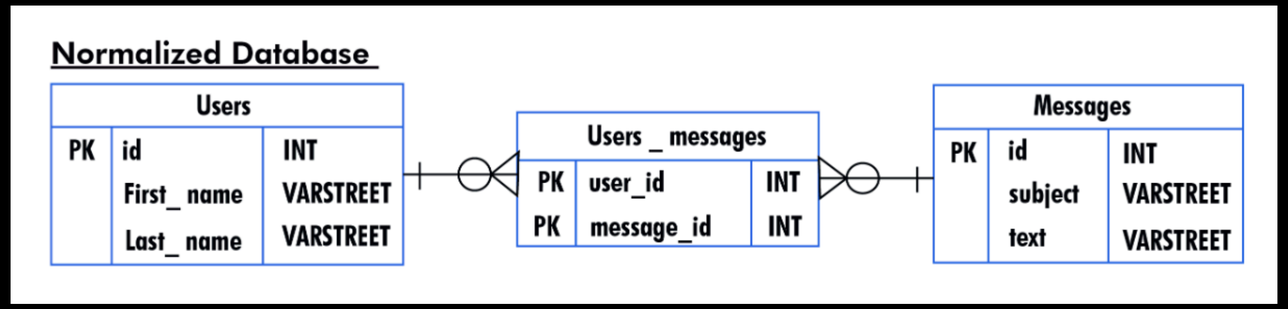


Normalization

7. Normalization

NAVER
DEVIEW
2023

Database Normalization



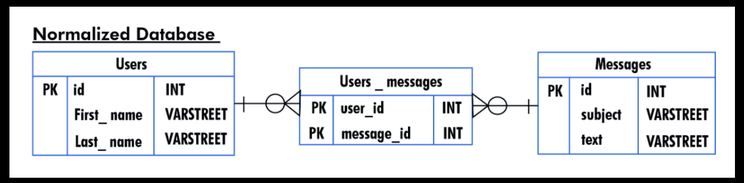
데이터의 중복을
관계와 참조로서 제거

→ 무결성 & 일관성을 지키기

7. Normalization

NAVER
DEVIEW
2023

Database Normalization



Client-Side Normalization?

↓

클라이언트 상태의 정규화

↓

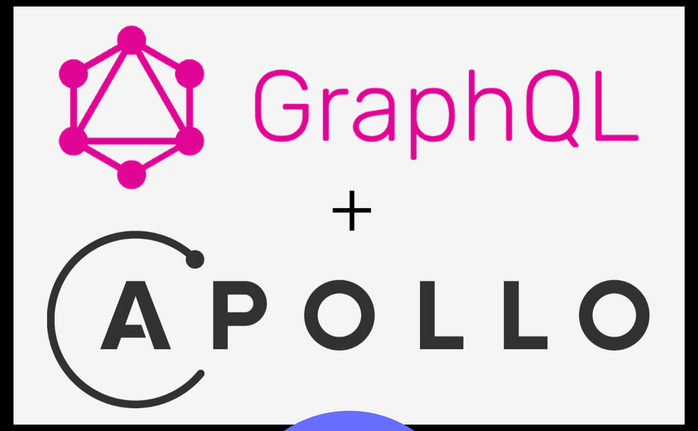
서버에서 받은 데이터

7. Normalization

NAVER
DEVIEW
2023

Client-Side Normalization 준비물

서버로부터 받는 모든 데이터의 Schema 를 미리 알고 있어야한다.



7. Normalization

NAVER
DEVIEW
2023

Normalization in Apollo-client

타입이름:아이디

unique cache id 로 “__typename:id” 를 이용

7. Normalization

NAVER DEVIEW 2023

query response

```
{
  "todos": [
    {
      "id": 1,
      "title": "영수증 리뷰 쓰기",
      // ...
      "__typename": "Todo"
    },
    {
      "id": 2,
      "title": "팔로잉 하기",
      // ...
      "__typename": "Todo"
    },
    {
      "id": 3,
      "title": "닉네임 바꾸기",
      // ...
      "__typename": "Todo"
    }
  ]
}
```

normalized cache

```
{
  "ROOT_QUERY" : {
    "todos": [
      {"__ref": "Todo:1"},
      {"__ref": "Todo:2"},
      {"__ref": "Todo:3"},
      {"__ref": "Todo:4"},
    ]
  }
}
```

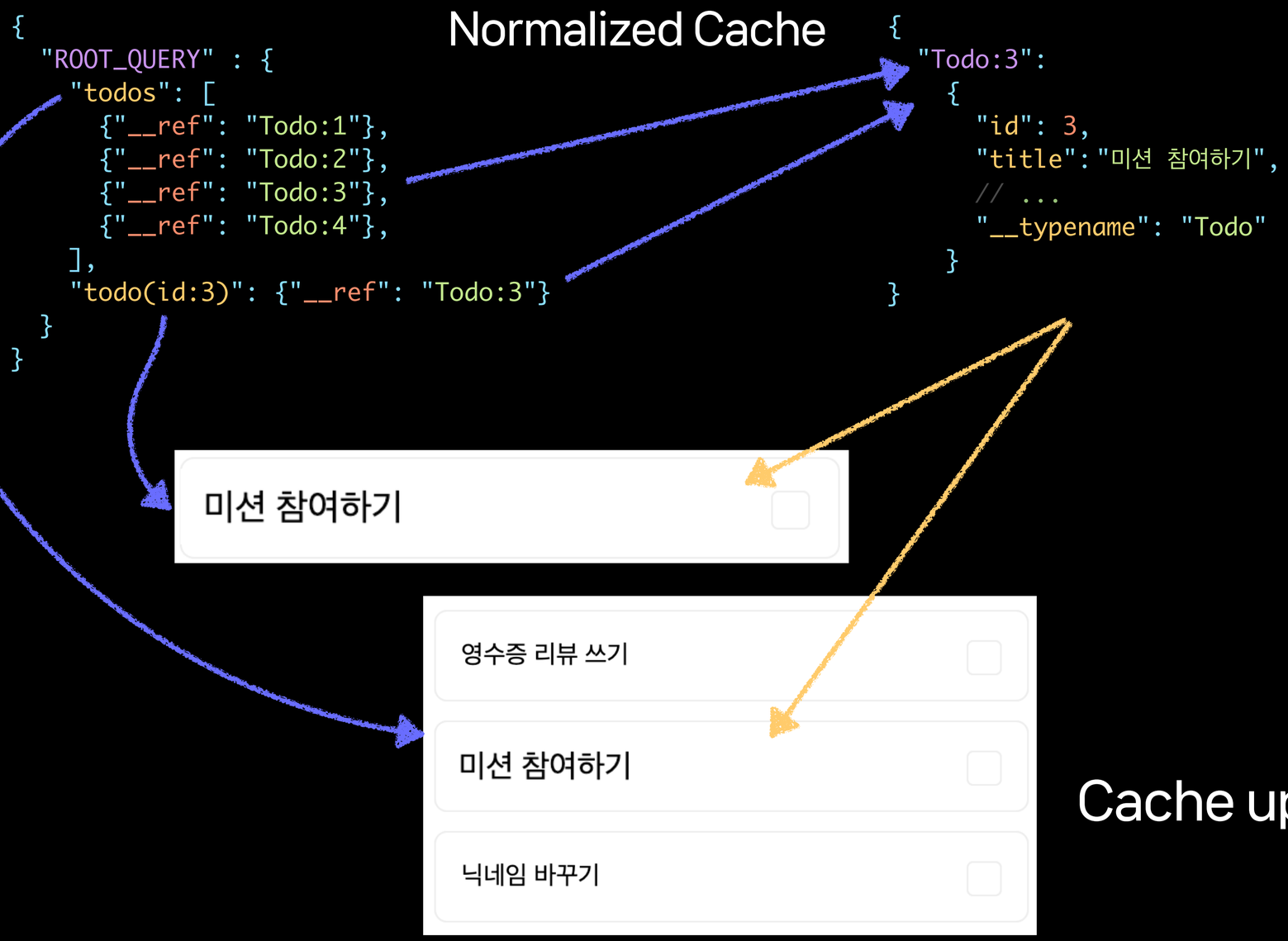
```
{
  "Todo:3":
    {
      "id": 3,
      "title": "닉네임 바꾸기",
      // ...
      "__typename": "Todo"
    }
}
```

__APOLLO_CLIENT__.cache.data.data

(connectToDevTools:true 설정 필요)

7. Normalization - Re-rendering

NAVER
DEVIEW
2023



Cache update 👉 UI update

7. Normalization

NAVER
DEVIEW
2023

Solution

```
type Mutation {
  following(id: ID!): Boolean!
}
```

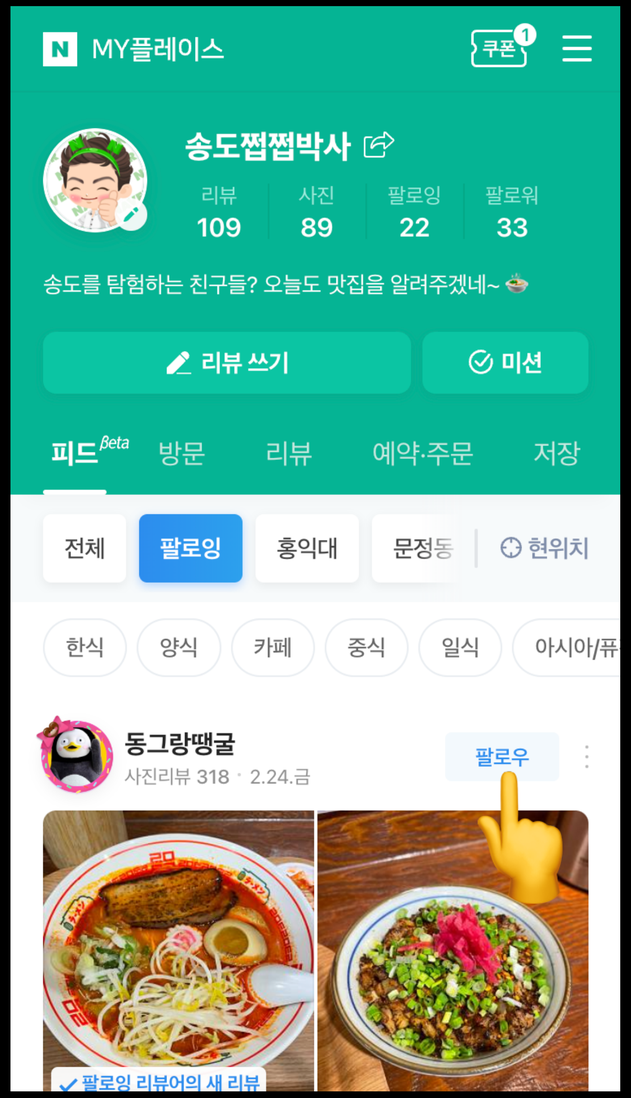
```
type Mutation {
  following(id: ID!): FollowingOutput
}

type FollowingOutput {
  me: User!
  followingUser: User!
}
```

7. Normalization

NAVER
DEVIEW
2023

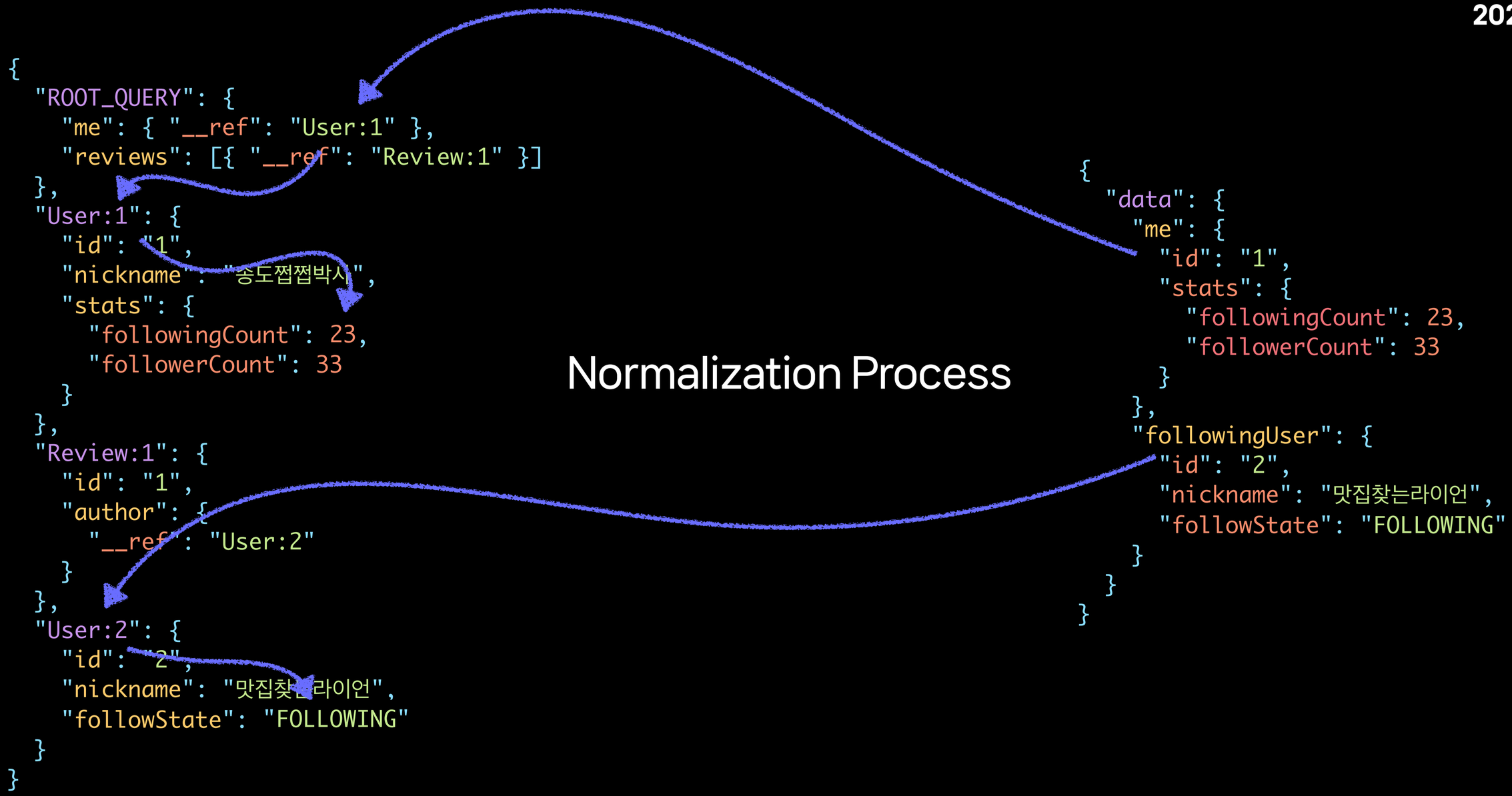
```
{
  "ROOT_QUERY": {
    "me": { "__ref": "User:1" },
    "reviews": [{ "__ref": "Review:1" }]
  },
  "User:1": {
    "id": "1",
    "nickname": "송도쩝쩝박사",
    "stats": {
      "followingCount": 22,
      "followerCount": 33
    }
  },
  "Review:1": {
    "id": "1",
    "author": {
      "__ref": "User:2"
    }
  },
  "User:2": {
    "id": "2",
    "nickname": "맛집찾는라이언",
    "followState": "NOT_FRIEND"
  }
}
```




```
mutation ($id: ID!) {
  following(id: $id) {
    me {
      id
      stats {
        followingCount
        followerCount
      }
    }
    followingUser {
      id
      nickname
      followState
    }
  }
}
```

7. Normalization

NAVER
DEVIEW
2023

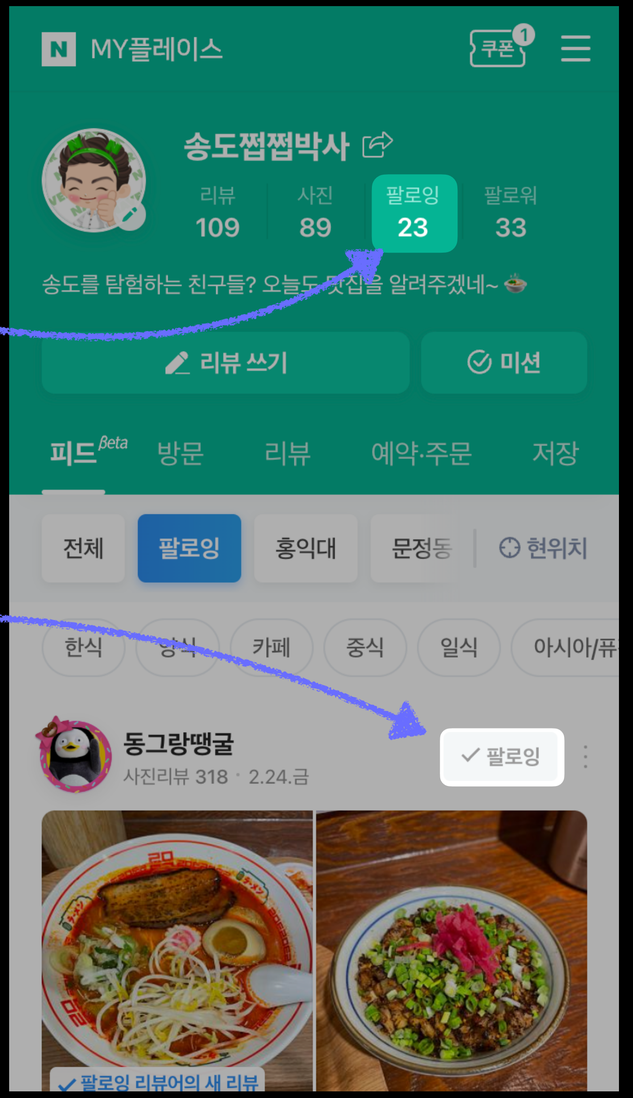


7. Normalization

NAVER DEVIEW 2023

```
{
  "ROOT_QUERY": {
    "me": { "__ref": "User:1" },
    "reviews": [{ "__ref": "Review:1" }]
  },
  "User:1": {
    "id": "1",
    "nickname": "송도쩝쩝박사",
    "stats": {
      "followingCount": 23,
      "followerCount": 33
    }
  },
  "Review:1": {
    "id": "1",
    "author": {
      "__ref": "User:2"
    }
  },
  "User:2": {
    "id": "2",
    "nickname": "맛집찾는라이언",
    "followState": "FOLLOWING"
  }
}
```

Cache update 👉 UI update



7. Normalization

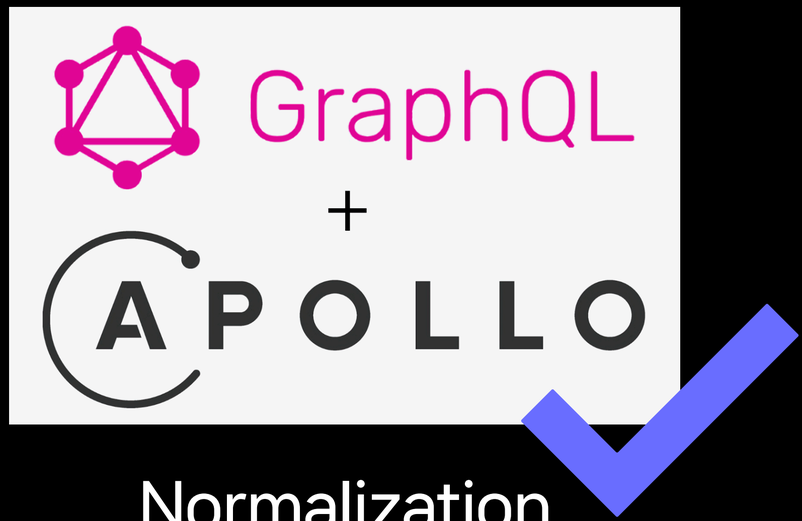
NAVER
DEVIEW
2023

필요했던 것

Follow Mutation → Response → ? → UI Update

Global state manager ✗

Refetch ✗



Normalization

Using Mutation Output

NAVER
DEVIEW
2023

Case.

쿼리를 보내는 곳과 쓰는 곳 사이의 거리

컴포넌트 이야기

NAVER
DEVIEW
2023

Server

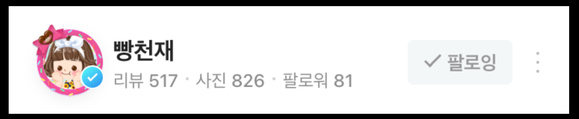
Request

```
query ReviewPage($reviewId: ID!) {
  review(id: $reviewId) {
    author {
      nickname
      thumbnailUrl
    }
  }
}
```

Response

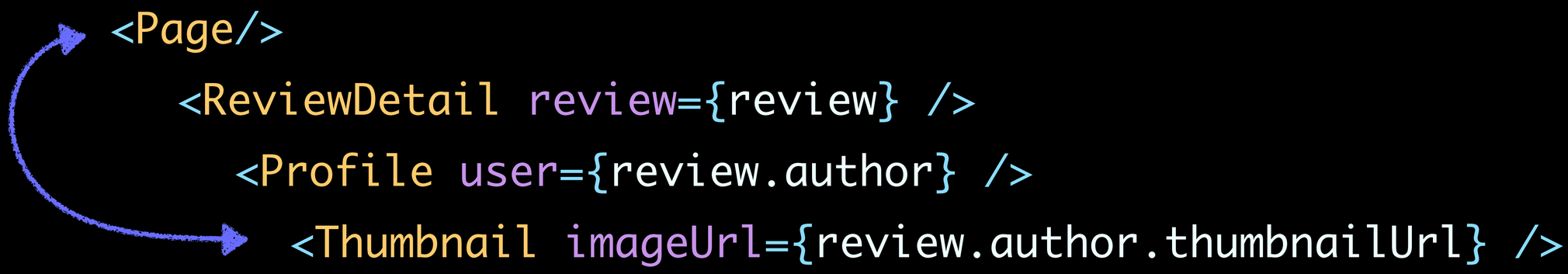
```
{
  "id": "1",
  "author": {
    "nickname": "리베로",
    "thumbnailUrl": "https://..."
  }
}
```


```
<Page/>
<ReviewDetail review={review} />
<Profile user={review.author} />
<Thumbnail imageUrl={review.author.thumbnailUrl} />
```



문제 상황.

NAVER
DEVIEW
2023



문제1. 데이터를 쓰는 곳과 불러오는 곳 사이에 거리

문제 상황.

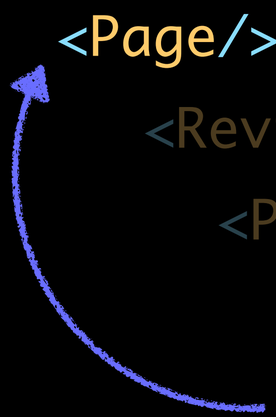
NAVER
DEVIEW
2023

새로운 데이터 의존성이 생기면?

```
<Page/>
  <ReviewDetail review={review} />
    <Profile user={review.author} />
      <Thumbnail imageUrl={review.author.thumbnailUrl} />
      <UserStat stat={review.author.stat} />
```

문제 상황.

NAVER
DEVIEW
2023



```
<Page/>
  <ReviewDetail review={review} />
    <Profile user={review.author} />
      <Thumbnail imageUrl={review.author.thumbnailUrl} />
      <UserStat stat={review.author.stat} />
```

1. 데이터 호출처를 찾는다

문제 상황.

NAVER
DEVIEW
2023

2. 해당 쿼리에 필드를 추가한다

```
query ReviewPage($reviewId: ID!) {
  review(id: $reviewId) {
    ...
    stat {
      reviewCount
      imageReviewCount
    }
  }
}
```

```
<Page/>
  <ReviewDetail review={review} />
    <Profile user={review.author} />
      <Thumbnail imageUrl={review.author.thumbnailUrl} />
      <UserStat stat={review.author.stat} />
```

Component tree가 더 복잡하고 호출처가 다양해진다면? 😵

문제 상황.

NAVER
DEVIEW
2023

문제2. 최상위 Query에서 필드별 사용 유무 판단이 어렵다

```
query ReviewPage($reviewId: ID!) {
  review(id: $reviewId) {
    author {
      nickname
      thumbnailUrl
    }
    stat {
      reviewCount
      imageReviewCount
    }
    createdDateTime
  }
}
```

쓰는 곳 없음

어느게 안쓰이는거지..?

<Page/>

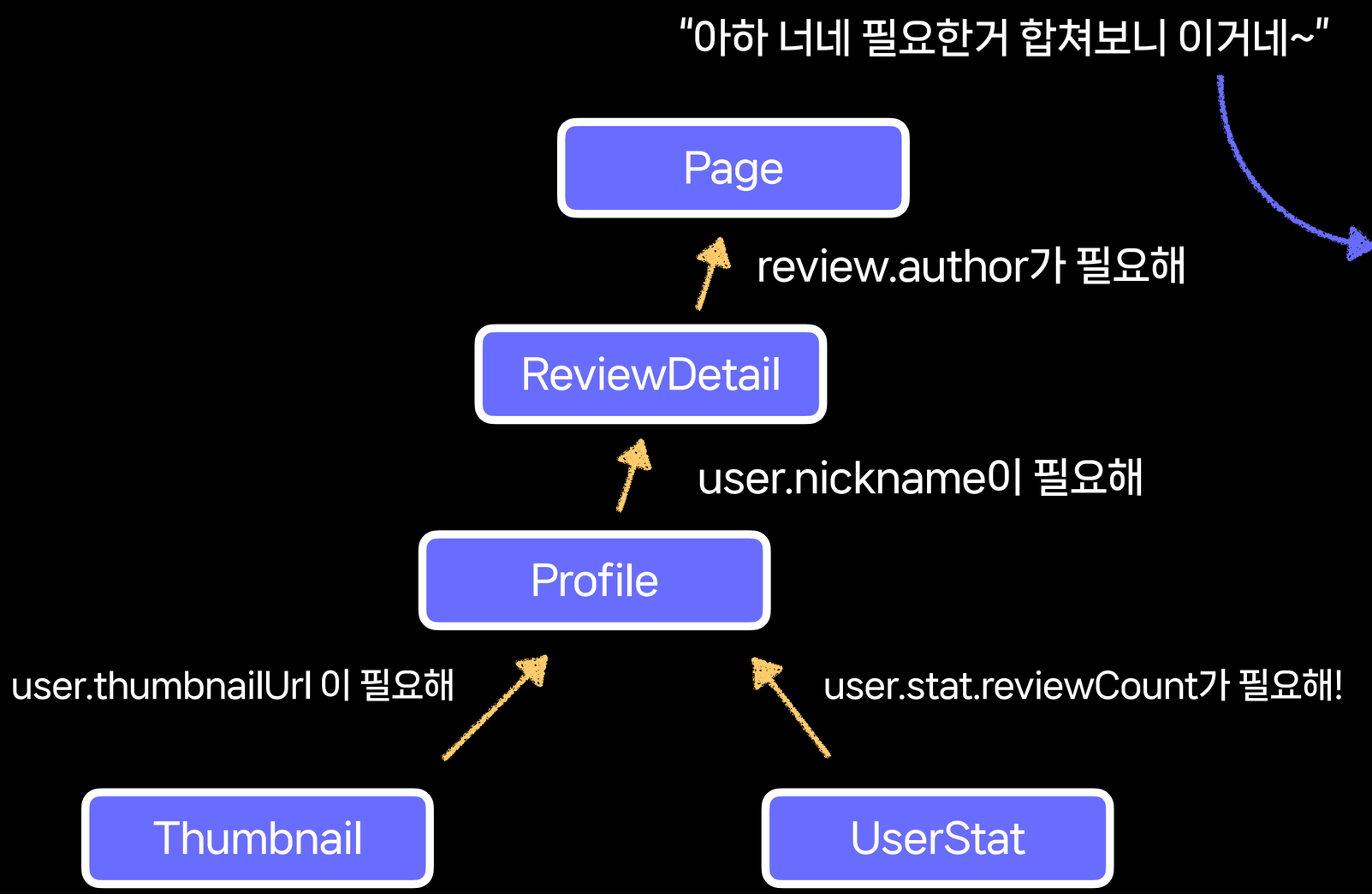
변경이 어려워진다.

What if...

Component 가 자신이 필요한 데이터 의존성을 선언만하고
Bottom-Up 으로 최상위 Query에 전달된다면??

Idea

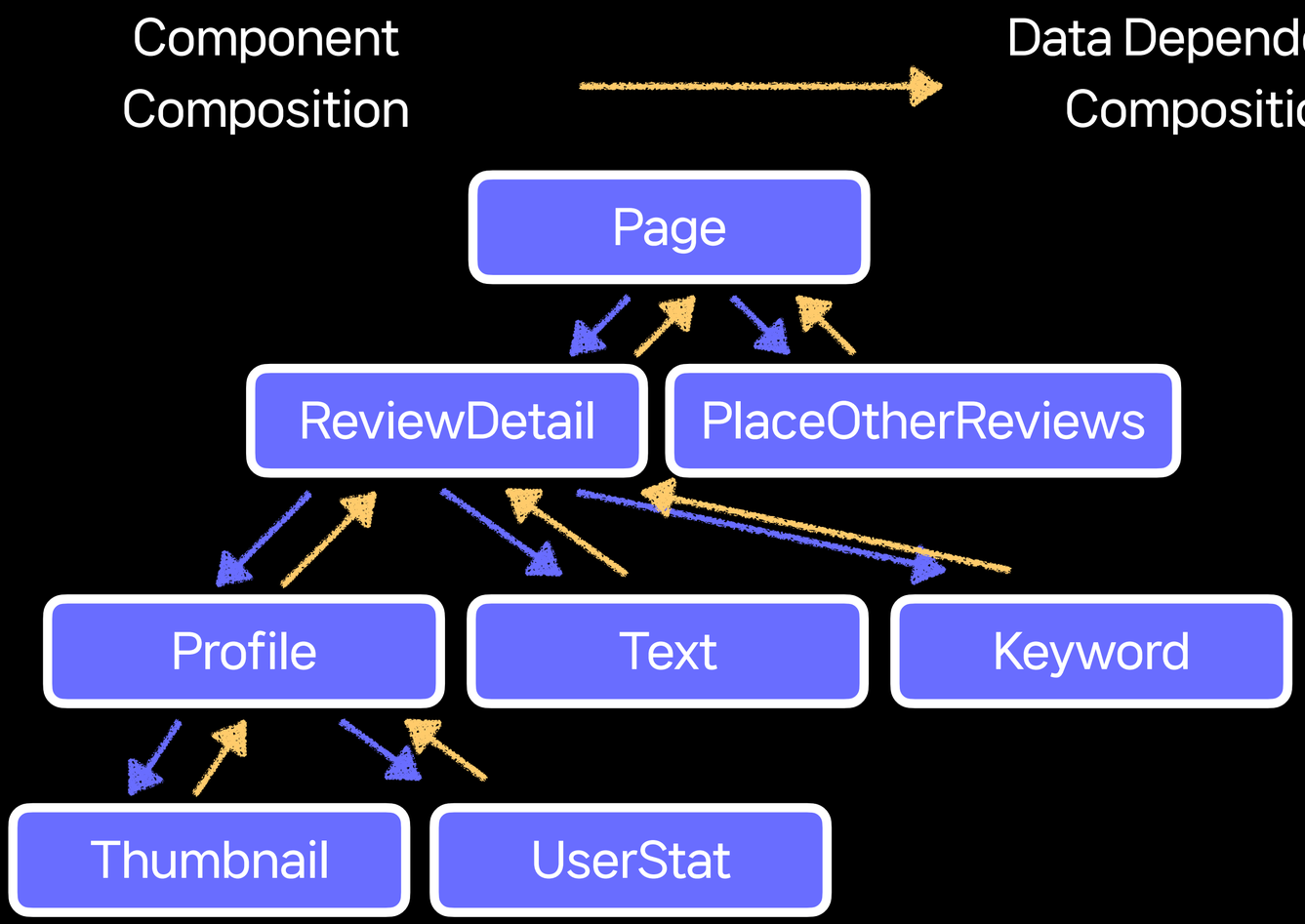
NAVER
DEVIEW
2023



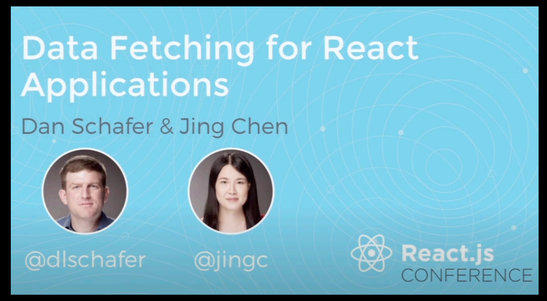

```
query ReviewPage($reviewId: ID!) {
  review(id: $reviewId) {
    author {
      nickname
      thumbnailUrl
    }
    stat {
      reviewCount
    }
  }
}
```

Idea

NAVER
DEVIEW
2023

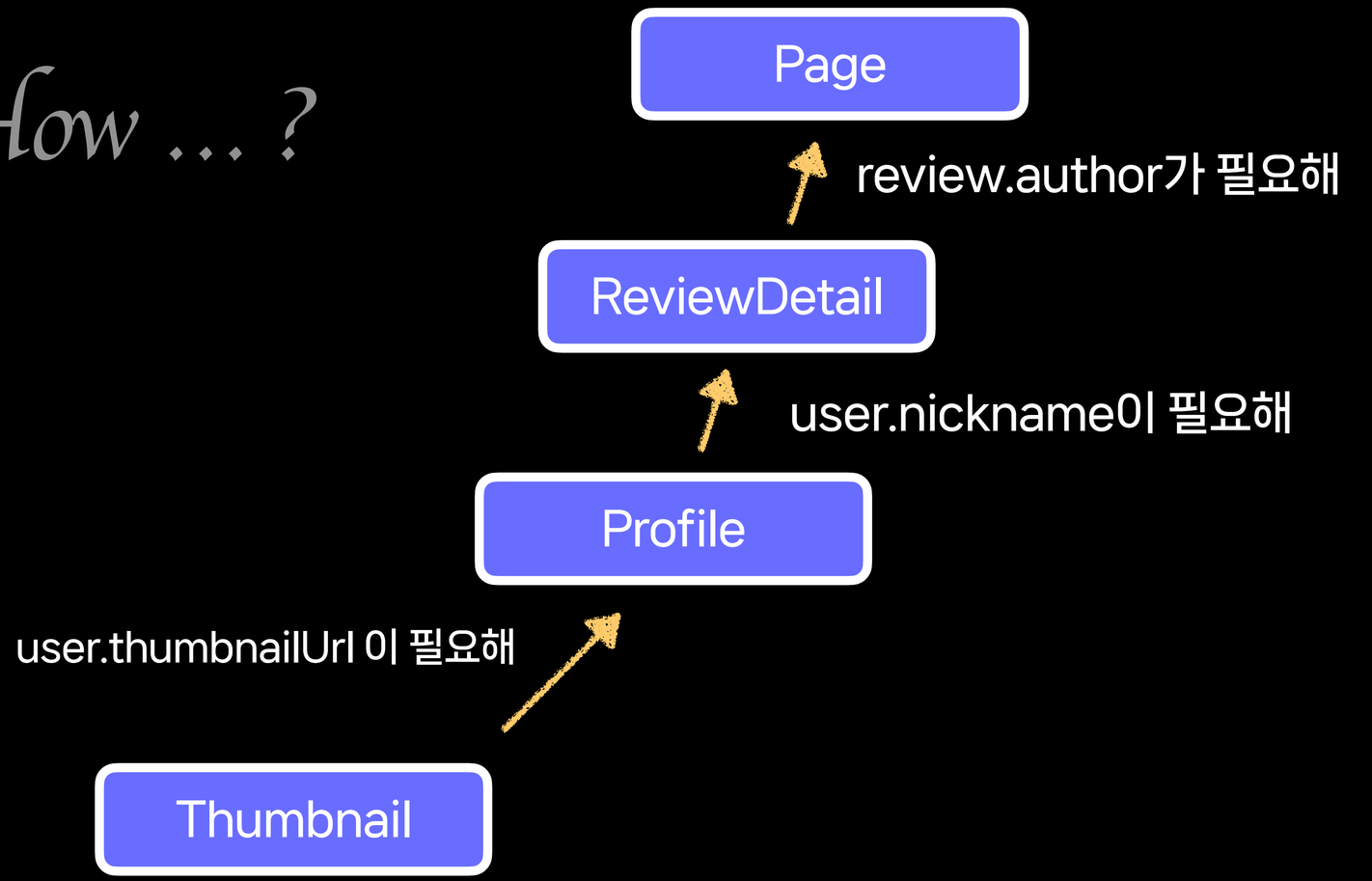


GraphQL & Relay ✨



Data fetching for React applications at Facebook

How ... ?


Page
review.author가 필요해
ReviewDetail
user.nickname이 필요해
Profile
user.thumbnailUrl 이 필요해
Thumbnail

```
fragment Thumbnail_User on User {
  thumbnailUrl
}
```

Fragment

8. Fragment

NAVER
DEVIEW
2023

Fragment

reusable units for data requirement

```
type User {
  id
  nickname
  thumbnailUrl
  stat {
    followingCount
    followerCount
  }
}
```

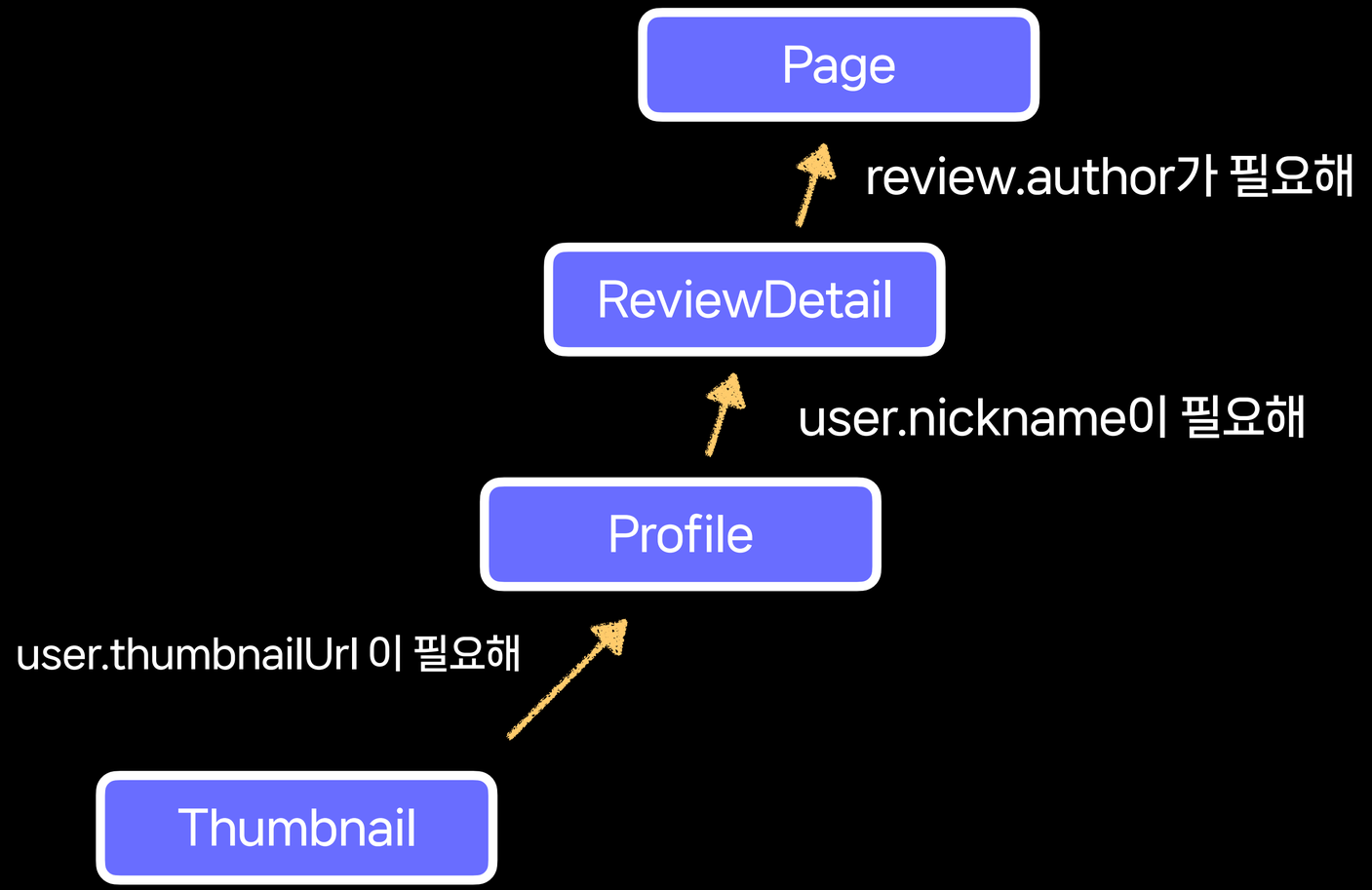
Thumbnail

user.thumbnailUrl 이 필요해

```
fragment Thumbnail_User on User {
  thumbnailUrl
}
```

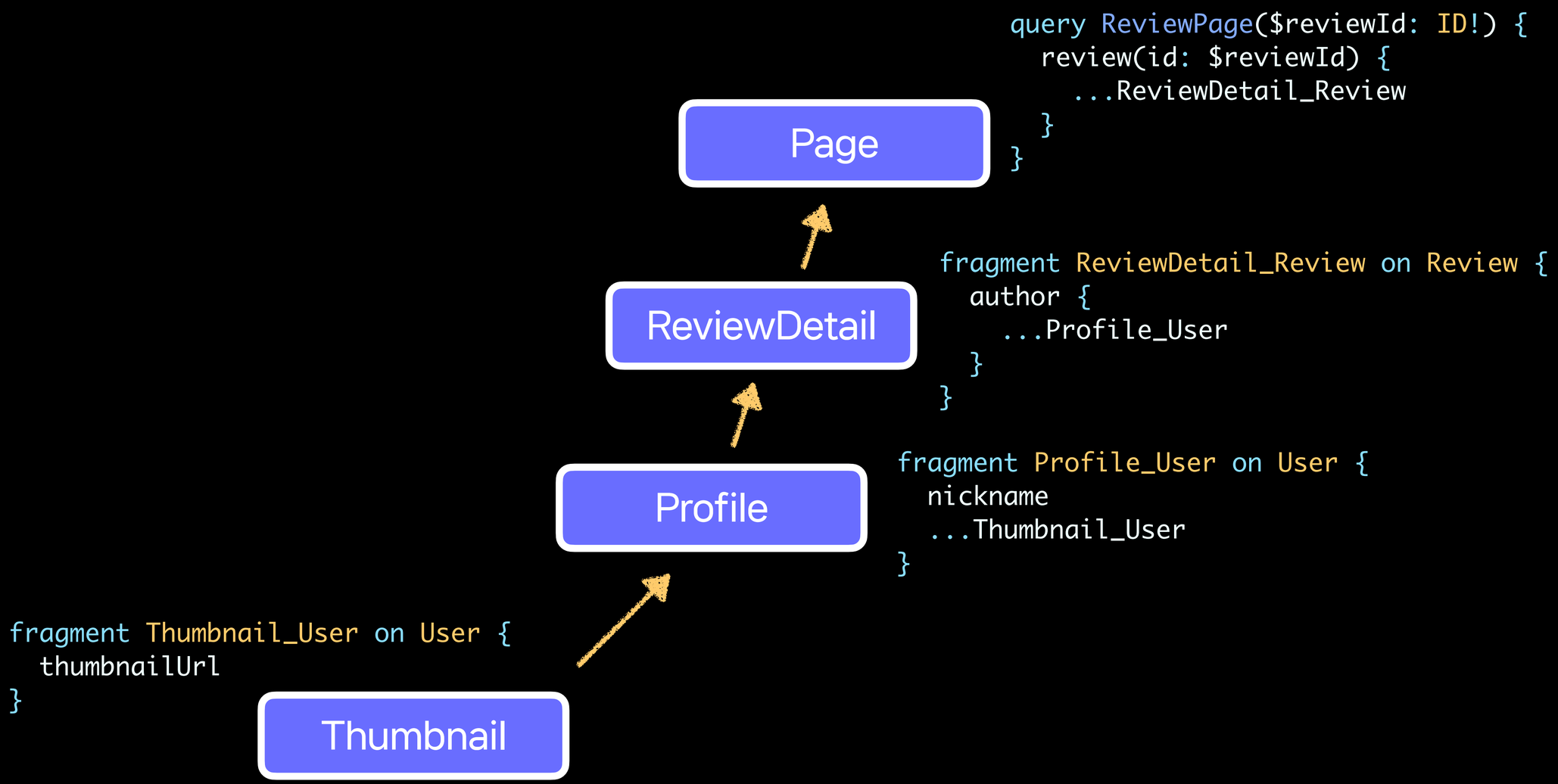
8. Fragment

NAVER
DEVIEW
2023



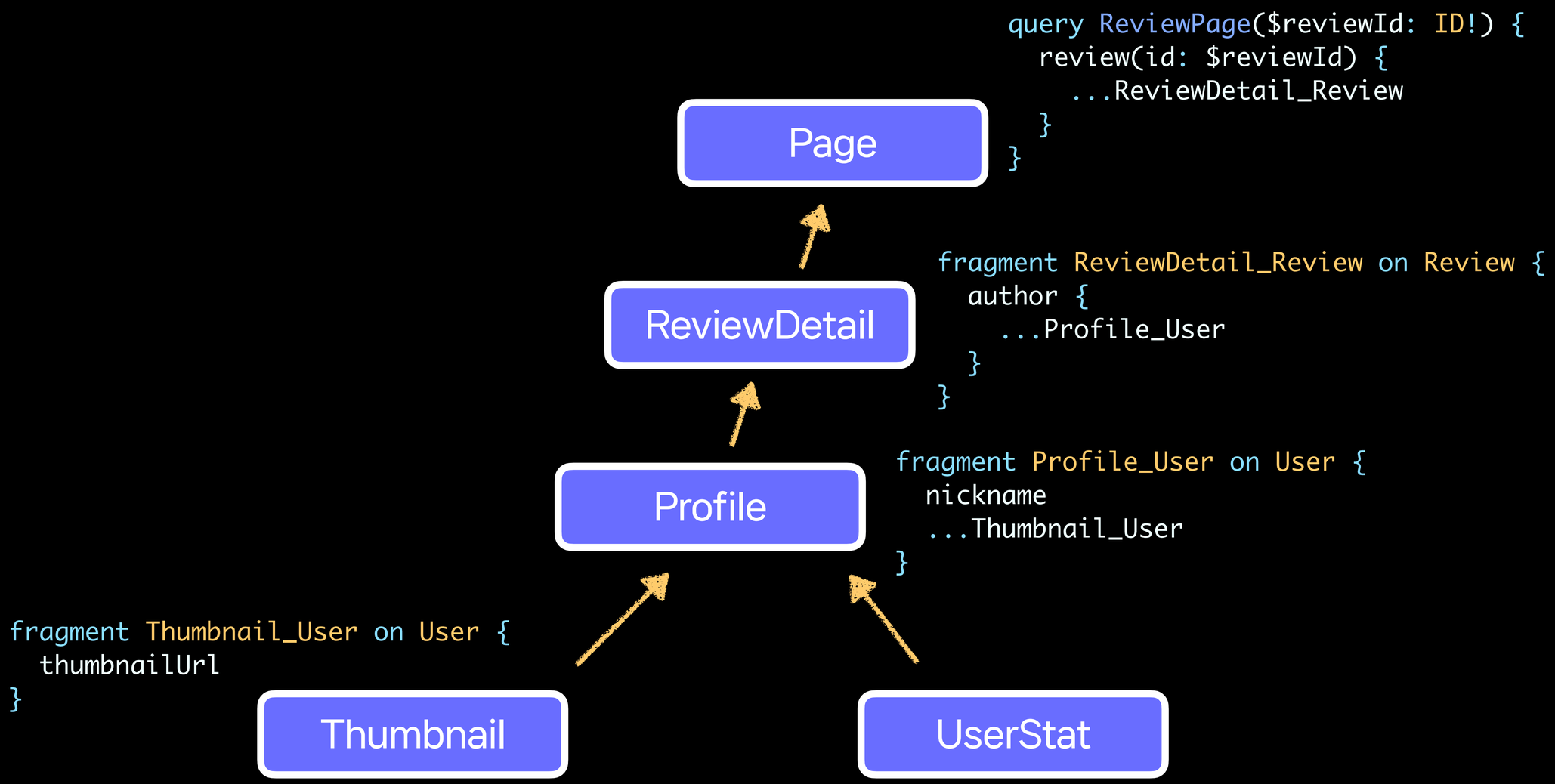
8. Fragment

NAVER DEVIEW 2023



8. Fragment

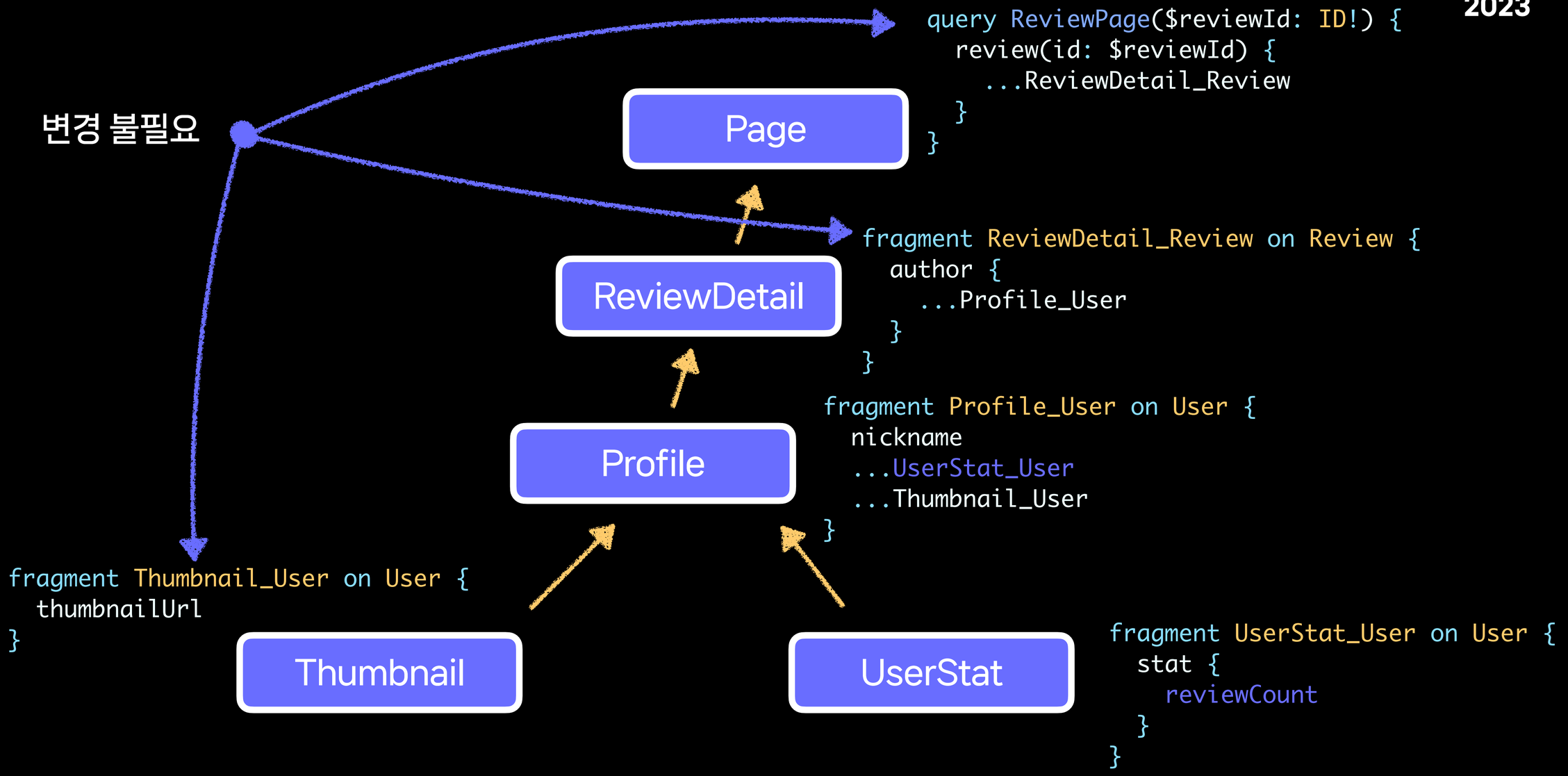
NAVER
DEVIEW
2023



새로운 데이터 의존성이 생기면?

8. Fragment

NAVER DEVIEW 2023



8. Fragment

NAVER
DEVIEW
2023

변경이 어려워진다.

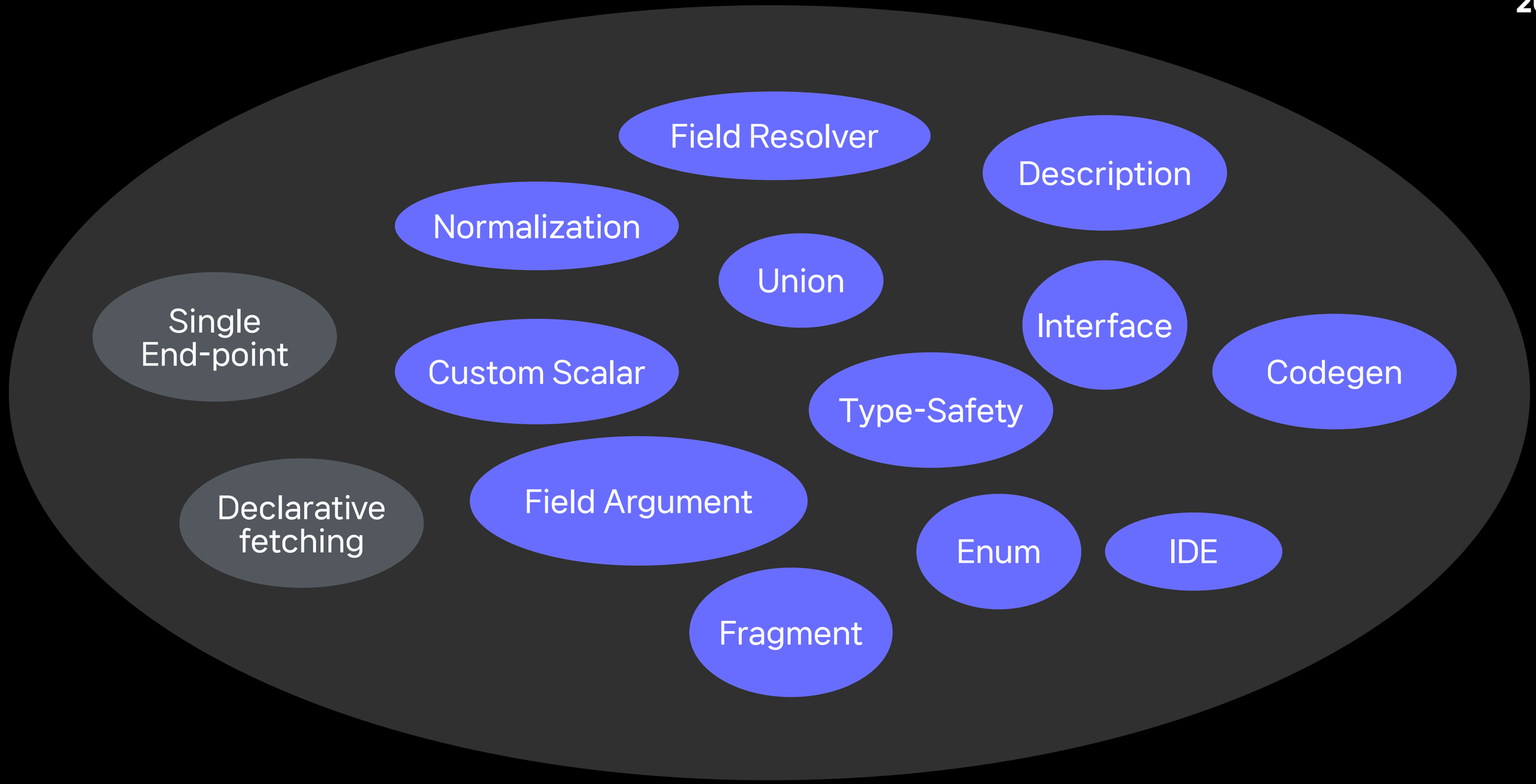
8. Fragment

NAVER
DEVIEW
2023

변경이 쉬워진다.

GraphQL의 가능성

NAVER
DEVIEW
2023



GraphQL의 가능성

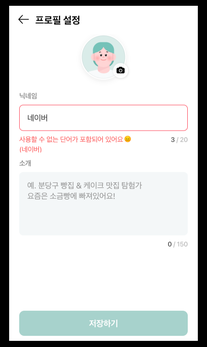
NAVER
DEVIEW
2023

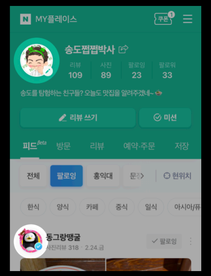
DX ⬆

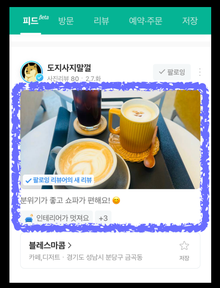
UX ⬆

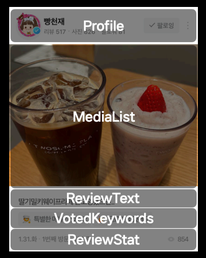
Recap

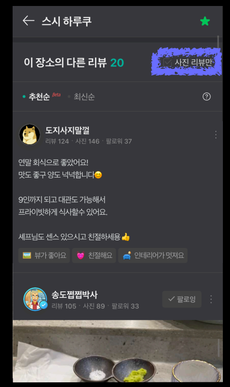
NAVER
DEVIEW
2023











1. Schema
2. Field Argument
3. Enum
4. Error Handling
5. Custom Scalar
6. Field Resolver
7. Normalization
8. Fragment

NAVER
DEVIEW
2023

Q&A