GraphQL 빠른 참조

스키마, 쿼리, 뮤테이션, 타입, 프래그먼트

스키마 정의

스키마 루트

```
schema {
  query: Query
  mutation: Mutation
}
```

객체 타입

```
type User {
  id: ID!
  name: String!
  email: String
}
```

쿼리

기본 쿼리

```
query {
  user(id: "1") {
    name
    email
  }
}
```

별칭이 있는 명명된 쿼리

```
query GetUsers {
  admin: user(role: ADMIN) { name }
  guest: user(role: GUEST) { name }
}
```

뮤테이션

기본 뮤테이션

```
mutation {
  createUser(input: { name: "Alice" }) {
    id
    name
  }
}
```

입력 타입

```
input CreateUserInput {
  name: String!
  email: String
}
```

구독

기본 구독

```
subscription {
  messageAdded(channel: "general") {
    text
    sender { name }
  }
}
```

개요

subscription	WebSocket을 통한 실시간 데이터
Server push	서버가 클라이언트에 업데이트 전송
Single field	구독당 루트 필드는 하나만

타입

스칼라 타입

Int	부호 있는 32비트 정수
Float	배정밀도 부동소수점
String	UTF-8 문자 시퀀스
Boolean	true 또는 false
ID	고유 식별자 (String으로 직렬화)

타입 수정자

String	null 허용 문자열
String!	null 불허 문자열
[String]	null 허용 문자열의 null 허용 목록
[String!]!	null 불허 문자열의 null 불허 목록

Enum & Union

```
enum Role { ADMIN USER GUEST }
union SearchResult = User | Post
interface Node { id: ID! }
```

인수 & 변수

변수

```
query GetUser($id: ID!) {
  user(id: $id) {
    name
  }
}
# Variables: { "id": "123" }
```

기본값

```
query GetUsers($limit: Int = 10) {
  users(limit: $limit) { name }
}
```

프래그먼트

명명된 프래그먼트

```
fragment UserFields on User {
  id
  name
  email
}
```

프래그먼트 사용

```
query {
  user(id: "1") { ...UserFields }
  admin: user(id: "2") { ...UserFields }
}
```

인라인 프래그먼트

```
query {
  search(text: "a") {
    ... on User { name }
    ... on Post { title }
  }
}
```

디렉티브

내장 디렉티브

@include(if: Boolean!)	조건이 참일 때만 필드 포함
@skip(if: Boolean!)	조건이 참이면 필드 건너뜀
@deprecated(reason: String)	필드를 deprecated로 표시

사용 예시

```
query GetUser($withEmail: Boolean!) {
  user(id: "1") {
    name
    email @include(if: $withEmail)
  }
}
```

인트로스펙션

타입 인트로스펙션

```
query {
  __type(name: "User") {
    name
    fields { name type { name } }
  }
}
```

스키마 인트로스펙션

```
query {
  __schema {
    types { name kind }
    queryType { name }
  }
}
```

인트로스펙션 필드

__schema	스키마의 타입과 디렉티브 쿼리
__type(name:)	이름으로 특정 타입 쿼리
__typename	객체의 타입 이름 반환

일반 패턴

페이지네이션 (Relay 스타일)

```
query {
  users(first: 10, after: "cursor") {
    edges { node { name } cursor }
    pageInfo { hasNextPage }
  }
}
```

에러 처리

```
{
  "data": { "user": null },
  "errors": [{ "message": "Not found",
    "path": ["user"] }]
}
```

모범 사례

Name operations	쿼리와 뮤테이션에 항상 이름 붙이기
Use variables	쿼리 문자열에 값을 직접 삽입하지 않기
Request only needed fields	정확한 선택으로 과도한 데이터 가져오기 방지
Use fragments	쿼리 간 필드 집합 공유