# Validation

타입 시스템을 사용하여 GraphQL 쿼리가 유효한지 여부를 미리 결정할 수 있습니다. 이를 통해 서버와 클라이언트는 런타임 검사에 의존하지 않고도 유효하지 않은 쿼리가 생성되었을 때 개발자에게 효과적으로 알릴 수 있습니다.[1)]

Star Wars 예제의 경우 starWarsValidation-test.ts 파일에는 다양한 무효성을 보여주는 여러 쿼리가 포함되어 있으며 참조 구현의 유효성 검사기를 실행하기 위해 실행할 수 있는 테스트 파일입니다.[2)]

시작으로 복잡한 유효한 쿼리를 사용하겠습니다. 이것은 이전 섹션의 예와 유사하지만 중복된 필드가 조각으로 포함된 중첩 쿼리입니다.[3)]

```
{
  hero {
    ...NameAndAppearances
    friends {
      ...NameAndAppearances
      friends {
        ...NameAndAppearances
      }
    }
  }
}

fragment NameAndAppearances on
Character {
  name
  appearsIn
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "appearsIn": [
        "NEWHOPE",
        "EMPIRE",
        "JEDI"
      ],
      "friends": [
        {
          "name": "Luke Skywalker",
          "appearsIn": [
            "NEWHOPE",
            "EMPIRE",
            "JEDI"
          ],
          "friends": [
            {
              "name": "Han Solo",
              "appearsIn": [
                "NEWHOPE",
                "EMPIRE",
                "JEDI"
              ]
            },
            {
              "name": "Leia
Organa",
              "appearsIn": [
                "NEWHOPE",
                "EMPIRE",
                "JEDI"
              ]
            },
            {
```

```
              "name": "C-3PO",
              "appearsIn": [
                "NEWHOPE",
                "EMPIRE",
                "JEDI"
              ]
            },
            {
              "name": "R2-D2",
              "appearsIn": [
                "NEWHOPE",
                "EMPIRE",
                "JEDI"
              ]
            }
          ]
        },
        {
          "name": "Han Solo",
          "appearsIn": [
            "NEWHOPE",
            "EMPIRE",
            "JEDI"
          ],
          "friends": [
            {
              "name": "Luke
Skywalker",
              "appearsIn": [
                "NEWHOPE",
                "EMPIRE",
                "JEDI"
              ]
            },
            {
              "name": "Leia
Organa",
              "appearsIn": [
                "NEWHOPE",
                "EMPIRE",
                "JEDI"
              ]
            },
            {
              "name": "R2-D2",
              "appearsIn": [
                "NEWHOPE",
                "EMPIRE",
                "JEDI"
              ]
            }
```

```
          ]
        },
        {
          "name": "Leia Organa",
          "appearsIn": [
            "NEWHOPE",
            "EMPIRE",
            "JEDI"
          ],
          "friends": [
            {
              "name": "Luke
Skywalker",
              "appearsIn": [
                "NEWHOPE",
                "EMPIRE",
                "JEDI"
              ]
            },
            {
              "name": "Han Solo",
              "appearsIn": [
                "NEWHOPE",
                "EMPIRE",
                "JEDI"
              ]
            },
            {
              "name": "C-3PO",
              "appearsIn": [
                "NEWHOPE",
                "EMPIRE",
                "JEDI"
              ]
            },
            {
              "name": "R2-D2",
              "appearsIn": [
                "NEWHOPE",
                "EMPIRE",
                "JEDI"
              ]
            }
          ]
        }
      ]
    }
  }
}
```

그리고 이 쿼리는 유효합니다. 잘못된 쿼리를 살펴보겠습니다.[4)]

프래그먼트는 자신을 참조하거나 순환을 생성할 수 없습니다. 결과가 제한되지 않을 수 있기 때문입니다! 다음은 위의 동일한 쿼리이지만 명시적인 3단계 중첩이 없습니다.[5)]

```
{
  hero {
    ...NameAndAppearancesAndFriends
  }
}

fragment
NameAndAppearancesAndFriends on
Character {
  name
  appearsIn
  friends {
    ...NameAndAppearancesAndFriends
  }
}
```

```
{
  "errors": [
    {
      "message": "Cannot spread
fragment
\"NameAndAppearancesAndFriends\"
within itself.",
      "locations": [
        {
          "line": 11,
          "column": 5
        }
      ]
    }
  ]
}
```

필드를 쿼리할 때 주어진 유형에 존재하는 필드를 쿼리해야 합니다. 따라서 hero가 Character를 반환하면 Character에 대한 필드를 쿼리해야 합니다. 해당 유형에는 favoriteSpaceship 필드가 없으므로 이 쿼리는 유효하지 않습니다.[6)]

```
# INVALID: favoriteSpaceship does
not exist on Character
{
  hero {
    favoriteSpaceship
  }
}
```

```
{
  "errors": [
    {
      "message": "Cannot query
field \"favoriteSpaceship\" on type
\"Character\".",
      "locations": [
        {
          "line": 4,
          "column": 5
        }
      ]
    }
  ]
}
```

필드를 쿼리하고 스칼라 또는 열거형이 아닌 다른 것을 반환할 때마다 필드에서 반환하려는 데이터를 지정해야 합니다. hero는 Character를 반환하고 우리는 name 및 appearsIn과 같은 필드를 요청했습니다. 이를 생략하면 쿼리가 유효하지 않습니다.

```
# INVALID: hero is not a scalar, so
fields are needed
{
  hero
}
```

```
{
  "errors": [
    {
      "message": "Field \"hero\" of
type \"Character\" must have a
selection of subfields. Did you
mean \"hero { ... }\"?",
      "locations": [
        {
          "line": 3,
          "column": 3
        }
      ]
    }
  ]
}
```

마찬가지로 필드가 스칼라인 경우 추가 필드를 쿼리하는 것은 의미가 없으며 그렇게 하면 쿼리가 무효화 됩니다.[7)]

```
# INVALID: name is a scalar, so
fields are not permitted
{
  hero {
    name {
      firstCharacterOfName
    }
  }
}
```

```
{
  "errors": [
    {
      "message": "Field \"name\"
must not have a selection since
type \"String!\" has no
subfields.",
      "locations": [
        {
          "line": 4,
          "column": 10
        }
      ]
    }
  ]
}
```

## 없는 필드

이전에 쿼리는 해당 유형의 필드만 쿼리할 수 있다는 점에 유의했습니다. Character를 반환하는 hero를 쿼리할 때 Character에 존재하는 필드만 쿼리할 수 있습니다. 하지만 R2-D2의 기본 기능을 쿼리하려는 경우 어떻게 됩니까?[8)]

```
# INVALID: primaryFunction does not
exist on Character
{
```

```
{
  "errors": [
    {
```

```
  hero {
    name
    primaryFunction
  }
}
```

```
      "message": "Cannot query
field \"primaryFunction\" on type
\"Character\". Did you mean to use
an inline fragment on \"Droid\"?",
      "locations": [
        {
          "line": 5,
          "column": 5
        }
      ]
    }
  ]
}
```

## Using Fragment

primaryFunction이 Character의 필드가 아니기 때문에 해당 쿼리는 유효하지 않습니다. Character가 Droid이면 primaryFunction을 가져오고 그렇지 않으면 해당 필드를 무시한다는 것을 나타내는 방법이 필요합니다. 이를 위해 앞서 소개한 조각을 사용할 수 있습니다. Droid에 정의된 프래그먼트를 설정하고 포함함으로써 정의된 primaryFunction만 쿼리하도록 합니다.[9)]

```
{
  hero {
    name
    ...DroidFields
  }
}

fragment DroidFields on Droid {
  primaryFunction
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "primaryFunction":
"Astromech"
    }
  }
}
```

이 쿼리는 유효하지만 약간 장황합니다. 명명된 조각은 위에서 여러 번 사용할 때 유용했지만 이 조각은 한 번만 사용합니다. 명명된 조각을 사용하는 대신 인라인 조각을 사용할 수 있습니다. 이것은 여전히 우리가 쿼리하는 유형을 나타낼 수 있지만 별도의 프래그먼트 이름을 지정하지 않습니다.[10)]

```
{
  hero {
    name
    ... on Droid {
      primaryFunction
    }
  }
}
```

```
{
  "data": {
    "hero": {
      "name": "R2-D2",
      "primaryFunction":
"Astromech"
    }
  }
}
```

이것은 검증 시스템의 표면을 긁은 것뿐입니다. GraphQL 쿼리가 의미적으로 의미가 있는지 확인하기 위한 여러 유효성 검사 규칙이 있습니다. 사양은 “ 검증 ” 섹션에서 이 주제에 대해 더 자세히 설명하고 GraphQL.js의 유효성 검사 디렉터리에는 사양을 준수하는 GraphQL 유효성 검사기를 구현하는 코드가 포함되어 있습니다.[11)]

## Continue Reading

- Execution

---

## 관련 문서

- Schemas and Types

1)
By using the type system, it can be predetermined whether a GraphQL query is valid or not. This allows servers and clients to effectively inform developers when an invalid query has been created, without having to rely on runtime checks.
2)
For our Star Wars example, the file starWarsValidation-test.ts contains a number of queries demonstrating various invalidities, and is a test file that can be run to exercise the reference implementation's validator.
3)
To start, let's take a complex valid query. This is a nested query, similar to an example from the previous section, but with the duplicated fields factored out into a fragment:
4)
And this query is valid. Let's take a look at some invalid queries...
5)
A fragment cannot refer to itself or create a cycle, as this could result in an unbounded result! Here's the same query above but without the explicit three levels of nesting:
6)
When we query for fields, we have to query for a field that exists on the given type. So as hero returns a Character, we have to query for a field on Character. That type does not have a favoriteSpaceship field, so this query is invalid:
7)
Similarly, if a field is a scalar, it doesn't make sense to query for additional fields on it, and doing so will make the query invalid:
8)
Earlier, it was noted that a query can only query for fields on the type in question; when we query for hero which returns a Character, we can only query for fields that exist on Character. What happens if we want to query for R2-D2s primary function, though?
9)
That query is invalid, because primaryFunction is not a field on Character. We want some way of indicating that we wish to fetch primaryFunction if the Character is a Droid, and to ignore that field otherwise. We can use the fragments we introduced earlier to do this. By setting up a fragment defined on Droid and including it, we ensure that we only query for primaryFunction where it is defined.
10)
This query is valid, but it's a bit verbose; named fragments were valuable above when we used them multiple times, but we're only using this one once. Instead of using a named fragment, we can use an

inline fragment; this still allows us to indicate the type we are querying on, but without naming a separate fragment:
[11)]

This has just scratched the surface of the validation system; there are a number of validation rules in place to ensure that a GraphQL query is semantically meaningful. The specification goes into more detail about this topic in the “Validation” section, and the validation directory in GraphQL.js contains code implementing a specification-compliant GraphQL validator.