GraphQL Quick Reference

Schemas, queries, mutations, types, fragments

Schema Definition

Schema Root

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

Object Type

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

Queries

Basic Query

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

Named Query with Alias

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

Mutations

Basic Mutation

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

Input Types

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

Subscriptions

Basic Subscription

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

Overview

`subscription`	Real-time data via WebSocket
`Server push`	Server sends updates to client
`Single field`	Only one root field per subscription

Types

Scalar Types

`Int`	Signed 32-bit integer
`Float`	Double-precision floating point
`String`	UTF-8 character sequence
`Boolean`	true or false
`ID`	Unique identifier (serialized as String)

Type Modifiers

`String`	Nullable string
`String!`	Non-null string
`[String]`	Nullable list of nullable strings
`[String!]!`	Non-null list of non-null strings

Enum & Union

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

Arguments & Variables

Variables

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

Default Values

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

Fragments

Named Fragment

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

Using Fragments

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

Inline Fragment

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

Directives

Built-in Directives

`@include(if: Boolean!)`	Include field only when condition is true
`@skip(if: Boolean!)`	Skip field when condition is true
`@deprecated(reason: String)`	Mark field as deprecated

Usage Example

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

Introspection

Type Introspection

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

Schema Introspection

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

Introspection Fields

`__schema`	Query the schema's types and directives
`__type(name:)`	Query a specific type by name
`__typename`	Returns the type name of any object

Common Patterns

Pagination (Relay-style)

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

Error Handling

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

Best Practices

`Name operations`	Always name queries and mutations
`Use variables`	Never interpolate values into query strings
`Request only needed fields`	Avoid over-fetching with precise selections
`Use fragments`	Share field sets across queries