1. **What is GraphQL?**
    - **Answer**: GraphQL is a query language for APIs and a runtime for executing those queries by providing a complete and understandable description of the data in your API. It allows clients to request exactly the data they need and nothing more, making APIs more efficient and flexible.
2. **How does GraphQL differ from REST?**
    - **Answer**: In REST, each resource is exposed via a URL and clients need to know multiple endpoints to fetch related data. In GraphQL, there's a single endpoint, and clients specify the structure of the response. GraphQL queries allow clients to request exactly the data they need, reducing over-fetching and under-fetching of data.
3. **What are the main components of a GraphQL schema?**
    - **Answer**: The main components are:
        - **Types**: Define the shape of the data.
        - **Queries**: Allow clients to request data.
        - **Mutations**: Allow clients to modify data.
        - **Resolvers**: Functions that resolve each field on a type.

Intermediate Questions

4. **What are resolvers in GraphQL?**
    - **Answer**: Resolvers are functions that handle fetching the data for each field in a GraphQL query. They are responsible for populating the data for a query, mutation, or subscription.
5. **Explain the difference between a query and a mutation in GraphQL.**
    - **Answer**:
        - **Query**: Used to read or fetch data.
        - **Mutation**: Used to write or modify data (e.g., create, update, delete).
6. **What is a GraphQL fragment?**
    - **Answer**: A fragment is a reusable piece of a GraphQL query that can be shared between multiple queries, mutations, or subscriptions. It helps avoid repetition and keeps queries DRY (Don't Repeat Yourself).
7. **What are subscriptions in GraphQL?**
    - **Answer**: Subscriptions are a way to push data from the server to the client in real-time. They are typically used for notifying clients about events or changes in the data.

Advanced Questions

8. **How can you handle authentication and authorization in GraphQL?**
    - **Answer**: Authentication can be handled by validating tokens (e.g., JWT) at the beginning of each request. Authorization can be managed in resolvers by checking user permissions before proceeding with the operation.
9. **How do you handle error management in GraphQL?**
    - **Answer**: Errors in GraphQL are handled through the `errors` field in the response. Custom error messages can be implemented in resolvers. Tools like `apollo-errors` can help standardize error handling.
10. **Explain the N+1 problem in GraphQL and how to solve it.**
    - **Answer**: The N+1 problem occurs when the server makes N additional database queries to resolve nested data. It can be solved using techniques like

DataLoader, which batches and caches requests to minimize the number of database queries.

Practical Questions

11. **How do you perform pagination in GraphQL?**
    - **Answer**: Pagination can be implemented using two main approaches:
      - **Offset-based pagination**: Uses `limit` and `offset` parameters.
      - **Cursor-based pagination**: Uses `first`, `last`, `before`, and `after` cursors for more efficient pagination in large datasets.
12. **What are some best practices for designing a GraphQL schema?**
    - **Answer**:
      - Design the schema around the needs of your clients.
      - Keep the schema simple and consistent.
      - Use descriptive naming for types and fields.
      - Avoid deeply nested queries.
      - Document the schema using descriptions and comments.
13. **How do you secure a GraphQL API?**
    - **Answer**: Security can be ensured by:
      - Validating inputs to prevent injection attacks.
      - Implementing rate limiting and depth limiting to prevent abuse.
      - Using proper authentication and authorization mechanisms.
      - Avoiding exposing sensitive information through introspection.

Hands-on Questions

14. **Can you show how to define a GraphQL schema and resolver for a simple user type?**
    - **Answer**:

```
javascript
Copy code
// schema.js
const { gql } = require('apollo-server-express');

const typeDefs = gql`
  type User {
    id: ID!
    name: String!
    email: String!
  }

  type Query {
    users: [User!]
    user(id: ID!): User
  }

  type Mutation {
    addUser(name: String!, email: String!): User!
  }
`;

module.exports = typeDefs;
javascript
```

```
Copy code
// resolvers.js
const users = [];

const resolvers = {
  Query: {
    users: () => users,
    user: (parent, args) => users.find(user => user.id ===
args.id),
  },
  Mutation: {
    addUser: (parent, { name, email }) => {
      const user = { id: users.length + 1, name, email };
      users.push(user);
      return user;
    },
  },
};

module.exports = resolvers;
```

15. **How do you test a GraphQL API?**
    - **Answer**: Testing a GraphQL API can be done using tools like Jest for unit tests and integration tests. `graphql-request` or `apollo-client` can be used to make queries and mutations in your tests.

```
javascript
Copy code
const { gql } = require('apollo-server-express');
const { createTestClient } = require('apollo-server-testing');
const { ApolloServer } = require('apollo-server');
const typeDefs = require('./schema');
const resolvers = require('./resolvers');

const server = new ApolloServer({ typeDefs, resolvers });
const { query, mutate } = createTestClient(server);

test('Add User Mutation', async () => {
  const ADD_USER = gql`
    mutation AddUser($name: String!, $email: String!) {
      addUser(name: $name, email: $email) {
        id
        name
        email
      }
    }
  `;

  const res = await mutate({
    mutation: ADD_USER,
    variables: { name: 'John Doe', email: 'john@example.com' },
  });

  expect(res.data.addUser).toHaveProperty('id');
  expect(res.data.addUser).toHaveProperty('name', 'John Doe');
  expect(res.data.addUser).toHaveProperty('email',
'john@example.com');
});
```