SQL database in Microsoft Fabric

Series Scenario – Conference session application

You need:

- to store and serve data for the conference webpage.
- to be able to run analysis over the data.
- an application that allows attendees to easily search for and find sessions of interest to them.
- to follow modern best practices for application lifecycle.
- to be able to monitor and troubleshoot database and query performance.

Course Overview and agenda

Episode 1: Introduction and Overview; Getting started

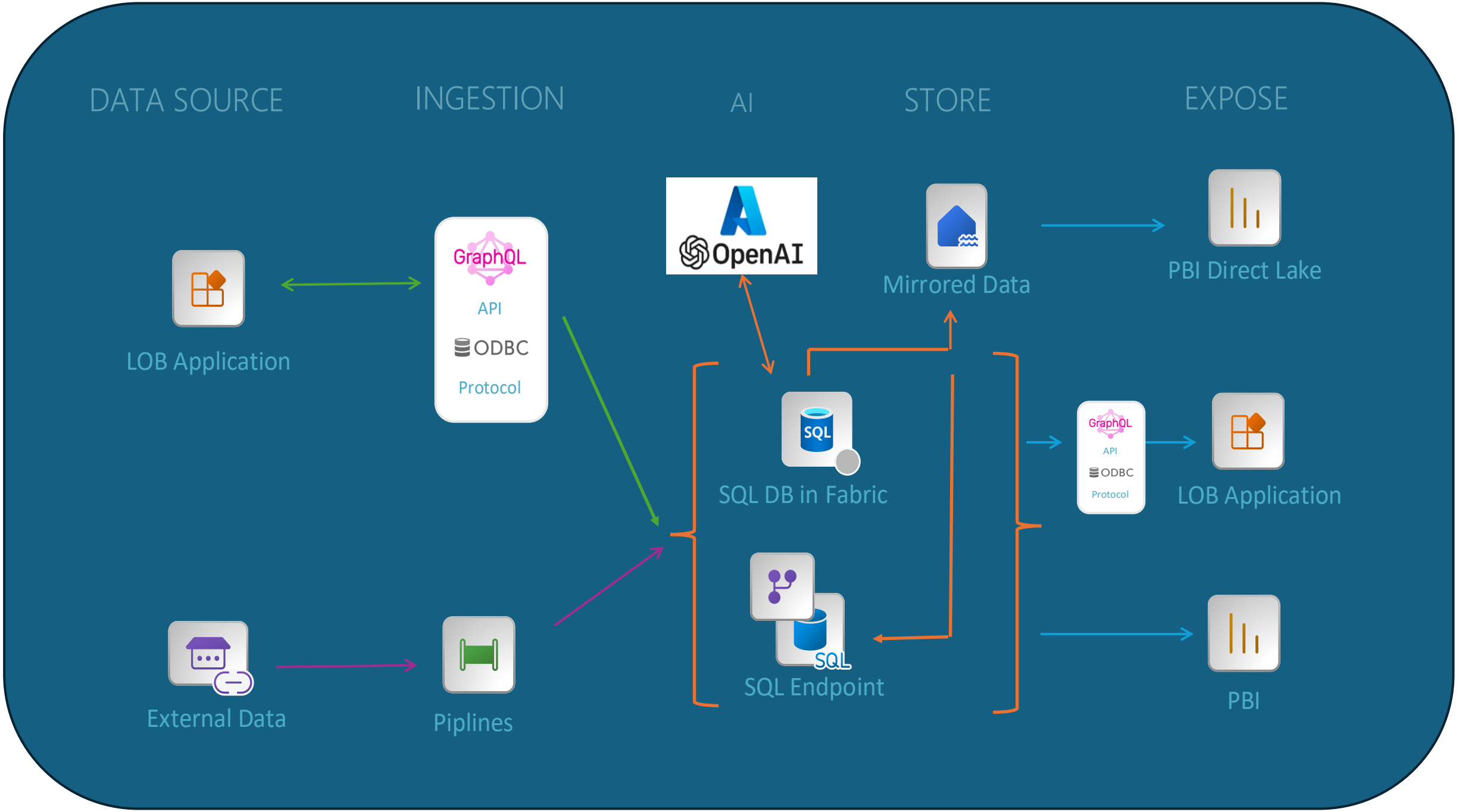
Episode 2: Dataflows, Notebooks, Reports

Episode 3: GenAI and vector databases

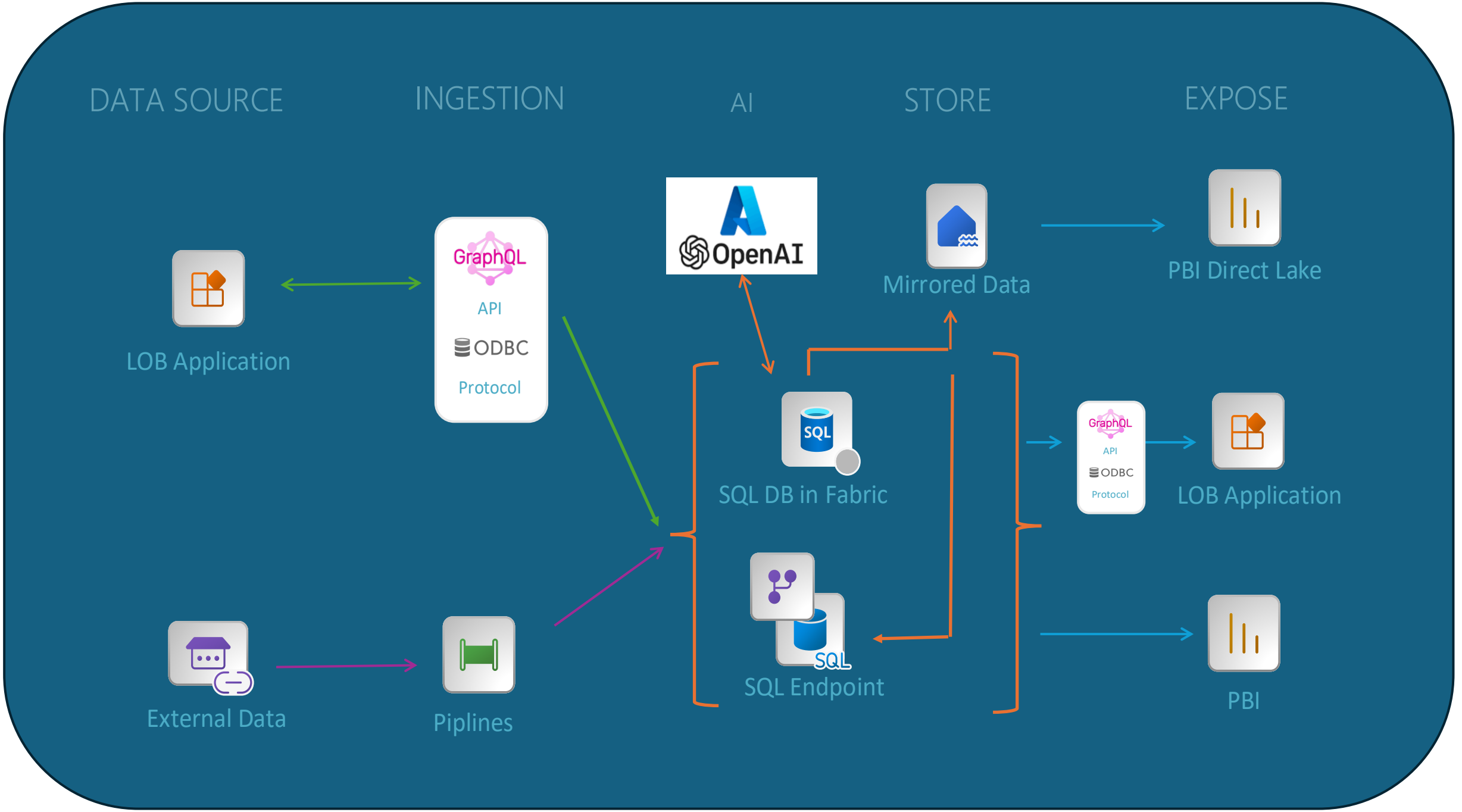
Episode 4: GraphQL and application development

Episode 5: Application lifecycle management

Episode 6: Performance Dashboard, Recap


DATA SOURCE
INGESTION
AI
STORE
EXPOSE
LOB Application
GraphQL
API
ODBC
Protocol
OpenAI
Mirrored Data
PBI Direct Lake
SQL
SQL DB in Fabric
GraphQL
API
ODBC
Protocol
LOB Application
SQL
SQL Endpoint
External Data
Piplines
PBI

Episode 4: Create application using GraphQL


DATA SOURCE
INGESTION
AI
STORE
EXPOSE
LOB Application
GraphQL
API
ODBC
Protocol
OpenAI
Mirrored Data
PBI Direct Lake
SQL
SQL DB in Fabric
GraphQL
API
ODBC
Protocol
LOB Application
SQL
SQL Endpoint
External Data
Piplines
PBI

Create find_sessions

About GraphQL

Using GraphQL

GraphQL Uses

Query and **mutate** data: Use a data service to store data and take advantage of Data API builder to use GraphQL to query and mutate data.

- **GraphQL for Databases**: Use Data API builder to automatically turn your databases into GraphQL endpoints
- **Relational data**: Use Azure SQL Database or Azure Database for PostgreSQL.
- **NoSQL data**: Use Azure Cosmos DB.

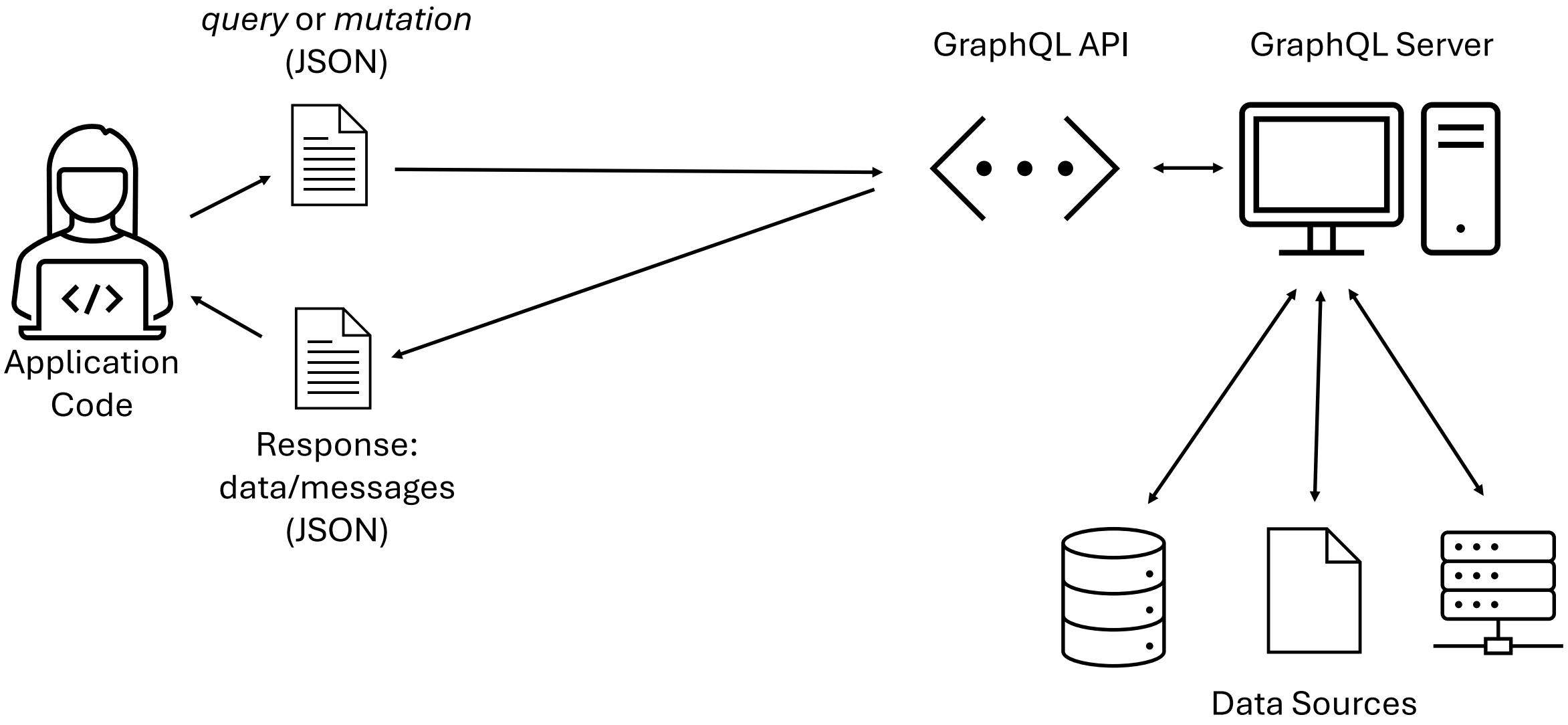
API layer: Use GraphQL APIs in Azure API Management

Hosting: You can bring your existing applications to Azure and take advantage of the benefits of Azure's web app hosting services. Which service depends on how you deploy your application.

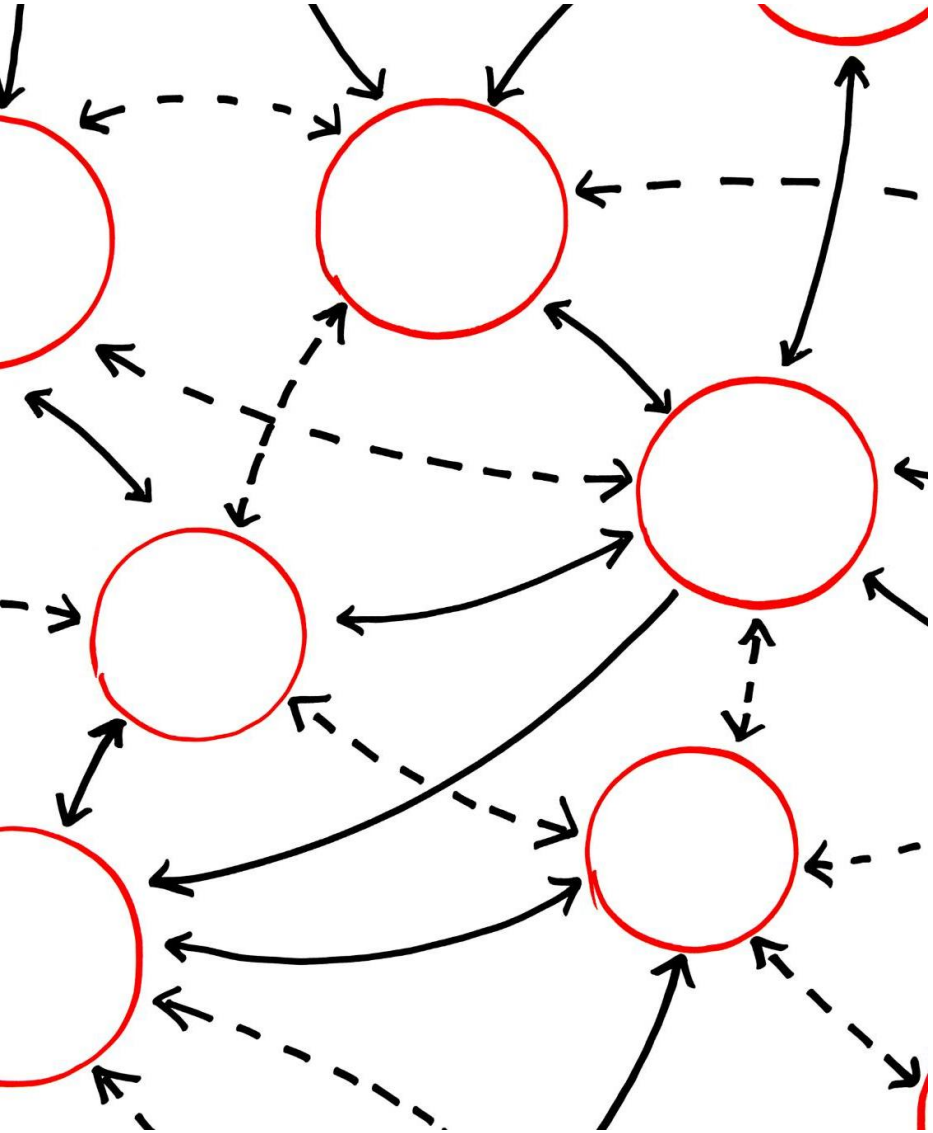
- **Static site**: use Azure Static Web Apps.
- **Server** or **full-stack**
  - Use **Azure App Service**.
  - Use **Azure Container Apps** for containerized applications.
- Serverless: use Azure Functions.

Authentication: Use an Identity platform to add authentication to your application.

GraphQL Data Path



Basic Concepts of GraphQL



GraphQL Schema

What is a GraphQL Schema?

A GraphQL schema defines the types of data that can be queried or mutated, and how they are related. It provides a clear and concise way to describe an API's capabilities.

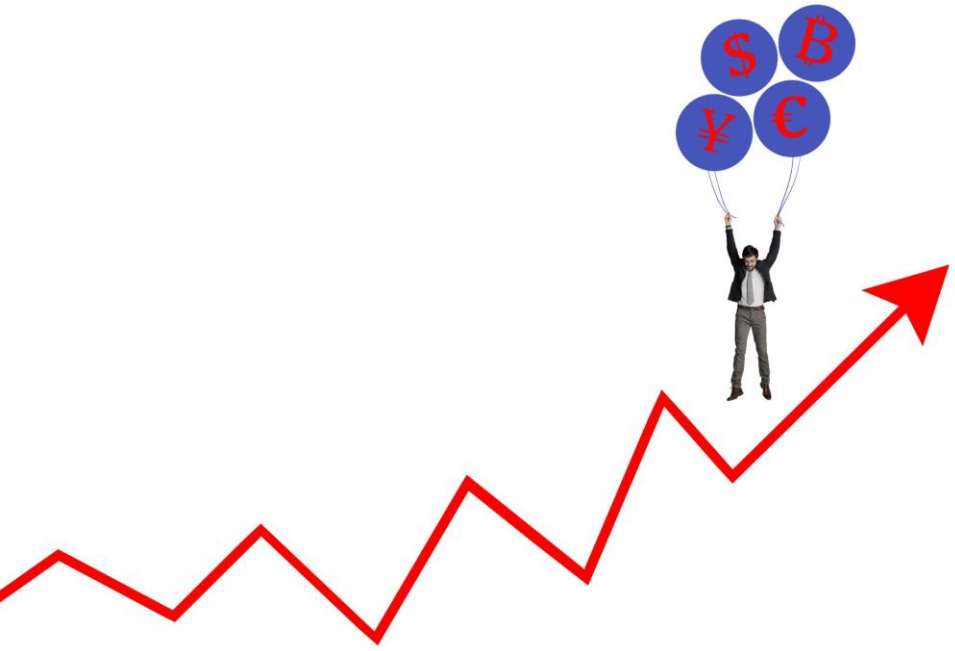
Objects in a GraphQL Schema

In a GraphQL schema, objects are used to represent data types. They provide a way to organize related data and define the fields that can be queried on each type.

Fields in a GraphQL Schema

Fields define the data that can be queried on an object type in a GraphQL schema. They can be scalar values or other object types and can have arguments to filter or modify the data.

Types, Queries, Mutations, and Subscriptions



Types

Types define the shape of the data that can be retrieved and manipulated through a GraphQL API. They provide a structured way to represent the data and define the relationships between different types.

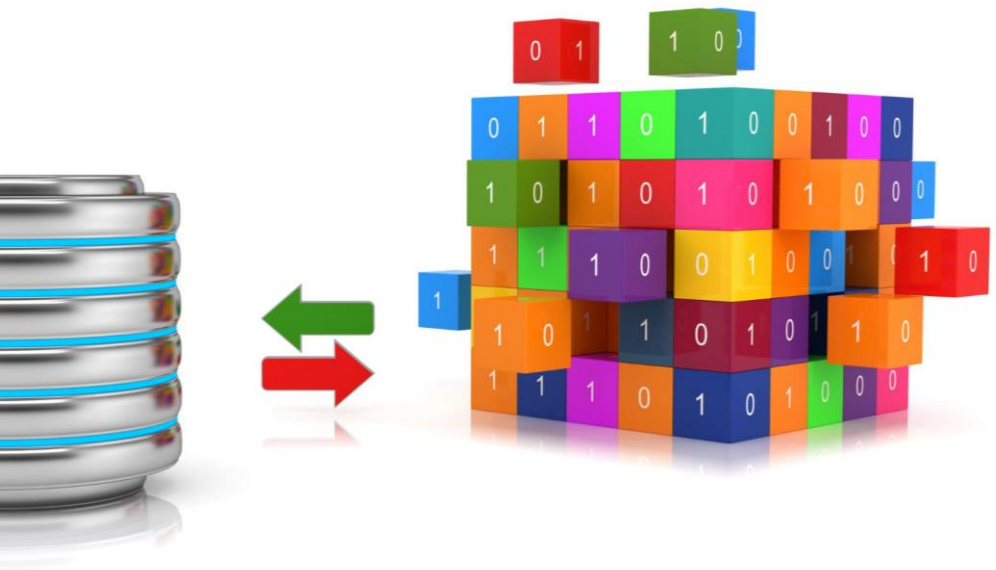
Queries

Queries are used to retrieve data from a GraphQL API. They define what data is being requested, what fields will be returned, and any arguments that may be needed to filter or paginate the data.

Mutations

Mutations are used to modify data in a GraphQL API. They allow for creating, updating, and deleting data, and can also be used to perform other side

GraphQL vs. SQL



GraphQL

GraphQL is a query language for APIs that provides a more efficient, powerful and flexible alternative to traditional REST APIs. GraphQL allows for precise queries and returns only the requested data, resulting in faster and more efficient data retrieval.

SQL

SQL is a standard language for managing and manipulating relational databases. SQL is known for its simplicity and ease of use but can lead to data redundancy and slower performance when handling large amounts of data.

Key Differences and Advantages of GraphQL

GraphQL allows for more precise queries, reduces data redundancy, provides superior performance and flexibility, and supports multiple data sources, making it increasingly popular for modern web and mobile applications.

Querying Data

Querying Data

```
-- Query for all countries
SELECT * FROM Country;
```

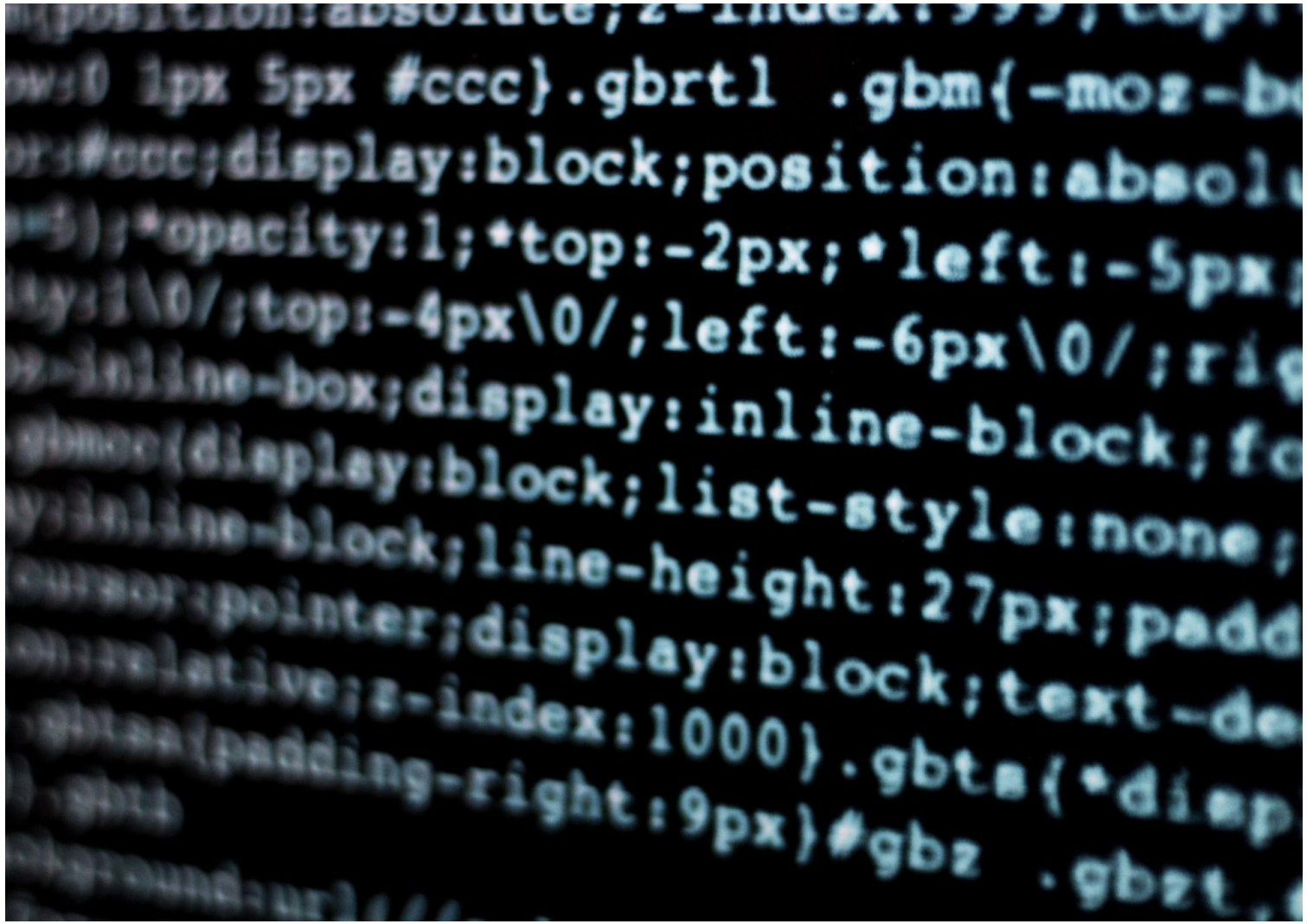
```
query {
  countries {
    code
    name
    capital
    awsRegion
    currency
    emoji
    emojiU
    native
    phone
  }
}
```

SQL

The SELECT statement in SQL is used to retrieve data from a table. It is a fundamental operation in SQL and is used extensively in applications that interact with databases.

GraphQL

GraphQL queries are used to retrieve data from APIs. In contrast to SQL, GraphQL allows clients to specify the structure of the data they want to retrieve, reducing over-fetching and under-fetching of data.



GraphQL Query

A GraphQL *query* is used to retrieve data from an API. The syntax for writing a GraphQL query includes defining fields and arguments, which allows users to specify what data they want to retrieve from the API.

Filtering Data

Filtering and Conditions

```
SELECT code, name, native,
rtl
FROM Language
WHERE code = 'en';
```

```
query
{
  languages(filter: { code: { eq: "en" } }) {
    code
    name
    native
    rtl
  }
}
```

SQL

The *WHERE* clause in SQL is used to filter data based on conditions. It allows us to select a subset of data that meets certain criteria.

GraphQL

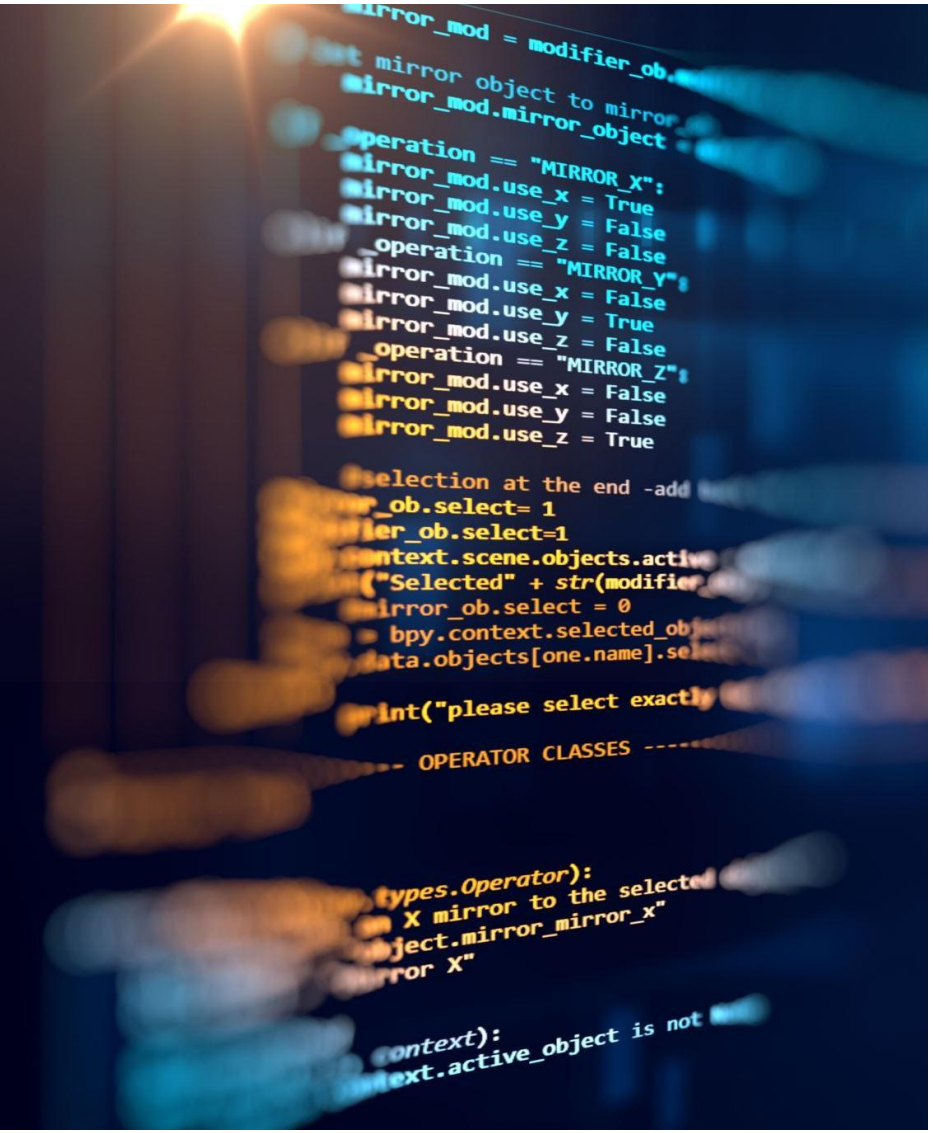
In GraphQL, filtering is done through *arguments* and *filter*. We can filter data based on the type of data, by using comparisons such as equals or not equals, or by using logical operators such as AND and OR.

Fields and Arguments in GraphQL

Fields and *arguments* are essential components of a GraphQL query used to retrieve specific data from an API.

Fields specify what data is requested, while *arguments* provide additional parameters to filter or sort the data.

Using Arguments for Filtering



Arguments in GraphQL

Arguments are used in GraphQL to filter data based on specific conditions. They are used in query fields, query parameters, and directive arguments.

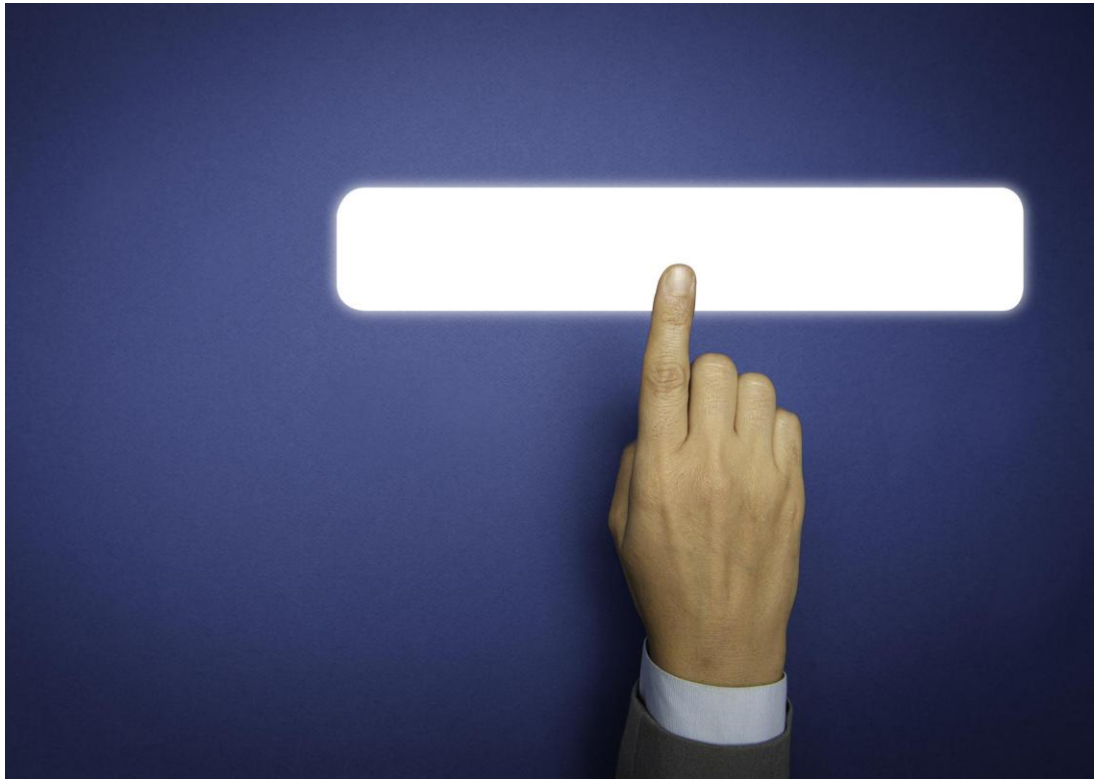
Using Arguments to Filter Data in a Query

Arguments are used in GraphQL queries to filter data based on specific conditions. For example, you can use arguments to filter data based on a date range, a location, or a user ID.

Using Arguments to Filter Data in a Mutation

Arguments are used in GraphQL mutations to filter data based on specific conditions just as in a Query.

Combining Multiple Conditions



Combining Conditions

In GraphQL, you can combine multiple conditions to filter data. This is useful when you want to filter data based on multiple criteria.

Using AND and OR Operators

You can use *AND* and *OR* operators to combine multiple conditions in GraphQL. The *AND* operator is used to combine two or more conditions where all conditions must be true. The *OR* operator is used to combine two or more conditions where at least one condition must be true.

Inserting and Updating Data

```
-- Inserting data
INSERT INTO Country
(code, name, capital, awsRegion, currency, emoji, emojiU,
native, phone, continent_code)
VALUES
('JP', 'Japan', 'Tokyo', 'ap-northeast-1', 'JPY', 'JP', 'U+1F1EF
U+1F1F5', '日本', '+81', 'AS');
```

```
mutation {
  addcountry(
    code: "JP",
    name: "Japan",
    capital: "Tokyo",
    awsRegion: "ap-northeast-1",
    currency: "JPY",
    emoji: "JP",
    emojiU: "U+1F1EF U+1F1F5",
    native: "日本",
    phone: "+81",
    continent_code: "AS"
  )
}
```

Inserting and Altering Data

SQL

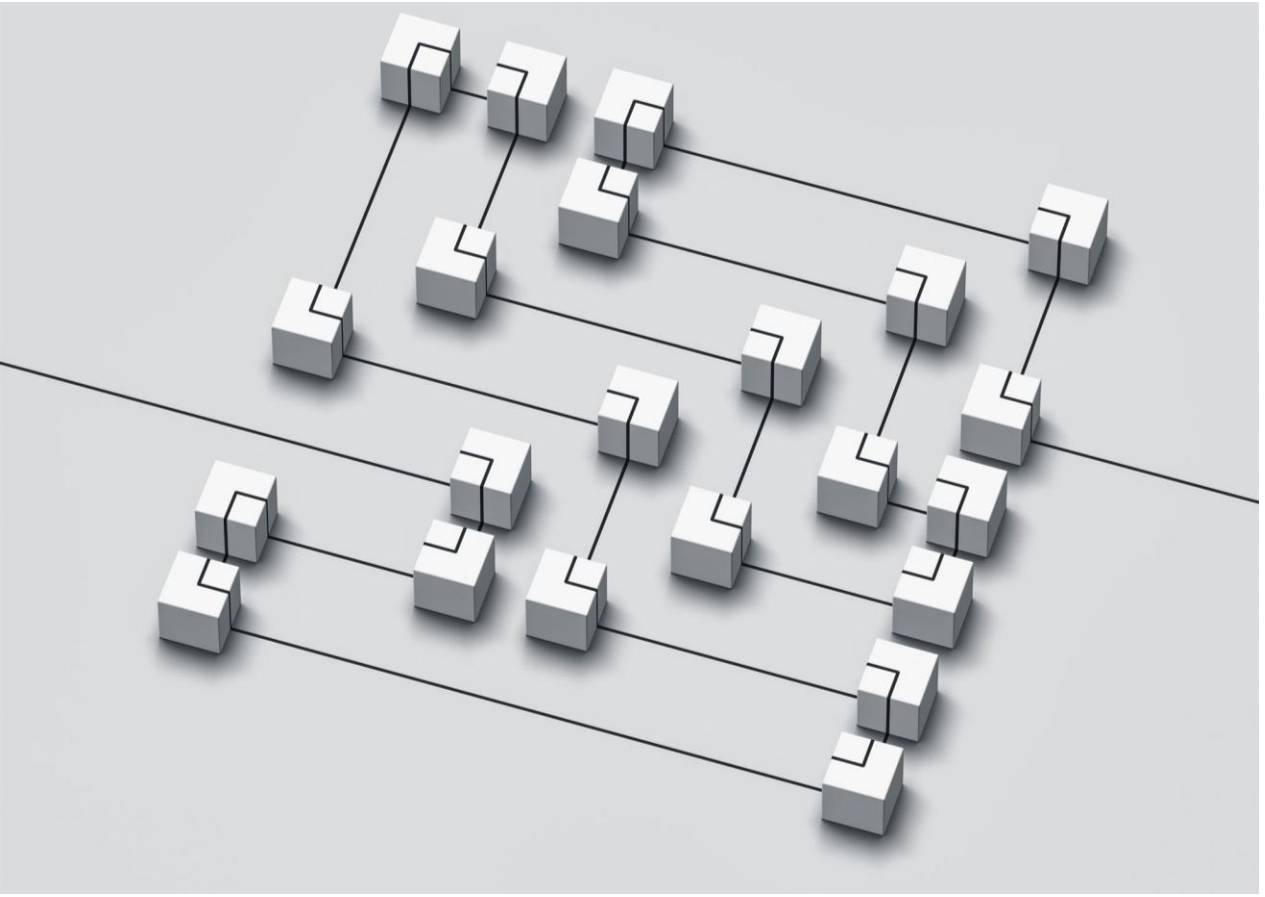
The *INSERT* statement in SQL is used to add new records to a table. *DELETE* removes data, and *UPDATE* changes Data.

GraphQL

GraphQL *mutations* are similar to SQL INSERT, UPDATE and DELETE statements in that they allow us to modify data in a database.

However, unlike SQL INSERT statements, GraphQL mutations can be used to modify multiple tables and perform complex operations. You will need a separate mutation for updates, deletes and inserts.

GraphQL Mutations



GraphQL *mutation* is an essential part of modifying data in the API. The syntax for writing a GraphQL mutation includes *input types* and *payloads* and is used to modify data on the server-side. You have to write or create these.

Dealing with Responses

Handling Responses

When you modify data in GraphQL, you will receive a response from the API for data and messages in JSON format

JavaScript Example

```
fetch('https://your-graphql-endpoint.com/graphql', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'Authorization': 'Bearer YOUR_AUTH_TOKEN'
  },
  body: JSON.stringify({
    query: `
      query {
        countries {
          code
          name
          capital
        }
      }
    `
  })
})
.then(response => response.json())
.then(data => {
  if (data.errors) {
    console.error('GraphQL errors:', data.errors);
  } else {
    console.log('GraphQL data:', data.data);
  }
})
.catch(error => {
  console.error('Network error:', error);
});
```

Create session recommender application

End Episode 4

Overview of GraphQL

Using GraphQL API builder to quickly bootstrap and application