PostgreSQL

Akash Pundir

System Programming

School of Computer Science and Engineering

What is a Database?

A database is **a structured collection of data organized for efficient retrieval, storage, and manipulation**. It serves as a central repository for storing and managing information in a structured manner.

DBMS (Database Management System)

It is **a software system** that facilitates the creation, organization, manipulation, and administration of databases. DBMS **serves as an interface between users or applications and the database itself**, providing a set of tools and functionalities for managing data efficiently

PostgresSQL

PostgreSQL (often abbreviated as Postgres) is a **powerful open-source relational database management system (RDBMS).**

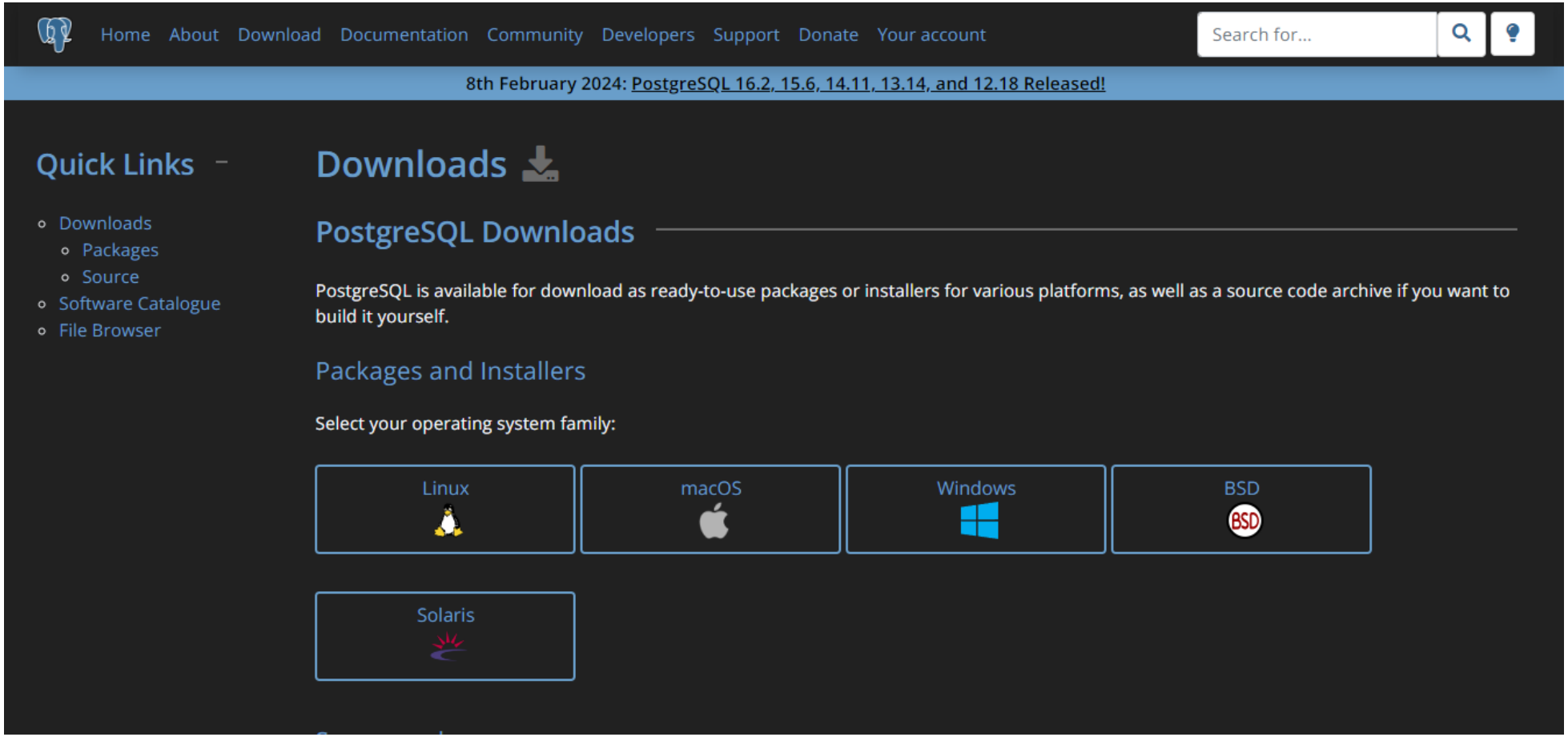
Relational Database Management System (RDBMS)

- In the relational model, data is organized into tables, also known as relations.
- Each table consists of rows (tuples) and columns (attributes).
- Rows represent individual records or instances, while columns represent attributes or properties of those records.

Structured Query Language

- SQL is the standard language for interacting with RDBMS.
- It provides a set of commands for defining, querying, manipulating, and managing relational databases.
- Common SQL commands include SELECT, INSERT, UPDATE, DELETE, CREATE TABLE, ALTER TABLE, DROP TABLE, and many others

Navigate to
https://www.postgresql.org/download/



8th February 2024: PostgreSQL 16.2, 15.6, 14.11, 13.14, and 12.18 Released!

Quick Links

- Downloads
  - Packages
  - Source
- Software Catalogue
- File Browser

Windows installers

Interactive installer by EDB

Download the installer certified by EDB for all supported PostgreSQL versions.

Note! This installer is hosted by EDB and not on the PostgreSQL community servers. If you have issues with the website it's hosted on, please contact webmaster@enterprisedb.com.

This installer includes the PostgreSQL server, pgAdmin; a graphical tool for managing and developing your databases, and StackBuilder; a package manager that can be used to download and install additional PostgreSQL tools and drivers. Stackbuilder includes management, integration, migration, replication, geospatial, connectors and other tools.

This installer can run in graphical or silent install modes.

The installer is designed to be a straightforward, fast way to get up and running with PostgreSQL on Windows.

Advanced users can also download a zip archive of the binaries, without the installer. This download is intended for users who wish to include PostgreSQL as part of another application installer.

Platform support

The installers are tested by EDB on the following platforms. They can generally be expected to run on other comparable versions, for example, desktop releases of Windows:

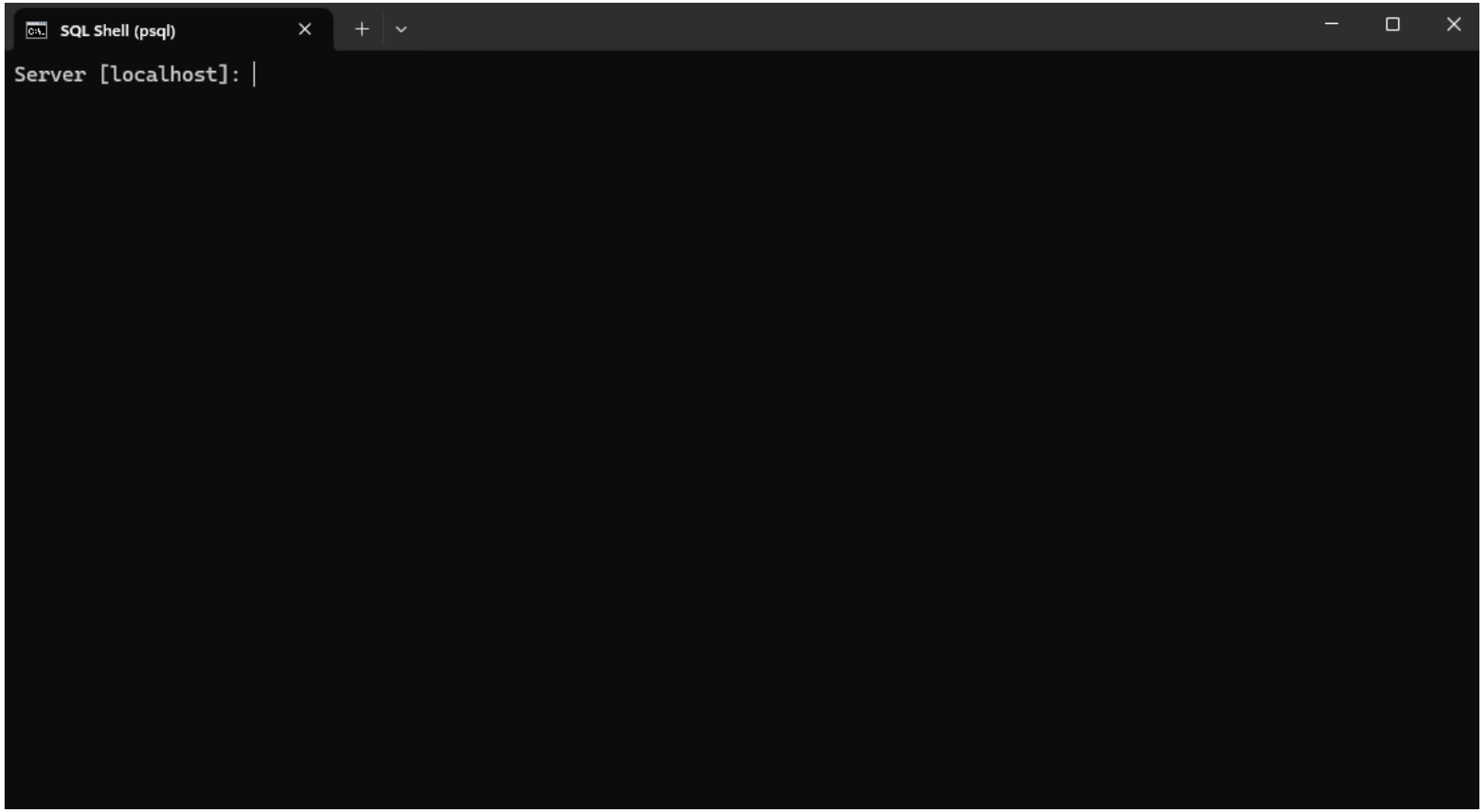
Download PostgreSQL

Open source PostgreSQL packages and installers from EDB

PostgreSQL Version	Linux x86-64	Linux x86-32	Mac OS X	Windows x86-64	Windows x86-32
16.2	postgresql.org	postgresql.org			Not supported
15.6	postgresql.org	postgresql.org			Not supported
14.11	postgresql.org	postgresql.org			Not supported
13.14	postgresql.org	postgresql.org			Not supported

Now, run the installer and complete setup

- **After that open sql shell**



Listing Databases

- To list all databases, use the \l command:

```
\l
```

Connecting to a Specific Database

- To connect to a specific database, use the \c command

\c database_name

Let’s try creating a table

```
CREATE TABLE todos (
   id SERIAL PRIMARY KEY,
   title TEXT NOT NULL,
   completed BOOLEAN NOT NULL
);
```

Listing Tables

- To list all tables in the current database, use the \dt command

\dt

Describing a Table

- To describe the structure of a specific table, use the \d command:

\d table_name

Viewing Data

- To view data from a table, use a simple SQL query

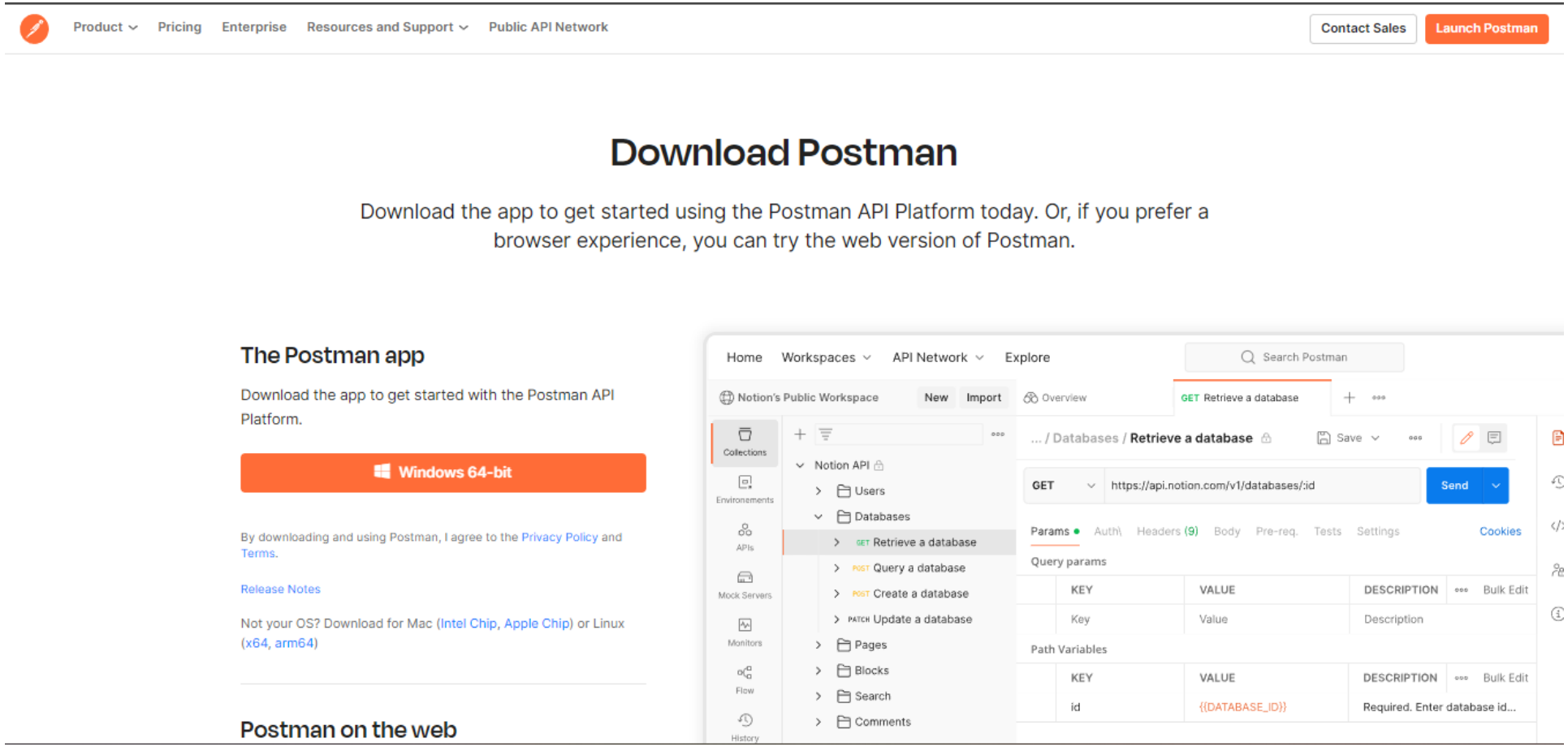
```
SELECT * FROM table_name;
```

Now, let’s install POSTMAN

- Postman simplifies the process of testing APIs by providing a user-friendly interface for sending HTTP requests and viewing responses.
- It supports various request types such as GET, POST, PUT, DELETE, PATCH, etc., allowing users to test different API endpoints and methods.

https://www.postman.com/downloads/

https://www.postman.com/downloads/



Let’s setup a new node project

```
npm init -y
```

Install necessary libraries

npm i express pg

Make a new index.js file in the same folder

```
const express = require('express');
const { Pool } = require('pg');

const app = express();
const port = 3000;

const pool = new Pool({
  user: 'postgres',
  host: 'localhost',
  database: ‘todos',
  password: 'root',
  port: 5432,
});

app.use(express.json());
```

```
// GET all todos
app.get('/todos', (req, res) => {
  pool.query('SELECT * FROM todos', (error, result) =>
{
    if (error) {
      console.error('Error fetching todos', error);
      res.status(500).json({ error: 'Internal server
error' });
    } else {
      res.json(result.rows);
    }
  });
});
```

```
// POST a new todo
app.post('/todos', (req, res) => {
  const { title, completed } = req.body;
  pool.query('INSERT INTO todos (title, completed) VALUES ($1, $2)',
[title, completed], (error) => {
    if (error) {
      console.error('Error creating todo', error);
      res.status(500).json({ error: 'Internal server error' });
    } else {
      res.status(201).json({ message: 'Todo created successfully' });
    }
  });
});
```

```
// PUT update todo
app.put('/todos/:id', (req, res) => {
  const { id } = req.params;
  const { title, completed } = req.body;
  pool.query('UPDATE todos SET title = $1, completed = $2 WHERE id = $3', [title,
completed, id], (error) => {
    if (error) {
      console.error('Error updating todo', error);
      res.status(500).json({ error: 'Internal server error' });
    } else {
      res.json({ message: 'Todo updated successfully' });
    }
  });
});
```

```
// DELETE todo
app.delete('/todos/:id', (req, res) => {
  const { id } = req.params;
  pool.query('DELETE FROM todos WHERE id = $1', [id], (error) => {
    if (error) {
      console.error('Error deleting todo', error);
      res.status(500).json({ error: 'Internal server error' });
    } else {
      res.json({ message: 'Todo deleted successfully' });
    }
  });
});
```

Object Relational Mapping

ORM is a programming technique that allows developers to work with relational databases using object-oriented programming languages, enabling them to interact with database entities as if they were ordinary objects in their code.

```java
@Entity
@Table(name="addresses")
public class Address {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name="id")
    private Long id;

    @Column(name="street")
    private String street;

    @Column(name="city")
    private String city;

    @Column(name="state")
    private String state;

    @Column(name="country")
    private String country;

    @Column(name="zip_code")
    private String zipCode;
```

addresses

- id BIGINT
- city VARCHAR(255)
- country VARCHAR(255)
- state VARCHAR(255)
- street VARCHAR(255)
- zip_code VARCHAR(255)
- order_id BIGINT

Indexes

```java
Address address = new Address();
address.setCity("Pune");
address.setStreet("Kothrud");
address.setState("Maharashtra");
address.setCountry("India");
address.setZipCode("411047");
```

Result Grid Filter Rows: Search

id	city	country	state	street	zip_code
2	Pune	India	Maharashtra	Kothrud	411047
3	Pune	India	Maharashtra	Kothrud	411047

Sequalize

Sequelize is **a popular Object-Relational Mapping (ORM) library for Node.js**, used with SQL databases such as PostgreSQL, MySQL, MariaDB, SQLite, and MSSQL

Key Features

- **Model Definition**: Sequelize allows developers to define models that map directly to database tables. These models specify the structure of the data and the relationships between different tables.
- **Querying**: Sequelize provides methods for executing SQL queries against the database, including selecting, inserting, updating, and deleting records. It supports various query options and conditions.
- **Data Validation**: Sequelize includes built-in support for data validation, allowing developers to define constraints on the data being saved to the database. This helps ensure data integrity and consistency.
- **Associations**: Sequelize enables developers to define relationships between different models, such as one-to-one, one-to-many, and many-to-many associations. These associations are reflected in the database schema and can be used to navigate between related records.

Create a new directory for your project and navigate into it via the terminal.

```
mkdir sequelize-postgres
cd sequelize-postgres
```

Initialize a new Node.js project:

```
npm init -y
```

Install Sequelize, PostgreSQL, and the pg driver:

```
npm install sequelize pg pg-hstore
```

- **sequelize**: This is the main library itself. Sequelize is an ORM that abstracts away the intricacies of SQL queries and provides a simple API for interacting with your database tables as JavaScript objects.

- **pg**: This is the PostgreSQL client for Node.js. Sequelize uses this package to communicate with PostgreSQL databases.

- **pg-hstore**: This is a module that Sequelize uses for managing JSON data in PostgreSQL.

Set up Sequelize:

Create a file named sequelize.js in your project directory:

```
const { Sequelize } = require('sequelize');

// Initialize Sequelize with your PostgreSQL database
credentials
const sequelize = new Sequelize('postgres', 'postgres',
'root', {
  host: 'localhost',
  dialect: 'postgres', // Specify the dialect for PostgreSQL
});

module.exports = sequelize;
```

Define a model:

Create a folder named models in your project directory, and within it, create a file named Todo.js:

```
const { DataTypes } = require('sequelize');
const sequelize = require('../sequelize');

const Todo = sequelize.define('Todo', {
  id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    autoIncrement: true
  },
  title: {
    type: DataTypes.STRING,
    allowNull: false
  },
```

```
completed: {
    type: DataTypes.BOOLEAN,
    allowNull: false,
    defaultValue: false
  }
}, {
  tableName: 'todos', // Match the table name with your
existing database table
  timestamps: false
});

module.exports = Todo;
```

Create an index.js file in your project directory to initialize Sequelize and synchronize the models with the database:

```
const express = require('express');
const sequelize = require('./sequelize');
const Todo = require('./models/Todo');

const app = express();
const PORT = 3000;
```

```
// Test the database connection
sequelize.authenticate()
  .then(() => {
    console.log('Connection has been established successfully.');

    // Synchronize defined models with the database
    return sequelize.sync({ alter: true });
  })
  .then(() => {
    console.log('All models were synchronized successfully.');
  })
  .catch((error) => {
    console.error('Unable to connect to the database:', error);
  });

app.use(express.json());
```

```
// Define endpoints
app.get('/todos', (req, res) => {
  Todo.findAll()
    .then((todos) => {
      res.json(todos);
    })
    .catch((error) => {
      res.status(500).json({ error: 'Internal
server error' });
    });
});
```

```
app.post('/todos', (req, res) => {
  const { title, completed } = req.body;
  Todo.create({ title, completed })
    .then((todo) => {
      res.status(201).json(todo);
    })
    .catch((error) => {
      res.status(400).json({ error: 'Bad request' });
    });
});
```

```
// PUT endpoint to update a todo item
app.put('/todos/:id', (req, res) => {
  const todoId = req.params.id;
  const { title, completed } = req.body;

  Todo.findByPk(todoId)
    .then(todo => {
      if (!todo) {
        return res.status(404).json({ error: 'Todo not found' });
      }

      // Update the todo
      todo.title = title;
      todo.completed = completed;

      // Save the updated todo
      return todo.save();
    })
    .then(updatedTodo => {
      res.json(updatedTodo);
    })
    .catch(error => {
      res.status(500).json({ error: 'Internal server error' });
    });
});
```

```
// DELETE endpoint to delete a todo item
app.delete('/todos/:id', (req, res) => {
  const todoId = req.params.id;

  Todo.findByPk(todoId)
    .then(todo => {
      if (!todo) {
        return res.status(404).json({ error: 'Todo not found' });
      }

      // Delete the todo
      return todo.destroy();
    })
    .then(() => {
      res.status(204).end(); // No content to send back
    })
    .catch(error => {
      res.status(500).json({ error: 'Internal server error' });
    });
});
```

TEST YOUR SKILLS

You are tasked with building a basic user management system using Node.js, Express.js, Sequelize, and PostgreSQL. The system should have the following features:

- **User Model: Create a User model with the following fields:**
  - **id (Primary Key, Auto-incrementing Integer)**
  - **username (String, Unique, Not Null)**
  - **email (String, Unique, Not Null)**
  - **password (String, Not Null)**
- **Endpoints**
  - **POST /users: Create a new user. The request body should contain username, email, and password. Return the created user in the response.**
  - **GET /users: Retrieve all users.**
  - **GET /users/:id: Retrieve a specific user by ID.**
  - **PUT /users/:id: Update a specific user by ID. Allow updating username, email, and password.**
  - **DELETE /users/:id: Delete a specific user by ID.**

- **Database Connection: Establish a connection to a PostgreSQL database using Sequelize. Ensure the connection is successful before starting the server.**