PENGEMBANGAN WEB (TEORI)

LAPORAN EKSPERIMEN MENGENAI PERBANDINGAN DATABASE CONNECTION POLLING VS. LAZY LOADING UNTUK MENGATASI MASALAH KINERJA AKSES DATABASE YANG LAMBAT PADA SPRING BOOT

Laporan ini disusun untuk memenuhi tugas 1 mata kuliah Pengembangan Web (Teori)

Disusun oleh kelompok B4:

Asri Husnul Rosadi	221524035
Faris Abulkhoir	221524040
Mahardika Pratama	221524044
Muhamad Fahri Yuwan	221524047
Najib Alimudin Fajri	221524053
Septyana Agustina	221524058
Sarah	221524059

Dosen Pengampu:
Joe Lian Min, M.Eng.

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA
PROGRAM STUDI D4 TEKNIK INFORMATIKA
POLITEKNIK NEGERI BANDUNG
2024

DAFTAR ISI

A. **IDENTIFIKASI PROBLEM**

Aplikasi yang dikembangkan menggunakan Spring Boot sering menghadapi masalah kinerja ketika berinteraksi dengan database, terutama dalam skenario yang melibatkan volume data besar dan akses data yang kompleks. Dua pendekatan yang umum digunakan untuk mengatasi masalah ini adalah **Database Connection Pooling** dan **Lazy Loading**. Database Connection Pooling berfokus pada manajemen koneksi database dengan mendaur ulang koneksi yang ada, sehingga mengurangi overhead dari pembuatan koneksi baru. Di sisi lain, Lazy Loading menunda pengambilan data sampai benar-benar dibutuhkan, menghindari pemuatan data yang tidak perlu dan mengurangi beban pada sistem. Identifikasi problem ini berfokus pada bagaimana kedua teknik ini mempengaruhi kinerja aplikasi, termasuk waktu respon dan efisiensi sumber daya, serta mencari solusi yang optimal untuk meningkatkan performa akses database.

B. **DESKRIPSI PROBLEM**

Masalah utama yang dihadapi adalah **kinerja akses database yang lambat** pada aplikasi Spring Boot, yang dapat disebabkan oleh cara aplikasi mengelola koneksi database dan pengambilan data. Ketika aplikasi berusaha untuk mengakses atau memproses data dari database, tanpa adanya teknik yang efisien, proses ini bisa menjadi lambat dan mempengaruhi kinerja keseluruhan aplikasi. **Database Connection Pooling** mencoba untuk memperbaiki masalah ini dengan mengurangi overhead dari pembuatan koneksi database yang baru setiap kali ada permintaan, sementara **Lazy Loading** berusaha mengurangi beban dengan hanya memuat data yang diperlukan pada saat itu saja. Masing-masing teknik ini memiliki kelebihan dan kekurangan tergantung pada kebutuhan aplikasi dan pola akses data. Evaluasi yang komprehensif diperlukan untuk menentukan metode mana yang lebih efektif dalam meningkatkan performa dan responsivitas aplikasi dalam skenario yang berbeda.

C. **METODOLOGI EKSPERIMEN**

1) Tujuan

   Tujuan dari eksperimen ini adalah untuk membandingkan efektivitas teknik **Database Connection Pooling** dan **Lazy Loading** dalam meningkatkan kinerja akses database pada aplikasi Spring Boot. Eksperimen ini bertujuan untuk menentukan teknik mana yang memberikan performa lebih baik dalam hal waktu respon, efisiensi

penggunaan sumber daya, dan skalabilitas saat menangani akses data yang kompleks dan volume data besar.

2) Desain Eksperimen

Eksperimen ini akan dilakukan dengan membandingkan dua konfigurasi aplikasi Spring Boot yang sama dalam hal struktur kode dan data yang diakses. Konfigurasi pertama akan menggunakan **Database Connection Pooling** untuk mengelola koneksi database, sedangkan konfigurasi kedua akan menggunakan **Lazy Loading** untuk penanganan data. Setiap konfigurasi akan diuji menggunakan skenario beban yang sama untuk memastikan perbandingan yang adil.

- **Konfigurasi A**: Menggunakan **Database Connection Pooling** dengan HikariCP.
- **Konfigurasi B**: Menggunakan **Lazy Loading** dengan Hibernate.

Kedua konfigurasi akan diuji dengan dataset yang sama dan skenario akses data yang serupa untuk menilai perbedaan kinerja secara objektif.

3) Variabel Eksperimen

- **Variabel Independen**: Teknik yang digunakan untuk pengelolaan data (Database Connection Pooling vs. Lazy Loading).
- Variabel Terikat:
    - Response Time: Waktu yang dibutuhkan untuk menyelesaikan permintaan database.
    - Throughput: Jumlah permintaan yang dapat diproses dalam satuan waktu.
    - Memory Usage: Penggunaan memori oleh aplikasi selama pengujian.
    - Error Rate: Jumlah kesalahan atau exception yang terjadi selama pengujian.
- Variabel Kontrol:
    - Volume Data: Ukuran dan kompleksitas dataset yang digunakan.
    - Load: Jumlah permintaan simultan yang dilakukan selama pengujian.
    - Aplikasi: Struktur aplikasi yang sama untuk kedua konfigurasi.

4) Langkah-Langkah Pengujian

- Persiapan:
    - Konfigurasikan dua instance aplikasi Spring Boot dengan masing-masing teknik yang akan diuji.

    - Siapkan database dan dataset yang identik untuk kedua konfigurasi.
    - Atur alat pengujian (seperti Apache JMeter) untuk mengukur kinerja.

- Implementasi:
    - Konfigurasi A: Implementasikan dan konfigurasi Database Connection Pooling menggunakan HikariCP pada aplikasi.
    - Konfigurasi B: Implementasikan dan konfigurasi Lazy Loading pada aplikasi menggunakan Hibernate.
- Pengujian:
    - Lakukan pengujian performa dengan melakukan serangkaian permintaan ke database menggunakan alat pengujian untuk mengukur Response Time, Throughput, dan Memory Usage.
    - Uji kedua konfigurasi di bawah beban yang sama untuk memastikan perbandingan yang adil.
- Pengumpulan Data:
    - Catat data kinerja dari setiap pengujian, termasuk waktu respon, throughput, penggunaan memori, dan tingkat error.
- Analisis Data:
    - Bandingkan hasil antara Database Connection Pooling dan Lazy Loading berdasarkan metrik yang diukur.
    - Identifikasi kelebihan dan kekurangan dari setiap teknik dalam konteks aplikasi yang diuji.

5) Analisis Hasil:

Setelah pengujian selesai, data yang dikumpulkan akan dianalisis untuk mengevaluasi efektivitas masing-masing teknik dalam meningkatkan kinerja aplikasi. Analisis akan mencakup:

- Perbandingan Response Time: Menilai teknik mana yang memberikan waktu respon lebih cepat dan bagaimana hal ini mempengaruhi pengalaman pengguna.

- Perbandingan Throughput: Mengidentifikasi teknik yang dapat menangani lebih banyak permintaan per unit waktu.
- Evaluasi Memory Usage: Membandingkan penggunaan memori untuk kedua teknik dan dampaknya pada performa aplikasi.
- Tingkat Error: Menilai seberapa stabil masing-masing teknik dalam hal jumlah kesalahan yang terjadi selama pengujian.

D. PELAKSANAAN EKSPERIMEN

Implementasi Database Connection Pooling dengan HikariCP

1) Pembuatan Database

```
USE test_db;


CREATE TABLE app_user (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE
);

INSERT INTO app_user (name, email) VALUES ('Alice',
'alice@example.com');
INSERT INTO app_user (name, email) VALUES ('Bob',
'bob@example.com');
INSERT INTO app_user (name, email) VALUES ('Charlie',
'charlie@example.com');
```

2) Buat Proyek Springboot:

   Anda bisa menggunakan Spring Initializr untuk membuat proyek Spring Boot dengan dependensi berikut:

   - Spring Web
   - Spring Data JPA
   - H2 Database (untuk database sederhana, dapat diganti dengan database lain sesuai kebutuhan)

3) Konfigurasikan application.properties

```
# application.properties
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=password
spring.datasource.driver-class-name=org.h2.Driver
```

```
# HikariCP specific settings
spring.datasource.hikari.connection-timeout=30000
spring.datasource.hikari.maximum-pool-size=10
spring.datasource.hikari.idle-timeout=600000
spring.datasource.hikari.max-lifetime=1800000
```

4) Pengembangan Aplikasi
Buat entitas sederhana untuk tabel database.

```
// src/main/java/com/example/demo/entity/User.java
package com.example.demo.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;

    // Getters and Setters
}
```

Buat repository untuk entitas.

```
//
src/main/java/com/example/demo/repository/UserRepository.java
package com.example.demo.repository;

import com.example.demo.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User,
Long> {
}
```

Buat service untuk melakukan operasi pada entitas.

```
// src/main/java/com/example/demo/service/UserService.java
package com.example.demo.service;

import com.example.demo.entity.User;
import com.example.demo.repository.UserRepository;
```

```
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    public List<User> getAllUsers() {
        return userRepository.findAll();
    }

    public User saveUser(User user) {
        return userRepository.save(user);
    }
}
```

Buat controller untuk API.

```
//
src/main/java/com/example/demo/controller/UserController.java
package com.example.demo.controller;

import com.example.demo.entity.User;
import com.example.demo.service.UserService;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/users")
public class UserController {

    @Autowired
    private UserService userService;

    @GetMapping
    public List<User> getAllUsers() {
        return userService.getAllUsers();
    }

    @PostMapping
    public User createUser(@RequestBody User user) {
        return userService.saveUser(user);
    }
}
```

5) Menjalankan Layanan
   a. Jalankan aplikasi dengan perintah:

```
./mvnw spring-boot:run   # Untuk Maven
./gradlew bootRun        # Untuk Gradle
```

6) Pengujian Beban dengan Apache-Jmeter
   Instalasi Apache JMeter
   Unduh dan instal Apache JMeter dari situs resmi Apache.
   Membuat Tes Beban
   a. Buka JMeter: Jalankan JMeter dengan menjalankan jmeter.bat (Windows) atau jmeter (Linux/Mac) dari direktori bin JMeter.
   b. Konfigurasi Tes:
      - Menambahkan Thread Group:
        - Klik kanan pada "Test Plan" > "Add" > "Threads (Users)" > "Thread Group".
        - Set jumlah thread (misalnya, 100), ramp-up period (misalnya, 10 detik), dan loop count (misalnya, 10).
      - Menambahkan HTTP Request Sampler:
        - Klik kanan pada "Thread Group" > "Add" > "Sampler" > "HTTP Request".
        - Set "Server Name or IP" ke localhost dan "Port Number" ke 3000.
        - Set "Path" ke /api/data dan "Method" ke GET atau POST sesuai kebutuhan.
        - Untuk POST, tambahkan parameter dalam tab "Body Data".
   c. Menambahkan Listener untuk Melihat Hasil:
      Klik kanan pada "Thread Group" > "Add" > "Listener" > "View Results Tree" atau "Summary Report".
   d. Menjalankan Tes:
      - Klik ikon start (berwarna hijau) di toolbar untuk menjalankan tes.
      - Monitor hasil pengujian di listener yang telah ditambahkan.

Implementasi Lazy Loading

1) Pembuatan Database

```
USE test_db;

CREATE TABLE user (
    id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(100) NOT NULL UNIQUE
);
```

```
CREATE TABLE profile (
    id INT AUTO_INCREMENT PRIMARY KEY,
    user_id INT NOT NULL,
    bio TEXT,
    FOREIGN KEY (user_id) REFERENCES user(id)
);


INSERT INTO user (name, email) VALUES ('Alice',
'alice@example.com');
INSERT INTO user (name, email) VALUES ('Bob',
'bob@example.com');
INSERT INTO user (name, email) VALUES ('Charlie',
'charlie@example.com');

INSERT INTO profile (user_id, bio) VALUES (1, 'Alice is a
software engineer.');
INSERT INTO profile (user_id, bio) VALUES (2, 'Bob is a
graphic designer.');
INSERT INTO profile (user_id, bio) VALUES (3, 'Charlie is a
data scientist.');
```

2) Buat Proyek Springboot:

   Anda bisa menggunakan Spring Initializr untuk membuat proyek Spring Boot dengan dependensi berikut:

   - Spring Web
   - Spring Data JPA
   - H2 Database (untuk database sederhana, dapat diganti dengan database lain sesuai kebutuhan

3) Konfigurasikan application.properties

```
# application.properties
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=password
spring.datasource.driver-class-name=org.h2.Driver

# HikariCP specific settings
spring.datasource.hikari.connection-timeout=30000
spring.datasource.hikari.maximum-pool-size=10
spring.datasource.hikari.idle-timeout=600000
spring.datasource.hikari.max-lifetime=1800000
```

4) Pengembangan Aplikasi
   Buat entitas sederhana untuk tabel database.

```
// src/main/java/com/example/demo/entity/User.java
package com.example.demo.entity;
```

```
import jakarta.persistence.*;
import java.util.HashSet;
import java.util.Set;

@Entity
public class User {

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

    private String name;

    private String email;

    @OneToOne(mappedBy = "user", fetch = FetchType.LAZY,
cascade = CascadeType.ALL)
    private Profile profile;

    // Constructors, getters, setters
}

// src/main/java/com/example/demo/entity/Profile.java
package com.example.demo.entity;

import jakarta.persistence.*;

@Entity
public class Profile {

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

    @OneToOne
    @JoinColumn(name = "user_id")
    private User user;

    private String bio;

    // Constructors, getters, setters
}
```

Buat repository untuk entitas.

```
//
src/main/java/com/example/demo/repository/UserRepository.java
package com.example.demo.repository;

import com.example.demo.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
```

```
public interface UserRepository extends JpaRepository<User,
Long> {
}

//
src/main/java/com/example/demo/repository/ProfileRepository.jav
a
package com.example.demo.repository;

import com.example.demo.entity.Profile;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ProfileRepository extends
JpaRepository<Profile, Long> {
}
```

Buat service untuk melakukan operasi pada entitas.

```
// src/main/java/com/example/demo/service/UserService.java
package com.example.demo.service;

import com.example.demo.entity.User;
import com.example.demo.repository.UserRepository;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import jakarta.transaction.Transactional;

@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    @Transactional
    public User getUser(Long id) {
        return userRepository.findById(id).orElse(null);
    }

    @Transactional
    public void saveUser(User user) {
        userRepository.save(user);
    }
}
```

Buat controller untuk API.

```
//
src/main/java/com/example/demo/controller/UserController.java
package com.example.demo.controller;
```

```
import com.example.demo.entity.User;
import com.example.demo.service.UserService;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/users")
public class UserController {

    @Autowired
    private UserService userService;

    @GetMapping("/{id}")
    public User getUser(@PathVariable Long id) {
        return userService.getUser(id);
    }
}
```

5) Menjalankan Layanan
   a. Jalankan aplikasi dengan perintah:

```
./mvnw spring-boot:run   # Untuk Maven

./gradlew bootRun        # Untuk Gradle
```

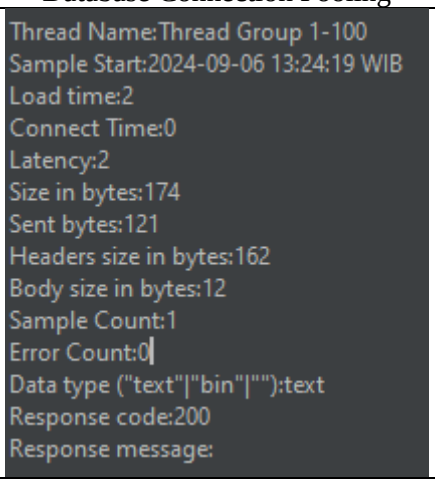
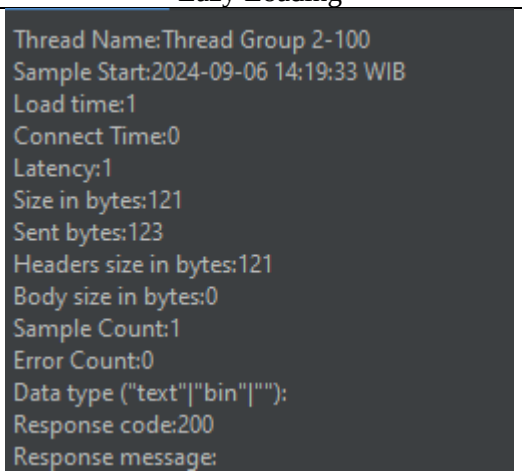
6) Pengujian Beban dengan Apache-Jmeter
   Instalasi Apache JMeter
   Unduh dan instal Apache JMeter dari situs resmi Apache.
   Membuat Tes Beban
   a. Buka JMeter: Jalankan JMeter dengan menjalankan jmeter.bat (Windows) atau jmeter (Linux/Mac) dari direktori bin JMeter.
   b. Konfigurasi Tes:
      - Menambahkan Thread Group:
        - Klik kanan pada "Test Plan" > "Add" > "Threads (Users)" > "Thread Group".
        - Set jumlah thread (misalnya, 100), ramp-up period (misalnya, 10 detik), dan loop count (misalnya, 10).
      - Menambahkan HTTP Request Sampler:

- Klik kanan pada "Thread Group" > "Add" > "Sampler" > "HTTP Request".
- Set "Server Name or IP" ke localhost dan "Port Number" ke 8080.
- Set "Path" ke /api/data dan "Method" ke GET atau POST sesuai kebutuhan.
- Untuk POST, tambahkan parameter dalam tab "Body Data".

c. Menambahkan Listener untuk Melihat Hasil:
Klik kanan pada "Thread Group" > "Add" > "Listener" > "View Results Tree" atau "Summary Report".

d. Menjalankan Tes:
   - Klik ikon start (berwarna hijau) di toolbar untuk menjalankan tes.
   - Monitor hasil pengujian di listener yang telah ditambahkan.

E. ANALISIS HASIL EKSPERIMEN

Database Connection Pooling	Lazy Loading
	

Berdasarkan hasil pengujian menggunakan Apache JMeter, analisis kinerja antara Database Connection Pooling dan Lazy Loading menunjukkan perbedaan dalam beberapa metrik penting. Database Connection Pooling memiliki waktu load sebesar 2 milidetik dengan latensi yang sama, yaitu 2 milidetik, dan tidak ada kesalahan yang tercatat selama pengujian, yang menunjukkan efisiensi dalam memproses permintaan. Hal ini mengindikasikan bahwa Connection Pooling dapat meningkatkan performa dengan memanfaatkan koneksi yang sudah ada, sehingga mengurangi waktu yang diperlukan untuk membuat koneksi baru setiap kali permintaan diterima.

Sementara itu, pengujian pada Lazy Loading menunjukkan waktu load yang lebih cepat, yaitu 1 milidetik dengan latensi yang sama, 1 milidetik, tanpa ada kesalahan yang terjadi. Hal ini menandakan bahwa Lazy Loading bekerja dengan baik dalam menunda inisialisasi objek sampai benar-benar diperlukan, yang mengurangi beban awal saat pengambilan data. Namun, Lazy Loading menghasilkan ukuran body yang lebih kecil dibandingkan dengan Database Connection Pooling, yang bisa menjadi indikasi bahwa hanya data minimum yang diakses sesuai kebutuhan saat itu.

Secara keseluruhan, kedua metode menunjukkan kinerja yang efisien dengan respons time yang rendah, tetapi Database Connection Pooling lebih unggul dalam konsistensi pengelolaan koneksi database, sementara Lazy Loading lebih efisien dalam memori dan waktu akses pada permintaan awal. Pemilihan antara kedua metode ini harus mempertimbangkan kebutuhan spesifik aplikasi, seperti frekuensi akses database dan kompleksitas data yang diambil.

F. **KESIMPULAN**

Kesimpulan dari praktikum ini menunjukkan bahwa kedua teknik, Database Connection Pooling dan Lazy Loading, memiliki keunggulan masing-masing dalam meningkatkan kinerja aplikasi berbasis Spring Boot yang terhubung dengan database.

Database Connection Pooling menunjukkan performa yang stabil dengan waktu load yang sedikit lebih tinggi namun konsisten, karena mekanisme ini mengelola koneksi database secara efisien dengan menggunakan kembali koneksi yang sudah ada, sehingga mengurangi overhead yang muncul saat membuat koneksi baru. Metode ini sangat efektif untuk aplikasi dengan beban koneksi database yang tinggi dan sering, karena dapat meminimalisir waktu tunggu dan meningkatkan throughput secara keseluruhan.

Di sisi lain, Lazy Loading memberikan waktu load yang lebih cepat dalam inisialisasi data, dengan menunda pengambilan data sampai benar-benar dibutuhkan. Hal ini membuat Lazy Loading sangat cocok untuk skenario di mana akses data tidak selalu diperlukan secara langsung atau untuk objek-objek yang jarang digunakan. Namun, implementasi ini perlu lebih berhati-hati pada kompleksitas data yang akan diload secara bertahap, karena dapat meningkatkan waktu akses jika tidak diatur dengan baik.

Dari hasil praktikum ini, dapat disimpulkan bahwa Database Connection Pooling lebih unggul dalam hal manajemen koneksi yang konsisten dan efisien untuk aplikasi dengan akses database yang intensif, sementara Lazy Loading lebih sesuai untuk skenario dengan kebutuhan akses data yang lebih jarang dan spesifik. Pemilihan metode yang tepat harus disesuaikan dengan kebutuhan aplikasi dan pola penggunaan data untuk memaksimalkan kinerja secara keseluruhan.