ЛЕКЦІЯ №2.4. РОБОТА З БАЗАМИ ДАНИХ В SPRING BOOT

Кафедра інформаційних технологій та систем електронних комунікацій
Спеціальність 122 «Комп'ютерні науки», ступінь вищої освіти «бакалавр»
VI семестр
Дисципліна «Web-технології та web-дизайн»
ЗМІСТОВИЙ МОДУЛЬ 2. РОЗРОБКА ВЕБ-ДОДАТКІВ НА SPRING BOOT

Вступ

Більшість сучасних веб-додатків працює з великими обсягами інформації, які необхідно зберігати, змінювати, аналізувати та швидко отримувати. Саме тому бази даних є одним із фундаментальних компонентів будь-якої backend-системи. У процесі розробки веб-застосунків програміст взаємодіє з користувачами, товарами, замовленнями, транзакціями, повідомленнями, журналами подій та іншими сутностями, які необхідно надійно зберігати у структурованому вигляді.

У Java-розробці одним із найпоширеніших рішень для роботи з реляційними базами даних є використання Spring Boot у поєднанні з Spring Data JPA та Hibernate. Подібна технологічна зв'язка дозволяє значно спростити доступ до бази даних, автоматизувати SQL-операції та мінімізувати кількість шаблонного коду.

Spring Boot забезпечує інтеграцію з великою кількістю систем управління базами даних, серед яких:

- MySQL;
- PostgreSQL;
- MariaDB;
- Oracle Database;
- Microsoft SQL Server;
- H2 Database.

Однією з ключових переваг Spring Boot є підтримка ORM-підходу (Object Relational Mapping). ORM дозволяє працювати з таблицями бази даних через Java-об'єкти, а не через прямі SQL-запити. Це значно спрощує розробку складних веб-додатків та підвищує підтримуваність програмного коду.

У межах лекції розглядаються принципи інтеграції Spring Boot із базами даних, конфігурація DataSource, Entity-класи, Hibernate, Spring Data JPA, Repository Layer, CRUD-операції, транзакції, зв'язки між таблицями та особливості ORM-підходу у сучасній Java-розробці.

Архітектура взаємодії Spring Boot із базою даних

У сучасних backend-системах доступ до бази даних зазвичай реалізується через багаторівневу архітектуру.

Типова структура містить:

- Controller Layer;
- Service Layer;
- Repository Layer;
- Database Layer.

Controller приймає HTTP-запити.

Service містить бізнес-логіку.

Repository виконує взаємодію з базою даних.

Database Layer забезпечує фізичне зберігання інформації.

Схематично взаємодію можна подати так:

Client
↓
Controller
↓
Service
↓
Repository
↓
Hibernate / JPA
↓
Database

Подібна архітектура дозволяє розділяти відповідальність між компонентами системи та спрощує підтримку коду.

Підключення бази даних у Spring Boot

Для роботи з реляційними базами даних у Spring Boot зазвичай використовуються starter-залежності.

Основною залежністю є:

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

Цей starter автоматично підключає:

- Hibernate;
- Spring Data JPA;

- EntityManager;
- транзакційний менеджер.

Для роботи з MySQL необхідно додатково підключити JDBC-driver:

```
<dependency>
   <groupId>com.mysql</groupId>
   <artifactId>mysql-connector-j</artifactId>
</dependency>
```

Після цього необхідно налаштувати DataSource.

Конфігурація DataSource

Параметри підключення до бази даних задаються через:

application.properties

Приклад:

```
spring.datasource.url=jdbc:mysql://localhost:3306/testdb
spring.datasource.username=root
spring.datasource.password=1234
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

У наведеному прикладі:

- spring.datasource.url задає URL бази даних;
- username та password визначають облікові дані;
- ddl-auto визначає поведінку Hibernate;
- show-sql дозволяє відображати SQL-запити у консолі.

Hibernate та ORM-підхід

Hibernate є ORM-фреймворком для Java.

ORM (Object Relational Mapping) — це технологія перетворення таблиць бази даних у Java-об’єкти.

Без ORM програміст повинен був би вручну:

- створювати SQL-запити;
- отримувати ResultSet;
- перетворювати рядки таблиці у Java-об’єкти.

Hibernate автоматизує цей процес.

Наприклад, таблиця users:

id	username	email
1	ivan	ivan@gmail.com

може бути представлена Java-класом:

```
@Entity
@Table(name = "users")
public class User {

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

    private String username;

    private String email;
}
```

Hibernate автоматично перетворює об’єкти User у SQL-запити.

Entity-класи у Spring Boot

Entity є Java-класом, який відповідає таблиці бази даних.

Основні анотації:

Анотація	Призначення
@Entity	позначає сутність
@Table	задає таблицю
@Id	первинний ключ
@GeneratedValue	генерація id
@Column	налаштування стовпця

Приклад Entity:

```
@Entity
@Table(name = "products")
public class Product {

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

    @Column(nullable = false)
    private String name;

    private Double price;
}
```

Hibernate автоматично створює таблицю products.

Repository Layer

Spring Data JPA дозволяє створювати Repository без написання SQL-коду.

Приклад:

```
@Repository
public interface ProductRepository
      extends JpaRepository<Product, Long> {
}
```

JpaRepository автоматично надає:

- save();
- findAll();
- findById();
- deleteById();
- count();
- existsById().

Таким чином значна частина CRUD-функціоналу генерується автоматично.

CRUD-операції у Spring Boot

CRUD є базовим набором операцій роботи з даними.

Create:

```
Product product = new Product();
product.setName("Laptop");
product.setPrice(50000.0);

repository.save(product);
```

Read:

```
List<Product> products = repository.findAll();
```

Update:

```
Product product = repository.findById(1L)
      .orElseThrow();

product.setPrice(55000.0);

repository.save(product);
```

Delete:

repository.deleteById(1L);

Spring Data JPA автоматично генерує SQL-запити.

Service Layer та бізнес-логіка

Service Layer є проміжним рівнем між Controller та Repository.

Його основним завданням є реалізація бізнес-логіки.

```
@Service
public class ProductService {

   private final ProductRepository repository;

   public ProductService(ProductRepository repository) {
      this.repository = repository;
   }

   public List<Product> getAllProducts() {
      return repository.findAll();
   }
}
```

Controller не повинен напряму працювати з базою даних.

Саме Service Layer забезпечує:

- інкапсуляцію логіки;
- тестованість;
- масштабованість.

Власні методи Repository

Spring Data JPA підтримує автоматичну генерацію SQL-запитів на основі назв методів.

Наприклад:

List<Product> findByName(String name);

або:

List<Product> findByPriceGreaterThan(Double price);

Spring автоматично генерує SQL-запит.

Це є однією з найважливіших особливостей Spring Data JPA.

JPQL та @Query

Для складніших запитів використовується JPQL.

JPQL працює з Entity, а не з таблицями.

Приклад:

```
@Query("SELECT p FROM Product p WHERE p.price > :price")
List<Product> findExpensiveProducts(
      @Param("price") Double price
);
```

Також можливе використання native SQL.

```
@Query(value = "SELECT * FROM products",
     nativeQuery = true)
List<Product> findAllNative();
```

Зв’язки між таблицями

Hibernate підтримує зв’язки між Entity.

Основні типи:

- OneToOne;
- OneToMany;
- ManyToOne;
- ManyToMany.

ManyToOne

Наприклад, багато замовлень можуть належати одному користувачу.

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

OneToMany

```
@OneToMany(mappedBy = "user")
private List<Order> orders;
```

Hibernate автоматично створює зв’язки між таблицями.

Транзакції у Spring Boot

Транзакція — це набір операцій, які виконуються як єдине ціле.

Spring Boot підтримує транзакції через анотацію:

@Transactional

Приклад:

```
@Transactional
public void transferMoney() {
}
```

Якщо під час виконання виникає помилка, всі зміни будуть скасовані.

Транзакції є критично важливими для:

- банківських систем;
- оплат;
- фінансових операцій.

Lazy Loading та Eager Loading

Hibernate підтримує два підходи завантаження даних:

- Lazy Loading;
- Eager Loading.

Lazy Loading відкладає завантаження пов’язаних даних.

Eager Loading завантажує їх одразу.

Приклад:

```
@OneToMany(fetch = FetchType.LAZY)
private List<Order> orders;
```

Неправильне використання Eager Loading може призводити до проблем продуктивності.

DTO та робота з даними

У production-системах не рекомендується повертати Entity напряму.

Для цього використовуються DTO.

```
public class ProductDto {

    private String name;

    private Double price;
}
```

DTO дозволяють:

- приховувати внутрішню структуру;
- контролювати API;
- підвищувати безпеку.

Валідація даних

Spring Boot підтримує Bean Validation.

```
@NotBlank
private String name;

@Positive
private Double price;
```

Валідація виконується автоматично.

```
public Product create(
      @Valid @RequestBody Product product
) {
}
```

H2 Database

Для навчання часто використовується H2 Database.

H2 є in-memory базою даних.

Конфігурація:

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.h2.console.enabled=true
```

H2 зручна для:

- тестування;
- навчання;
- швидкого запуску проєкту.

SQL Logging

Spring Boot дозволяє відображати SQL-запити.

```
spring.jpa.show-sql=true
```

Також можна форматувати SQL:

```
spring.jpa.properties.hibernate.format_sql=true
```

Це значно спрощує дебаг.

Проблема N+1 Query

Однією з типових проблем Hibernate є N+1 Query Problem.

Вона виникає, коли Hibernate виконує надмірну кількість SQL-запитів.

Наприклад:

- 1 запит для користувачів;
- N запитів для замовлень.

Для вирішення використовуються:

- JOIN FETCH;
- EntityGraph;
- оптимізація Lazy Loading.

Flyway та міграції бази даних

У production-проєктах структура бази даних змінюється поступово.

Для керування міграціями використовується Flyway.

Flyway дозволяє:

- версіонувати SQL;
- автоматично оновлювати схему БД;
- синхронізувати структуру бази.

Приклад міграції:

```
CREATE TABLE users (
   id BIGINT PRIMARY KEY,
   username VARCHAR(255)
);
```

Висновки

Робота з базами даних є одним із найважливіших аспектів сучасної backend-розробки. Саме база даних забезпечує довготривале зберігання інформації та підтримку бізнес-процесів веб-додатка.

У межах лекції було розглянуто інтеграцію Spring Boot із реляційними базами даних, конфігурацію DataSource, Hibernate, Spring Data JPA, Entity-класи, Repository Layer, CRUD-операції, транзакції та ORM-підхід.

Особливу увагу приділено багаторівневій архітектурі та принципам розділення відповідальності між компонентами системи.

Розуміння роботи Spring Boot із базами даних є фундаментом для створення сучасних enterprise-систем, REST API та мікросервісної архітектури.

Отримані знання є основою для подальшого вивчення:

- Spring Security;
- JWT;
- Docker;
- PostgreSQL;
- Redis;
- Kafka;
- cloud-native systems.

Завдання на самопідготовку

1. Підключити MySQL до Spring Boot.
2. Створити Entity-клас Product.
3. Реалізувати Repository.
4. Створити CRUD-операції.
5. Реалізувати зв’язок ManyToOne.
6. Додати DTO.
7. Увімкнути SQL Logging.
8. Спробувати використати H2 Database.
9. Створити власний JPQL-запит.

Література

1. Spring Data JPA Documentation.
2. Hibernate Documentation.
3. Craig Walls. Spring in Action.
4. Martin Fowler. Patterns of Enterprise Application Architecture.
5. Oracle Java Documentation.
6. Spring Boot Documentation.
7. Herbert Schildt. Java: The Complete Reference.