**UNIT-5**

**Persisting data reactively:** Understanding Spring Data’s reactive story (Spring Data reactive distilled, Converting between reactive and non-reactive types, Developing reactive repositories), Working with reactive Cassandra repositories, Enabling Spring Data Cassandra, Understanding Cassandra data modeling, Mapping domain types for Cassandra persistence, Writing reactive Cassandra repositories, Writing reactive MongoDB repositories (Enabling Spring Data MongoDB, Mapping domain types to documents, Writing reactive MongoDB repository interfaces).

**Textbook 1: Chapter 12:12.1 to 12.3**

**Persisting Data Reactively**

**🎯 Learning Objectives:**

**By the end of this lecture, students will be able to:**

* **Understand the fundamentals of reactive persistence in Spring Data.**
* **Explore the reactive programming model supported by Spring Data.**
* **Implement reactive repositories for Cassandra and MongoDB.**
* **Differentiate between reactive and non-reactive types.**
* **Map domain types to persistent stores in a reactive fashion.**

**1. 🌱 Introduction to Spring Data Reactive**

**Spring Data provides reactive support for NoSQL and relational databases using Project Reactor, which is the default reactive programming library in Spring WebFlux.**

**🔍 Why Reactive Persistence?**

* **Traditional database operations are blocking.**
* **Reactive data access allows non-blocking, backpressure-enabled communication with the database.**
* **Ideal for applications requiring high concurrency, low latency, and asynchronous data access.**

**Reactive programming is a programming paradigm that deals with asynchronous data streams and the propagation of change. In Spring, this is supported via the Project Reactor library and its integration into the Spring ecosystem (e.g., WebFlux, Spring Data Reactive).**

**Why Reactive Persistence?**

* **Efficient resource usage (non-blocking I/O)**
* **Better scalability for data-intensive applications**
* **Seamless integration with reactive web layers (e.g., WebFlux)**

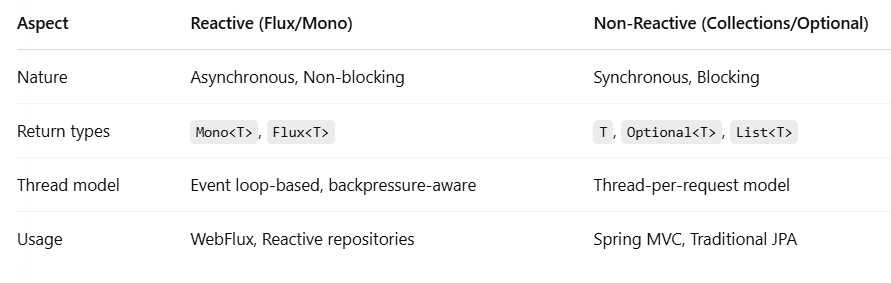
**2. 🧠 Spring Data’s Reactive Story**

**a. Reactive Repositories**

* **Spring Data introduces interfaces like ReactiveCrudRepository and ReactiveMongoRepository.**
* **They return Publisher types like Mono<T> and Flux<T>.**
  + **Mono<T> – Emits zero or one item.**
  + **Flux<T> – Emits zero or more items.**

**Spring Data Reactive provides support for non-blocking data access using reactive repositories. It uses reactive types like Mono and Flux from Project Reactor to support asynchronous data flows.**

**2.1 Reactive vs. Non-Reactive Types**

****

**Conversion Example:**

**Mono<User> reactiveUser = Mono.just(new User());**

**User user = reactiveUser.block(); // Converts Mono to User (blocking)**

**User user = new User();**

**Mono<User> monoUser = Mono.just(user); // Converts User to Mono**

**3. Developing Reactive Repositories**

**Spring provides out-of-the-box support for reactive repositories for data stores that support reactive access, such as MongoDB and Cassandra.**

**3.1 Key Interfaces**

* **ReactiveCrudRepository<T, ID>**
* **ReactiveMongoRepository<T, ID>**
* **ReactiveCassandraRepository<T, ID>**

**Example:**

**public interface UserRepository extends ReactiveMongoRepository<User, String> {**

**Flux<User> findByRole(String role);**

**}**

**4. Working with Reactive Cassandra Repositories**

**4.1 Enabling Spring Data Cassandra**

**To use Cassandra with reactive Spring Data:**

* **Add the dependency:**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-data-cassandra-reactive</artifactId>**

**</dependency>**

* **Configure Cassandra connection in application.yml:**

**spring:**

**data:**

**cassandra:**

**keyspace-name: demo**

**contact-points: localhost**

**port: 9042**

**4.2 Cassandra Data Modeling**

**Cassandra favors denormalized, query-driven designs:**

* **Model tables based on access patterns.**
* **Use primary key, partition key, and clustering columns.**

**4.3 Mapping Domain Types**

**Use annotations like @Table, @PrimaryKey, @Column.**

**@Table**

**public class User {**

**@PrimaryKey**

**private String id;**

**@Column**

**private String name;**

**}**

**4.4 Writing Reactive Cassandra Repositories**

**public interface UserRepository extends ReactiveCassandraRepository<User, String> {**

**Flux<User> findByName(String name);**

**}**

**5. Writing Reactive MongoDB Repositories**

**5.1 Enabling Spring Data MongoDB**

**Add dependency:**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-data-mongodb-reactive</artifactId>**

**</dependency>**

**Configure MongoDB:**

**spring:**

**data:**

**mongodb:**

**database: test**

**host: localhost**

**port: 27017**

**5.2 Mapping Domain Types**

**Use @Document instead of @Entity.**

**@Document(collection = "users")**

**public class User {**

**@Id**

**private String id;**

**private String name;**

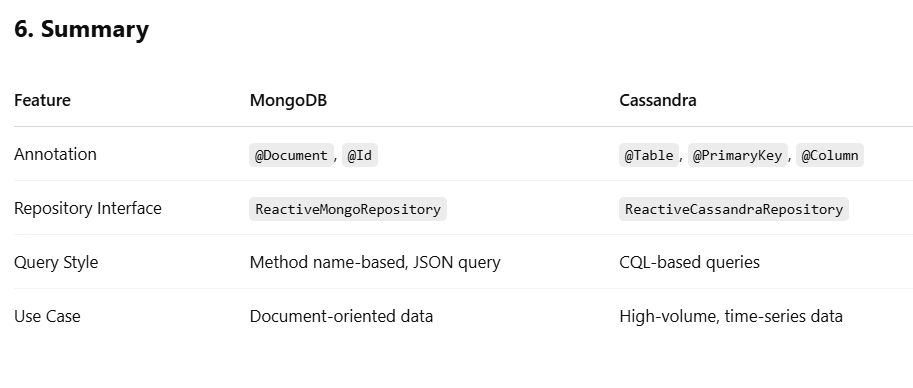
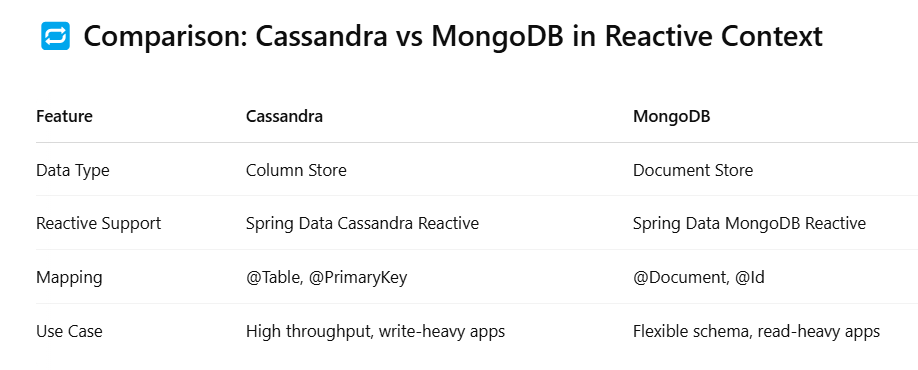
**}**

**5.3 Writing Reactive MongoDB Repository Interfaces**

**public interface UserRepository extends ReactiveMongoRepository<User, String> {**

**Flux<User> findByName(String name);**

**}**

****