Spring ORM

1. The Spring Boot Framework integrates well with ORM frameworks like Hibernate, Java Persistence API (JPA), Java Data Objects (JDO) and iBATIS SQL Maps
2. Spring ORM (Object-Relational Mapping) is a module within the Spring Framework that provides integration and support for working with relational databases and mapping Java objects to database tables
3. It simplifies the process of data access and persistence by allowing developers to interact with the database using Java objects instead of writing raw SQL queries.
4. Spring ORM supports various ORM frameworks, with Hibernate being the most commonly used one. Hibernate is a popular and powerful ORM tool that provides features like object-oriented querying, caching, lazy loading, and transaction management.

Some key features of Spring ORM include:

Entity Mapping: Spring ORM allows you to define entity classes that represent database tables. These entity classes are annotated with JPA (Java Persistence API) annotations to define the mapping between the Java objects and the database tables.
Repository Support: Spring ORM provides a repository support layer that allows you to create interfaces for data access operations (e.g., save, delete, find) without writing the implementation code. Spring takes care of generating the necessary SQL queries and managing transactions.
Transaction Management: Spring ORM offers robust transaction management capabilities. It allows you to declaratively manage transactions using annotations or XML configuration. Transactions are essential for ensuring data consistency and integrity.
JPQL and Criteria API: Spring ORM supports JPQL (Java Persistence Query Language) and Criteria API, which provide ways to perform complex queries and retrieve data from the database based on object-oriented criteria.
Caching: ORM frameworks like Hibernate support caching mechanisms, which can significantly improve application performance by reducing the number of database queries.

To use Spring ORM in your application, you typically configure a data source (database connection) in the Spring configuration file, set up entity classes with appropriate annotations, and define repositories or DAOs (Data Access Objects) to interact with the database. Spring will handle the underlying plumbing to manage the database operations.

```xml
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Is required from Annotation in Controller class

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

Is used to connectivity using JPA framework

Following is the description of key attributes of application.properties.

```
#datasource configurations

spring.datasource.url=jdbc:mysql://localhost:3306/springorm?useSSL=false&allowP
ublicKeyRetrieval=true

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

```
# show SQL

spring.jpa.show-sql: true

# DDL generation

spring.jpa.generate-ddl=true
```

spring.datasource − Database specific attributes like connection url, username, password, driver class etc.

spring.jpa − jpa specific attributes like to show sql, to allow create tables etc.