**SPRING DATA JPA HIBERNATE**

**1. Explain the need and benefit of ORM**

**What is ORM (Object-Relational Mapping)?**

* Technique that maps object-oriented domain models (Java classes) to relational database tables.
* Automates conversion between Java objects and SQL records.

**Need:**

* Bridging the impedance mismatch between object-oriented code and relational databases.
* Avoid manual SQL and repetitive JDBC boilerplate.

**Pros:**

1. You get to write in the language you are already using anyway. If we’re being honest, we probably aren’t the greatest at writing SQL statements. SQL is a ridiculously powerful language, but most of us don’t write in it often. We do, however, tend to be much more fluent in one language or another and being able to leverage that fluency is awesome!
2. It abstracts away the database system so that switching from MySQL to PostgreSQL, or whatever flavor you prefer, is easy-peasy.
3. Depending on the ORM you get a lot of advanced features out of the box, such as support for transactions, connection pooling, migrations, seeds, streams, and all sorts of other goodies.
4. Many of the queries you write will perform better than if you wrote them yourself.

**Cons :**

1. If you are a master at SQL, you can probably get more performant queries by writing them yourself.
2. There is overhead involved in learning how to use any given ORM.
3. The initial configuration of an ORM can be a headache.
4. As a developer, it is important to understand what is happening under the hood. Since ORMs can serve as a crutch to avoid understanding databases and SQL, it can make you a weaker developer in that portion of the stack.

---------------------------------------------------------------------------------------------------------

**2. Demonstrate the need and benefit of Spring Data JPA**

**Evolution:**

* Old style Hibernate → verbose XML configuration.
* Annotations in Hibernate → better, but still boilerplate.
* Spring Data JPA → minimal boilerplate, powerful query generation.

**Need:**

* Avoid repetitive repository/DAO code.
* Simplify transaction and session handling.
* Rapid development with Spring integration.

**Benefits:**

* Repository interfaces with automatic query generation.
* Database-agnostic (via JPA spec).
* Declarative transaction management.
* Easily integrates with Spring Boot.

**3. Explain about core objects of Hibernate framework**

**Core Objects:**

* **SessionFactory**: Heavyweight, thread-safe factory for Session. Configured once.
* **Session**: Lightweight, single-threaded. Main object for CRUD. Wraps JDBC connection.
* **Transaction**: Encapsulates a unit of work. Begin, commit, rollback.
* **TransactionFactory**: Produces Transaction instances.
* **ConnectionProvider**: Manages underlying JDBC connections.

Hibernate is an Object-Relational Mapping (ORM) tool used in Java applications. It was created by Gavin King in 2001 and is available as an open-source framework. Hibernate makes it easier for developers to save and retrieve data from a database by handling most of the repetitive and complex code that would normally be required.

It works by mapping Java classes to database tables and converting Java data types into SQL data types. This means developers don’t have to write a lot of SQL manually—it does around 95% of the work related to data persistence on its own.

Hibernate acts as a link between the Java application and the database. It takes care of saving and loading objects to and from the database using efficient and well-known object-relational mapping techniques.

**Hibernate Advantages**

* Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
* Provides simple APIs for storing and retrieving Java objects directly to and from the database.
* If there is change in the database or in any table, then you need to change the XML file properties only.
* Abstracts away the unfamiliar SQL types and provides a way to work around familiar Java Objects.
* Hibernate does not require an application server to operate.
* Manipulates Complex associations of objects of your database.
* Minimizes database access with smart fetching strategies.
* Provides simple querying of data.

---------------------------------------------------------------------------------------------------------

**4.Explain ORM implementation with Hibernate XML Configuration and Annotation Configuration**

**A. XML Configuration**

* **Persistence Class:** POJO with properties.
* **Mapping XML:** Defines property-to-column mappings.
* **Configuration XML:** Database connection and mapping file references.
* **Lifecycle Steps:**
  + Load configuration.
  + Build SessionFactory.
  + Open Session.
  + Begin Transaction.
  + Save/update/delete objects.
  + Commit transaction.
  + Close session

**B. Annotation Configuration**

* Uses annotations in Java class:
  + @Entity, @Table, @Id, @Column.
* Removes need for separate mapping XML.

**@Entity Annotation**

* The EJB 3 standard annotations are contained in the **javax.persistence** package, so we import this package as the first step. Second, we used the **@Entity** annotation to the Employee class, which marks this class as an entity bean, so it must have a no-argument constructor that is visible with at least protected scope.

**@Table Annotation**

* The @Table annotation allows you to specify the details of the table that will be used to persist the entity in the database.
* The @Table annotation provides four attributes, allowing you to override the name of the table, its catalogue, and its schema, and enforce unique constraints on columns in the table. For now, we are using just table name, which is EMPLOYEE.

**@Id and @GeneratedValue Annotations**

* Each entity bean will have a primary key, which you annotate on the class with the **@Id** annotation. The primary key can be a single field or a combination of multiple fields depending on your table structure.

-------------------------------------------------------------------------------------------------

1. **Explain the difference between Java Persistence API, Hibernate, and Spring Data JPA**

| **Feature** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification (JSR 338) | Implementation of JPA, ORM framework | Abstraction layer over JPA |
| Provides | Interfaces, annotations | Actual ORM implementation | Auto-generated repositories |
| Boilerplate | Requires implementation | Still needs DAOs and session management | Eliminates most boilerplate |
| Vendor | Part of Java EE / Jakarta EE | Red Hat | Spring / Pivotal |

------------------------------------------------------------------------------------------------------

**6. Demonstrate implementation of DML using Spring Data JPA on a single database table**

**Configuration (application.properties):**

ini

CopyEdit

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

**Repository Interface:**

java

CopyEdit

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

List<Employee> findByLastName(String lastName);

}

**Example Usage in Service:**

java

CopyEdit

@Autowired

private EmployeeRepository employeeRepository;

// Insert or Update

employeeRepository.save(new Employee("John", "Doe"));

// Fetch

employeeRepository.findById(1L);

employeeRepository.findByLastName("Doe");

// Delete

employeeRepository.deleteById(1L);

---------------------------------------------------------------------------------------------------