Difference Between JPA, Hibernate, and Spring Data JPA

When we build Java applications that need to store and use data from databases, we use special tools that make it easier to handle this data. Three of the main tools used for this in Java are JPA, Hibernate, and Spring Data JPA. They are connected, but each one has its own job. Let's talk about what each one does in simple terms with easy examples.

What is JPA?

JPA stands for Java Persistence API. It is like a **rulebook or guideline** for how Java programs should store and read data from a database. But JPA **doesn't actually do the work**—it just tells how it should be done.

Important Points:

- It is just a set of rules (a specification)
- It provides annotations like @Entity, @Id, etc.
- It needs some tool to actually follow and implement these rules

So, to use JPA, we need something like Hibernate that actually does the job according to JPA rules.

What is Hibernate?

Hibernate is a tool that actually follows the rules set by JPA. It **connects Java objects to database tables** and makes it easier to save, update, or delete data.

Key Points:

- It follows JPA rules (it implements JPA)
- Also gives extra features like its own query language (HQL)
- You need to write more code for session and transaction handling

Example Code Using Hibernate:

```
public Integer addEmployee(Employee employee){
    Session session = factory.openSession();
    Transaction tx = null;
    Integer employeeID = null;

    try {
        tx = session.beginTransaction();
        employeeID = (Integer) session.save(employee);
        tx.commit();
    } catch (HibernateException e) {
        if (tx != null) tx.rollback();
        e.printStackTrace();
    } finally {
        session.close();
    }
    return employeeID;
}
```

What is Happening in This Code?

- We open a session to talk to the database
- Start a transaction

- Save employee data
- Commit the transaction and close the session

This works well, but there's a lot of repeated code (boilerplate).

What is Spring Data JPA?

Spring Data JPA is built on top of JPA and Hibernate. It helps **reduce all the repeated code** we saw in Hibernate. It uses Spring's features to make data access much easier.

Key Points:

- Works with JPA and Hibernate
- Reduces boilerplate code
- Automatically creates queries based on method names
- Works smoothly with Spring Boot

Example Code Using Spring Data JPA:

EmployeeRepository.java

```
public interface EmployeeRepository extends JpaRepository<Employee,
Integer> {
}
EmployeeService.java
@Autowired
private EmployeeRepository employeeRepository;

@Transactional
public void addEmployee(Employee employee) {
    employeeRepository.save(employee);

}
```

What is Happening in This Code?

- We don't open sessions or manage transactions manually
- Spring handles everything using annotations
- The repository interface takes care of saving data

This makes our code much cleaner and easier to understand.

Simple Comparison Table

Feature	JPA (Rules)	Hibernate (Tool)	Spring Data JPA (Helper)
Type	Specification	JPA Implementation	Abstraction over JPA
Who provides it?	Java EE / Jakarta EE	Red Hat	Spring Framework
Extra Code Required	Medium	High	Very Low
Needs Implementation?	Yes	No (already is one)	Yes (uses Hibernate)
Manages Transactions?	No	Yes (manually)	Yes (automatically)
Main Benefit	Portability	Control + Features	Simplicity + Less Code

Conclusion

To sum it up:

- **JPA** gives the rules.
- **Hibernate** follows the rules and connects Java code with the database.
- **Spring Data JPA** makes our job even easier by reducing the code we need to write.

If you're working on a basic project and want full control, Hibernate is a good choice. But if you're working with Spring Boot and want fast development, Spring Data JPA is perfect. All three are connected and used together in real-world projects.

That’s why it’s important to understand the difference and use the right tool for the right job.