<https://www.stackchief.com/blog/Spring%20Data%20JPA%20Interview%20Questions>

## JPA

JPA is a specification which specifies how to access, manage and persist information/data between java objects and relational databases. It provides a standard approach for ORM, Object Relational Mapping.

## What is Hibernate?

Hibernate is a JPA provider. This means it provides an implementation for the JPA specification.

Hibernate is an implementation of JPA. It provides a lightweight framework and is one of the most popular ORM tool used.

## What does JPA stand for?

Java Persistence API

## What is the difference between Spring Data JPA and Hibernate?

Spring Data JPA provides an abstraction for more easily working with a JPA provider like Hibernate.

## What does the @Id annotation do?

The @Id annotation marks a field as the primary key for that particular table. This is a unique identifier for each entry in the table. This annotation is typically used with @GeneratedValue to automatically generate an unique id for each entry in the table.

## What does the @Entity annotation do?

The @Entity annotation indicates a class represents a relational table in the database. The JPA specification includes any class marked with @Entity in the persistence setup.

## What is the difference between FetchType.Eager and FetchType.Lazy?

FetchType attribute indicates how whether records will be eagerly or lazily loaded from the database. When records are eagerly loaded, JPA returns these objects regardless of whether they are accessed by the client or not. When records are lazily loaded the actual objects are only retrieved when directly accessed. This can save memory and processing when appropriate.

LAZY = fetch when needed

EAGER = fetch immediately

## Is the CrudRepository interface part of JPA?

No. CrudRepository is an interface exposed by Spring Data framework for more easily interacting with JPA implementations like Hibernate. While this interface saves a lot of boilerplate code, it isn't part of the JPA specification.

## Is Spring Data JPA an implementation of the JPA specification?

No. Spring Data simply makes it easier to interface with a JPA specification like Hibernate. Spring Data JPA abstracts away a lot of the configuration associated with these implementations but is not an implementation itself.

## Is the @Column annotation required for mapping fields to columns?

No. The @Column field allows you to optionally override the name of the column that the entity class field maps to in the database table. It is not required.

## What does the @EnableJpaRepositories annotation do?

This annotation enables the automatic generation of JPA repositories. Any class which implements CrudRepository interface will generate a repository when this annotation is present.

## What is Object Relational Mapping (ORM)

ORM is a mechanism for maintaining the relationship between object oriented data structures and relational tables in a database. Hibernate is an ORM tool that implements the JPA specifications. ORM allows objects (POJOS) to represent database tables and serves as an abstraction for database querying.

## What are some of the most popular ORM frameworks?

Hibernate, TopLink, ORMLite, iBATIS, JPOX

## What does the @Query Annotation do?

The @Query annotation allows you to define a Spring Data Repository method with custom SQL. Using @Query, you can map Spring Data repository methods to actual SQL statements.

## What's the difference between a CrudRepository and JpaRepository in Spring Data JPA?

CrudRepository extends REpository interface. JpaRepostitory extends PagingandSortingRepository interface. CrudRepository is for CRUD methods only where batch operations are better handled by extending JpaRepository.

## How does the CrudRepository save() method work in Spring Data JPA

The save() method effectively "upserts" a record. If the record doesn't exist in the database, then persist() is called. If the record does exist, then merge() is called to perform an update.

## What is Spring Data JPA?

Spring Data JPA provides an abstraction for working with JPA providers like Hibernate. Using Spring Data JPA allows developers to quickly implement data access repositories without writing boilerplate code associated with JPA providers.

## What is a JPA provider anyways?

A JPA provider implements the Java Persistence API (JPA) specification. The JPA specification defines how Java objects can represent and persist data stored in database tables. You can think of the JPA as a set of interfaces that need to be implemented. ORM libraries like Hibernate provide those implementations.

## Spring Data JPA vs Hibernate

Spring Data JPA and Hibernate are not competitors. In fact, Spring Data JPA uses Hibernate (or alternate JPA provider) under the hood. Spring Data JPA simply serves as an abstraction for more easily working with ORM libraries like Hibernate.

For more information be sure to check out [Spring Data JPA vs Hibernate](https://www.stackchief.com/blog/Spring%20Data%20JPA).

## What is Spring Data JPA?

Spring Data JPA makes it easier to work with JPA providers. A JPA provider is an object relational mapping (ORM) tool that implements the JPA specification. The JPA specification defines how Java objects represent relational database tables.

Spring Data JPA is an abstraction for working with JPA providers such as Hibernate. Using Spring Data JPA, you can avoid the boilerplate code associated with managing transactions and entity managers for providers like Hibernate.

## 1. Creating the project

You can easily create a project with all the necessary dependencies using maven.

spring-boot-starter-data-jpa

## 2.Configuring the database

Spring provides abstractions that hide the implementation details of connecting to a MySQL instance. Specifying the following in **application.properties** configures Spring Data JPA to work with the database:

spring.datasource.url=jdbc:mysql://localhost:3306/books\_service

spring.datasource.username=<DATABASE USERNAME>

spring.datasource.password=<DATABASE PASSSWORD>

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

spring.jpa.database-platform = org.hibernate.dialect.MySQL55Dialect

spring.jpa.generate-ddl=true

spring.jpa.hibernate.ddl-auto = create

spring.jpa.show-sql=true

## 3. Create the entities

Think of entities as tables in your database. For each table you can create an entity class whose members represent the table's columns.

**@Entity** signifies the class represents a table in the database.

**@Id** signifies the member is the primary key for this table.

**@GeneratedValue** means the id should be automatically generated. This means you don't have to worry about generating the id yourself.

**@Column** adds optional metadata for a given entity member. Using @Column, you can specify name, implement constraints (unique, nullable, etc).

**@OneToMany** and @ManyToOne specify the association between the two tables. An owner of the association is defined via mappedBy = "author" pointing to the field on the owning side of the relationship (in this case Book).

Entities are classes that map to tables in the database. For each table in the database, you can create a corresponding entity class:

The @Entity annotation specifies a class corresponds to a table in the database. Each member of the class corresponds to a column in the database. For example, the author table will have a column first\_name and last\_name.

You can override default column names via @Column. This annotation allows you to specify optional name and constraints/validations like uniqueness and required.

The @Id annotation signifies a class member is the primary key for that table. This is often used in combination with @GeneratedValue to automatically manage id generation for you.

## Managing relationships between entities

Notice the use of @OneToMany and @ManyToOne. These annotations specify a one-to-many association between authors and books. One author can have many books.

This type of relationship requires the book table have a foreign key referencing the author table. In this sense, the book table "owns the relationship".

To achieve this relationship, @OneToMany is used to associate a set of books with an author. The mappedBy = "authors" attribute points to the associated field on the owning entity in the relationship. For these reasons, the @ManyToOne annotation is used on the author field for Book.

## FetchType.EAGER vs FetchType.LAZY

https://stackoverflow.com/questions/2990799/difference-between-fetchtype-lazy-and-eager-in-java-persistence-api

A FetchType is also specified. While FetchType.EAGER loads associated records at the time data is accessed, FetchType.LAZY only loads associated records when they are explicitly accessed by the application.

FetchType.LAZY saves memory and processing but FetchType.EAGER makes sense if the associated data is always being used by the application.

Notice how a FetchType is specified for each one-to-many annotation. This indicates whether associated data will be eagerly or lazily loaded. Lazy loading can save memory and processing as associated records are only retrieved when asked for. Eager loading can be better if associated data will always be accessed by the application.

## 4. Create the repositories

Repositories are how the application interacts with the database. Extending Spring Data JPA interfaces like CrudRepository allows for an easy data access layer implementation.

Repositories are the gateway to interacting with the database. By simply extending Spring Data JPA defined interfaces, you can quickly perform CRUD operations without boilerplate code.

## JPA Specification???

The Java Persistence API (JPA) is a specification for mapping Java objects to database tables. Annotations like @Entity point to JPA provider (Hibernate) implementations of JPA specifications. AKA JPA specifies what an Entity is and Hibernate provides the implementation for that interface or specification.

This sounds a bit confusing because JPA originated from ORMs like Hibernate. JPA standardizes how ORM tools should look and behave.

## What is Spring Data JPA

Spring Data JPA is an abstraction that makes it easier to work with a JPA provider. Specifically Spring Data JPA provides a set of interfaces for easily creating data access repositories.

## Spring Data JPA vs Hibernate: The Key Difference

Spring Data JPA is really a set of dependencies that makes it easier to work with a JPA provider. Hibernate is one of several JPA providers. This means you can use Spring Data JPA without using Hibernate (if you really wanted to).

Hibernate is a JPA provider and ORM that maps Java objects to relational database tables. Spring Data JPA is an abstraction that makes working with the JPA provider less verbose. Using Spring Data JPA you can eliminate a lot of the boilerplate code involved in managing a JPA provider like Hibernate.