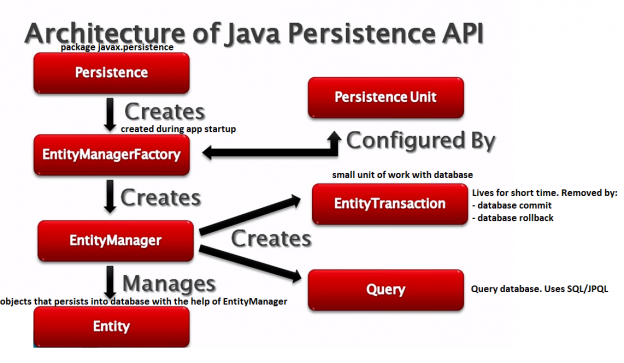
****

What is JPA?

**Java Persistence API**

**JPA** is a specification while **Hibernate** is a particular implementation of that specification to perform persistence logic

To work on JPA we need to manage some component every time, have a look on below diagram



So to work with JPA we need to create **EntityManagerFactory** then from factory we need to lookup **EntityManager** then using **Entitymanager** we need to create Query and to avoid data inconsistency we need to use **EntityTransaction**

So for every operation we need to bootstrap those component it seems boiler plate code

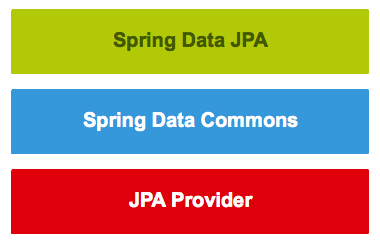
The problem of this approach is that we still have to write the code that creates our database queries and invokes them. To make matters worse, we have to do this every time when we want to create a new database query. **This is a waste of time**.

So to overcome this issue spring provided integration with JPA i.e. Spring Data JPA to make our work easy

**What is Spring Data JPA?**

It is a library / framework that adds an extra layer of abstraction on the top of our JPA provider

If we decide to use Spring Data JPA, the repository layer of our application contains three layers that are described like below



**Spring Data JPA** provides support for creating JPA repositories by extending the Spring Data repository interfaces.

**Spring Data Commons** provides the infrastructure that is shared by the data store specific Spring Data projects

**JPA Provider** implements the Java Persistence API

So now we got some basic internal idea about Spring Data JPA .To work with Spring Data JPA we no need to pay attention on actual implementation but we should aware on Spring Data Repository interface

**Spring Data Commons project provides the following interfaces**

1. The Repository<T, ID extends Serializable> interface is a marker interface that has two purposes:

🡪It captures the type of the managed entity and the type of the entity’s id.

🡪 It helps the spring container to discover the “concrete” repository interfaces during class path scanning.

2. The CrudRepository<T, ID extends Serializable> interface provides CRUD operations for the managed entity.

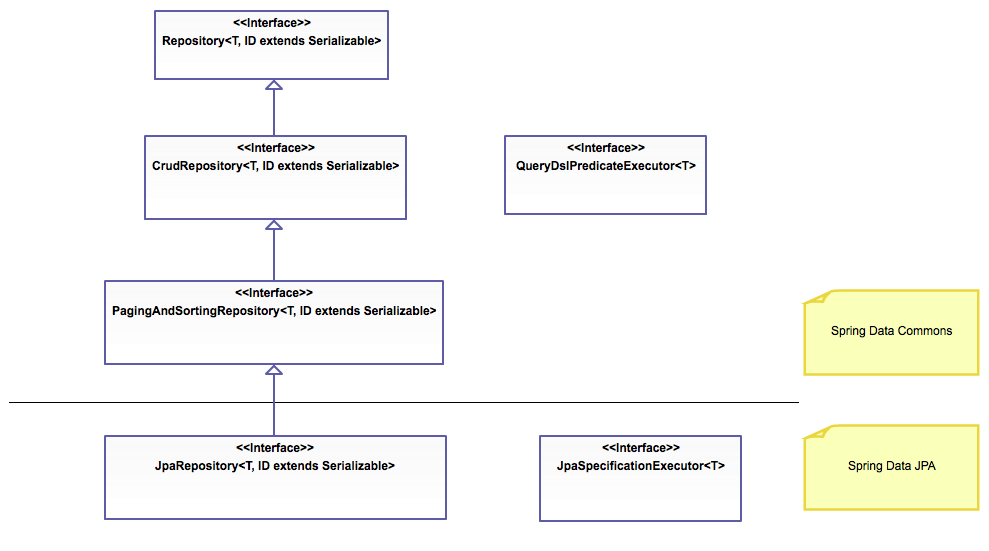
3. The PagingAndSortingRepository<T, ID extends Serializable> interface declares the methods that are used to sort and paginate entities that are retrieved from the database.

**Spring Data JPA project provides the following interfaces:**

1. The JpaRepository<T, ID extends Serializable> interface is a JPA specific repository interface that combines the methods declared by the common repository interfaces behind a single interface.

2. The JpaSpecificationExecutor<T> interface is not a “repository interface”. It declares the methods that are used to retrieve entities from the database by using Specification<T> objects that use the JPA criteria API.

**The repository hierarchy looks as follows**

****

That is nice, but how can we use them?

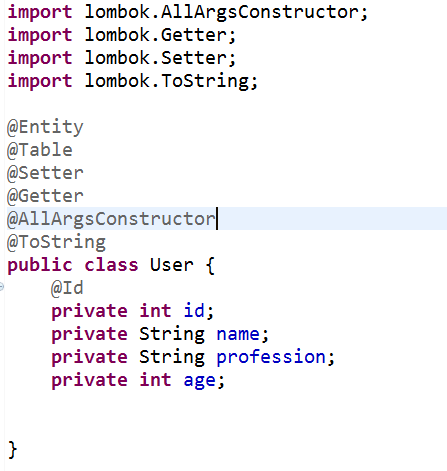
We have to follow these steps:

1. Create a repository interface and extend one of the repository interfaces provided by Spring Data.
2. Add custom query methods to the created repository interface (if we need them that is).

Now let’s have one small example How to develop one Repository using Spring Data and some of features of query optimization.

As my focus here to explain how to work with Spring Data JPA that’s why am using JpaRepository<T, Serializable> interface

Assume I have model class which I want to persist in Database



So for this Entity I want to perform some DB Operation without writing any Database code manually, so to achieve that we need to create our repository using Spring Data Jpa

Let’s create Repository now



Here in JpaRepository<User, Integer>

JpaRepository 🡪one interface

User 🡪Model class

Integer 🡪Datatype of our Model class id field

If we follow Spring Data provided syntax as above then, our model will register with spring data then we can get all inbuilt method to perform CRUD operation

Few inbuilt method provided by Spring Data



Apart from this above method we can write our own custom Query method to perform DB operation like below.

Assume few scenario.

**Requirement: 1**

Fetch List of User based on Profession

Normal Query: select \* from User where profession=?

Using Spring Data Jpa



If you mark here Profession field is one attribute of my model class, so based on model variable we can perform Query

Note: If mention field not available in your model class then we will get QuerySyntaxException

**Requirement: 2**

Fetch User object based on name

Normal Query: select \* from User where name=?

Using Spring Data Jpa



Here findBy is common syntax next is your field name of your entity, through which field condition you want to fetch Object from DB.

Similarly fetch Object based on Age or id or any field you can write method syntax accordingly like FindByAge (int age) and findById (int id)

Method Syntax: findByField (datatype field)

**Requirement: 3**

**Derived Count Query**

Assume I have to fetch employee counts who belongs from IT profession

Normal Query: select count (\*) from User where profession=?

Using Spring Data Jpa



**Requirement: 4**

**Derived remove Query**

Assume I have to remove employee with any field like removing by id or name or profession

SQL: delete from User where id=? Or delete from User where name=?

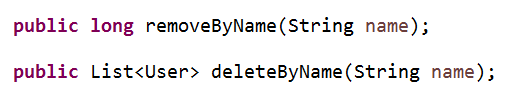
Using Spring Data Jpa

1. removeByName(String name);
2. deleteByName(String name);

Here we have 2 methods to perform delete operation but the difference between these 2 methods

RemoveByField will always return long (no of record deleted count)

DeleteByField will return list of model (list of available records present after delete)



**Requirement: 5**

**Multiple condition Query**

Assume I have requirement to fetch Employee object based on id and name

SQL: select \* from User where id=?, name=?

Using Spring Data Jpa



**Requirement: 6**

**Enabling ignoring case for an individual property**

Assume I want to fetch Employee based on name but I want to name as case insensitive

SQL: select \* from User where upper (name) =? ;

Or

Select \* from User where lower (name) =? ;

Using Spring Data Jpa



**Requirement: 7**

**Enabling static ORDER BY for a query**

Assume I want to fetch list of Employee based on profession and in sorting order by id

SQL: select \* from User orderby id asc where profession=? ;

Using Spring Data Jpa



We can apply inbuilt sort method as well

Like List<User> findAll (new Sort (“pass field name”));

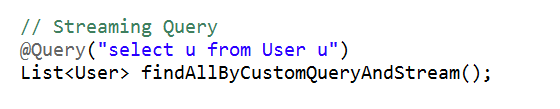
Suppose you want to sort by name then pass name in Sort constructor similar for which field you want to sort pass that field name in new Sort (“pass field name”));

**Requirement: 8**

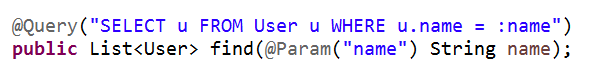
**Custom query**

Assume I want to fetch list of Employee using SQL query in spring Jpa

Using Spring Data Jpa



I want to fetch Employee based on name with SQL and spring data Jpa



**Requirement: 9**

**Limiting the result size of a query with Top**

Assume I want to fetch top 5 User object based on age

SQL: select \* from User where age in (select distinct top 5 age from User order by age desc)

In SQL we need to write nested sub query

Using Spring Data Jpa



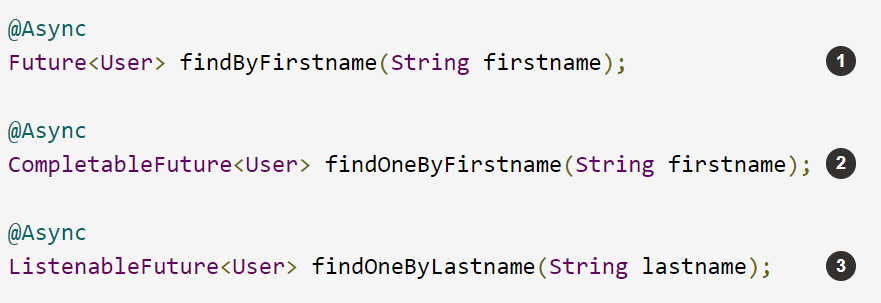
**Requirement: 10**

**Async query results**

Assume I have huge record in DB and I want to execute the query in asynchronous manner like want to execute using multi thread concept

So for this requirement we need to manually write threading logic to split the task and then after getting result join the task

Without writing own logic how we ca achieve it Using Spring Data Jpa



This means the method will return immediately upon invocation and the actual query execution will occur in a task that has been submitted to a Spring TaskExecutor

**Requirement: 11**

**Aggregate Function Query**

Aggregate function means find less than or greater than or max

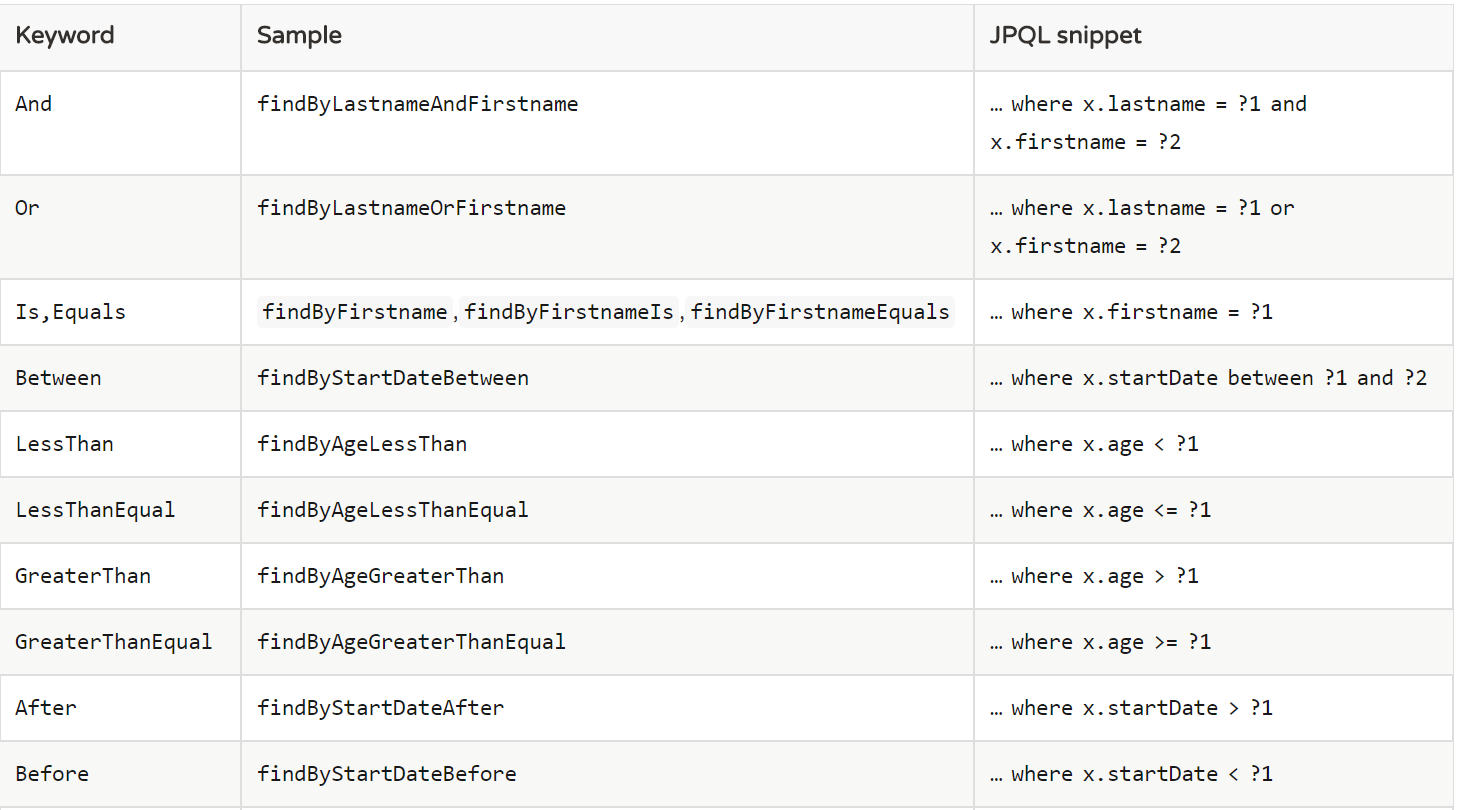
Ex: Fetch list of user whose age is less than 25, similarly we can write for greater than or equals

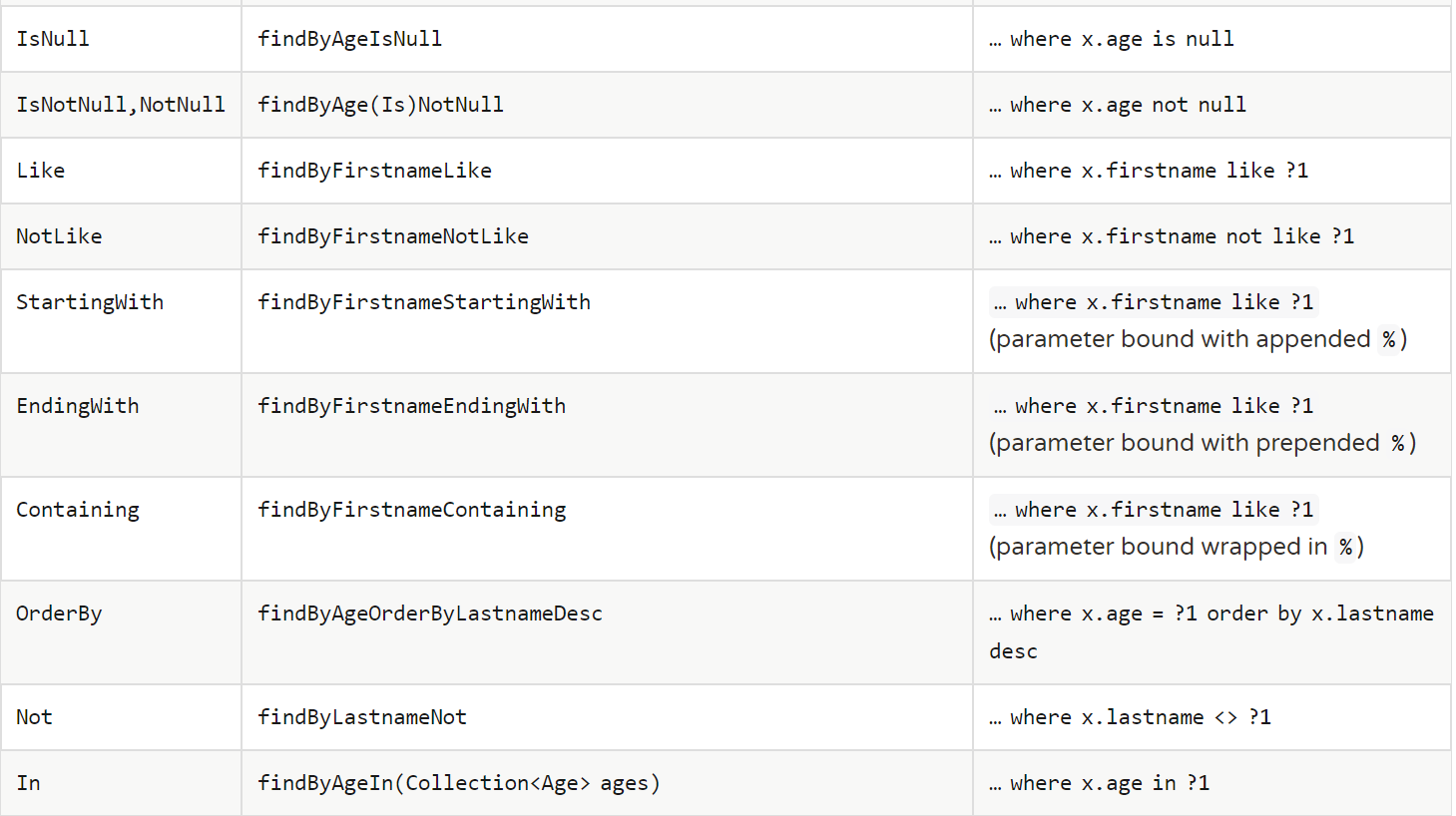
SQL: Select \* from User where age<=?

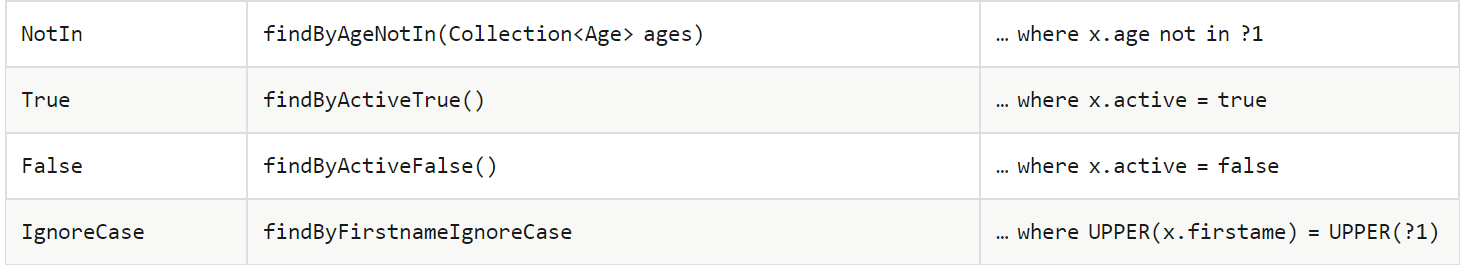
Using Spring Data Jpa



Let’s have a look few of keywords which Supported inside method names







In all above example we have disused related only select Query now let’s find the solution how to perform any non-select operation like update and delete

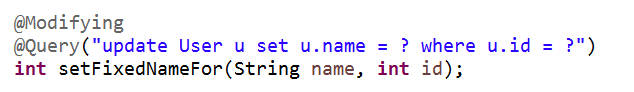
**Requirement: 12**

**Modifying queries**

Let’s update user name based on user id

SQL: update User u set u.name =? Where u.id =?

Using Spring Data Jpa



This is the way we can perform our Database logic without writing query manually it can achieve through only Spring Data by declaring methods with Spring data provided keywords.

Some more features also there like we can perform stored Procedure, auditing using Spring Data Jpa that we will cover in next blogs

[Official Link:](https://docs.spring.io/spring-data/jpa/docs/current/reference/html/)

**Thanks,**

**Basant Hota**