**Spring Data JPA:**

* Spring Data JPA, it is part of the Spring Data Family, aims to provide the JPA based repositories that aims to simplify the implementation of data access layer using JPA.

**Spring Data Repositories:**

* When we are working with spring data technologies, it is important to understand the repositories concept. Spring data provides the abstract repositories that implemented at run-time by the spring container and perform the **CRUD** operations. As a developer we have to just provide the abstract methods in the interface. This reduce the amount of boilerplate code required to data access layers.

There are three base interfaces defined in the spring data commons project.

1. **Repository:**
   * It’s the central interface in spring data repository abstraction.
   * This is a **Marker Interface**.
   * If we are extending this interface, we have to declare our own methods and implementations will be provided by the spring at **run-time**.
   * For this interface also we have to pass two parameters: **type of entity and type of the entity’s id field**.
   * This is super interface for **CrudRepository.**
2. **CrudRepository:**
   * **CrudRepository** provides methods for the **CRUD** operation.
   * This interface extends **Repository** interface.
   * When we define CrudRepository, we have to pass two parameters: **type of entity and type of the entity’s id field**.
   * There are following methods in CrudRepository:
   * **public** **interface** CrudRepository<T, ID> **extends** Repository<T, ID> {

|  |  |
| --- | --- |
| <S **extends** T> S save(S entity) | Saves a given entity. |
| <S **extends** T> Iterable<S> saveAll(Iterable<S> entities) | Saves all given entities. |
| Optional<T> findById(ID id) | Retrieves an entity by its id. |
| **boolean** existsById(ID id) | Returns whether an entity with the given id exists. |
| Iterable<T> findAll() | Returns all instances of the type. |
| Iterable<T> findAllById(Iterable<ID> ids) | Returns all instances of the type T with the given IDs. |
| **long** count() | Returns the number of entities available. |
| **void** deleteById(ID id) | Deletes the entity with the given id. |
| **void** delete(T entity) | Deletes a given entity. |
| **void** deleteAll(Iterable<? **extends** T> entities) | Deletes the given entities. |
| **void** deleteAll(); | Deletes all entities managed by the repository. |

* If we are extending the CrudRepository, there is no need for implementing our own methods. Just extends this interface and leave it as blank. Required implementation is provided at run-time.
* Example:
* **public** **interface** TicketRepository **extends** JpaRepository<Ticket, Integer> {}
* We have used **JpaRepositoory**, which is special version specific to the JPA technology. Unless until we are using any of the JPA specific things in application, it is highly recommended to use **CrudRepository,** becauseit will not tie application with any specific store implementation.

1. **PagingAndSortingRepository:**
   * This is extension of **CrudRepository.**
   * It is specialized version for the page operations.
   * **public** **interface** PagingAndSortingRepository<T, ID> **extends** CrudRepository<T, ID> {

|  |  |
| --- | --- |
| Iterable<T> findAll(Sort sort) | Returns all entities sorted by the given options. |
| Page<T> findAll(Pageable pageable) | Returns a [**Page**](https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Page.html) of entities meeting the paging restriction provided in the Pageable object. |

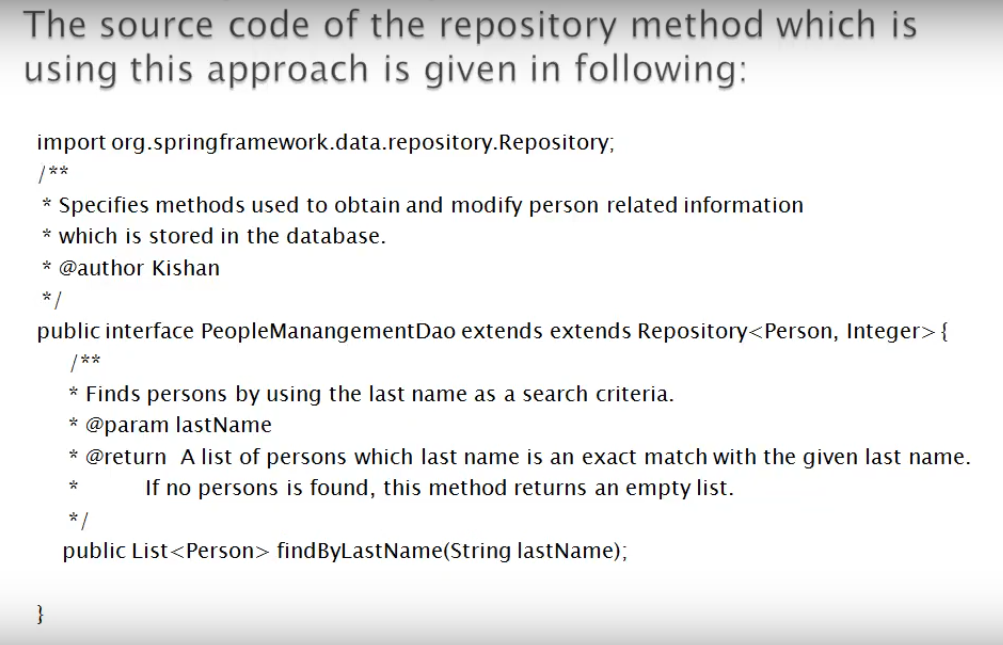
}

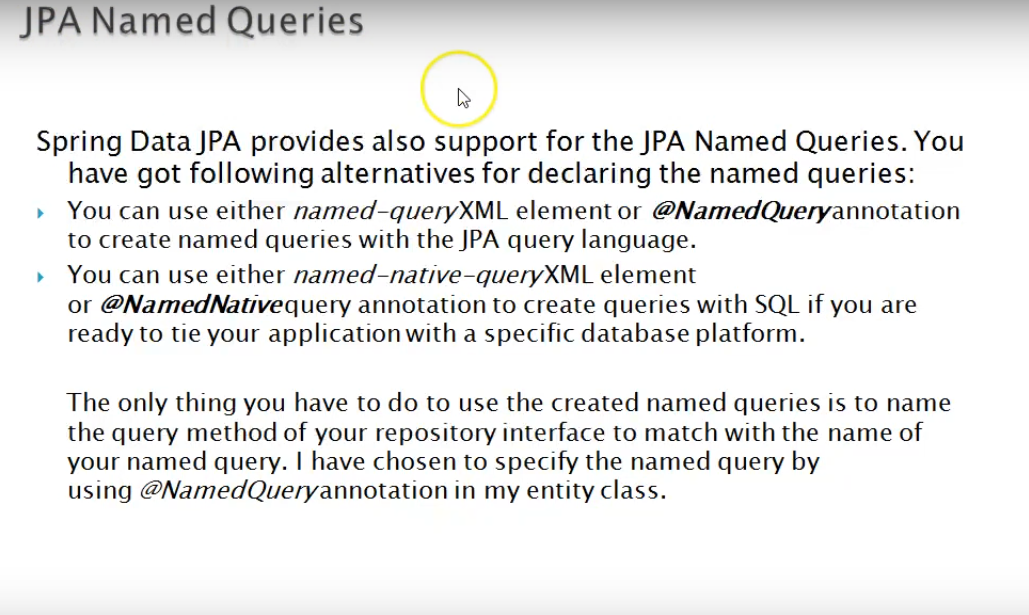
**Spring Data JPA Custom Queries with Query Methods**

* Spring Data JPA provides three different approaches for creating custom queries with query method:
  1. **Creating Database Queries From Method Names:** describe**s** how we can createdatabase queries from the method names of our query methods.
  2. **Creating Database Queries from Named Queries:** describe**s** how we can createdatabase queries by using named queries.
  3. **Creating Database Queries with the @Query Annotation:** describe**s** how we can createdatabase queries by annotating our query method with the **@Query** annotation.

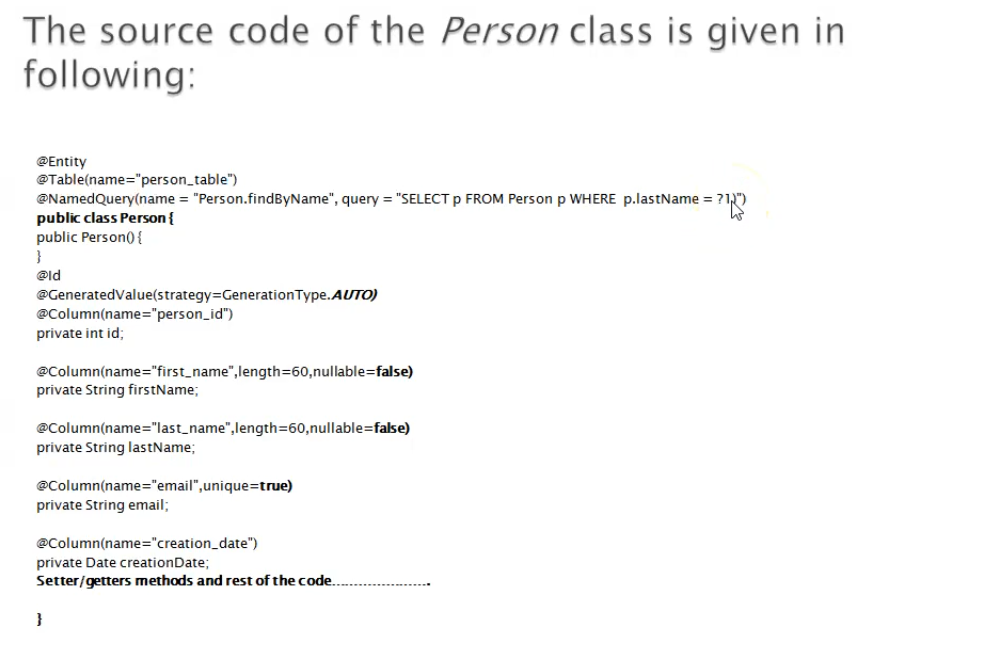
**Queries Creation from Method Names:**

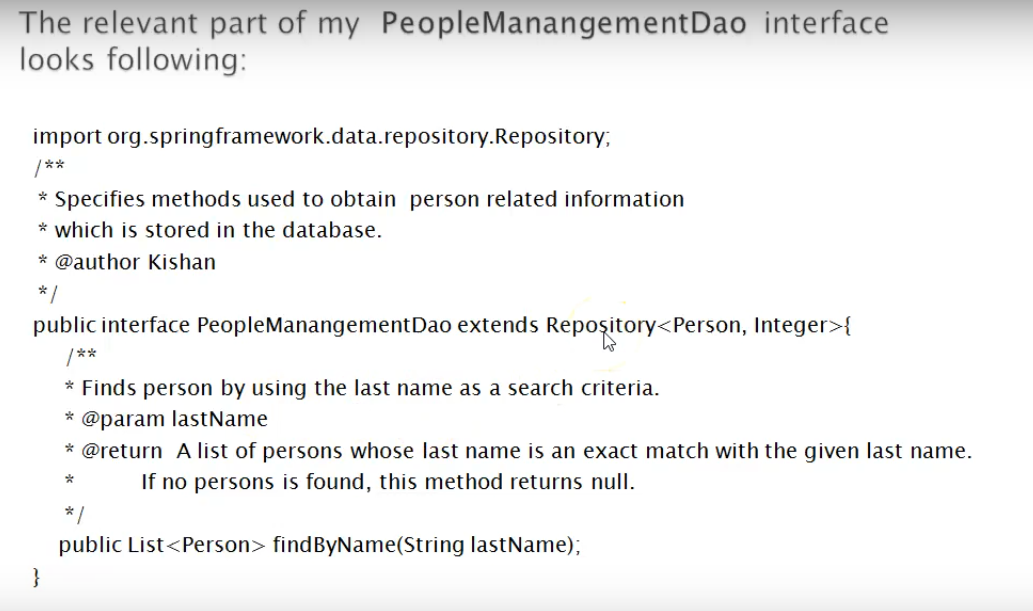
* Spring Data JPA has a built in query creation mechanism which can be used for parsing queries straight from the method name of a query method.
* This mechanism first removes common prefixes from method name and parses the constraint of the query from rest of the method name.
* The query builder mechanism is described with more details in the **Defining Query** **Method Subsection of Spring Data JPA reference documentation.**
* Using this approach is quite simple. All we have to do is to ensure that the method name of our repository interface are created by combining the property names of an entity object and the **Supported Keywords**.
* The **Query Creation Subsection of the Spring Data JPA reference documentation** has nice examples concerning the usages of supported keyword.
* <https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods>

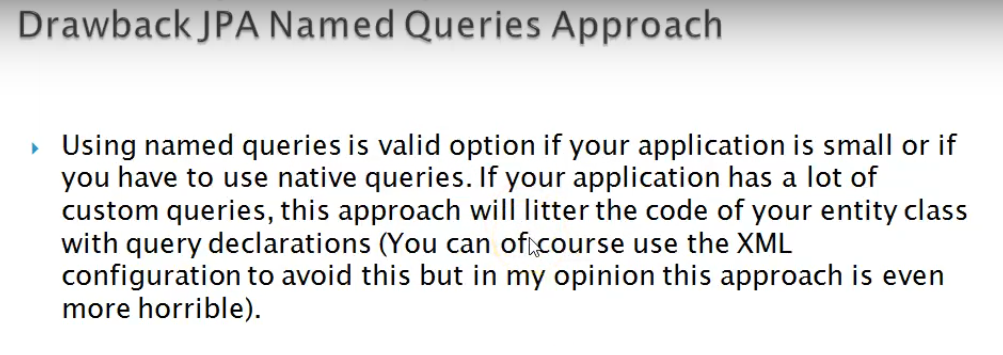


**JPA Named Queries:**

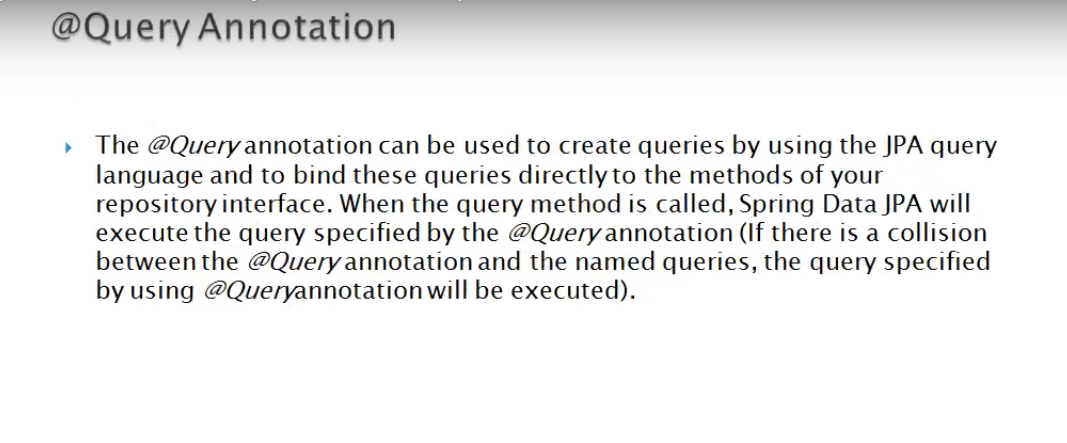
Example:



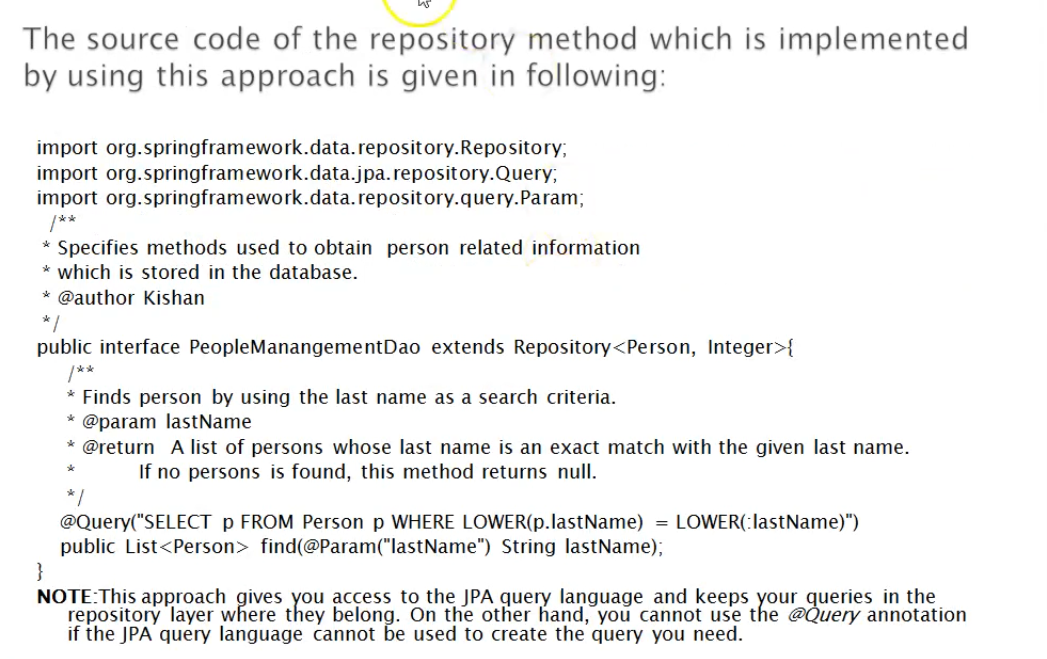


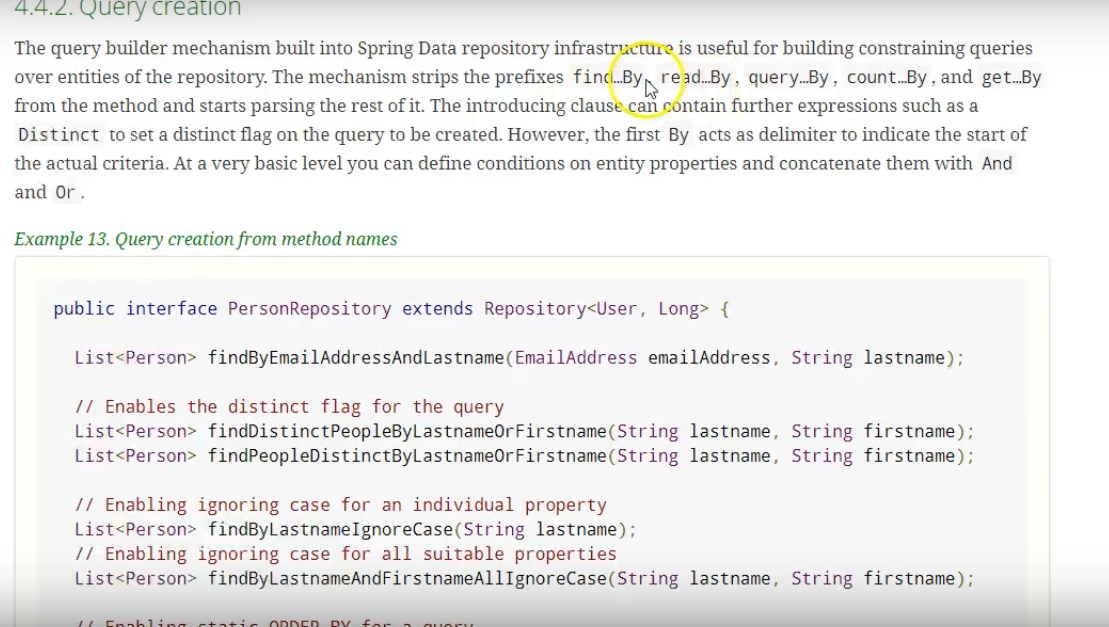


**@Query Annotation**

****

**Example:**

****

****