

Till now, our *Repository interface* looks like this

```
@Repository
public interface UserDetailsRepository extends
    JpaRepository<UserDetails, Long> {

}
```

And in service class, we used to invoke methods which are available in JPA framework

```
@Service
public class UserDetailsService {

    @Autowired
    UserDetailsRepository userDetailsRepository;

    public UserDetails saveUser(UserDetails user) {
        return userDetailsRepository.save(user);
    }

    public UserDetails findById(Long primaryKey) {
        return userDetailsRepository.findById(primaryKey).get();
    }

}
```

Then, we have something called: **Derived Query**

- Automatically generates queries from the methods.
- Need to follow a specific naming convention.
- Derived query used for GET/REMOVE operations but not for INSERT/UPDATE
 - Insert and Update operations is supported though "save()"

PartTree.java

```
private static final String QUERY_PATTERN = "find|read|get|query|search|stream";
private static final String COUNT_PATTERN = "count";
private static final String EXISTS_PATTERN = "exists";
private static final String DELETE_PATTERN = "delete|remove";
private static final Pattern PREFIX_TEMPLATE = Pattern.compile( //
    "^(" + QUERY_PATTERN + "|" + COUNT_PATTERN + "|" + EXISTS_PATTERN + "|" + DELETE_PATTERN + ")((\\p{Lu}.*)??)??By"
```

"^(find|read|get|query|search|stream|count|exists|delete|remove)((\\p{Lu}.*)??)??By"



Method name should start with either One of these values : find or read or get etc..

Uppercase Letter (ex: A,B,C etc..)

0 or More characters

'By' at the end of the String

```
@Repository
public interface UserDetailsRepository extends
    JpaRepository<UserDetails, Long> {

    List<UserDetails> findUserDetailsByName(String userName);
}
```

```
@Table(name = "user_details")
@Entity
public class UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;

    @Column(name = "user_name")
    private String name;
    private String phone;

    //getters and setters
}
```

Query in which it get translates too:

```
Hibernate:
select
    ud1_0.user_id,
    ud1_0.user_name,
    ud1_0.phone
from
    user_details ud1_0
where
    ud1_0.user_name=?
```

Different Use cases:

And:

```
List<UserDetails> findUserDetailsByNameAndPhone(String userName, String phone);
```

```
Hibernate:
select
    ud1_0.user_id,
    ud1_0.user_name,
    ud1_0.phone
from
    user_details ud1_0
where
    ud1_0.user_name=?
    and ud1_0.phone=?
```

Or:

```
List<UserDetails> findUserDetailsByNameAndPhoneOrUserId(String userName, String phone, Long id);
```

```
Hibernate:
```

```

select
    ud1_0.user_id,
    ud1_0.user_name,
    ud1_0.phone
from
    user_details ud1_0
where
    ud1_0.user_name=?
    and ud1_0.phone=?
    or ud1_0.user_id=?

```

Comparison:

Part.java

```

BETWEEN(2, "IsBetween", "Between"),
IS_NOT_NULL(0, "IsNotNull", "NotNull"),
IS_NULL(0, "IsNull", "Null"),
LESS_THAN("IsLessThan", "LessThan"),
LESS_THAN_EQUAL("IsLessThanEqual", "LessThanEqual"),
GREATER_THAN("IsGreaterThan", "GreaterThan"),
GREATER_THAN_EQUAL("IsGreaterThanEqual", "GreaterThanEqual"),
BEFORE("IsBefore", "Before"),
AFTER("IsAfter", "After"),
NOT_LIKE("IsNotLike", "NotLike"),
LIKE("IsLike", "Like"),
STARTING_WITH("IsStartingWith", "StartingWith", "StartsWith"),
ENDING_WITH("IsEndingWith", "EndingWith", "EndsWith"),
IS_NOT_EMPTY(0, "IsNotEmpty", "NotEmpty"),
IS_EMPTY(0, "IsEmpty", "Empty"),
NOT_CONTAINING("IsNotContaining", "NotContaining", "NotContains"),
CONTAINING("IsContaining", "Containing", "Contains"),
NOT_IN("IsNotIn", "NotIn"),
IN("IsIn", "In"),
NEAR("IsNear", "Near"),
WITHIN("IsWithin", "Within"),
REGEX("MatchesRegex", "Matches", "Regex"),

```

```
EXISTS(0, "Exists"),  
TRUE(0, "IsTrue", "True"),  
FALSE(0, "IsFalse", "False"),  
NEGATING_SIMPLE_PROPERTY("IsNot", "Not"),  
SIMPLE_PROPERTY("Is", "Equals");
```

```
List<UserDetails> findUserDetailsByNameIsIn(List<String> userName);
```

```
Hibernate:  
select  
    ud1_0.user_id,  
    ud1_0.user_name,  
    ud1_0.phone  
from  
    user_details ud1_0  
where  
    ud1_0.user_name in (?)
```

```
List<UserDetails> findUserDetailsByNameLike(String userName);
```

```
Hibernate:  
select  
    ud1_0.user_id,  
    ud1_0.user_name,  
    ud1_0.phone  
from
```

```

from
    user_details ud1_0
where
    ud1_0.user_name like ? escape '\'

```

Delete:

- Need to add @Transactional annotation.

```

@Transactional
void deleteByName(String userName);

```

```

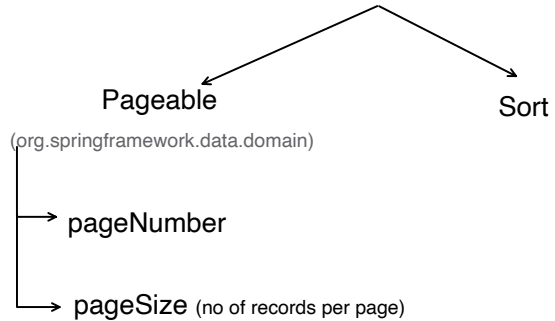
Hibernate:
    select
        ud1_0.user_id,
        ud1_0.user_name,
        ud1_0.phone
    from
        user_details ud1_0
    where
        ud1_0.user_name=?
Hibernate:
    delete
    from
        user_details
    where
        user_id=?
Hibernate:
    delete
    from
        user_details
    where
        user_id=?
Hibernate:
    delete
    from

```

```
from  
    user_details  
where  
    user_id=?
```

Paginations and Sorting in Derived Query:

. JPA provides 2 interfaces to support Pagination and Sorting i.e.




```
public interface UserDetailsRepository extends
    JpaRepository<UserDetails, Long> {

    List<UserDetails> findUserDetailsByNameStartingWith(String userName, Pageable page);
}

```

```
@Service
public class UserDetailsService {

    @Autowired
    UserDetailsRepository userDetailsRepository;

    public UserDetails saveUser(UserDetails user) {
        return userDetailsRepository.save(user);
    }

    public List<UserDetails> findByNameDerived(String name) {
        Pageable pageable = PageRequest.of( pageNumber: 0, pageSize: 5); // Page 0, 5 records per page
        return userDetailsRepository.findUserDetailsByNameStartingWith(name, pageable);
    }
}

```

If we need more info about Pages, then we can use "Page" as return type

```
@Repository
public interface UserDetailsRepository extends
    JpaRepository<UserDetails, Long> {

    Page<UserDetails> findUserDetailsByNameStartingWith(String userName, Pageable page);
}

```

```
public List<UserDetails> findByNameDerived(String name) {

```

```

public List<UserDetails> findByNameDerived(String name) {
    Pageable pageable = PageRequest.of( pageNumber: 0, pageSize: 5); // Page 0, 5 records per page
    Page<UserDetails> userDetailsPage = userDetailsRepository.findUserDetailsByNameStartingWith(name, pageable);
    List<UserDetails> userDetailsList = userDetailsPage.getContent();
    System.out.println("total pages: " + userDetailsPage.getTotalPages());
    System.out.println("is first page: " + userDetailsPage.isFirst());
    System.out.println("is last page: " + userDetailsPage.isLast());
    return userDetailsList;
}

```

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM USER_DETAILS

SELECT * FROM USER_DETAILS:

USER_ID	PHONE	USER_NAME
1	12312	A
2	12312	AB
3	12312	ABC
4	12312	ABCD
5	12312	ABCDE
6	12312	ABCDEF

(6 rows, 3 ms)

GET localhost:8080/api/user/byname_derived/A

Params Authorization Headers (6) Body Scripts Settings

Query Params

Key	Value
Key	Value

Body Cookies Headers (5) Test Results -i

Pretty Raw Preview Visualize JSON -i

```

1 [
2   {
3     "userId": 1,
4     "name": "A",
5     "phone": "12312"
6   },
7   {
8     "userId": 2,
9     "name": "AB",
10    "phone": "12312"
11  },
12  {
13    "userId": 3,
14    "name": "ABC",
15    "phone": "12312"
16  },
17  {
18    "userId": 4,
19    "name": "ABCD",
20    "phone": "12312"
21  },
22  {
23    "userId": 5,
24    "name": "ABCDE",
25    "phone": "12312"
26  }
27 ]

```

```

public List<UserDetails> findByNameDerived(String name) {
    Pageable pageable = PageRequest.of( pageNumber: 1, pageSize: 5); // Page 0, 5 records per page
    Page<UserDetails> userDetailsPage = userDetailsRepository.findUserDetailsByNameStartingWith(name, pageable);
    List<UserDetails> userDetailsList = userDetailsPage.getContent();
    System.out.println("total pages: " + userDetailsPage.getTotalPages());
}

```

```
System.out.println("is first page: " + userDetailsPage.isFirst());
System.out.println("is last page: " + userDetailsPage.isLast());
return userDetailsList;
}
```

RunRun SelectedAuto completeClearSQL statement:

SELECT * FROM USER_DETAILS

SELECT * FROM USER_DETAILS;

	USER_ID	PHONE	USER_NAME
1	12312	A	
2	12312	AB	
3	12312	ABC	
4	12312	ABCD	
5	12312	ABCDE	
6	12312	ABCDEF	

(6 rows, 3 ms)

GET

localhost:8080/api/user/byname_derived/A

Params

Authorization

Headers (6)

Body

Scripts

Settings

Query Params

Key	Value
Key	Value

Body

Cookies

Headers (5)

Test Results

Pretty

Raw

Preview

Visualize

JSON

1

[

2

{

3

"userId": 6,

4

"name": "ABCDEF",

5

"phone": "12312"

6

}

7

]

```
Hibernate:
select
    ud1_0.user_id,
    ud1_0.user_name,
    ud1_0.phone
from
    user_details ud1_0
where
    ud1_0.user_name like ? escape '\'
offset
    ? rows
fetch
    first ? rows only
total pages: 2
is first page: false
is last page: true
```

Paginations with Sorting:

```
public List<UserDetails> findByNameDerived(String name) {
    Pageable pageable = PageRequest.of( pageNumber: 0, pageSize: 5, Sort.by(...properties: "name").descending()); // Page 0, 5 records per page
    Page<UserDetails> userDetailsPage = userDetailsRepository.findUserDetailsByNameStartingWith(name, pageable);
    List<UserDetails> userDetailsList = userDetailsPage.getContent();
    System.out.println("total pages: " + userDetailsPage.getTotalPages());
    System.out.println("is first page: " + userDetailsPage.isFirst());
    System.out.println("is last page: " + userDetailsPage.isLast());
    return userDetailsList;
}
```

GET localhost:8080/api/user/byname_derived/A

Params Authorization Headers (6) Body Scripts Settings

Query Params

Key
Key

Body Cookies Headers (5) Test Results ⓘ

Pretty Raw Preview Visualize JSON ⌵

```
1 [
2   {
3     "userId": 6,
4     "name": "ABCDEF",
5     "phone": "12312"
6   },
7   {
8     "userId": 5,
9     "name": "ABCDE",
10    "phone": "12312"
11  },
12  {
13    "userId": 4,
14    "name": "ABCD",
15    "phone": "12312"
16  },
17  {
18    "userId": 3,
19    "name": "ABC",
20    "phone": "12312"
21  },
22  {
23    "userId": 2,
24    "name": "AB",
25    "phone": "12312"
26  }
27 ]
```

```

26 |     }
27 | }

```

Only Sorting:

```

@Repository
public interface UserDetailsRepository extends
    JpaRepository<UserDetails, Long> {

    List<UserDetails> findUserDetailsByNameStartingWith(String userName, Sort sort);

}

```

```

public List<UserDetails> findByNameDerived(String name) {
    return userDetailsRepository.findUserDetailsByNameStartingWith(name, Sort.by(...properties: "name").descending());
}

```

GET localhost:8080/api/user/byname_derived/A

Params Authorization Headers (6) Body Scripts Settings

Body Cookies Headers (5) Test Results ↺

Pretty Raw Preview Visualize JSON ⌵

```

1  [
2    {
3      "userId": 6,
4      "name": "ABCDEF",
5      "phone": "12312"
6    },
7    {
8      "userId": 5,
9      "name": "ABCDE",
10     "phone": "12312"

```

```

11  },
12  {
13      "userId": 4,
14      "name": "ABCD",
15      "phone": "12312"
16  },
17  {
18      "userId": 3,
19      "name": "ABC",
20      "phone": "12312"
21  },
22  {
23      "userId": 2,
24      "name": "AB",
25      "phone": "12312"
26  },
27  {
28      "userId": 1,
29      "name": "A",
30      "phone": "12312"
31  }
32 ]

```

- Sort.by accepts multiple fields.
- When multiple fields provided, sorting applied in order.
- first it sort by first field and if there are duplicates then second field is used and so on.

```

public List<UserDetails> findByNameDerived(String name) {
    return userDetailsRepository.findUserDetailsByNameStartingWith(name, Sort.by(...properties: "name", "phone").ascending());
}

```

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM USER_DETAILS

GET localhost:8080/api/user/byname_derived/A

Params Authorization Headers (6) Body Scripts Settings

Body Cookies Headers (5) Test Results ↺

Pretty Raw Preview Visualize JSON ↕

```

1  [
2  | 4

```

```
SELECT * FROM USER_DETAILS;
```

USER_ID	PHONE	USER_NAME
1	2	A
2	1	A
3	3	B

(3 rows, 3 ms)

```
1 | | |
2 | | |
3 | | |
4 | | |
5 | | |
6 | | |
7 | | |
8 | | |
9 | | |
10 | | |
11 | | |
12 | | |
```

```
{
  "userId": 2,
  "name": "A",
  "phone": "1"
},
{
  "userId": 1,
  "name": "A",
  "phone": "2"
}
```

- If we need different sorting order for different fields

```
public List<UserDetails> findByNameDerived(String name) {
    Sort sort = Sort.by(
        Sort.Order.asc( property: "name"),
        Sort.Order.desc( property: "phone")
    );
    return userDetailsRepository.findUserDetailsByNameStartingWith(name, sort);
}
```

Queries which are little complex and can't be handled via Derived Query, we can use:

JPQL:

- . Java Persistence Query Language.
- . Similar to SQL but works on **Entity Object** instead of direct database.
 - . Its database independent
 - . Works with Entity name and fields and not with table column names.

Syntax:

Entity alias, returns all the fields

This is an entity, not a table name

This is an entity field name,
not a column name

```
@Query("SELECT u FROM UserDetails u WHERE u.name = :userFirstName")  
List<UserDetails> findByUserName(@Param("userFirstName") String userName);
```

Binds, method
parameter
with named
parameter in
the query

There is no strict rule for Return type:

- you can return List or
- Single object

But, if say there are more than one rows, but in return type, we return Single

Object, then JPQL will throw an exception

JPQL query with JOIN

- OneToOne

```
@Table(name = "user_details")
@Entity
public class UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;

    @Column(name = "user_name")
    private String name;
    private String phone;

    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "user_address")
    private UserAddress userAddress;

    //getters and setters
}
```

```
@Entity
@Table(name = "user_address")
public class UserAddress {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String street;
    private String city;
    private String state;
    private String country;
    private String pinCode;

    //getters and setters
}
```

```
@Repository
public interface UserDetailsRepository extends
    JpaRepository<UserDetails, Long> {

    @Query("SELECT ud FROM UserDetails ud JOIN ud.userAddress ad WHERE ud.name = :userFirstName")
    List<UserDetails> findUserDetailsWithAddress(@Param("userFirstName") String userName);
}
```

→ We don't specifically need to put "On" here, JPA will automatically do that

```
Hibernate:
select
    ud1_0.user_id,
    ud1_0.user_name,
    ud1_0.phone,
    ud1_0.user_address
from
    user_details ud1_0
join
    user_address ua1_0
    on ua1_0.id=ud1_0.user_address
where
    ud1_0.user_name=?
```

GET localhost:8080/api/user/byname_derived/AB

Params Authorization Headers (6) Body Scripts Settings

☒ none ☐ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   {
3     "userId": 1,
4     "name": "AB",
5     "phone": "123",
6     "userAddress": {
7       "id": 1,
8       "street": null,
9       "city": "cityNameA",
10      "state": null,
11      "country": "countryNameA",
12      "pinCode": null
13    }
14  }
15 }
```

```
@Repository
public interface UserDetailsRepository extends
    JpaRepository<UserDetails, Long> {
```

```

    @Query("SELECT ud.name, ad.country FROM UserDetails ud JOIN ud.userAddress ad WHERE ud.name = :userFirstName")
    List<Object[]> findUserDetailsWithAddress(@Param("userFirstName") String userName);
}

```

```

public class UserDTO {

    String userName;
    String country;

    // Constructor to populate from UserDetails entity
    public UserDTO(String userName, String country) {
        this.userName = userName;
        this.country = country;
    }

    //getters and setters
}

```

```

public List<UserDTO> findByNameDerived(String name) {
    List<Object[]> dbOutput = userDetailsRepository.findUserDetailsWithAddress(name);
    List<UserDTO> output = new ArrayList<>();
    for(Object[] val : dbOutput) {
        String userName = (String) val[0];
        String country = (String) val[1];
        UserDTO dto = new UserDTO(userName, country);
        output.add(dto);
    }
    return output;
}

```

If we don't, want Object[] to be used, we can also return direct custom DTO

```

@Repository
public interface UserDetailsRepository extends
    JpaRepository<UserDetails, Long> {

    @Query("SELECT new com.conceptandcoding.learningspringboot.jpa.DTO.UserDTO(ud.name, ad.country) FROM UserDetails ud JOIN ud.userAddress ad WHERE ud.name = :userFirstName")
    List<UserDTO> findUserDetailsWithAddress(@Param("userFirstName") String userName);
}

```

- OneToMany

```
@Table(name = "user_details")
@Entity
public class UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;

    @Column(name = "user_name")
    private String name;
    private String phone;

    @OneToMany(cascade = CascadeType.ALL)
    @JoinColumn(name = "user_id") //fk in user address table
    private List<UserAddress> userAddressList = new ArrayList<>();

    //getters and setters
}
```

```
@Entity
@Table(name = "user_address")
public class UserAddress {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String street;
    private String city;
    private String state;
    private String country;
    private String pinCode;

    //getters and setters
}
```

```
@Query("SELECT ud FROM UserDetails ud JOIN ud.userAddressList ad WHERE ud.name = :userFirstName")
List<UserDetails> findUserDetailsWithAddress(@Param("userFirstName") String userName);
```