N+1 Problem and its Solution:

Problem :

Say, 1 User can have Many Addresses.
And our Query is such that, it can fetch more than 1 Users. Then this problem can occurs.

So, say we have 'N' Users. Then below queries will be hit by JPA:

- 1 query to fetch all the USERS.
- For each User it will fetch ADDRESSES, so for N users, it will fetch N times.

So total number of query hit : N+1.

So we need to find the way, so that only 1 QUERY it hit instead of N+1.

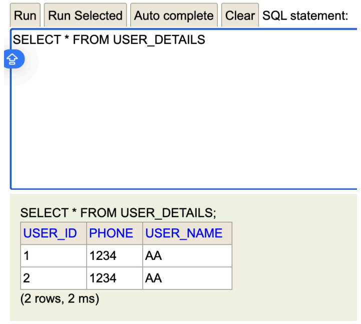
Before going for the solution for this problem, One question might be coming to our mind:

What if, we use EAGER initialization, then can we avoid this issue?

NO because EAGER initialization do not work, when our query tries to fetch multiple PARENT rows and that also have multiple CHILD.

In previous video, we tested EAGER with ***"findByID(id)"*** method, in which it make sure that, our query is fetching only 1 PARENT and that can have many CHILD, that's fine. In that JPA internally draft a JOIN query.

But when Multiple parent with Multiple child get involved, EAGER do not work in just 1 query, it first fetches all the parent and then for each parent, it fetch all its child.

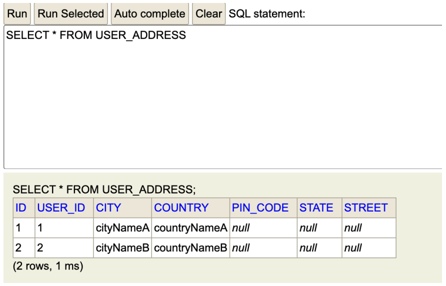


```
SELECT * FROM USER_DETAILS
```

SELECT * FROM USER_DETAILS;

USER_ID	PHONE	USER_NAME
1	1234	AA
2	1234	AA

(2 rows, 2 ms)



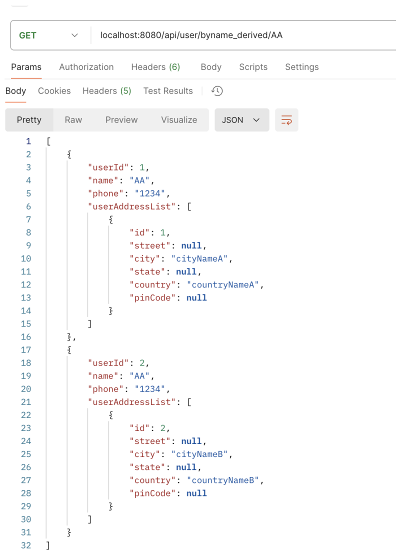
```
SELECT * FROM USER_ADDRESS
```

SELECT * FROM USER_ADDRESS;

ID	USER_ID	CITY	COUNTRY	PIN_CODE	STATE	STREET
1	1	cityNameA	countryNameA	*null*	*null*	*null*
2	2	cityNameB	countryNameB	*null*	*null*	*null*

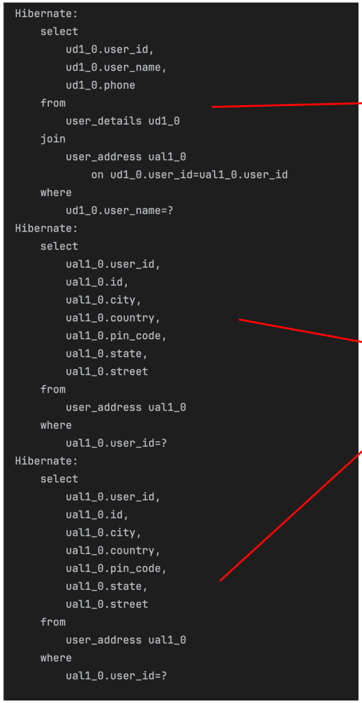
(2 rows, 1 ms)

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



```
localhost:8080/api/user/byname_derived/AA
```

```
[
    {
        "userId": 1,
        "name": "AA",
        "phone": "1234",
        "userAddressList": [
            {
                "id": 1,
                "street": null,
                "city": "cityNameA",
                "state": null,
                "country": "countryNameA",
                "pinCode": null
            }
        ]
    },
    {
        "userId": 2,
        "name": "AA",
        "phone": "1234",
        "userAddressList": [
            {
                "id": 2,
                "street": null,
                "city": "cityNameB",
                "state": null,
                "country": "countryNameB",
                "pinCode": null
            }
        ]
    }
]
```



```
Hibernate:
    select
        ud1_0.user_id,
        ud1_0.user_name,
        ud1_0.phone
    from
        user_details ud1_0
    join
        user_address ual1_0
            on ud1_0.user_id=ual1_0.user_id
    where
        ud1_0.user_name=?
Hibernate:
    select
        ual1_0.user_id,
        ual1_0.id,
        ual1_0.city,
        ual1_0.country,
        ual1_0.pin_code,
        ual1_0.state,
        ual1_0.street
    from
        user_address ual1_0
    where
        ual1_0.user_id=?
Hibernate:
    select
        ual1_0.user_id,
        ual1_0.id,
        ual1_0.city,
        ual1_0.country,
        ual1_0.pin_code,
        ual1_0.state,
        ual1_0.street
    from
        user_address ual1_0
    where
        ual1_0.user_id=?
```

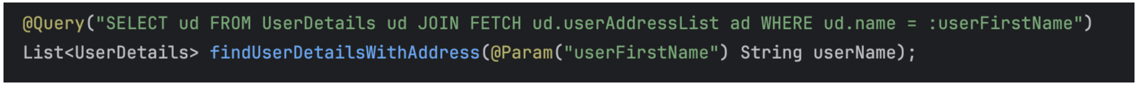
1 query to fetch all users
with Name "AA".
So it will return 2 users.

For each user
Its fetching all its addresses.

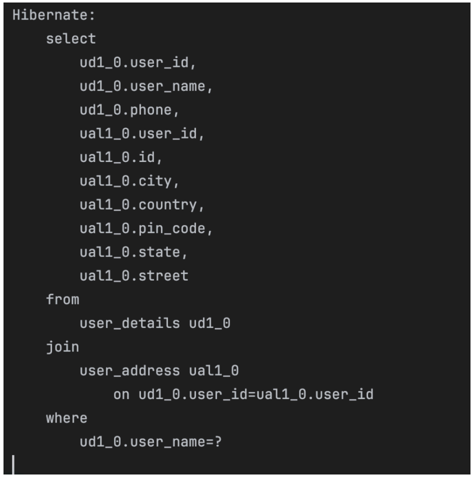
So for 2 users,
2 select query on child table

So, how to solve this, N+1 problem?

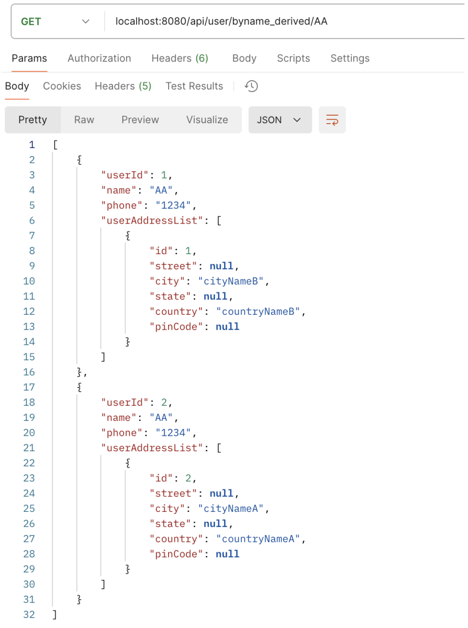
Solution1: using ***JOIN FETCH*** (JPQL)



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



```
Hibernate:
    select
        ud1_0.user_id,
        ud1_0.user_name,
        ud1_0.phone,
        ual1_0.user_id,
        ual1_0.id,
        ual1_0.city,
        ual1_0.country,
        ual1_0.pin_code,
        ual1_0.state,
        ual1_0.street
    from
        user_details ud1_0
    join
        user_address ual1_0
            on ud1_0.user_id=ual1_0.user_id
    where
        ud1_0.user_name=?
```

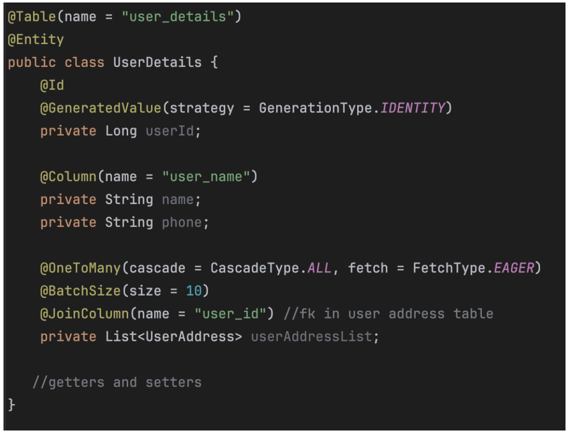


GET localhost:8080/api/user/byname_derived/AA

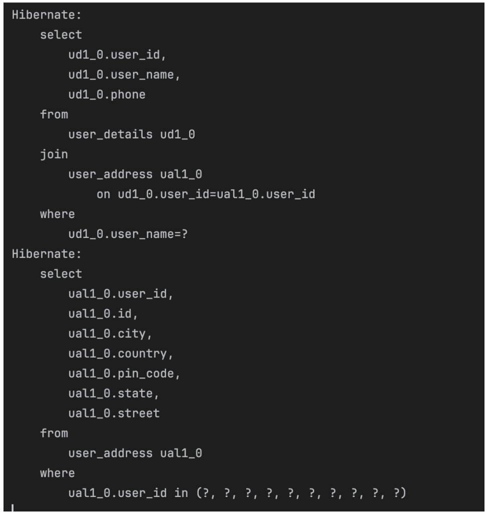
```json
[
    {
        "userId": 1,
        "name": "AA",
        "phone": "1234",
        "userAddressList": [
            {
                "id": 1,
                "street": null,
                "city": "cityNameB",
                "state": null,
                "country": "countryNameB",
                "pinCode": null
            }
        ]
    },
    {
        "userId": 2,
        "name": "AA",
        "phone": "1234",
        "userAddressList": [
            {
                "id": 2,
                "street": null,
                "city": "cityNameA",
                "state": null,
                "country": "countryNameA",
                "pinCode": null
            }
        ]
    }
]
```

Solution2: using *@BatchSize(size=10)*

- It wont make only 1 query, but it will reduce it, as it will divide it into batches



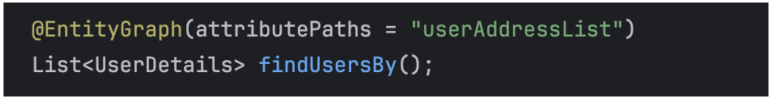
```java
@Table(name = "user_details")
@Entity
public class UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;

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

    @OneToMany(cascade = CascadeType.ALL, fetch = FetchType.EAGER)
    @BatchSize(size = 10)
    @JoinColumn(name = "user_id") //fk in user address table
    private List<UserAddress> userAddressList;

   //getters and setters
}
```



```
Hibernate:
    select
        ud1_0.user_id,
        ud1_0.user_name,
        ud1_0.phone
    from
        user_details ud1_0
    join
        user_address ual1_0
            on ud1_0.user_id=ual1_0.user_id
    where
        ud1_0.user_name=?
Hibernate:
    select
        ual1_0.user_id,
        ual1_0.id,
        ual1_0.city,
        ual1_0.country,
        ual1_0.pin_code,
        ual1_0.state,
        ual1_0.street
    from
        user_address ual1_0
    where
        ual1_0.user_id in (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
```

Solution3: using ***@EntityGraph(attributePaths="userAddressList")***

- Used over method (helpful in derived methods)
- Tell JPA to fetch all the entries of UserAddress along with user details.



```
@EntityGraph(attributePaths = "userAddressList")
List<UserDetails> findUsersBy();
```

How to join Many tables?

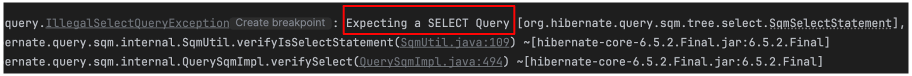
Its almost same as SQL only

Say, we have
Table A has one to many relationship with Table B
Table B has one to many relationship with Table C

```
@Query("SELECT a FROM A a  JOIN a.bList b  JOIN b.cList c WHERE c.someProperty =
:someValue")
List<A> findAWithBAndC(@Param("someValue") String someValue);
```

@Modifying Annotation

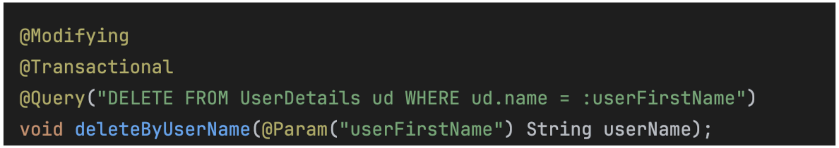
- when @Query annotation used, by-default JPA expects ***SELECT*** query.
- If we try to use "DELETE" or "INSERT" or "UPDATE" query with @Query, JPA will throw error, that:



```
query.IllegalSelectQueryException Create breakpoint : Expecting a SELECT Query [org.hibernate.query.sqm.tree.select.SqmSelectStatement],
ernate.query.sqm.internal.SqmUtil.verifyIsSelectStatement(SqmUtil.java:109) ~[hibernate-core-6.5.2.Final.jar:6.5.2.Final]
ernate.query.sqm.internal.QuerySqmImpl.verifySelect(QuerySqmImpl.java:494) ~[hibernate-core-6.5.2.Final.jar:6.5.2.Final]
```

- @Modifying annotation, is to tell JPA that, expect either "DELETE" or "INSERT" or "UPDATE" query with @Query

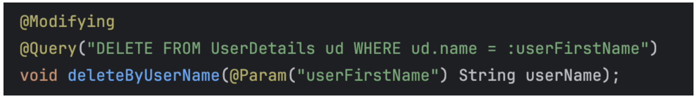
- Since we are trying to update the DB, we also need to use @Transactional annotation.



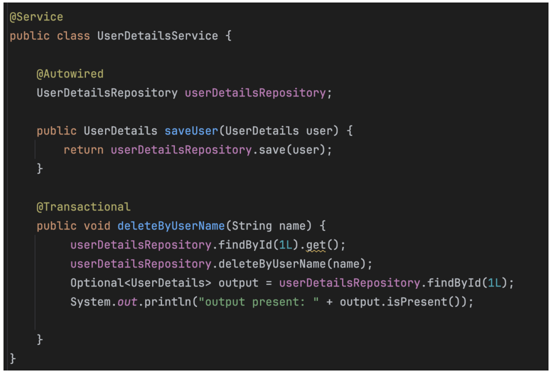
```java
@Modifying
@Transactional
@Query("DELETE FROM UserDetails ud WHERE ud.name = :userFirstName")
void deleteByUserName(@Param("userFirstName") String userName);
```

Understanding Usage of Flush and Clear:

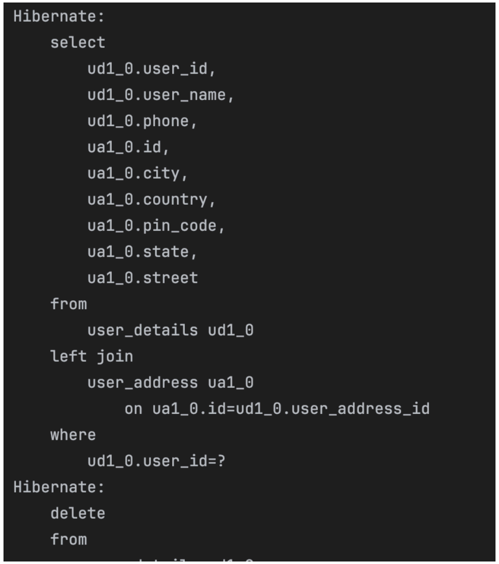
- **As we know, Flush just pushed the persistence context changes to DB but hold the value in persistence context.**
- **Clear, purge the persistence context, and required fresh DB call**



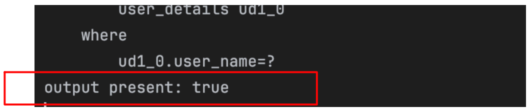
```java
@Modifying
@Query("DELETE FROM UserDetails ud WHERE ud.name = :userFirstName")
void deleteByUserName(@Param("userFirstName") String userName);
```



```java
@Service
public class UserDetailsService {

    @Autowired
    UserDetailsRepository userDetailsRepository;

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

    @Transactional
    public void deleteByUserName(String name) {
        userDetailsRepository.findById(1L).get();
        userDetailsRepository.deleteByUserName(name);
        Optional<UserDetails> output = userDetailsRepository.findById(1L);
        System.out.println("output present: " + output.isPresent());

    }
}
```

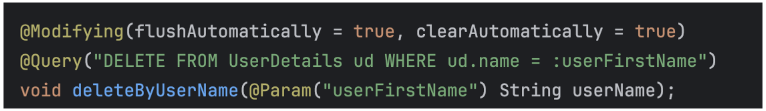


```
Hibernate:
    select
        ud1_0.user_id,
        ud1_0.user_name,
        ud1_0.phone,
        ua1_0.id,
        ua1_0.city,
        ua1_0.country,
        ua1_0.pin_code,
        ua1_0.state,
        ua1_0.street
    from
        user_details ud1_0
    left join
        user_address ua1_0
            on ua1_0.id=ud1_0.user_address_id
    where
        ud1_0.user_id=?
Hibernate:
    delete
    from
```

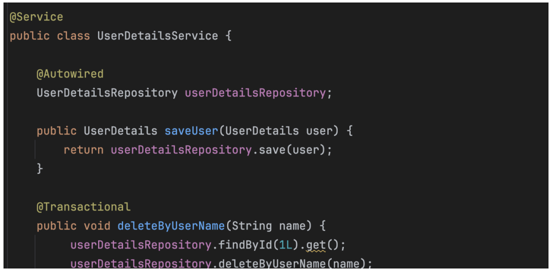


```
        user_details ud1_0
    where
        ud1_0.user_name=?
output present: true
```

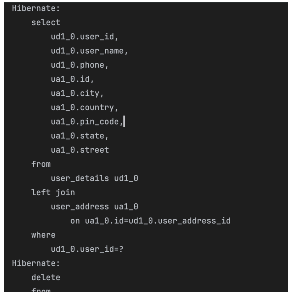
Now using, Flush and Clear



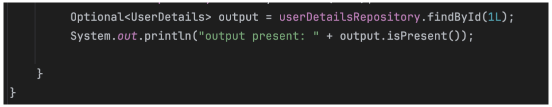
```java
@Modifying(flushAutomatically = true, clearAutomatically = true)
@Query("DELETE FROM UserDetails ud WHERE ud.name = :userFirstName")
void deleteByUserName(@Param("userFirstName") String userName);
```



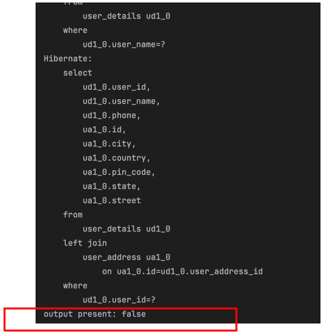
```java
@Service
public class UserDetailsService {

    @Autowired
    UserDetailsRepository userDetailsRepository;

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

    @Transactional
    public void deleteByUserName(String name) {
        userDetailsRepository.findById(1L).get();
        userDetailsRepository.deleteByUserName(name);
```



```
Hibernate:
    select
        ud1_0.user_id,
        ud1_0.user_name,
        ud1_0.phone,
        ua1_0.id,
        ua1_0.city,
        ua1_0.country,
        ua1_0.pin_code,
        ua1_0.state,
        ua1_0.street
    from
        user_details ud1_0
    left join
        user_address ua1_0
            on ua1_0.id=ud1_0.user_address_id
    where
        ud1_0.user_id=?
Hibernate:
    delete
    from
```



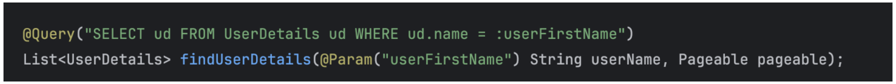
```
        Optional<UserDetails> output = userDetailsRepository.findById(1L);
        System.out.println("output present: " + output.isPresent());

    }
}
```



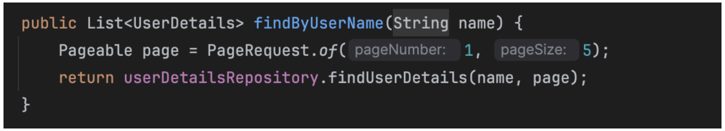
```
            user_details ud1_0
        where
            ud1_0.user_name=?
Hibernate:
        select
            ud1_0.user_id,
            ud1_0.user_name,
            ud1_0.phone,
            ua1_0.id,
            ua1_0.city,
            ua1_0.country,
            ua1_0.pin_code,
            ua1_0.state,
            ua1_0.street
        from
            user_details ud1_0
        left join
            user_address ua1_0
                on ua1_0.id=ud1_0.user_address_id
        where
            ud1_0.user_id=?
output present: false
```

Pagination and Sorting in JPQL

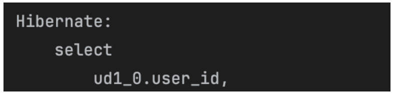
Same like discussed in derived query method



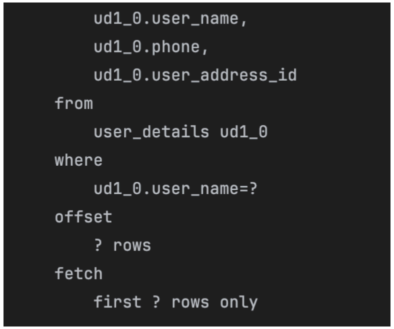
```
@Query("SELECT ud FROM UserDetails ud WHERE ud.name = :userFirstName")
List<UserDetails> findUserDetails(@Param("userFirstName") String userName, Pageable pageable);
```



```
public List<UserDetails> findByUserName(String name) {
    Pageable page = PageRequest.of( pageNumber: 1,  pageSize: 5);
    return userDetailsRepository.findUserDetails(name, page);
}
```



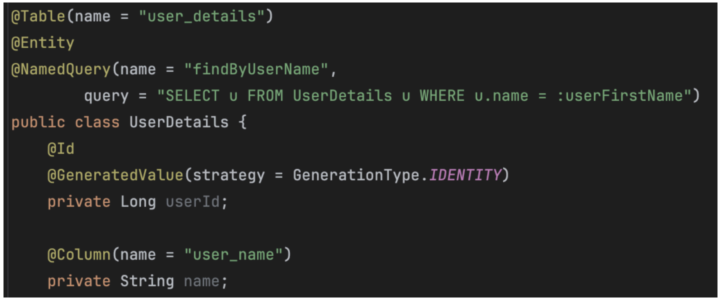
```
Hibernate:
    select
        ud1_0.user_id,
```



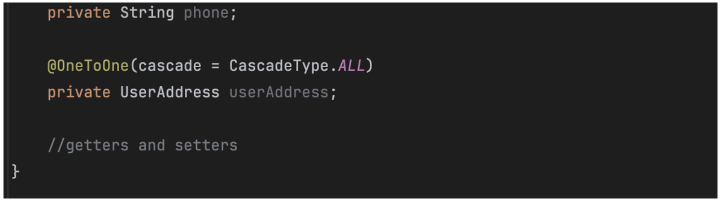
```
        ud1_0.user_name,
        ud1_0.phone,
        ud1_0.user_address_id
    from
        user_details ud1_0
    where
        ud1_0.user_name=?
    offset
        ? rows
    fetch
        first ? rows only
```

@NamedQuery Annotation

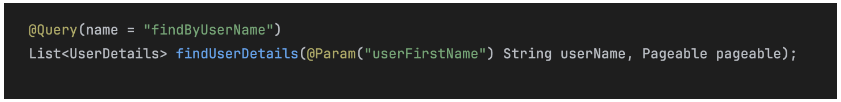
- We can name our Query, so that we can reuse it.



```
@Table(name = "user_details")
@Entity
@NamedQuery(name = "findByUserName",
        query = "SELECT u FROM UserDetails u WHERE u.name = :userFirstName")
public class UserDetails {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long userId;

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



```java
    private String phone;

    @OneToOne(cascade = CascadeType.ALL)
    private UserAddress userAddress;

    //getters and setters
}
```



```java
@Query(name = "findByUserName")
List<UserDetails> findUserDetails(@Param("userFirstName") String userName, Pageable pageable);
```