**Fetch Strategy**

By default GORM service defines relationship between entities, extrac related entities based on defined relationship. When extracting related Entity, Lazy Loading returns to Proxy object without executing SQL before object is actually required by default. However, if it is processed as Lazy Loading, a problem occurs in that query should be performed several times as necessary at the required time in case when Lazy Loading is not performed. To solve such problem, several Fetch strategies exist including **data inquiry using batch**, **data inquiry using Sub-Query**, **the method to inquire data all at once using Join Fetch**. However, this service is defined in implement Hibernate, rather than JPA standards.

**Data Search Using the Batch**

In case the BatchSize is designated on the entity class, the query execution number decreased by n/batch size +1 by loading relevant object for the designated number of times. Next is a part of department class file as an example of batch-size.

**Config Source**

```
@Entity
public class Department implements Serializable {
    ...
    @org.hibernate.annotations.BatchSize(size=2)
    private Set<User> users ;
    ...
}
```

Designate BatchSize applied when extracting Set<User> that belongs to Department as 2.

**Sample Source**

```
    qlBuf.append("FROM Department");
    Query query = em.createQuery(qlBuf.toString());

    List deptList = query.getResultList();

    assertEquals("fail to match the size of department list.", 3, deptList.size());

    for (int i = 0; i < deptList.size(); i++) {

        Department department = (Department) deptList.get(i);

        if (i == 0) {
            assertEquals("fail to match a department name.", "HRD", department.getDeptName());

            Set users = department.getUsers();
            assertEquals("fail to match the size of user list.", 2, users.size());

        } else if (i == 1) {
            assertEquals("fail to match a department name.", "PD", department.getDeptName());

            Set users = department.getUsers();
            assertEquals("fail to match the size of user list.", 1, users.size());
    ...
```

Following is SQL Query created automatically when calling department.getUsers() and query.getResultList() by sample and above Config.

**Generated SQL**

```
SELECT ... FROM USER user0_
```

```
SELECT ...
FROM USER users0_
WHERE users0_.DEPT_ID IN (?, ?)
```

The number '?' above where sentence [in (?,?)] is BatchSize. Since it is set as 2, 2 are designated and inquired.

**Data inquiry using Sub-Query**

When the FetchMode is designated as SUBSELECTET in the entity class, subquery form SELECT option is executed and loaded all at the same time. Next is a part of the example of SUBSELECT configuration, USER Class.

**Config Source**

```
@Entity
public class User implements Serializable {
    ...
    @org.hibernate.annotations.Fetch(org.hibernate.annotations.FetchMode.SUBSELECT)
    private Set<Role> roles = new HashSet(0);
    ...
}
```

Designate the FetchMode applied when extracting Set<Role> that belongs to user as SUBSELECT.

**Sample Source**

```
    qlBuf.append("FROM User");
    Query query = em.createQuery(qlBuf.toString());
    List userList = query.getResultList();

    assertEquals("fail to match the size of user list.", 3, userList.size());

    for (int i = 0; i < userList.size(); i++) {
        User user = (User) userList.get(i);

        if (i == 0) {
            assertEquals("fail to match a user name.", "kim" , user.getUserName());
            assertEquals("fail to match a user password.", "kim123" , user.getPassword());

            Set roles = user.getRoles();
            assertEquals("fail to match the size of role list.", 2 , roles.size());
        } else if (i == 1) {
            assertEquals("fail to match a user name.", "lee" , user.getUserName());
            assertEquals("fail to match a user password.", "lee123" , user.getPassword());
            ...
```

Following is SQL Query automatically created when user.getRoles() and query.getResultList() calling by Sample and above Config.

**Generated SQL**

```
SELECT ... FROM USER user0_

SELECT ...
FROM AUTHORITY roles0_ LEFT OUTER JOIN ROLE role1_ ON roles0_.ROLE_ID=role1_.ROLE_ID
WHERE roles0_.USER_ID IN (SELECT user0_.USER_ID FROM USER user0_)
```

Above where sentence [in (select user0_.USER_ID from USER user0_)] shows that all Roles information is extracted for all users corresponding in Sub Query type.

**How to search data using join fetch**

Without separate configuration in the entity class, the join fetch can be written in QL execution to extract all child entities at once.

**Sample Source**

```
qlBuf.append("SELECT user ");
// Use JOIN FETCH .
qlBuf.append("FROM User user join fetch user.roles role ");
qlBuf.append("WHERE role.roleName = ?");
Query query = em.createQuery(qlBuf.toString());
query.setParameter(1, "Admin");
List userList = query.getResultList();

assertEquals("fail to match the size of user list.", 2, userList.size());

for (int i = 0; i < userList.size(); i++) {
    User user = (User) userList.get(i);

    if (i == 0) {
        assertEquals("fail to match a user name.", "kim",user.getUserName());
        assertEquals("fail to match a user password.", "kim123",user.getPassword());

        Set roles = user.getRoles();
        assertEquals("fail to match the size of role list.", 1, roles.size());
    } else if (i == 1) {
        assertEquals("fail to match a user name.", "lee",user.getUserName());
        assertEquals("fail to match a user password.", "lee123",user.getPassword());
    ...
```

Above example shows that keyword of join fetch was used in [FROM User user join fetch user.roles role]. SQL created by this is as follows.

**Generated SQL**

```
SELECT ...
FROM USER user0_ INNER JOIN AUTHORITY roles1_ ON user0_.USER_ID=roles1_.USER_ID
                 INNER JOIN ROLE role2_ ON roles1_.ROLE_ID=role2_.ROLE_ID
WHERE role2_.ROLE_NAME=?
```

Above SQL shows that ROLE information is extracted related to JOIN as SQL executed by query.getResultList().