Entity Operation

This section will study the method of input, update, find, remove, disposition, the data on a certain DB using the ORM service.

Input

The persist() method in the EntityManager should be called to add the data case by case on the DB. The entity in calling the persist() method should be transmitted as the input element.

Sample Source

```
private Department addDepartment() throws Exception {
    // 1. insert a new Department information
    Department department = new Department();
    String DepartmentId = "DEPT-0001";
    department.setDeptId(DepartmentId);
    department.setDeptName("SaleDept");
    department.setDesc("Sales Department");

    em.persist(department);
    ...
    return department;
}
```

Above example delivers and processes the Entity of department as input factor to persis() method of EntityManager.

Update

Updating data is possible in two ways. First, it can be updated by calling the merge() method in the EntityManager, and revising case by case on the DB. When a certain object is in persistent state and there were changes in properties of the object, the change is checked when terminating transaction without calling the method directly and the change is reflected on the DB.

merge Sample Source with calling

```
public void testUpdateDepartment() throws Exception {
    // 1. insert a new Department information
    Department department = addDepartment();

    // 2. update a Department information
    department.setDeptName("Purchase Dept");

    // 3. explicit method calling
    em.merge(department);
}
```

Above example delivers and processes the Entity of department as input factor to merge() method of EntityManager.

merge Sample Source without calling

```
public void testUpdateDepartment() throws Exception {
    // 1. insert a new Department information
    Department department = addDepartment();

    // 2. update a Department information
    department.setDeptName("Purchase Dept");
```

```
    // commit. successful update!!!
}
```

Above example changes through setDeptName() and is changed at the time of Commit processing.

Find

One case data can be inquired from the DB by calling the find() method in the entity manager. The ID of the subject entity in calling the find() method should be transmitted as an input factor.

Sample Source

```
private void assertDepartmentInfo(String departmentId, Department department)
                throws Exception {

    Department result = (Department) em.find(Department.class, departmentId);

    //...
}
```

Above example delivers and processes the Entity Id of departmentId as input factor to the find() method of EntityManager.

Remove

One case data can be inquired from the DB by calling the remove() method in the entity manager. Deliver the Entity that becomes the target at the time of remove() method calling as input factor.

Sample Source

```
public void testDeleteDepartment() throws Exception {

    // 1. insert a new Department information
    Department department = addDepartment();

    // 2. delete a Department information
    em.remove(department);

}
```

Above example delivers and processes the Entity of department to remove() method of EntityManager. However, it should be noted that in above example, since department object is same object as the one registered in DB, remove cannot be used as it is. But, if only key is created new identically, remove cannot be used directly. In this case, process as follows.

Sample Source

```
public void testDeleteDepartment() throws Exception {

    Department department = new Department();
    department.setDeptId = "DEPT_1";

    // 2. delete a Department information
    em.remove(em.getReference(Department.class, department.getDeptId()));

}
```

In above example, call getReference method, extract the object information corresponding to Id of Entity and call remove with that information as input factor.

Disposition

One case data can be inquired from the DB by calling the persist() method in the entity manager.Delete items in the memory by calling flush(),clear() with certain term in order to prevent OutOfMemoryException.

Sample Source

```
public void testMultiSave() throws Exception {

    for (int i = 0; i < 900 ; i++) {

        Department department = new Department();
        String DeptId = "DEPT-000" + i;
        department.setDeptId(DeptId);
        department.setDeptName("Sale" + i);
        department.setDesc("Sales Department" + i);

        em.persist(department);
        logger.debug("=== DEPT-000"+i+" ===");

        // To avoid OutOfMemoryException
        if (i != 0 && i % 9 == 0) {
            em.flush();
            em.clear();
        }
    }
}
```

Callback Methods

Callback function is defined by defining entity classes or EntityListner to separate the logics as business logic checking into before and after the actual entity operation.

Callback Methods

Method Name	Description	Related Operation
PrePersist	Execute at the timing before Persist	persist
PostPersist	Execute at the timing after Persist	persist
PreRemove	Execute at the timing before Remove	remove
PostRemove	Execute at the timing after Remove	remove
PreUpdate	Execute at the timing before Update	merge
PostUpdate	Execute at the timing after Update	merge
PostLoad	Execute at the timing after Find	find

Defining in Entity

The callback function is subscribed on the annotation directly on the entity class to define methods.

Define Source – Define in Entity class

```
@Entity
public class User {
    @PrePersist
    @PreUpdate
    protected void validateCreate() throws Exception {
        if (getSalary() < 2000000 )
            throw new Exception("Insufficient Salary !");
    }
```

```
}
```

Above example shows whether salary is below 2,000,000 before the timing before Persist, Update. Update corresponds to the case of using merge() of EntityManager and the case of performing update using ql.

Sample Source - merge() use case

```
@Test
public void testUpdateFailUser() throws Exception {

    newTransaction();

    User getUser = (User) em.find(User.class, "User1");
    assertEquals(getUser.getSalary(), sal );
    user.setSalary(1000000);

    em.merge(user);

    // 2. Update User , execute update when transaction terminates rather than above merge() calling.

    try{
        closeTransaction();
    }
        catch( Exception e ){
        e.printStackTrace();
        assertTrue("fail to PreUpdate.",e instanceof Exception);
    }
}
```

Above example shows that Exception occurs if the salary is set to below 2000000 and updated.

Sample Source - Update case through ql

```
@Test
public void testUpdateFail2User() throws Exception {

    newTransaction();

    User getUser = (User) em.find(User.class, "User1");

    StringBuffer ql = new StringBuffer();
    ql.append("UPDATE User user ");
    ql.append("SET user.salary = :salary ");
    ql.append("WHERE user.userId = :userId ");

    Query query = em.createQuery(ql.toString());
    query.setParameter("salary", 1000000);
    query.setParameter("userId",getUser.getUserId());

    // 2. Update User , Update executed when terminating Transaction rather than above merge()
calling.
    try{
        closeTransaction();
    }
        catch( Exception e ){
        e.printStackTrace();
        assertTrue("fail to PreUpdate.",e instanceof Exception);
    }
}
```

Above example shows that exception occurs if the salary is set to below 2000000 and updated. It can be also checked that Exception occurs identically when requesting the Update using ql.

Defining EntityListener

The EntityListener is indicated on the entity class. Indicate the annotation on the EntityListener to define methods.

Define Source – Define to EntityListener class

```
@Entity
@EntityListeners(egovframework.sample.model.callback.AlertMonitor.class)
public class User {
}

// AlertMonitor class defined above
public class AlertMonitor {
    @PostPersist
    public void newUserAlert(User user) {
        System.out.println(user.getUserName()+" Created!");
    }

    @PostLoad
    public void usrGetAlert(User user) {
        System.out.println(user.getUserName()+" Get!");
    }
}
```

It differs in that the location of definition differs and original entity should be transferred to the parameter, but it works same as the designation in entity class.