Native SQL

The basic API of JPA or QL can be used to executed CRUD. However, it supports use of Native SQL to use the function provided by specific DBMS.

Basic method

entityManager.createNativeQuery() method is used to execute Native SQL.

Inquire basic list

Inquiry can be executed for one table through SQL.

Sample Source

```
StringBuffer qlBuf = new StringBuffer();

qlBuf.append("SELECT * ");
qlBuf.append("FROM DEPARTMENT ");
qlBuf.append("WHERE DEPT_NAME like :condition ");
qlBuf.append("ORDER BY DEPT_NAME");

Query query = em.createNativeQuery(qlBuf.toString(),Department.class);
query.setParameter("condition", "%%");

List deptList = query.getResultList();
```

List of a Department object matching the inquiry conditions is returned through SQL sentence defined as above. Inquiry condition can be completed through Named Paramenter in WHERE statement using ':'. Value of inquiry condition is designated through setParameter() method of Query. In addition, it Entity class (Department.class) to return as the second factor of createNativeQuery.

List inquiry through JOIN

Inquiry using Native SQL (Inner Join) can be executed through two related tables.

Basic Source

```
StringBuffer qlBuf = new StringBuffer();

qlBuf.append("SELECT user.* ");
qlBuf.append("FROM USER user ");
qlBuf.append("join AUTHORITY auth on user.USER_ID = auth.USER_ID ");
qlBuf.append("join ROLE role on auth.ROLE_ID = role.ROLE_ID ");
qlBuf.append("WHERE role.ROLE_NAME = ?");

Query query = em.createNativeQuery(qlBuf.toString(),User.class);
query.setParameter(1, "Admin");

List userList = query.getResultList();
```

Inner Join can be performed using join keyword as above code. In addition, inquiry task can be performed using Native SQL(Right Outer Join) on 2 tables in Relation.

RIGHT JOIN Source

```
StringBuffer qlBuf = new StringBuffer();

qlBuf.append("SELECT distinct role.* ");
qlBuf.append("FROM USER user ");
qlBuf.append("right join AUTHORITY auth on user.USER_ID=auth.USER_ID ");
```

```
qlBuf.append("right join ROLE role on auth.ROLE_ID=role.ROLE_ID ");
qlBuf.append("ORDER BY role.ROLE_NAME ASC ");

Query query = em.createNativeQuery(qlBuf.toString(),Role.class);

List roleList = query.getResultList();
```

In addition, to select the inquired results in the value of object joined, indicate and execute the name defined in @SqlResultSetMapping in the second factor of createNativeQuery.

Multi Entity Result Source

```
StringBuffer qlBuf = new StringBuffer();

qlBuf.append("SELECT distinct user.*, department.* ");
qlBuf.append("FROM USER user, DEPARTMENT department ");
qlBuf.append("WHERE user.DEPT_ID = department.DEPT_ID ");
qlBuf.append("AND department.DEPT_NAME = :condition1 ");
qlBuf.append("AND user.USER_NAME like :condition2 ");

Query query = em.createNativeQuery(qlBuf.toString(), "UserAndDept" ) ;
query.setParameter("condition1", "HRD");
query.setParameter("condition2", "%%");

List userList = query.getResultList();
```

Above example shows that it defines the Entity class to return in the name of UserAndDept in User Entity Class.

SqlResultSetMapping Define Source

```
@SqlResultSetMapping(name="UserAndDept",entities={@EntityResult(entityClass=User.class),
                                                  @EntityResult(entityClass=Department.class)
                                                 }
                     )
@Entity
public class User implements Serializable {
}
```

Above example shows that UserAndDept is defined through Annotation in User Entity class. In addition, each can be extracted as shown below.

Multi Entity Result Use Source

```
Object[] results = (Object[]) userList.get(0);

User user1 = (User)results[0];
Department dept1 = (Department)results[1];
```

Named Query

The name of SQL defined by annotation on thin entity class file can be input.

Sample Source

```
Query query = em.createNamedQuery("nativeFindDeptList");
query.setParameter("condition", "%%");

List deptList = query.getResultList();
```

As shown above, deliver query name to createNamedQuery() method as above and execute by finding QL sentence matching this name. Following is the part of Department Entity class containing nativeFindDeptList.

Entity Source

```
@Entity
@NamedNativeQuery(name="nativeFindDeptList",
            resultClass=Department.class,
                  query="SELECT * FROM DEPARTMENT department " +
                        "WHERE department.DEPT_Name like :condition "+
                        "ORDER BY department.DEPT_Name" )
public class Department implements Serializable {
...
}
```

As shown above, it differs from NamedQuery of QL in that it explicitly indicates resultClass.

Processing Paging

Processing paging is objected to decrease the DB or application memory load by limiting the inquiry in one page. Let's examine the method to obtain the inquiry results paged when executing Native SQL. Execute the inquiry tasks using Native SQL targeting specific table(USER table in the example). At this time, paging becomes possible by defining the number of Row(FirstResult) to start inquiry and the number of inquiry list(MaxResult).

Sample Source

```
StringBuffer qlBuf = new StringBuffer();

qlBuf.append("SELECT * ");
qlBuf.append("FROM User ");
Query query = em.createNativeQuery(qlBuf.toString(),User.class);

query.setFirstResult(1);
query.setMaxResults(2);

List userList = query.getResultList();
```

If defining as above, create SQL matching each DB according to hibernate.dialect property defined in persistence.xml file in QL. It is not to deliver the data number that belongs to the relevant page after reading all data at the time of Pagination, but to read the data to inquire, that is, the data as many as the data that belongs to relevant page.

Calling Function

Native SQL can be executed and confirm results using the function created in the DB.

Sample Source

```
StringBuffer qlBuf = new StringBuffer();

qlBuf.append("SELECT * FROM USER_TBL ");
qlBuf.append("WHERE salary > FIND_USER(:condition)");

Query query = em.createNativeQuery(qlBuf.toString(),User.class);
query.setParameter("condition", "User1");

List userList = query.getResultList();
```

The above example shows that function of FIND_USER is called and comparison is performed in WHERE sentence. In case of Procedure, it was impossible to check how to process input/output factor processing and to explain it as an example.