**Query Language**

In JPA, separate query language is provided to enable inquiries on certain object and independent query definition on DB types based on instantiation aspect. The elements and creating method is as following.

**Elements**

The QL statements are two types: SELECT statement, and UPDATE and DELETE statement.

- SELECT statement: SELECT option + FROM option + WHERE option + ORDER BY option + GROUP BY option
- UPDATE&DELETE statement: UPDATE/DELETE option + WHERE option

Let's examine each option in the following.

**SELECT Option**

Defined to indicate inquiry data in detail.
SELECT [object or property], Aggregate Functions, etc.

If inquiring several cases of data, result value can be defined as List, Map or user-defined Type. (Default = Object[])

SELECT new List (prop1, prop2, …)

Possible Aggregate Functions

COUNT: Return to Long
MAX, MIN: Return to a defined field
AVG: Return to Double
SUM: Long in case of integral type; Double in case of float type; BigInteger in case of BigInteger; BigDecimal in case of BigDecimal

**Main Functions used in QL**

**String Functions**

| Function | Description |
|---|---|
| CONCAT(str1, str2) | Connect two strings |
| SUBSTRING(str, idx, length) | Calculate strings of length size in designated idx location |
| TRIM([type] str) | Delete leading and trailing blanks (When the type is BOTH, delete leading and trailing blanks, when the type is leading, delete the leading blank, and when the type is trailing, delete the trailing blank) |
| LOWER(str) | Convert to lower case |
| UPPER(str) | Convert to upper case |
| LENGTH(str) | Locate total length |
| LOCATE(str, s, idx) | Locate 's' defined in related str. The starting location is idx. |

**Arithmetic Functions**

| Function | Description |
|---|---|

| | |
|---|---|
| ABS(num) | Calculate the absolute number. |
| SQRT(num) | Calculate square root of the number. |
| MOD(num1,num2) | Calculate the remaining value from dividing num1 by num 2 |
| SIZE(collection value) | Calculate the entry number included in the collection |

**DateTime Functions**

| Function | Description |
|---|---|
| CURRENT_DATE | Calculate current data |
| CURRENT_TIME | Calculate current time |
| CURRENT_TIMESTAMP | Calculate current date and time |

**FROM Option**

Defines inquiring object, when SELECT option is omitted, the object defined in the FROM option becomes the subject to be transmitted.

FROM [object] ((AS) alias), …

**JOINS**

JOIN can be used in FROM statement. Following is the type of JOIN.

| JOIN Type | Example | Description |
|---|---|---|
| Inner Joins | SELECT c FROM Customer c **JOIN** c.orders o WHERE c.status = 1 | Extracts the case where both for comparison exists (extract the customers with orders only) |
| Left Outer Joins | SELECT c FROM Customer c **LEFT JOIN** c.orders o WHERE c.status = 1 | Extracts even if it exists only in one part(the customer without Order also extracts) |
| Fetch Joins | SELECT d FROM Department d LEFT JOIN **FETCH** d.employees WHERE d.deptno = 1 | Child list following FETCH statement is extracted together (extract employee list that has attribute of Department, in case of lazy loading, employee information is not extracted if not fetched.) |

**WHERE Option**

Defined to make more detailed classification on the result of inquiry.
WHERE [condition], …

There are several expressions that indicate the conditions. Following are the main examples.

| Conditions | Description | Example |
|---|---|---|
| Path Expressions | Indicate attribute of entity class | user.roles |
| Named Parameters | Indicate the parameter indicating the name. The value can be designated through setParameter | WHERE department.deptName like:condition |
| Positional Parameters | Can designate the parameter indicating the location and designate the value through setParameter. | WHERE role.roleName = ?1 |
| Collection Member Expressions | Process Collection type Attribute as condition in the expression of ”[NOT] MEMBER [OF]”. | user MEMBER OF role.users |

Besides, expressions such as IN , LIKE , IS NULL , EXISTS , Function are supported.

**ORDER BY Option**

Defines how to order results.

ORDER BY [condition] (ASC or DESC), …

**GROUP BY Option**

Defines grouping of results

GROUP BY [condition], …
[HAVING] [condition]

**Basic methods**

Representative method of use will be explained based on the example sources. Basic CRUD and JOIN method is as following.

**Example**

Inquiry can be executed on one table through QL

**Sample Source**

```
StringBuffer qlBuf = new StringBuffer();
qlBuf.append("FROM Department department ");
qlBuf.append("WHERE department.deptName like:condition ");
qlBuf.append("ORDER BY department.deptName");

Query qlQuery = em.createQuery(qlBuf.toString());
qlQuery.setParameter("condition", "%%");

List departmentList = qlQuery.getResultList();
```

List of Department object matching the search conditions are returned through QL statement defined as above. Search condition of WHERE option can be defined in the object name. Attribute name(department.deptName) and search condition can be completed through Named Paramenter using ':'. Value of inquiry condition is designated through setParameter() method of Query.

**List Inquiry through JOIN**

Execute INNER JOIN and LEFT OUTER JOIN. The example is as following.

**INNER JOIN (1)**

```
StringBuffer qlBuf = new StringBuffer();

qlBuf.append("SELECT user ");
qlBuf.append("FROM User user join user.roles role ");
qlBuf.append("WHERE role.roleName = ?1");

Query query = em.createQuery(qlBuf.toString());
query.setParameter(1, "Admin");

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

Through QL statement defined as above, List of Department object matching the inquiry condition is returned. It was processed INNER JOIN using JOIN in FROM option and the inquiry condition of WHERE option can be defined as object name. Attribute name (department.deptName). Inquiry condition can

be completed using '?!' through Positional Paramenter. Value of search condition is designated through setParameter() method of Query.

**INNER JOIN (2)**

```
StringBuffer qlBuf = new StringBuffer();

qlBuf.append("SELECT distinct user ");
qlBuf.append("FROM User user, Department department ");
qlBuf.append("WHERE user.department.deptId = department.deptId ");
qlBuf.append("AND department.deptId =:condition1 ");
qlBuf.append("AND user.userName like:condition2 ");

Query query = em.createQuery(qlBuf.toString());
query.setParameter("condition1", "Dept1");
query.setParameter("condition2", "%%");

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

Through QL statement defined as above, List of Department object matching the inquiry condition is returned. Through WHERE option, it was processed INNER JOIN through '=' and the inquiry condition can be defined as object name. Attribute name(department.deptName) using '?!' and completed through Positional Paramenter. Value of inquiry condition is designated through set Parameter() method of Query.

**LEFT OUTER JOIN**

```
StringBuffer qlBuf = new StringBuffer();

qlBuf.append("SELECT distinct role ");
qlBuf.append("FROM Role role left outer join role.users user ");
qlBuf.append("ORDER BY role.roleName ASC ");

Query query = em.createQuery(qlBuf.toString());

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

Through QL statement defined as above, the List of role object matching condition is returned. FROM option 에서 LEFT OUTER JOIN processing was performed in FROM option. Since it is LEFT OUTER JOIN, it is extracted even if information in RIGHT is extracted. In above example, ROLE information without USER information is all listed.

**Defined Return Type**

After inquiry, the result can be transmitted as desired object type.It can be used to return as composite class rather than Persistence class mapped to one table if joining several tables.

**Invoking as Certain Object Formats**

It receives inquiry results as a certain object (user object format in the example) using QL (INNER JOIN) in two related tables.

```
StringBuffer qlBuf = new StringBuffer();
    qlBuf.append("SELECT new User(user.userId as userId, ");
    qlBuf.append("user.userName as userName, user.password as password, ");
    qlBuf.append("role.roleName as roleName, ");
    qlBuf.append("user.department.deptName as deptName) ");
    qlBuf.append("FROM User user join user.roles role ");
    qlBuf.append("WHERE role.roleName =:condition");

    Query query = em.createQuery(qlBuf.toString());
```

```
query.setParameter("condition", "Admin");

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

What should be noted that the creator is called through new User(…) and this creator should be defined in User class. In addition, to get out the value corresponding to each attribute in returned value, get out each user object from list and use the getter method.

```
User user1 = (User) userList.get(0);
user1.getUserName());
User user2 = (User) userList.get(1);
user2.getUserName());
```

**Invoking as Map Formats**

It receives inquiry results as a map format using QL (INNER JOIN) in two related tables.

```
StringBuffer qlBuf = new StringBuffer();

qlBuf.append("SELECT new Map(user.userId as userId, ");
qlBuf.append("user.userName as userName, user.password as password, ");
qlBuf.append("role.roleName as roleName, ");
qlBuf.append("user.department.deptName as deptName) ");
qlBuf.append("FROM User user join user.roles role ");
qlBuf.append("WHERE role.roleName =:condition");

Query query = em.createQuery(qlBuf.toString());
query.setParameter("condition", "Admin");

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

If defining as above, the inquiry result becomes the form of Map List. At this time, userId, userName, password, roleName, deptName defined as alias become the key value of map. Accordingly, result value can be inquired through key value defined as Map as shown below.

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

Map user1 = (Map) userList.get(0);
user1.get("userId");
user1.get("userName");
...
```

**Invoking as List Format**

It receives inquiry results as a list format using QL (INNER JOIN) in two related tables.

```
StringBuffer qlBuf = new StringBuffer();

qlBuf.append("SELECT new List(user.userId as userId, ");
qlBuf.append("user.userName as userName, user.password as password, ");
qlBuf.append("role.roleName as roleName, ");
qlBuf.append("user.department.deptName as deptName) ");
qlBuf.append("FROM User user join user.roles role ");
qlBuf.append("WHERE role.roleName =:condition");

Query query = em.createQuery(qlBuf.toString());
query.setParameter("condition", "Admin");

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

If defining as above, inquiry result becomes the form of list of list. Follow the order defined to get out the result value from the List.

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

List user1 = (List) userList.get(0);
user1.get(1); //userId
user1.get(2); //userName
...
```

**Named Query**

The execution can be enabled by inserting name on the QL statement defined as annotation within the entity class file.

**Sample Source**

```
Query qlQuery = em.createNamedQuery("findDeptList");
qlQuery.setParameter("condition", "%%");

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

If transferring the query name to createNamedQuery() method as above, execute by finding QL statement matching this name. Following is the part of Department Entity class source containing findDeptList.

**Entity Source**

```
@Entity
@NamedQuery(name = "findDeptList",
            query = "FROM Department department WHERE department.deptName like:condition
ORDER BY department.deptName")
public class Department implements Serializable {
...
}
```

**Processing Paging**

Processing paging limits the inquiry list shown in one page to reduce DB or application memory overload. Let's examine the method to obtain the inquiry results processed paging when executing QL. Execute the inquiry tasks using QL on the specific table(USER table in the example). At this time, by defining the number(MaxResult) of inquiry list and inquiry list of Number(FirstResult) of Row to start inquiry, paging processing becomes possible.

**Sample Source**

```
StringBuffer qlBuf = new StringBuffer();

qlBuf.append("FROM User user ");
Query query = em.createQuery(qlBuf.toString());
//Number of item to inquire first
query.setFirstResult(1);
//Total number of inquiry items
query.setMaxResults(2);

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

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

**CUD using QL**

In CUD (create, update, delete) using JPA, it generally uses basic API (Refer to Basic CRUD in this manual). However, in specific cases, basic CUD should occur through QL in specific cases.

**INSERT**

Next is an example of using INSERT statement using QL.

```
StringBuffer ql = new StringBuffer();

ql.append("INSERT INTO Department (deptId,deptName) ");
ql.append("SELECT CONCAT(deptId,'UPD'), CONCAT(deptName,'UPD') ");
ql.append("FROM Department department ");
ql.append("WHERE deptId =:deptId");

Query query = em.createQuery(ql.toString());
query.setParameter("deptId", "Dept1");

query.executeUpdate();
```

If created as above, register new Department information using QL.

**UPDATE**

Next is an example of using UPDATE statement using QL.

```
StringBuffer ql = new StringBuffer();

ql.append("UPDATE Department department ");
ql.append("SET department.desc =:desc ");
ql.append("WHERE department.deptId =:deptId and department.deptName =:deptName ");

Query query = em.createQuery(ql.toString());
query.setParameter("desc", "Human Resource");
query.setParameter("deptId", "Dept1");
query.setParameter("deptName", "HRD");

query.executeUpdate();
```

Above example updates the Department information using QL and factor value is set through setParameter() method of Query.

**DELETE**

Next is an example of using DELETE statement using QL.

```
StringBuffer ql = new StringBuffer();
ql.append("DELETE Department department ");
ql.append("WHERE department.deptId =:deptId ");

Query query = em.createQuery(ql.toString());
query.setParameter("deptId", "Dept1");

query.executeUpdate();
```

Above example deletes the Department information using QL and sets the factor value through setParameter() method of Query.