EJBQL 3.0 Quick Reference Guide

Query Structure

Query clauses must be in the specified order.

select [**distinct**] [**new**] <result>
from <identification-variable>
[[**left** | **inner**] **join** [**fetch**] <objects to join>]
[**where** <filter>]
[**group by** <grouping-clause>]
[**having** <filter>]
[**order by** <ordering-clause>]

update < identification-variable >
set < identification-variable.field = new_value >
[**where** <filter>]

delete from < identification-variable >
[**where** <filter>]

Query Examples

The following examples use this sample class and assume an EntityManager named "em".

```
@Entity(access=AccessType.FIELD)
public class Person {
 @Id private int id;
 @Version private long version;
 private int age;
 private String firstName;
 private String lastName;
 @OneToOne private Address address;
 @OneToMany private Set<Person> children;
}
```

Basic Example

```
Query q = em.createQuery("select p from "
  + "Person p where p.firstName='John'");
List<Person> people = (List<Person>)
  q.getResultList();
for (Person person : people)
  log(person.getLastName());
```

Parameters

Parameters can be specified in query strings by placing a colon in front of the identifier. (i.e. :param)

Simple parameters:

Find all people named "John". Parameters allow using known data in the query, in this case a String.

```
Query q = em.createQuery("select p from "
  + "Person p where p.firstName=:param");
q.setParameter("param", "John");
List<Person> people = (List<Person>)
  q.getResultList();
```

Bulk Update and Delete

Performed for a single type of entity per statement.

Update:

Replace nickname Johnny with name John.

```
Query q = em.createQuery("update Person p "
  + "set p.firstName = 'John' where "
  + "p.firstName = 'Johnny'");
int numUpdates = q.executeUpdate();
```

Delete:

Delete all those with no children. Delete does not cascade

```
Query q = em.createQuery("delete from "
  + "Person p where p.children is empty");
int numDeleted = q.executeUpdate();
```

Subqueries

Find the youngest people in the company.

```
select p from Person p where p.age =
  (select min(p.age) from Person p)
```

Find those whose last name is part of a street address.

```
select p from Person p where p.lastName in
  (select a.street from Address a)
```

Ordering Results

Order query results by age, oldest first.

```
select p from Person p order by p.age desc
```

Order query results by name, from A to Z, and then by age, from oldest to youngest.

```
select p from Person p
  order by p.firstName asc, p.age desc
```

Optimizations

Limiting/paging query results:

Return a subset of the results, here, items 10 through 30.

```
Query q = em.createQuery(...);
q.setMaxResults(20);
q.setFirstResult(10);
```

Flush mode:

Set flushes to occur at commit or before query execution. If the flush mode is set to FlushModeType.COMMIT, changes made during the transaction might not be visible in the query execution results.

```
query.setFlushMode(FlushModeType.AUTO);
query.setFlushMode(FlushModeType.COMMIT);
```

Single Result

Specify that only one result is expected and to return only the single instance instead of a List.

```
Query q = em.createQuery("select distinct p"
  + " from Person p where p.firstName = "
  + "'UniqueName'");
Person john = (Person) q.getSingleResult();
```

Aggregates, Projections, and Grouping

Grouping allows aggregates and projections to be grouped by a given field and optionally limited using "having". Available aggregates are min, max, sum, avg, and count.

Simple grouping:

```
select avg(p.age) from Person p
  group by p.firstName
```

Limiting grouping with "having" expression:

Group by firstName where the firstName starts with "J".

```
select count(p) from Person p
  group by p.firstName
  having lower(p.firstName) like 'j%'
```

EJBQL from JDO

Execute an EJBQL query via the JDO Query interface.

```
Query q = pm.newQuery(
  kodo.query.QueryLanguages.LANG_EJBQL,
  "select p from Person p where "
    + "p.address.state = :stateCode");
List<Person> people = (List<Person>)
  q.execute("MA");
```


Content based upon EJB3.0 public draft http://solarmetric.com/resources/ejbql-quickref.pdf

EJBQL 3.0 Quick Reference Guide

SQL Queries

Queries can use direct SQL.
Find people whose first name is "John".

```
Query sql = em.createNativeQuery("SELECT"
  + " FIRST, LAST, AGE FROM PERSON"
  + " WHERE FIRST = 'JOHN'",
  com.example.Person.class);
List<Person> people = (List<Person>)
    sql.execute();
```

Named Queries

Query definition:

```
@NamedQuery(name="minorsByFirstName",
  queryString="select p from Person p
  where  p.age < 18 group by
  p.firstName")
```

Use the named query in code:

```
List<Person> minors = (List<Person>)
  em.createNamedQuery(
  "minorsByFirstName").getResultList();
```

Keywords

Reserved keywords are not case sensitive.

```
select, from, where, update, delete,
join, outer, inner, group, by, having,
fetch, distinct, object, null, true,
false, not, and, or, between, like, in,
as, unknown, empty, member, of, is, avg,
max, min, sum, count, order, by, asc,
desc, mod, upper, lower, trim, position,
character_length, char_length,
bit_length, current_time, current_date,
current_timestamp, new, left, exists,
all, any, some, size
```

Where Clause Methods

The following methods may be used as part of a query where clause. For example, "where p.firstName like ('%ohn%')" or "where sqrt(p.age) > 6".

General	`is [not] null,` `[not] exists`

Collection	`size(Collection)`
	`member [of],`
	`[not] in`
	`is [not] empty,`
	`all | any | some`

String	`[not] like,`
	`concat(String, String),`
	`trim([leading | trailing |`
	`both] <char to trim>`
	`[from] String)`
	`lower(String),`
	`upper(String),`
	`length(String),`
	`locate(String, String[, int`
	`startpoint]),`
	`substring(int, int)`

Math	`abs(numeric),`
	`sqrt(numeric),`
	`mod(numeric, numeric)`

Date	`[not] between`

Regular Expressions

The "like" expression accepts limited wildcard matching.

_ represents a single character
% represents multiple characters

All people with 'rick' in their name – Patrick, Rick, etc.

```
select p from Person p
  where lower(p.firstName) like '%rick%'
```

Where Clause Operators

=	equal (*note: can be used with Strings*)
<>	not equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
AND	conditional AND
OR	conditional OR
-	subtract or convert sign
+	add
*	multiply
/	divide
NOT	logical complement

Result Classes and Aliases

You can have query results placed directly into a custom class. This example uses the custom class Name below.

```
public class Name
{
  private String first;
  private String last;
  public Name(String f, String l) {
    this.first = f;
    this.last = l;
  }
  // ...
}

Query q = em.createQuery("select "
  + "new com.example.Name(p.firstName, "
  + "p.lastName) from Person p "
  + "where p.age > :param");
List<Name> names = (List<Name>)
  q.setParameter("param", 30).
  getResultList();
for(Name name : names)
  printLabel(name);
```

Joins

Inner join:

Select all people named John that have children.

```
select p from Person p join p.children c
  where p.firstName = 'John'
```

Use of the keyword 'inner' is optional

The prior query is equivalent to this EJB 2.1 query.

```
select object(p) from Person p,
  in(p.children) c where p.firstName='John'
```

Left outer join:

Select the last name of every person and the age of each of his/her children, if any.

```
select p.lastName, c.age from Person p
  left join p.children c
```

Fetch join:

Select everyone named John and eagerly fetch information about any children.

```
select distinct p from Person p
  left join fetch p.children
  where p.firstName='John'
```


Content based upon EJB3.0 public draft http://solarmetric.com/resources/ejbql-quickref.pdf