JDO3.1 Query Structure

Items in square brackets are optional

```
select [unique] [ <result> ] [into <result-class-name>] 
from <candidate-class-name> [exclude subclasses] [where <filter>] 
[variables <variable -list>] [parameters <parameter-list>] [imports <import-list>] 
[group by <grouping-clause>] [order by <ordering-clause>] [range <from-range>,<to-range>]
```

Query Examples

The following examples use this sample class.

```java
class Person
{
  private int age;
  private String firstName;
  private String lastName;
  private Address address;
  private Set<Person> children;
}
```

Example: Get objects with first name "John".
```
Query q = pm.newQuery("select from com.example.Person where firstName == :name");
List<Person> people = (List<Person>) q.execute("John");
```

Parameters

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

Simple parameters:

- Find all people named "John".
  - Query q = pm.newQuery("select from "+"com.example.Person where firstName == :name");
  - List<Person> people = (List<Person>) q.execute("John");

Using persistent instances as parameters:

- Find all people over 21 with a given address.
  - Address a = (Address) pm.getObjectById(Address.class, id);
  - Query q = pm.newQuery("select from "+"com.example.Person where address == :param1 && "+"&& age > :param2");
  - List<Person> people = (List<Person>) q.execute(a, new Integer (21));

Variables

Variables allow queries on multi-value relationships or on unrelated classes.

- Querying on a related instance:
  - Find all parents of a Person named "John".
    - Query q = pm.newQuery("select from "+"com.example.Person where children.contains(p) "+"&& p.firstName == :name");
    - List<Person> parents = (List<Person>) q.execute("John");

Ordering Results

Order query results by age, oldest first.
```
select from com.example.Person
order by age desc
```

Order by name, A first, then by age, oldest first.
```
select from com.example.Person
order by firstName asc, age desc
```

Keywords

Keywords must appear in either all upper-case or all lower-case characters.
```
as, asc, ascending, avg, by, count, desc, descending, distinct,
exclude, from, group, having, imports, into, max, min, order,
parameters, range, select, subclasses, sum, to, unique, variables,
where
```

Optimizations

These represent a few of the available methods to speed up JDO queries.

Limiting and paging query results:

The query can be configured to only return a subset of the results so that unused elements will not be instantiated. The start point is included, while the element at the limit is not.
```
select from com.example.Person range 10, 20
```

Ignore PersistenceManager cache:

Setting this parameter to true can speed up queries because changes made during the transaction do not need to be included in the results.
```
query.setIgnoreCache(true);
```

Indicate unique result:

Specifies that only one result is expected and to return only the single instance instead of a List.
```
Query q = pm.newQuery("select unique "+"from com.example.Person where firstName == :name");
Person john = (Person) q.execute("John");
```

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:
```
Query q = pm.newQuery("select avg(age) "+"from com.example.Person group by firstName");
```

Limiting grouping with "having" expression:

Group by firstName where the firstName starts with "J".
```
Query q = pm.newQuery("select count(this) "+"from com.example.Person group by firstName "+"having firstName.startsWith(:string)");
q.execute("J");
```
Where Clause Methods
The following methods may be used in a where clause.
For example, “where Math.abs (balance) > 500”.

<table>
<thead>
<tr>
<th>Collection</th>
<th>contains(Object), isEmpty(), size()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>containsKey(Object), isEmpty(), size(), containsValue(Object), get(Object)</td>
</tr>
<tr>
<td>List</td>
<td>get(int)</td>
</tr>
<tr>
<td>String</td>
<td>startsWith(String), endsWith(String), indexOf(String), indexOf(String, int), substring(int), substring(int, int), toLowerCase(), toUpperCase(), matches(String)</td>
</tr>
<tr>
<td>JDOHelper</td>
<td>getObjectId(Object)</td>
</tr>
<tr>
<td>Math</td>
<td>abs(Number), sqrt(Number)</td>
</tr>
</tbody>
</table>

The methods follow their counterparts in standard Java, so refer to the JDK javadocs for details.

Example: All people with ‘ick’ in their name – Patrick, Rick, etc.

select from com.example.Person
where firstName.matches('.*ick.*(?!)' )

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

```java
public class Name {
    public String first;
    public String last;
}
```

Query q = pm.newQuery ("SELECT firstName AS first,
                          public String last;
                       ");
List<Name> names = (List<Name>) q.execute(30);

SQL Queries
Queries can use SQL when accessing a relational database.

Find people whose first name is "John".

```sql
Query q = pm.newQuery(Query.SQL, "SELECT * "
                       + "FROM PERSON WHERE FIRSTNAME = ?")
q.setClass (Person.class);
List<Person> people = (List<Person>)
q.execute("John");
```

Named Queries
Named queries are defined in XML or annotations and consist of a name, query language, unmodifiable attribute, and the query itself.

```xml
<class name="Person">
  <query name="adultsByFirstName" unmodifiable="false">
    select where age > 18 group by firstName
  </query>
</class>
```

List<Person> adults = (List<Person>) pm.newNamedQuery (Person.class, "adultsByFirstName").execute ();

In-Memory Queries
JDOQL queries can be evaluated against an in-memory collection of persistent or transactional types.

```java
Query q = pm.newQuery("select from " + "com.example.Person where firstName == 'John'" );
q.setCandidates (allPeople);
filteredPeople = (List<Person>) q.execute();
```

JDOQL Subqueries
Find the oldest people in the company.

```java
Query q = pm.newQuery("select from " + "com.example.Person where age == " + "(select max(p.age) from Person p)" );
```

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

```java
Query q = pm.newQuery("select from " + "com.example.Person where (select " + "a.streetAddress from Address a).contains (lastName)" );
```