Поиск экземпляров сущностей

Для выполнения статических (т.н. именованных) и динамических запросов на языках Java Persistence Query Language и SQL используется интерфейс `Query`.

Создание объекта типа `Query` для статического запроса (описанного в XML-дескрипторе или с помощью аннотации `@NamedQuery` в классе сущности) выполняется с помощью метода `EntityManager.createNamedQuery(String queryName)`. Для создания динамического запроса предназначен метод `EntityManager.createQuery(String qlString)`.

Пример определения статического запроса:

```
@NamedQuery( name="findAllCustomersWithName",
             query="SELECT c FROM Customer c WHERE c.name LIKE :custName")
@Entity
public class Customer { ... }
```

Рассмотрим подробнее методы интерфейса `Query`., которые можно разделить на две группы.

1. Подготовка запроса.

Для установки именованных и позиционных параметров запроса предназначены методы `Query setParameter(String name, Object value)` и `Query setParameter(int position, Object value)`, соответственно.

Управление постраничным возвратом результатов осуществляется методами `Query setFirstResult(int startPosition)` и `Query setMaxResults(int maxResult)`, первый из которых устанавливает начало страницы, а второй – предельное количество результатов на странице.

Большое значение при выполнении запроса в рамках транзакции имеет режим сброса, установить который можно с помощью метода `Query setFlushMode(FlushModeType flushMode)`. При установленном по умолчанию автоматическом режиме сброса (`FlushModeType.AUTO`) СОРП должно обеспечить видимость в результатах запроса всех изменений состояния сущностей в рамках контекста персистентности, например, путем сброса изменений в базу данных перед выполнением запроса. В режиме сброса `FlushModeType.COMMIT` влияние изменений сущностей на результат запроса не определено.

2. Выполнение запроса.

Для выполнения запросов на изменение и удаление объектов предназначен метод `int executeUpdate()`, возвращающий количество измененных или удаленных сущностей.

Метод `List getResultList()` выполняет SELECT-запрос и возвращает список результатов поиска.

Метод `Object getSingleResult()` выполняет SELECT-запрос и возвращает единственный результат. Если запрос возвращает более одного результата, то выбрасывается исключение `NonUniqueResultException`. Если запрос не возвращает результатов, то выбрасывается исключение `NoResultException`.

Возврат объекта типа `Query` из методов подготовки запроса позволяет организовать выполнение запроса в виде следующей цепочки вызовов:

```
List result = em.createQuery("SELECT c FROM Customer c WHERE c.name LIKE
:custName")
.setParameter("custName", name)
.setMaxResults(10)
.getResultList();
```

Запросы могут записываться на языке SQL. Результат SQL-запроса может содержать сущности. В этом случае запрос должен выбирать все столбцы таблицы, которые отображаются на класс сущности, включая внешние ключи на связанные сущности. Кроме того, для SQL-запроса необходимо определить отображение результата на сущности.

Java Persistence Query Language

Данный язык предназначен для определения запросов к сущностям и их постоянному состоянию, то есть запросы на данном языке оперируют не таблицами и столбцами, а сущностями, их атрибутами и ассоциациями. Язык JP QL является расширением языка запросов к компонентам-сущностям EJB QL. Оба указанных языка напоминают язык запросов к реляционным базам данных SQL.

Запросы на языке JP QL не выполняются непосредственно по графу экземпляров сущностей в памяти, а преобразуются в SQL-запросы при установке модуля персистентности. Это позволяет оптимизировать выполнение запросов для конкретной СУБД при сохранении их переносимости (мобильности).

В составе языка запросов Java Persistence QL входят операторы SELECT, UPDATE и DELETE для выборки, изменения или удаления сущностей, соответственно.

Оператор SELECT

Оператор SELECT конструируется из следующих предложений: SELECT (список выбора), FROM, WHERE, GROUP BY, HAVING, ORDER BY:

```
SELECT FROM [WHERE] [GROUP BY [HAVING]] [ORDER BY]
```

Сущность в запросе задается своим именем, которое по умолчанию совпадает с неквалифицированным именем класса сущности. Имена сущностей должны быть уникальны в рамках модуля персистентности.

При обращении к полям сущностей используются их имена в том виде, в котором они были описаны разработчиком класса сущности. При этом используются точечные выражения (path expressions).

В запросах JP QL ключевые слова регистронезависимы, имена сущностей и их полей указывают с учетом регистра.

From

Предложение FROM имеет следующий синтаксис:

```
FROM объявление_переменной {, объявление_переменной}*
```

Объявление_переменной определяет сущность (т.н. домен предложения from), с которой будет работать запрос. Имена переменных используются затем в других предложениях оператора SELECT для обращения к полям сущностей. Также объявление переменной может включать соединения. Полный синтаксис объявления переменной выглядит так:

```
имя_сущности [AS] переменная
     { [ LEFT [OUTER] | INNER ] JOIN ассоциация [AS]  переменная]
     | [ LEFT [OUTER] | INNER ] JOIN FETCH ассоциация }*
```

Как следует из синтаксиса, в языке JP QL поддерживаются внутреннее и левое внешнее соединение.

Ассоциация представляет собой точечное выражение со значением типа коллекции.

Особым видом соединения является т.н. fetch join, при котором не вводится новая переменная, но в результате запроса соответствующая ассоциация будет обязательно загружена в память (независимо от настройки типа загрузки – `FetchType` – для ассоциации).

По умолчанию все запросы полиморфны, то есть в результаты запроса входят удовлетворяющие условиям запроса экземпляры сущностей не только тех классов, которые указаны в предложении FROM, но также и всех их подклассов.

Пример 1: `from Reader as r`. Запрос будет работать с сущностями класса `Reader`.

Пример 2: `from Reader r, Book b`. Запрос будет работать с сущностями классов `Reader` и `Book`.

Пример 3: `from Reader r join r.books as b`. Запрос будет работать с сущностями классов `Reader` и `Book`., причем будет проведено внутреннее соединение сущностей `Reader` и `Book`.

Пример 4: `from Reader r join fetch r.books`. Запрос будет работать с сущностями класса `Reader`, причем для каждого экземпляра сущности будет получена коллекция связанных с ним сущностей класса `Book` (то есть список книг, взятых читателем).

Where

Предложение WHERE имеет следующий синтаксис:

`WHERE` *условное_выражение*

В этом предложении указывается выражение логического типа, которое используется для отбора экземпляров сущности из домена, ограниченного предложением FROM. Отбираются только те экземпляры, для которых результат вычисления выражения дает `true`.

В условном выражении можно использовать литералы, переменные, определенные в предложении FROM, точечные выражения, входные параметры и вложенные SQL-запросы.

Точечные выражения дозволяют обращаться к полями сущностей и сравнивать их между собой. Точечные выражения бывают 2-х видов:

- Точечное выражение типа коллекции (collection valued path expression)
- Точечное выражение скалярного типа (single valued path expression)

Точечное выражение типа коллекции идентифицирует множество сущностей одного класса, либо соответствует постоянному полю сущности типа коллекции. Точечное выражение типа коллекции может быть использовано в следующих выражениях:

- проверка сущности на вхождение в коллекцию (`MEMBER OF`);
- проверка на пустоту коллекции (`IS EMPTY`).

Входные параметры делятся на позиционные и именованные, которые запрещается смешивать в одном запросе.

Позиционные параметры обозначаются знаком вопроса с номером параметра, например, `?1`. Номера параметров начинаются с 1. Параметр с одним и тем же номером можно использовать в запросе несколько раз. Кроме того, номер параметра не обязательно должен соответствовать порядку использования параметра в запросе.

Именованный параметр обозначается двоеточием с идентификатором, например, `:name`.

В условном выражении допускается использовать операции, приведенные в табл. 1.

Приоритет	Операции
	Арифметические операторы:
1	унарные +,-
2	*,/
3	+,-
	Операторы сравнения:

4	=, >, >=, <, <=, <> [NOT] BETWEEN – проверка на вхождение в диапазон, [NOT] LIKE – сравнение по шаблону, [NOT] IN – проверка на вхождение в список, IS [NOT] NULL – проверка на null-значение, IS [NOT] EMPTY – проверка на пустоту коллекции, [NOT] MEMBER [OF] – проверка сущности на вхождение в коллекцию
	Логические операторы:
5	NOT
6	AND
7	OR

Group By, Having

Предложение GROUP BY имеет следующий синтаксис:

```
GROUP BY объект_группировки {, объект_группировки}*
```

Данное предложение позволяет агрегировать результаты запроса по объектам группировки. В роли объекта группировки может выступать переменная или точечное выражение со скалярным значением.

Предложение HAVING позволяет включить в итоговую выборку только те группы, которые удовлетворяют заданным условиям:

```
HAVING условное_выражение
```

В этом предложении условное выражение может включать операции и агрегирующие функции над объектами группировки.

Пример. Следующий запрос возвращает список стран, в которых проживают заказчики, а также количество заказчиков из каждой страны:

```
SELECT c.country, COUNT(c)
FROM Customer c
GROUP BY c.country
HAVING COUNT(c.country) > 3
```

Select

Предложение SELECT представляет список выбора SELECT-запроса и имеет следующий синтаксис:

```
SELECT [DISTINCT] выражение_выборки {, выражение_выборки}*
```

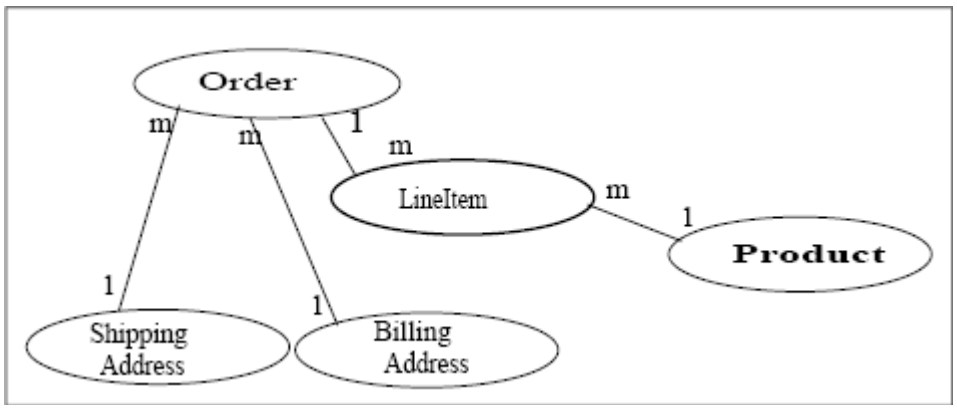
В качестве *выражения_выборки* можно использовать:

- точечное выражение скалярного типа;
- агрегирующее выражение;
- переменную;
- конструктор объекта.

Если в списке выбора содержится единственное выражение, то список результатов содержит объекты (`Object`), в противном случае – массивы объектов (`Object[]`).

Примеры

В примерах используются следующие взаимосвязанные сущности:



1. Найти все заказы, которые необходимо доставить во Владимир:

```
SELECT o FROM Order o WHERE o.shippingAddress.city = 'Vladimir'
```

2. Найти все заказы, у которых нет строк:

```
SELECT o FROM Order o WHERE o.lineItems IS EMPTY
```

3. Найти все заказы на определенную книгу, название которой передается в качестве именованного параметра:

```
SELECT DISTINCT o FROM Order o JOIN o.lineItems l
WHERE l.product.type = 'book' AND l.product.name = :bookname
```

Операторы UPDATE и DELETE

Данные операторы предназначены для одновременного изменения или удаления множества объектов. Операторы UPDATE и DELETE применяются к единственному классу сущностей, однако свойство полиморфизма для них сохраняется.

Результат выполнения операторов UPDATE и DELETE не отражается на состоянии контекста персистентности, что может привести к ошибкам (потере изменений или обращению к удаленной записи). Поэтому данные операторы рекомендуется выполнять в отдельной транзакции или в самом начале транзакции, до получения экземпляров сущностей из базы данных.

Синтаксис рассматриваемых операторов:

```
UPDATE имя_сущности [[AS] переменная]
       SET поле = новое_значение {, поле = новое_значение}*
       [WHERE условное_выражение]
DELETE FROM имя_сущности [[AS] переменная] [WHERE условное_выражение]
```

Пример обновления статуса заказчиков, имеющих баланс более 10000 и сделавших более 1000 заказов (обратите также внимание на коррелированный вложенный SELECT-запрос, возвращающий количество заказов, сделанных заказчиком):

```
UPDATE Customer c
       SET c.status = 'outstanding'
       WHERE c.balance > 10000
            AND 1000 < (SELECT COUNT(o) FROM customer cust JOIN cust.order o)
```