Запросы в JPA

Язык запросов Java Persistence Query Language

- Java Persistence Query Language (JP QL) предназначен для определения запросов к сущностям и их постоянному состоянию
- В запросах JP QL используются сущности, их атрибуты и ассоциации
- JP QL — расширение языка запросов к компонентам-сущностям EJB QL
- Операторы SELECT, UPDATE и DELETE для выборки, изменения и удаления сущностей
- Все запросы полиморфны
- Запросы JP QL перед выполнением преобразуются в SQL-запросы

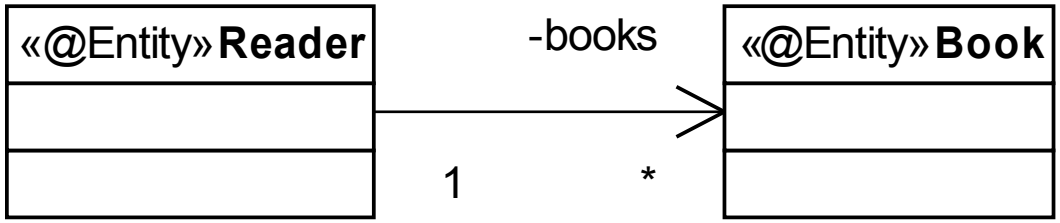
Оператор SELECT

- **`SELECT FROM [WHERE]`**
  `[GROUP BY [HAVING]] [ORDER BY]`
- Сущность в запросе задается своим именем
  - По умолчанию имя сущности совпадает с неквалифицированным именем класса сущности
- Имена полей сущностей используются в том виде, в каком они описаны в классе сущности
  - При этом используются точечные выражения (path expressions)
- Ключевые слова регистронезависимы, имена сущностей и полей указывают с учетом регистра

Оператор SELECT: предложение FROM

- **FROM** *объявление_переменной* {, *объявление_переменной*}*
- *Объявление_переменной* определяет сущность, с которой будет работать запрос
  - *имя_сущности* [**AS**] *переменная*
    { [**LEFT**[**OUTER**]|**INNER**] **JOIN** *ассоциация* [**AS**] *переменная*]
    | [**LEFT**[**OUTER**]|**INNER**] **JOIN FETCH** *ассоциация* }*
- *Ассоциация* - точечное выражение со значением типа коллекции
- Fetch join - в результате запроса соответствующая ассоциация будет обязательно загружена в память
  - По умолчанию ассоциации @OneToMany, @ManyToMany загружаются только при первом обращении

Оператор SELECT: предложение FROM



- Примеры:
  - `FROM Reader AS r`
  - `FROM Reader r, Book b`
  - `FROM Reader r JOIN r.books AS b`
    - По умолчанию выполняется внутреннее соединение (INNER)
  - `FROM Reader r JOIN FETCH r.books`

Оператор SELECT: предложение WHERE

- **WHERE** *условное_выражение*
- В *условное_выражение* можно использовать:
  - литералы
  - переменные, определенные в предложении `FROM`
  - точечные выражения
    - типа коллекции (в операциях `MEMBER OF` и `IS EMPTY`)
    - скалярного типа
  - входные параметры
  - вложенные SELECT-запросы

Оператор SELECT: предложение WHERE

- Входные параметры:
  - Позиционные параметры - обозначаются знаком вопроса с номером параметра
    - `?1`
    - Номера параметров начинаются с 1
    - Параметр с одним номером можно использовать несколько раз
    - Номер параметра не обязан соответствовать порядку использования параметра
  - Именованные параметры - обозначаются двоеточием с идентификатором
    - `:name`
  - Нельзя смешивать позиционные и именованные параметры в одном запросе

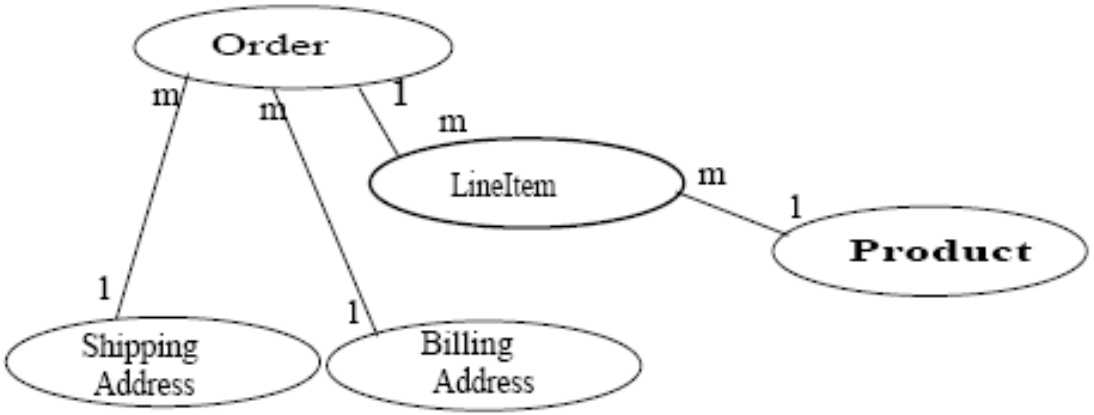
Оператор SELECT: предложение WHERE

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

Оператор SELECT: предложение SELECT

- **SELECT** [**DISTINCT**] *выражение_выборки*
  {, *выражение_выборки*}*
- В качестве *выражения_выборки* можно использовать:
  - точечное выражение скалярного типа
  - агрегирующее выражение
    - `COUNT, MAX, MIN, AVG, SUM`
  - переменную, определенную в предложении `FROM`
  - конструктор объекта
    - `NEW com.acme.example.CustomerDetails(c.id, o.count)`
- Если в списке выбора содержится единственное выражение, то список результатов содержит объекты (`Object`), в противном случае – массивы объектов (`Object[]`)

Оператор SELECT: примеры



- Найти все заказы, которые необходимо доставить во Владимир:
- `SELECT o FROM Order o WHERE o.shippingAddress.city = 'Vladimir'`
- Найти все заказы, у которых нет строк:
- `SELECT o FROM Order o WHERE o.lineItems IS EMPTY`
- 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

- Предназначены для одновременного изменения или удаления множества объектов
- Результат выполнения не отражается на состоянии контекста персистентности
  - > Потеря изменений или обращение к удаленной записи
  - > Данные операторы рекомендуется выполнять в отдельной транзакции или в самом начале транзакции, до получения экземпляров сущностей из БД
- **UPDATE** *имя_сущности* [[**AS**] *переменная*]
  SET *поле* = *новое_значение* {, *поле* = *новое_значение*}*
  [**WHERE** *условное_выражение*]
- **DELETE FROM** *имя_сущности* [[**AS**] *переменная*]
  [**WHERE** *условное_выражение*]

Поиск экземпляров сущностей

- По первичному ключу - **`EntityManager.find()`**
- По запросу на языке JP QL - интерфейс **`Query`**
  - Динамические запросы - создаются в программе
    - `EntityManager.createQuery(String qlString)`
  - Статические (именованные) запросы - описываются в XML-дескрипторе или в классе сущности
    - ```
      @NamedQuery(name="findAllCustomersWithName",
          query="SELECT c FROM Customer c WHERE
                 c.name LIKE :custName")
      @Entity public class Customer { ... }
      ```
    - `EntityManager.createNamedQuery(String queryName)`
      ```

Подготовка запроса

- Установка именованных и позиционных параметров запроса: `Query.setParameter()`
- Управление постраничным возвратом результатов: `Query.setFirstResult()` и `Query.setMaxResults()`
- Установка режима сброса изменений в контексте: `Query.setFlushMode()`
  - По умолчанию - автоматический режим сброса, при этом в результатах запроса «видны» все изменения состояния сущностей в рамках контекста персистентности

Выполнение запроса

- **`Query.executeUpdate()`** - выполнение UPDATE- и DELETE-запросов
- **`Query.getResultList()`** - выполнение SELECT-запроса, возвращающего список результатов
- **`Query.getSingleResult()`** - выполнение SELECT-запроса, возвращающего единственный результат
  - `NonUniqueResultException` - если >1 результата
  - `NoResultException` - если 0 результатов
- Пример подготовки и выполнения запроса:

```
List result =
  em.createNamedQuery("findAllCustomersWithName")
    .setParameter("custName", name)
    .setMaxResults(10)
    .getResultList();
```

Native Query

- Запрос на языке SQL - интерфейс **`Query`**
  - Статический (именованный)
  - Динамический - `EntityManager.createNativeQuery(String sqlQuery, ...)`
  - Результат может содержать сущности
    - Запрос должен выбирать все столбцы таблицы, которые отображаются на класс сущности, включая внешние ключи на связанные сущности
  - Необходимо определить отображение результата на сущности
  - Подготовка и выполнение запроса - так же, как для запроса на языке JP QL
    - Именованные запросы позволяют плавно перейти от SQL к JP QL

Native Query - пример 1

```
@NamedNativeQuery(name="getAllRoute",
    query="SELECT r.idRoute as route_id, r.routeName
as route_name, r.departureDateTime as
route_departure, r.arrivalDateTime as route_arrival,
r.seat as route_seat, r.tarif as route_tarif,
r.allSeat as route_all_seat
FROM route r, link l, destination d
WHERE DATE(r.departureDateTime)=?1 and
d.destinationName=?2 and r.idRoute=l.idRoute and
d.idDestination=l.idDestination",
    resultSetMapping="Route")
```

```
@SqlResultSetMapping(name="Route", entities={
  @EntityResult(entityClass=Route.class, fields={
    @FieldResult(name="idRoute", column="route_id"),
    @FieldResult(name="routeName", column="route_name"),
    @FieldResult(name="departureDateTime",
                 column="route_departure"),
    @FieldResult(name="arrivalDateTime",
                 column="route_arrival"),
    @FieldResult(name="seat", column="route_seat"),
    @FieldResult(name="tarif", column="route_tarif"),
    @FieldResult(name="allSeat", column="route_all_seat")
})})
```

Native Query - пример 2

- ```
  @NamedNativeQuery(name="getAllDepartureDate",
      query="SELECT DISTINCT DATE(departureDateTime)
              as result FROM route r",
      resultSetMapping="SingleResult")
  ```
- ```
  @SqlResultSetMapping(name="SingleResult",
      columns={@ColumnResult(name="result")})
  ```