JPQL коротко все механизмы

Запросы Select

Запросы Select могут использоваться, чтобы считать объекты из базы данных. Запросы Select могут возвратить единственный объектный или элемент данных, список объектов или элементов данных или объектного массива многократных объектов и данных.

***Запросы Select***

*// Запрос для Списка объектов.*

Query query = em.createQuery("Select e FROM Employee e WHERE e.salary > 100000");

List<Employee> result = query.getResultList();

*// Запрос для единственного объекта.*

Query query = em.createQuery("Select e FROM Employee e WHERE e.id = :id");

query.setParameter("id", id);

Employee result2 = (Employee)query.getSingleResult();

*// Запрос для единственного элемента данных.*

Query query = em.createQuery("Select MAX(e.salary) FROM Employee e");

BigDecimal result3 = (BigDecimal)query.getSingleResult();

*// Запрос для Списка элементов данных.*

Query query = em.createQuery("Select e.firstName FROM Employee e");

List<String> result4 = query.getResultList();

*// Запрос для Списка массивов элемента.*

Query query = em.createQuery("Select e.firstName, e.lastName FROM Employee e");

List<Object[]> result5 = query.getResultList();

**SELECT**

SELECT пункт может содержать объектные выражения, выражения атрибута, функции, выборки, конструкторы и функции агрегации.

**Функции агрегации**

Функции агрегации могут включать итоговую информацию о ряде объектов. Они включают MIN, MAX, AVG, SUM, GRAPH. Эти функции могут использоваться, чтобы возвратить единственный результат или могут использоваться с GROUP BY, чтобы возвратить многократные результаты.

**SELECT** **COUNT**(e) **FROM** Employee e

**SELECT** **MAX**(e.salary) **FROM** Employee e

**Конструкторы**

Оператор NEW может использоваться с полностью определенным именем класса, чтобы возвратить объекты данных из запроса JPQL. Они не будут управляемыми объектами, и класс должен определить конструктор, который соответствует параметрам конструктора и их типов. Запросы конструктора могут использоваться, чтобы выбрать частичные данные или создать отчеты о данных по объектам, и возвратить экземпляр класса вместо объектного массива.

**SELECT** **NEW** com.acme.reports.EmpReport(e.firstName, e.lastName, e.salary) **FROM** Employee e

**FROM**

FROM определяет то, что запрашивается. Типичное FROM будет содержать запрашиваемое имя объекта и присваивать его псевдоним.

**SELECT** e **FROM** Employee e

JPQL допускает многократные корневые объекты уровня, которые будут запрошены. Предостережение: должно использоваться при выполнении этого, поскольку оно может привести к Декартовым произведениям двух таблиц. Оператор Where должен гарантировать, что два объекта присоединяются в некотором роде.

**SELECT** e, a **FROM** Employee e, MailingAddress a **WHERE** e.address = a.address

Название объекта, используемое в JPQL, происходит от атрибута имени @Entity аннотации или XML. Оно принимает значение по умолчанию к простому имени класса объекта. Некоторые провайдеры JPA также допускают полное имя класса.

**JOIN**

Пункт JOIN может также использоваться в FROM. Пункт JOIN позволяет любому из отношений объекта быть присоединенным в запрос, таким образом, они могут использоваться в операторе Where. JOIN не означает, что отношения будут выбраны, если опция FETCH не будет включена.

**SELECT** e **FROM** Employee e **JOIN** e.address a **WHERE** a.city = :city

JOIN может использоваться с OneToOne, ManyToOne, OneToMany, ManyToMany и отображениями ElementCollection. Когда используется с отношением набора Вы можете обратиться к тому же отношению многократно, чтобы запросить многократные независимые значения.

**SELECT** e **FROM** Employee e **JOIN** e.projects p **JOIN** e.projects p2 **WHERE** p.name = :p1 **and** p2.name = :p2

**JOIN FETCH**

Опция FETCH может использоваться в JOIN, чтобы выбрать связанные объекты в единственном запросе. Это избегает дополнительных запросов для каждого из отношений объекта и гарантирует, что отношения были выбраны, если они были LAZY.

**SELECT** e **FROM** Employee e **JOIN** **FETCH** e.address

не позволяет псевдоним в спецификации JPA, но некоторые провайдеры JPA могут позволить его.

**LEFT JOIN**

По умолчанию все внутренние объединения участвует в JPQL. Это означает, что выборки, у которых нет отношении, будут фильтроваться от результатов запроса. Чтобы избежать этого, соединение может быть определено как OUTER объединение, используя LEFT опции.

**SELECT** e **FROM** Employee e **LEFT** **JOIN** e.address a **ORDER** **BY** a.city

**ON (JPA 2.1)**

Условие объединения, используемое для соединения, прибывает из объединяющих столбцов отображения. Это означает, что пользователь JPQL обычно избавлен от необходимости знать, как к каждому отношению присоединяются. В некоторых случаях желательно добавить дополнительные условия к условию объединения, обычно в случае внешних объединений. Это может быть сделано через ON пункте. ON пункте определен в JPA 2.1 specifiation и может поддерживаться некоторыми провайдерами JPA.

**SELECT** e **FROM** Employee e **LEFT** **JOIN** e.address a **ON** a.city = :city

**ORDER BY**

ORDER BY позволяет выборке результатов быть отсортированной. Многократные значения могут быть упорядочены, или возрастающий (ASC) или убывающий (DESC). JPA 1.0 и 2.0 BNFs ограничивают использование функций, и подвыбирает в ORDER BY, но проект JPA 2.1 удаляет большинство из них.

**SELECT** e **FROM** Employee e **ORDER** **BY** e.lastName **ASC**, e.firstName, **ASC**

**SELECT** e **FROM** Employee e **ORDER** **BY** **UPPER**(e.lastName)

**GROUP BY**

GROUP BY допускает итоговую информацию, которая будет вычислена на ряде объектов. GROUP BY обычно используется в сочетании с функциями агрегации.

**SELECT** **AVG**(e.salary), e.address.city **FROM** Employee e **GROUP** **BY** e.address.city

**SELECT** **AVG**(e.salary), e.address.city **FROM** Employee e **GROUP** **BY** e.address.city **ORDER** **BY** **AVG**(e.salary)

**SELECT** e, **COUNT**(p) **FROM** Employee e **LEFT** **JOIN** e.projects p **GROUP** **BY** e

**HAVING**

Пункт HAVING допускает результаты GROUP BY, который будет фильтроваться.

**SELECT** **AVG**(e.salary), e.address.city **FROM** Employee e **GROUP** **BY** e.address.city **HAVING** **AVG**(e.salary) > 100000

**UNION**

JPA не поддерживает SQL UNION, INTERSECT и EXCEPT. Некоторые провайдеры JPA могут поддерживать их.

WHERE

Оператор Where обычно - основная часть запроса, поскольку оно определяет условия, которые фильтруют выборку. Оператор Where может использовать любой оператор сравнения, логическую операцию, функции, атрибуты, объекты, и подвыборки. Операции сравнения включают =, <, >, <=, >=, <>, LIKE, BETWEEN, IS NULL,IN. NOT Может также использоваться с любым оператором сравнения (NOT LIKE, NOT BETWEEN, IS NOT NULL, NOT IN). Логические операции включают AND, OR, и NOT.

**Операции сравнения**

|  |  |  |
| --- | --- | --- |
| **Операция** | **Описание** | **Пример** |
| = | равно | e.firstName = 'Bob' |
| < | меньше чем | e.salary < 100000 |
| > | больше чем | e.salary > :sal |
| <= | меньше или равно | e.salary <= 100000 |
| >= | больше или равно | e.salary >= :sal |
| LIKE | оценивает, если два соответствия последовательности, '%' и '\_' являются допустимыми подстановочными знаками, и Символ ESC дополнительный | e.firstName **LIKE** 'A%' **OR** e.firstName **NOT** **LIKE** '%.\_%' **ESCAPE** '.' |
| BETWEEN | оценивает, если значение между двумя значениями | e.firstName **BETWEEN** 'A' **AND** 'C' |
| IS NULL | сравнивает значение с нулем, базы данных могут не позволить или иметь неожиданные результаты при использовании = с нулем | e.endDate **IS** **NULL** |
| IN | оценивает, если значение содержится в списке | e.firstName **IN** ('Bob', 'Fred', 'Joe') |

Операция IN допускает список значений или параметров, единственного параметра списка или подвыбора.

e.firstName **IN** (:name1, :name2, :name3)

e.firstName **IN** (:name1)

e.firstName **IN** :**names**

e.firstName **IN** (**SELECT** e2.firstName **from** Employee e2 **WHERE** e2.lastName = 'Smith')

Подвыбор может использоваться с любой работой, если это возвращает единственное значение, или если ALL или ANY опции используются. ALL указывает, что атрибут должен быть истиной для всех элементов, возвращенных подвыбором, ANY указывает, что атрибутдолжен быть истиной для любого из элементов, возвращенных подвыбором.

e.firstName = (**SELECT** e2.firstName **from** Employee e2 **WHERE** e2.id = :id)

e.salary < (**SELECT** e2.salary **from** Employee e2 **WHERE** e2.id = :id)

e.firstName = **ANY** (**SELECT** e2.firstName **from** Employee e2 **WHERE** e.id <> e.id)

e.salary <= **ALL** (**SELECT** e2.salary **from** Employee e2)

Запросы на обновление

Вы можете выполнить массовое обновление объектов с оператором UPDATE. Этот оператор воздействует на единственный тип объекта и устанавливает одно или более однозначных свойств объекта, подвергающегося условию в WHERE. Запросы на обновление обеспечивают эквивалент оператору SQL UPDATE, но с условными выражениями JPQL.

Запросы на обновление не позволяют соединения, но действительно поддерживают выборку. OneToOne и отношения ManyToOne могут быть пересечены в операторе Where. Отношения набора могут все еще быть запрошены посредством использования EXISTS в операторе Where с подвыбором. Запросы на обновление могут только обновить атрибуты объекта или его embeddables, его отношения не могут быть обновлены. Сложные запросы на обновление зависят от поддержки обновления базы данных и могут использовать временные таблицы на некоторых базах данных.

Запросы на обновление должны использоваться только для массовых обновлений, регулярные обновления объектов должны быть сделаны при помощи методов набора объекта в транзакции и фиксации изменений.

Запросы на обновление возвращают число измененных строк на базе данных (количество строки).

Этот пример демонстрирует, как использовать запрос на обновление, чтобы дать сотрудникам повышение. WHERE пункт содержит условное выражение.

**Пример запроса на обновление**

Query query = em.createQuery("UPDATE Employee e SET e.salary = 60000 WHERE e.salary = 50000");

int rowCount = query.executeUpdate();

Контекст персистентности не обновлен, чтобы отразить результаты операций обновления. Если Вы используете ограниченный по объему транзакцией контекст персистентности, Вы должны или выполнить объемную работу в транзакции совершенно отдельно или являетесь первой работой в транзакции. Это вызвано тем, что любой объект, которым активно управляет контекст персистентности, останется не знать, что фактические изменения происходят на уровне базы данных.

Объекты в совместно используемом кэше, которые соответствуют запрос на обновление, будут лишены законной силы, чтобы гарантировать, чтобы последующие контексты персистентности видели обновленные данные.

Запросы на удаление

Вы можете выполнить объемное удаление объектов с оператором DELETE . Запросы на удаление обеспечивают эквивалент оператору SQL DELETE , но с условными выражениями JPQL.

Запросы на удаление не позволяют соединения, но действительно поддерживают, подвыбирает. OneToOne и отношения ManyToOne могут быть пересечены в операторе Where. Отношения набора могут все еще быть запрошены посредством использования EXISTS в операторе Where с подвыбором. Сложные запросы на удаление зависят от базы данных, удаляют поддержку и может использовать временные таблицы на некоторых базах данных.

Запросы на удаление должны только использоваться для объема, удаляет, регулярный удаляет к объектам, должен быть выполнен посредством вызова EntityManager , удаляют () API.

Запросы на удаление возвращают число удаленных строк на базе данных (количество строки).

Этот пример демонстрирует, как использовать запрос на удаление, чтобы удалить всех сотрудников, которые не присвоены отделу. WHERE пункт содержит условное выражение.

**Пример запроса на удаление**

Query query = em.createQuery("DELETE FROM Employee e WHERE e.department IS NULL").executeUpdate();

Запросы на удаление полиморфные: будут удалены любые экземпляры подкласса объекта, которые соответствуют критериям запроса на удаление. Однако запросы на удаление не соблюдают каскадные правила: никакие объекты кроме типа, на который ссылаются в запросе и его подклассах, не будут удалены, даже если у объекта будут отношения к другим объектам с каскадом, удаляет, включал. Запросы на удаление удалят строки из таблиц набора и соединения.

Контекст персистентности не может быть обновлен, чтобы отразить, что результаты удаляют операции. Если Вы используете ограниченный по объему транзакцией контекст персистентности, Вы должны или выполнить объемную работу в транзакции совершенно отдельно или являетесь первой работой в транзакции. Это вызвано тем, что любой объект, которым активно управляет контекст персистентности, останется не знать, что фактические изменения происходят на уровне базы данных.

Параметры

JPA определяет названные параметры и позиционные параметры. Названные параметры могут быть определены в JPQL использование синтаксиса :<name>. Позиционные параметры могут быть определены в JPQL использование синтаксиса ? или ?<position>. Позиционные параметры запускаются в позиции 1 не 0 .

***Пример запроса названного параметра***

Query query = em.createQuery("SELECT e FROM Employee e WHERE e.firstName = :first and e.lastName = :last");

query.setParameter("first", "Bob");

query.setParameter("last", "Smith");

List<Employee> list = query.getResultList();

***пример запроса позиционного параметра***

Query query = em.createQuery("SELECT e FROM Employee e WHERE e.firstName = ? and e.lastName = ?");

query.setParameter(1, "Bob");

query.setParameter(2, "Smith");

List<Employee> list = query.getResultList();

Литералы

Литеральные значения могут быть встроены в JPQL для стандартных типов Java. В целом обычно лучше использовать параметры вместо того, чтобы встроить значения. Встроенные параметры будут препятствовать тому, чтобы JPQL извлек выгоду из кэша синтаксического анализатора EclipseLink JPQL и могут потенциально подать заявку, уязвимую для атак инъекций JPQL.

Каждый Java типы определяет свой собственный синтаксис встраивания:

* String - '<string>'
* **SELECT** e **FROM** Employee e **WHERE** e.name = 'Bob'
  + Чтобы определить **'**(кавычка) символ в последовательности, кавычка дважды заключена в кавычки, т.е. 'Baie-D''Urfé'.
* Integer - +|-<digits>
* **SELECT** e **FROM** Employee e **WHERE** e.id = 1234
* Long - +|-<digits>L
* **SELECT** e **FROM** Employee e **WHERE** e.id = 1234L
* Float - +|-<digits>.<decimale><exponent>F
* **SELECT** s **FROM** Stat s **WHERE** s.ratio > 3.14F
* Double - +|-<digits>.<decimale><exponent>D
* **SELECT** s **FROM** Stat s **WHERE** s.ratio > 3.14e32D
* Boolean - TRUE | FALSE
* **SELECT** e **FROM** Employee e **WHERE** e.active = **TRUE**
* Date - {d'yyyy-mm-dd'}
* **SELECT** e **FROM** Employee e **WHERE** e.startDate = {d'2012-01-03'}
* Time - {t'hh:mm:ss'}
* **SELECT** e **FROM** Employee e **WHERE** e.startTime = {t'09:00:00'}
* Timestamp - {ts'yyyy-mm-dd hh:mm:ss.nnnnnnnnn'} -
* **SELECT** e **FROM** Employee e **WHERE** e.**version** = {ts'2012-01-03 09:00:00.000000001'}
* Enum - package.class.enum
* **SELECT** e **FROM** Employee e **WHERE** e.gender = org.acme.Gender.MALE
* null - NULL
* **UPDATE** Employee e **SET** e.manager = **NULL** **WHERE** e.manager = :manager

Функции

JPQL поддерживает несколько функций базы данных. Эти функции баз данных, независимы от имя и синтаксиса, но требуют поддержки базы данных. Если база данных поддерживает эквивалентный функциональный или различный синтаксис, стандартная функция JPQL поддерживается, если база данных не обеспечивает способа выполнить функцию, то это не поддерживается. Для математических функций (+,-/, \*) применяются правила BEDMAS. Некоторые провайдеры JPA может поддерживать дополнительные функции.

**Поддерживаемые JPQL функции**

|  |  |  |
| --- | --- | --- |
| **Функция** | **Описание** | **Пример** |
| - | вычитание | e.salary - 1000 |
| + | сложение | e.salary + 1000 |
| \* | умножение | e.salary \* 2 |
| / | деление | e.salary / 2 |
| ABS | модуль числа | **ABS**(e.salary - e.manager.salary) |
| CASE | определяет оператор выбора | **CASE** e.status **WHEN** 0 **THEN** 'active' **WHEN** 1 **THEN** 'consultant' **ELSE** 'unknown' **END** |
| COALESCE | оценивает к первому не нулевое значение аргумента | COALESCE(e.salary, 0) |
| CONCAT | конкатинирует строки | CONCAT(e.firstName, ' ', e.lastName) |
| CURRENT\_DATE | текущая дата | **CURRENT\_DATE** |
| CURRENT\_TIME | текущее время | **CURRENT\_TIME** |
| CURRENT\_TIMESTAMP | текущее дата и время | **CURRENT\_TIMESTAMP** |
| LENGTH | длинна строки или двоичного значения | **LENGTH**(e.lastName) |
| LOCATE | индекс строки в последовательности, дополнительно начинающиейся по индексу запуска | LOCATE('-', e.lastName) |
| LOWER | преобразование значение последовательности в нижний регистр | **LOWER**(e.lastName) |
| MOD | вычисляет остаток от деления первого целого числа вторым | **MOD**(e.hoursWorked / 8) |
| NULLIF | возвращает null, если первый параметр будет равняться второму параметру, иначе возвращает первый параметр | **NULLIF**(e.salary, 0) |
| SQRT | вычисляет квадратный корень числа | SQRT(o.**result**) |
| SUBSTRING | подстрока от строки, начинающиеся по индексу, дополнительно с размером подстроки | **SUBSTRING**(e.lastName, 0, 2) |
| TRIM | обрезание пробелов в конце и начале строки | **TRIM**(**TRAILING** **FROM** e.lastName), **TRIM**(e.lastName), **TRIM**(**LEADING** '-' **FROM** e.lastName) |
| UPPER | преобразование значение последовательности в верхний регистр | **UPPER**(e.lastName) |

Специальные операторы

JPQL определяет несколько специальных операторов, которые не являются функциями базы данных, но имеют особое значение в JPQL. Они включают INDEX, KEY, SIZE, IS EMPTY, TYPE, FUNCTION и TREAT.

**JPQL специальные операторы**

|  |  |  |
| --- | --- | --- |
| **Функция** | **Описание** | **Пример** |
| INDEX | индекс упорядоченного элемента Списка, только поддерживаемого с @OrderColumn | **SELECT** toDo **FROM** Employee e **JOIN** e.toDoList toDo **WHERE** **INDEX**(toDo) = 1 |
| KEY | ключ элемента Карты | **SELECT** p **FROM** Employee e **JOIN** e.priorities p **WHERE** **KEY**(p) = 'high' |
| SIZE | размер отношений набора, это оценивает к подвыбору | **SELECT** e **FROM** Employee e **WHERE** **SIZE**(e.managedEmployees) < 2 |
| IS EMPTY, IS NOT EMPTY | оценивает к истине, если отношение набора пусто, это оценивает к подвыбору | **SELECT** e **FROM** Employee e **WHERE** e.managedEmployees **IS** EMPTY |
| MEMBER OF, NOT MEMBER OF | оценивает к истине, если отношение набора содержит значение, это оценивает к подвыбору | **SELECT** e **FROM** Employee e **WHERE** 'write code' MEMBER **OF** e.responsibilities |
| TYPE | значение различителя наследования | **SELECT** p **FROM** Project p **WHERE** **TYPE**(p) = LargeProject |
| TREAT | обработка (cast) объект в качестве своего значения подкласса (JPA 2.1) | **SELECT** e **FROM** Employee **JOIN** **TREAT**(e.projects **as** LargeProject) p **WHERE** p.budget > 1000000 |
| FUNCTION | вызовите функцию базы данных (JPA 2.1) | **SELECT** p **FROM** Phone p **WHERE** **FUNCTION**('TO\_NUMBER', p.areaCode) > 613 |