Objektorientierte Datenbanken

Vorlesung 6
Sebastian Iwanowski
FH Wedel

JDO: 4. Teil: JDOQL (2. Teil)

Deklarationen in JDOQL

„Normaler“ Weg (JDO 1.0.1):

Interface Query

public void declareParameters (String declarationOfParams);

public void declareVariables (String declarationOfVariables);

Manchmal sind Importe notwendig:

public void declareImports (String declarationOfImports);

- **declarationOfImports: normale Java-Syntax**
- **Alle Klassen aus dem Package java.lang sind per default bekannt.**
- **Alle Klassen aus dem Package der zur Query gehörenden Kandidatenklasse sind per default bekannt.**
- **Alle anderen Klassen müssen hier importiert werden (normale Java-Syntax).**

Weitere Wege (nur für JDO 2.0):

- **optional in single strings der Select-Queries**
- **durch implizite Vereinbarungen (siehe Spec. 2.0, Kap. 14.6.3, S. 153)**

Unterschiede der Filtersprache zu Java

Aus der Java-Dokumentation der Methode `Query.setFilter (String filter)`:

Rules for constructing valid expressions follow the Java language, except for these differences:

- Equality and ordering comparisons between primitives and instances of wrapper classes are valid.
- Equality and ordering comparisons of **`Date`** fields and **`Date`** parameters are valid.
- White space (non-printing characters space, tab, carriage return, and line feed) is a separator and is otherwise ignored.
- The assignment operators =, +=, etc. and pre- and post-increment and -decrement are not supported. Therefore, there are no side effects from evaluation of any expressions.
- Methods, including object construction, are not supported, except for **`Collection.contains(Object o)`**, **`Collection.isEmpty()`**, **`String.startsWith(String s)`**, and **`String.endsWith(String e)`**. Implementations might choose to support non-mutating method calls as non-standard extensions.
- Navigation through a **`null`**-valued field, which would throw **`NullPointerException`**, is treated as if the filter expression returned **`false`** for the evaluation of the current set of variable values. Other values for variables might still qualify the candidate instance for inclusion in the result set.
- Navigation through multi-valued fields (**`Collection`** types) is specified using a variable declaration and the **`Collection.contains(Object o)`** method.

Unterschiede der Filtersprache zu Java

Zusammenfassung der wichtigsten Unterschiede:

- **keine Veränderungen der abgefragten Objekte möglich**
- **keine Methodenaufrufe möglich (mit wenigen Ausnahmen)**
- **Wenn Abfragen nicht möglich sind (Exceptions erfordern würden), wird `false` zurückgegeben.**

Unterschiede der Filtersprache zu Java

In Version 2.0 gibt es mehr Möglichkeiten (siehe Spec. 2.0, Kap. 14.6.2, S. 150 ff.)***:***

Table 4: Query Operators

Operator	Description
==	equal
!=	not equal
>	greater than
<	less than
>=	greater than or equal
<=	less than or equal
&	boolean logical AND (not bitwise)
&&	conditional AND
|	boolean logical OR (not bitwise)
||	conditional OR
~	integral unary bitwise complement
+	binary addition, unary plus, or String concatenation
-	binary subtraction or unary numeric sign inversion
*	times
/	divide by
!	logical complement
%	modulo operator
instanceof	instanceof operator

Table 5: Query Methods

Method	Description
contains(Object)	applies to Collection types
get(Object)	applies to Map types
containsKey(Object)	applies to Map types
containsValue(Object)	applies to Map types
isEmpty()	applies to Map and Collection types
size()	applies to Map and Collection types
toLowerCase()	applies to String type
toUpperCase()	applies to String type
indexOf(String)	applies to String type; 0-indexing is used
indexOf(String, int)	applies to String type; 0-indexing is used
matches(String)	applies to String type; only the following regular expression patterns are required to be supported and are portable: global "(?i)" for case-insensitive matches; and "." and ".*" for wild card matches. The pattern passed to matches must be a literal or parameter.
substring(int)	applies to String type
substring(int, int)	applies to String type
startsWith(String)	applies to String type
endsWith(String)	applies to String type
Math.abs(numeric)	static method in java.lang.Math, applies to types of float, double, int, and long
Math.sqrt(numeric)	static method in java.lang.Math, applies to double type
JDOHelper.getObjectId(Object)	static method in JDOHelper, allows using the object identity of an instance directly in a query.

Vergleich von JDOQL mit weiteren SQL-Funktionalitäten

Bsp.: Sortieren der Antwort

Select-Query:

- Finde alle Studenten, die von Iwanowski geprüft wurden:

select s **from** Studenten

where s.wurdeGeprüft.contains (p) && p.Prüfer.Name == „Iwanowski“

order by s.Semesterzahl **DESC**, s.Name **ASC**

Filter-Query ?

Achtung:
Diese Lösung setzt voraus,
dass wurdeGeprüft ein mehrwertiges Attribut ist,
d.h. jeder Student mehrere Prüfungen macht.

Vergleiche mit der unterschiedlichen
Voraussetzung in Folie OODB 5-10

Funktionalität von JDOQL-Queries

Sortieren der Antwort für Filter-Queries:

Interface Query

public void setOrdering (String orderingInfo);

- **orderingInfo enthält Sortieranweisungen, durch Komma getrennt, die von links nach rechts lexikographisch befolgt werden.**
- **Jede Sortieranweisung hat die Form: `<Ausdruck> descending` oder `<Ausdruck> ascending`. Hierbei ist `<Ausdruck>` ein Wert, der von einem Element der Antwort-Collection gebildet werden kann und eine Anordnung hat.**
- **Es sind nur navigierende Zugriffe auf Attribute erlaubt (also keine beliebigen Rechenoperationen).**
  Als Datentypen zugelassen sind alle primitiven (außer boolean), alle Wrapper-Typen (außer Boolean) sowie zugelassene Spezialfälle (aber nicht eigendefinierte Objekte)

Existenzquantoren

SQL-ähnliche Frage:

- Finde die Vorlesungen, in denen weibliche Studenten sitzen und die von Iwanowski gehalten werden:

select v

from v **in** Vorlesungen

where (v.gelesenVon.Name = „Iwanowski“) and (exists s in v.Hörer: s.female)

JDOQL-Lösung ?

2. Variante:

- 1. Frage: Finde die Vorlesungen, in denen weibliche Studenten sitzen.
- 2. Frage: Finde unter den Vorlesungen aus der 1. Frage die, die von Sokrates gelesen werden.

Allquantoren

SQL-ähnliche Frage:

- Finde die Vorlesungen, in denen nur weibliche Studenten sitzen und die von Iwanowski gehalten werden:

select v

from v **in** Vorlesungen

where (v.gelesenVon.Name = „Iwanowski“) and (**for all** s in v.Hörer: s.female)

JDOQL-Lösung ?

max,min im Filter

SQL-ähnliche Frage:

- Finde alle Professoren, die umfangreiche Vorlesungen halten:

```
select p
from p in Professoren
where max(select v.SWS from v in p.liest) >= 4
```

JDOQL-Lösung ?

Allgemeine Verwendung von Aggregatfunktionen: max, min, sum, avg, count

SQL-ähnliche Frage:

- Finde alle Professoren, die viel zu tun haben:

```
select p
from p in Professoren
where sum(select v.SWS from v in p.liest) >= 14
```

In JDO gar nicht möglich !

Projektionen

SQL-ähnliche Frage:

- Finde Anzahl und Durchschnittsalter aller jungen Professoren

```
select count (p), avg (p.alter)
from p in Professoren
where p.alter <= 40
```

JDOQL-Lösung ?

Nur in JDO 2.0 möglich !

Projektionen

Auszug aus Spec. 2.0, Kap. 14.6.9, S. 157:

The result expressions include:

- "`this`": indicates that the candidate instance is returned
- <field>: this indicates that a field is returned as a value; the field might be in the candidate class or in a class referenced by a variable
- <variable>: this indicates that a variable's value is returned as a persistent instance
- <aggregate>: this indicates that an aggregate of multiple values is returned
  - `count`(<expression>): the count of the number of instances of this expression is returned; the expression can be "`this`" or a variable name
  - `sum`(<numeric field expression>): the sum of field expressions is returned
  - `min`(<field expression>): the minimum value of the field expressions is returned
  - `max`(<field expression>): the maximum value of the field expressions is returned
  - `avg`(<numeric field expression>): the average value of all field expressions is returned
- <field expression>: the value of a numeric expression using any of the numeric operators allowed in queries applied to fields is returned
- <navigational expression>: this indicates a navigational path through single-valued fields or variables as specified by the Java language syntax; the navigational path starts with the keyword "this", a variable, a parameter, or a field name followed by field names separated by dots.
- <parameter>: one of the parameters provided to the query.

Funktionalität von JDOQL-Queries

Weitere Funktionen:

Interface Query

public void close (Object result);

public void compile ();

public void setIgnoreCache (boolean transactionChangesAreNotConsidered);

public long deletePersistentAll ();

public long deletePersistentAll (Map parameters);

public long deletePersistentAll (Object[] parameters);

public long setUnique (boolean unique);

Zusammenspiel mit anderen JDO-Funktionen:

- **Einbettung einer JDOQLQuery in eine Transaktion nicht zwingend erforderlich, aber empfohlen**

Funktionalität von JDOQL-Queries: Zusammenfassung

- JDOQL verlangt Filter für Boolesche Auswertungen
- Die Auswertungen beziehen sich auf beliebige Eigenschaften von Attributen der dem Filter übergebenen Elemente
- Andere als Booleschen Operationen können im Filter nicht durchgeführt werden (mit wenigen Ausnahmen)
- Die Eingabe in den Filter ist eine Menge von Elementen derselben Klasse (Collection oder Extent)
- Die Elemente können durch Parameter verallgemeinert werden
- Existenzeigenschaften von Collection-wertigen Attributen können über **`contains`** mit Variablen nachgeprüft werden
- Die Ausgabe vom Filter ist eine Collection von Elementen der Eingabeklasse
- Die Ausgabe kann bereits durch den Filter sortiert werden
- Select-Queries (nur JDO 2.0) bieten nur eine SQL-ähnlichere Syntax, aber keine andere Funktionalität als Filter-Queries

Analogien bzgl. der Funktionalität

SINGLE STRING (Select-Query)	TRADITIONAL (Filter-Query)
SELECT [UNIQUE] [<result>]	setUnique() setResult()
[INTO <result-class>]	setResultClass()
[FROM <candidate-class> [EXCLUDE SUBCLASSES]]	setCandidates(Extent), setClass()
[WHERE <filter>]	setFilter()
[VARIABLES <variable declarations>]	declareVariables()
[PARAMETERS <parameter declarations>]	declareParameters()
[IMPORTS <import declarations>]	declareImports()
[GROUP BY <grouping>]	setGrouping()
[ORDER BY <ordering>]	setOrdering()
[RANGE <start>, <end>]	setRange()
	----------
	getFetchPlan()...
	setCandidates(Collection)
	setExtensions()
	setIgnoreCache()
	setUnmodifiable()

Aufstellung von Maximilian Herold