

Objektorientierte Datenbanken

Vorlesung 7 vom 27.05.2004

Dr. Sebastian Iwanowski

FH Wedel

JDO

- **Persistenzkonzept**

Persistenzfähige Klassen, objektspezifische Persistenz, Persistenz durch Erreichbarkeit

- **Transaktionskonzept**

mehrere Transaktionsmanagementstrategien

- **Anfragesprache (JDOQL)**

mit objektorientiertem Fokus, Java-Syntax

- **Konzept zum Management von Datenveränderungen**

über so genannte Lebenszykluszustände von Daten
definiert Mechanismen für den Lebenszyklusübergang

- **Datenidentitätskonzepte**

berücksichtigt unterschiedliche Anforderungen von Datenbank und Programm

Inhalt heute:

Die JDO-Anfragesprache JDOQL:

Motivation für JDOQL

Im Detail: Funktionalität von JDOQL

Motivation für JDOQL

Datenbankanfragemöglichkeiten ohne Anfragesprache

- a) Extraktion eines einzelnen Objects (spezifiziert durch Name / **ID**)
- b) Extraktion des **Extents** einer Klasse

Ungeeignet für das Problem:

Finde Objekte mit bestimmten Eigenschaften !

- a) Hoher Programmieraufwand
- b) Hoher Hauptspeicherbedarf

Ziele der Objektanfragesprache **OQL**

1) gezielte Extraktion von Objekten mit bestimmten Eigenschaften

mehr Komfort, mehr Effizienz

~~**2) Abfragemöglichkeiten ohne detaillierte Java-Kenntnisse**~~

~~politischer und sozialer Grund~~

Für **JDOQL gilt nur das 1. Ziel !**

Wesentliche Merkmale von JDOQL-Queries

- **Jede Query bezieht sich auf genau eine Klasse**
- **Jede Query sucht aus einer Menge von Elementen dieser Klasse diejenigen Elemente, die eine bestimmte Eigenschaft erfüllen**
- **Die gesuchte Eigenschaft einer Query wird durch eine Booleschen Ausdruck bestimmt (*Filter* genannt)**

ODMG-OQL-Beispiel

```
Database db = new Database ();
db.open("fastObjects://LOCAL/MyBase", Database.OPEN_READ_WRITE );
Transaction txn = new Transaction( db );
txn.begin();
Extent AlleProfessoren = new Extent (db, Professoren.class);
```

```
// Frage formulieren:
String queryString = " select p.Name from p in AlleProfessoren " +
                    " where p.Rang = \"C4\" ";

// Frageobjekt erzeugen:
OQLQuery query = new OQLQuery( queryString );

// Frage stellen:
Object result = query.execute();

// Ergebnis auswerten (testen, welcher Typ zurückgegeben wurde):
if (SetOfObject.class.isInstance (result))
    SetOfObject names = (SetOfObject) result;
else if (ArrayOfObject.class.isInstance (result))
    ArrayOfObject names = (ArrayOfObject) result;
```

```
txn.commit();
db.close();
```


Dasselbe Beispiel in JDOQL

```
Properties pmfProps = new java.util.Properties();
pmfProps.put("javax.jdo.PersistenceManagerFactoryClass",
            "com.poet.jdo.PersistenceManagerFactories" );
pmfProps.put("javax.jdo.option.ConnectionURL", "fastobjects://LOCAL/MyBase" );
PersistenceManagerFactory pmf = JDOHelper.getPersistenceManagerFactory( pmfProps );
PersistenceManager pm = pmf.getPersistenceManager();
Transaction txn = pm.currentTransaction();
txn.begin();
Extent AlleProfessoren = pm.getExtent (Professoren.class, true);
```

```
// Frage formulieren:
String queryString = " Rang == \"C4\" ";
// Frageobjekt erzeugen:
Query query = pm.newQuery( AlleProfessoren, queryString );
// Frage stellen:
Collection result = (Collection) query.execute();
// Ergebnis auswerten (aus Professorenmenge muss Namenmenge gemacht werden):
Iteration iter = result.iterator();
HashSet names = new HashSet (); Professoren professor;
while (iter.hasNext()) {
    professor = (Professoren) iter.next();
    names.add (professor.name);}
```

```
txn.commit();
```

***Im Detail:
Funktionalität von JDOQL-Queries***

Funktionalität von JDOQL-Queries

Grundfunktionalität:

Interface PersistenceManager

public Query newQuery ();

public Query newQuery (Class class);

public Query newQuery (Class class, Collection elements);

public Query newQuery (Class class, Collection elements, String filter);

public Query newQuery (Extent elements);

public Query newQuery (Extent elements, String filter);

Interface Query

public void setClass (Class class);

public void setCandidates (Collection elements);

public void setCandidates (Extent elements);

public void setFilter (String booleanExpression);

public Object execute (); **gibt immer eine Collection zurück**

Funktionalität von JDOQL-Queries

Filterdefinition:

- Der Filter entspricht einer Java-Methode, die `boolean` zurückgibt.
- Die Syntax des Filters ist gleich normaler Java-Syntax.
- Es sind fast alle Java-Operationen zulässig (Ausnahmen später).

Filterauswertung:

- Der Filter wird auf jedes Element der Kandidatenmenge angewandt.
- Die Attribute der Kandidatenelemente können im Filtercode direkt angesprochen werden. Das angesprochene Attribut bezieht sich auf das Element, das gerade ausgewertet wird. Alle Elemente, für die der Filter `true` ergibt, werden im `execute` zurückgegeben.
- Wenn kein Filter explizit definiert wurde, wird die gesamte Kandidatenmenge im `execute` zurückgegeben.

Motivation für Fragen mit Parametern

OQL-Anfrage:

- Finde die Namen aller Studenten, die von Sokrates geprüft wurden:

select s.Name

from s **in** AlleStudenten

where s.wurdeGeprüft.Prüfer.Name = „Sokrates“

JDOQL-Lösung ?

Wie kann man erreichen, dass dieselbe Frage für beliebige Professoren gestellt werden kann ?

Fiktive OQL-Anfrage:

- Finde die Namen aller Studenten, die von einem bestimmten Professor geprüft wurden:

select s.Name

from s **in** AlleStudenten

where s.wurdeGeprüft.Prüfer.Name = \$beliebigerProfessorenname\$

JDOQL-Lösung ?

Funktionalität von JDOQL-Queries

Fragen mit Parametern:

Interface Query

public void declareParameters (String declarationOfParams);

public Object execute (Object param1);

public Object execute (Object param1, Object param2);

public Object execute (Object param1, Object param2, param3);

- **declarationOfParams: normale Java-Syntax**
- **Typen und Anzahl der Parameter müssen zueinander passen**
Ausnahme: primitive Datentypen als formale Parameter werden mit zugehörigen Objektdatentypen als aktuelle Parameter aufgerufen, Beispiel: `int` mit `Integer`
- **im Filter dürfen die Parameter wie lokale Variablen benutzt werden**
(mit den Namen, die in `declarationOfParams` deklariert wurden)

Zugriff auf Attribute, die selbst Collections sind

OQL-Anfrage:

- Finde die Namen aller Studenten, die bei Sokrates Vorlesung haben:

select s.Name

from s **in** AlleStudenten, v **in** s.hört

where v.gelesenVon.Name = „Sokrates“

JDOQL-Lösung ?

Funktionalität von JDOQL-Queries

Zugriff auf Attribute, die selbst Collections sind:

Zugriff auf Collection-wertige Attribute im Filtercode mit den Methoden:

- `Collection.isEmpty ()`
- `Collection.contains (varname)`
 - varname muss als Variable deklariert werden.
 - varname bindet ein Element der Collection
 - und sollte in einem zweiten Booleschen Ausdruck weiter eingeschränkt werden

Interface Query

public void declareVariables (String declarationOfVariables);

- **declarationOfVariables: normale Java-Syntax**
- **im Filter dürfen die Variablen als lokale Variablen benutzt werden (mit den Namen, die in declarationOfVariables deklariert wurden)**

Funktionalität von JDOQL-Queries

Importieren von Klassen für Parameter- und Variablendeklarationen:

Interface Query

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 (Syntax wie in Java).**

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 sehr wenigen Ausnahmen)
- Wenn Abfragen nicht möglich sind (Exceptions erfordern würden), wird `false` zurückgegeben.

Sortieren der Antwort

OQL-Anfrage:

- Finde alle Studenten, die von Sokrates geprüft wurden:

select s

from s **in** AlleStudenten

where s.wurdeGeprüft.Prüfer.Name = „Sokrates“

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

JDOQL-Lösung ?

Funktionalität von JDOQL-Queries

Sortieren der Antwort:

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.

Beim nächsten Mal:

JDOQL, 2. Teil