Objektorientierte Datenbanken

Vorlesung 5

Sebastian Iwanowski
FH Wedel

Inhalt heute:

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

Persistenzkonzept

Persistenzkonzept: Persistenzfähige Klassen

Definition in XML-Datei

Ausschnitt der DTD von Sun (`http://java.sun.com/dtd`):

```
<!ELEMENT package ((class)+, (extension)*)>
<!ATTLIST package name CDATA #REQUIRED>

<!ELEMENT class (field|extension)*>
<!ATTLIST class name CDATA #REQUIRED>
<!ATTLIST class requires-extent (true|false) 'true'>
<!ATTLIST class persistence-capable-superclass CDATA #IMPLIED>

<!ELEMENT field ((collection|map|array)?, (extension)*)?>
<!ATTLIST field name CDATA #REQUIRED>
<!ATTLIST field persistence-modifier
                        (persistent|transactional|none) #IMPLIED>
<!ATTLIST field embedded (true|false) #IMPLIED>

<!ELEMENT extension (extension)*>
<!ATTLIST extension vendor-name CDATA #REQUIRED>
<!ATTLIST extension key CDATA #IMPLIED>
<!ATTLIST extension value CDATA #IMPLIED>
```

Persistenzkonzept: Persistenzfähige Klassen

Definition in XML-Datei

Klassendefinitionen:

```
<!ELEMENT class (field|extension)*>
<!ATTLIST class name CDATA #REQUIRED>
<!ATTLIST class requires-extent (true|false) 'true'>
<!ATTLIST class persistence-capable-superclass CDATA #IMPLIED>
```

Persistenzkonzept: Persistenzfähige Klassen

Definition in XML-Datei

Attributmanagement (field management) :

```
<!ELEMENT field ((collection|map|array)?, (extension)*)?>
<!ATTLIST field name CDATA #REQUIRED>
<!ATTLIST field persistence-modifier
                        (persistent|transactional|none) #IMPLIED>
<!ATTLIST field embedded (true|false) #IMPLIED>
```

Persistenzkonzept: Attributmanagement

Drei Typen für die Persistenz durch Erreichbarkeit:

- **persistent**

  normale Persistenz durch Erreichbarkeit, **`rollback()`** bei Transaktionen wirksam

  Zugelassene Attributdatentypen:

  - alle einfachen Datentypen
  - zu einfachen Datentypen gehörende Objektdatentypen
  - alle selbst definierten persistenzfähige Klassen
  - weitere spezielle Klassen

- **transactional**

  Daten sind immer transient, aber **`rollback()`** bei Transaktionen wirksam

  Zugelassene Attributdatentypen: Alle

- **none**

  Daten sind immer transient, **`rollback()`** bei Transaktionen setzt sie nicht zurück

  Zugelassene Attributdatentypen: Alle

Persistenzkonzept: Attributmanagement

Drei Typen für die Persistenz durch Erreichbarkeit:

- **persistent**

  unveränderbare Objekte: Instanzen von

  - allen einfachen Datentypen (z.B. **`int`**)
  - zu einfachen Datentypen gehörenden Klassen (z.B. **`Integer`**)
  - **`Locale, BigDecimal, BigInteger, String`**
  - allen selbst definierten persistenzfähigen Klassen ohne Änderungsmöglichkeiten

  veränderbare Objekte: Instanzen von

  - **`Date, HashSet`**
  - weiteren optionalen Klassen wie **`ArrayList, Vector,`** etc.
  - allen selbst definierten persistenzfähigen Klassen mit Änderungsmöglichkeiten

Unterscheidung aus Anwendersicht:

Nur unveränderbare Objektinstanzen dürfen mit
gleicher Identität an mehreren Stellen verwendet werden !

Persistenzkonzept: Attributmanagement

Zwei Typen für die Objektreferenzierung in der Datenbank:

- **First Class Objects (`embedded = false`):**

  eigene ObjektId

  existiert auch von alleine in der Datenbank

  notwendig, falls mehrere Instanzen auf dieses Objekt zeigen

- **Second Class Objects (`embedded = true`):**

  keine eigene ObjektId

  existiert nur in Verbindung mit referenzierendem Objekt in der Datenbank

  nur möglich, wenn nur eine Instanz auf dieses Objekt zeigt

Die persistenzfähigen Systemklassen sind second class per default

in vielen JDO-Implementierungen gar nicht erlaubt als first class

für second-class-Systemklassen: **`deletePersistent()`** nicht nötig und nicht erlaubt

Persistenzkonzept: Collections

Definition in XML-Datei

Ausschnitt der DTD von Sun (`http://java.sun.com/dtd`):

```
<!ELEMENT field ((collection|map|array)?, (extension)*)?>

<!ELEMENT collection (extension)*>
<!ATTLIST collection element-type CDATA #IMPLIED>
<!ATTLIST collection embedded-element (true|false) #IMPLIED>

<!ELEMENT map (extension)*>
<!ATTLIST map key-type CDATA #IMPLIED>
<!ATTLIST map embedded-key (true|false) #IMPLIED>
<!ATTLIST map value-type CDATA #IMPLIED>
<!ATTLIST map embedded-value (true|false) #IMPLIED>

<!ELEMENT array (extension)*>
<!ATTLIST array embedded-element (true|false) #IMPLIED>
```

Persistenzkonzept: FastObjects-Besonderheiten

(siehe Kapitel 14 aus `doc/JDOProgrammersGuide.pdf`)

```
<!ATTLIST field embedded (true|false) #IMPLIED>
<!ATTLIST collection embedded-element (true|false) #IMPLIED>
<!ATTLIST map embedded-key (true|false) #IMPLIED>
<!ATTLIST map embedded-value (true|false) #IMPLIED>
<!ATTLIST array embedded-element (true|false) #IMPLIED>
```

wird in FastObjects ignoriert !

Stattdessen benutzt FastObjects Extensions:

```
<!ELEMENT extension (extension)*>
<!ATTLIST extension vendor-name CDATA #REQUIRED>
<!ATTLIST extension key CDATA #IMPLIED>
<!ATTLIST extension value CDATA #IMPLIED>
```

Beispiel:

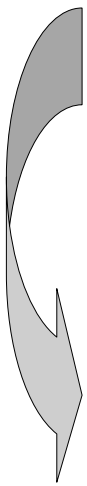
```
<class name = "Prüfung">
   <extension vendor-name = "FastObjects„
              key = "embedded" value = "true" />)
</class>
```

➔ **Entweder sind alle Instanzen einer Klasse second class oder keine**

Transaktionskonzepte

Wdh.: Transaktionseigenschaften

Atomicity	Alle Aktionen auf einmal beim commit()
Consistency	hängt von der Sorgfalt des Programmierers ab
Isolation	Eine Transaktion darf nicht die Daten für eine andere ändern !
Durability	wird durch die Persistenz der Datenbank erreicht



Lösungsalternativen:

- **Nur eine Transaktion zur selben Zeit**
- **Sperrung der von einer Transaktion benutzten Daten für alle anderen**

Isolationsproblem bei Transaktionen

4 Isolationsniveaus (nach ISO):

- **Lesen aus nicht abgeschlossenen Transaktionen (Uncommitted Read)**

  überhaupt keine Gewähr der Konsistenz

- **Lesen nur aus der Datenbank (Committed Read)**

  keine gegenseitigen Überschreibungen von gleichzeitig laufenden Transaktionen
  wiederholtes Lesen aus Datenbank kann unterschiedliche Resultate liefern

- **Wiederholungssicheres Lesen (Repeatably Read)**

  wiederholtes Lesen aus Datenbank liefert immer dieselben Resultate
  wiederholte Anfragen an die Datenbank kann unterschiedliche Resultate liefern

- **Lesen nur von Daten mit Schreibsperre (Completely Isolated)**

  wiederholtes Lesen aus Datenbank liefert immer dieselben Resultate
  wiederholte Anfragen an die Datenbank liefern immer dieselben Resultate

Isolationsproblem bei Transaktionen

2 Sperrmöglichkeiten

- **Lese- und Schreibsperre**

  Exklusive Sperre: Eine Transaktion blockiert alle anderen

- **nur Schreibsperre**

  Gemeinsame Sperre: Keine Transaktion darf schreiben, aber alle dürfen lesen

  Exklusive Sperre: Nur eine Transaktion darf schreiben, aber alle dürfen lesen

JDO-Transaktionskonzept

3 Transaktionsstrategien:

- **Normale (pessimistische) Transaktionen**

  Bei Zugriff wird eine Sperre auf das betreffende Datenbankobjekt gelegt.

  Erst bei Transaktionsende wird die Sperre aufgehoben.

- **Optimistische Transaktionen**

  Es gibt während der Transaktion die meiste Zeit keine Sperre.

  Es wird vor dem Commit nachgeprüft, ob die benutzten Daten sich während der Transaktion geändert haben (dabei gibt es eine Sperre).

  Falls sich die Daten während der Transaktion geändert haben,
  erfolgt eine Nachricht an die Transaktion als Exception und kein Commit.

- **Datenbankzugriff außerhalb von Transaktionen**

  für Objekte in bestimmten Zuständen (wird später besprochen)

JDO-Transaktionskonzept

Technische Realisierung:

- **in jedem PersistenceManager nur eine Transaktion gleichzeitig**

  erhältlich durch **`PersistenceManager.currentTransaction()`**

  Verschachtelte Transaktionen sind verboten (anderenfalls **`JDOUserException`**).

- **verschiedene PersistenceManager in verschiedenen Threads**

  Damit sind parallele Transaktionen möglich.

- **individuelle Einstellung des Transaktionstyps**

  durch **`Transaction.setOptimistic(bool)`**

- **Prüfmöglichkeit auf zwischenzeitliche Änderungen**

  durch **`PersistenceManager.refresh(obj)`**
  oder **`PersistenceManager.refreshAll()`**

FastObjects-Transaktionskonzept

Prinzip:

Der JDO-Standard wird um die ODMG-Implementierung erweitert

Zusätzliche Funktionalitäten:

- **verschachtelte Transaktionen sind erlaubt**
  durch **`PersistenceManagerFactories.setNestedTransactionEnabled()`**
- **es gibt die Möglichkeit, Zwischenstände abzuspeichern**
  durch **`Transactions.checkpoint(txn)`**
- **Transaktionen haben Zugriffsrechte und lösen Sperren aus**
  Schreibsperren abhängig vom Typ (pessimistisch oder optimistisch)
  Schreibzugriff kann abgeschaltet werden (durch Property-Key **`readOnly`**)
- **viele weitere Funktionalitäten in den Klassen**
  `PersistenceManagerFactories, PersistenceManagers, Transactions`

Beim nächsten Mal:
JDO-Anfragen