Tipps und Tricks bei der Nutzung von EclipseLink (JPA)

Falko Steglich
Robotron Datenbank-Software GmbH
Dresden

Schlüsselworte
Java, EclipseLink, JPA, Persistence, Framework, Criteria API

Einleitung

Die Nutzung vom Persistenz-Framework EclipseLink stellt den Softwareentwickler vor neue Chancen, aber auch vor Risiken. Der Vortrag gibt Einblick in die Verwendung von EclipseLink zur nachhaltigen Softwareentwicklung. Es wird ein Überblick gegeben, welche Vorgehensweisen sich im Projekt bewährt haben. Die Aspekte des agilen Vorgehens werden ebenso beleuchtet wie Methoden zur Performance-Optimierung.

Entwurf des Datenbankmodells

Schon ein wohlüberlegter Entwurf des Datenbankmodells kann dem Software-Entwickler viel Unmut in seiner Arbeit ersparen. Bestehende Datenmodelle sollten, soweit möglich, auch an die Anforderungen des Frameworks angepasst werden. Dazu gehört die Einführung von ID-Spalten als Primärschlüssel und die Reduzierung von zusammengesetzten Schlüsseln. Zusammengesetzte Schlüssel können als Unique nachgebildet werden. Das Framework erstellt für den Schlüssel ein gesondertes Objekt, eine Indirektionsebene, welche schwerer lesbaren Code bedingt.

Auch Festgelegte Schlüsselwörter können als Spaltenname verwendet werden. In der Annotation sollte der Spaltenname jedoch maskiert werden, z.B. @Column(„\“Rand\““);

Für die Umsetzung von optimistischen Sperren empfiehlt sich die Einführung einer Versionspalte. Diese muss mit „@Version“ annotiert werden. Es hat sich als günstig erwiesen, die Eigenschaft in einer Basisklasse zu hinterlegen. Die notwendigen Datenbank-Trigger können automatisiert erzeugt werden. Zu beachten ist bei der Arbeit mit den Version-Objekten, dass auch die Datenbankprozeduren die Versionsnummer bei Änderungen modifizieren müssen.

Es sollte durch vorherige Tests festgestellt werden, ob die Größe des Datenmodells zum herkömmlichen EclipseLink-Ansatz kompatibel ist. Zum Erstellungszeitpunkt der Persistenzschicht werden die Metainformationen der im Datenmodell verwendeten Objekte geladen. Bei großen Schemas ist die Instanziierung zeit- und speicheraufwändig. In solchen Fällen empfiehlt es sich, über eine striktere Trennung des Datenmodells nachzudenken oder den Ansatz vom dynamischen Metamodell zu verfolgen.

Nutzung der Criteria API

Große Abfragen (Abfragen mit vielen Join-Bedingungen) sollte man vor der Umsetzung mit der Criteria API erst auf Kompatibilität überprüfen. Wegen der Komplexität von speziellen Datenbankmodellen ist EclipseLink nicht immer in der Lage, die optimale Query zu erstellen. In

solchen Fällen empfiehlt es sich die Abfrage selbstständig zu erstellen. Allerdings sind dann Überprüfungen zu integrieren, sodass keine SQL-Injection ermöglicht wird.

Um den Anforderungen der agilen Softwareentwicklung gerecht zu werden, muss die Struktur des Datenbankmodells flexibel gestaltet werden. Die Critera API kann nur bedingt auf Datenbankänderungen reagieren. Im Normalfall werden die Properties stringbasiert in der Abfrage eingebunden. Es empfiehlt sich, alle Properties einer Klasse als Enumeration nachzubilden. Sind Änderungen an den Metadaten notwendig, können alle Referenzen des Enumerationswertes überarbeitet werden. Sollten dennoch hardcodierte Statements genutzt werden, besteht die latente Gefahr von Laufzeitfehlern. Um zu überprüfen, ob die Statements valide sind, empfiehlt sich der Einsatz von automatisierten Tests. Allein die Ausführung der Abfrage auf der Datenbank, auch ohne Ergebnisse, gibt Auskunft über die Korrektheit der Abfrage.

Die Criteria API sollte weitestgehend getypt verwendet werden. EclipseLink bietet in den Methoden dafür Typparameter an. Es empfiehlt sich Abfragen ohne Typisierung noch in der Serviceschicht auf Laufzeitobjekte darzustellen, damit Fehler der Ergebnismengen frühzeitig erkannt werden.

Die vom Framework bereitgestellten Methoden getSingleResult() und getResultList() sollten verwendet werden. Die dazu passenden Exceptions können Aussagen über die zu erwartende Ergebnismenge geben. Diese Methoden geben frühzeitig über die erwartete Ergebnismenge Auskunft.

EclipseLink generierte Statements

Die von EclipseLink generierten Statements entsprechen einer vordefinierten Reihenfolge. Diese wird vom Framework selbst festgelegt. Zuerst werden Löschoperationen, dann Insertoperationen und schlussendlich Updateoperationen an die Datenbank gesendet. Diese Reihenfolge kann mittels dem JpaHelper und der UnitOfWork geändert werden. Die Update-Operationen müssen in sich nicht konsistent sein, d.h. es kann zur Verletzung von Constraints kommen. Es empfiehlt sich, die Constraints der Datenbank nur defered prüfen zu lassen.

Procedures/Functions verwenden

Prozeduren und Funktionen können mittels den von EclipseLink vorgegebenen Annotationen erstellt und aufgerufen werden. Es ist sinnvoll, einen Generator für die Aufrufe zu erstellen. Dieser muss den Anforderungen genügen, typisierte Parameter anzunehmen und ein typisiertes Objekt als Ergebnis zu liefern. Das Standardverhalten stellt die Ergebnisse von Datenbankoperationen nur als Object-Array zur Verfügung. Aus Gründen der Lesbarkeit und Typensicherheit sollten die Ergebnisse aufbereitet und als Datentransferobjekt zur Verfügung gestellt werden.

Es hat sich als günstig erwiesen, die Ergebnisse einer Prozedur zu kapseln. Diese Kapselung sollte zudem Aussagen ermöglichen, ob die Ausführungseinheit erfolgreich beendet wurde. Diverse Fehlerquellen sind beim Aufruf von Prozeduren und Funktionen denkbar. Eine Analyse, ob die Prozesseinheit nicht gefunden wurde, oder ob sie abgebrochen wurde, hilft in den meisten Fällen die Fehlersuche zu vereinfachen.

Umgang mit mehreren Sessions

Bei der parallelen Verwendung mehrerer Datenbanksessions ist es notwendig, bestimmte Vorgehensmuster einzuhalten.

Objekte einer Session dürfen niemals in anderen Session verwendet werden. EclipseLink wertet die Objekte als neu erstellte Entität und fügt sie der Session an. Beim Speichern wird das in der Datenbank schon vorhandene Objekt nochmals eingefügt. Üblicherweise quittiert die Datenbank dieses Vorgehen mit einer Primärschlüsselverletzung.

Durch die Arbeit mit mehreren Sessions ist es möglich, in zwei Sessions dasselbe Objekt zu laden. Änderungen am Objekt in Session 1 werden in Session 2 nicht sichtbar, solange das Objekt in Session 2 nicht neu geladen wurde. Das erneute Laden darf in diesem Falle nicht aus dem Cache geschehen! Es empfiehlt sich das Objekt in von Session 2 zu trennen und neu hinzuzufügen.

Für die Aktualisierung von Objekten empfiehlt sich auch die Verwendung der REFRESH-Flags in den Annotationen der Persistenzschicht. Werden diese angegeben, dann laden sich die abhängigen Elemente bei gleich mit.

EclipseLink und Performance

Der Performance-Aspekt spielt in vielen Anwendungen eine zentrale Rolle. Deshalb ist es wichtig, die interne Arbeitsweise von EclipseLink zu verstehen. Sogenannte Komfortfunktionen können dem Softwareentwickler viel Arbeit ersparen, im Hintergrund aber auch zu einer Belastung der Zugriffsschicht führen.

Ein Hauptindikator der Performance ist die Latenz. Meist ist es günstiger, ein Gesamt-Statement anstelle von vielen Teil-Statements auszuführen. Um zu ermitteln, wie viele Datenanforderungen an die Datenhaltungsschicht übermittelt werden, hilft es, den Log-Level der persistence.xml auf FINE zu setzen. In der Ausgabe (Konsole oder Datei) werden dann alle gesendeten SQL-Befehle im Klartext protokolliert. Die Ausführung mit der Angabe eines geringen Log-Levels darf keinesfalls als Indikator für die Schnelligkeit der Anwendung dienen, weil die Ausgaben viele Schreiboperationen in das Dateisystem nach sich ziehen.

Es ist möglich, eine Tabelle als Grundlage mehrerer Entitäten zu verwenden. Der Nutzen liegt dabei in der Erstellung von anwendungsspezifischen Sichten, ohne der Erstellung einer View. Es ist denkbar, eine Entität für die Pflege von Elementen zu besitzen, und eine zweite Entität mit derselben Basistabelle. Die zweite Entität enthält nur die für einen Auswahldialog notwendigen Eigenschaften, die ID und keine Beziehung. Mittels dieser Vorgehensweise ist die Anzahl an Statements begrenzbar, überflüssige Joins entfallen.

Vermeiden Sie den direkten Zugriff auf Child-Collections um Elemente zu zählen! In diesem Falle initialisiert das Framework selbstständig alle Listenelemente und gegebenenfalls deren Abhängigkeiten. Hier bietet sich Verwendung einer COUNT-Abfrage an.

```
Region.Countries().lenght -- Countries werden initalisiert!
```

Es ist bei der Erstellung der Mappings darauf zu achten, dass die direkt modellierten Abhängigkeiten vom Framework bei der Initialisierung des Objektes mitgeladen werden. Das kann ärgerlich sein, wenn nur ein Datenobjekt anzeigen möchte, EclipseLink jedoch alle Abhängigkeiten nachlädt. Besonders in Umgebungen mit geringer Latenz führt die Verhaltensweise zu Performanceeinbußen.

Falls nicht zwingend notwendig, sollten die Beziehungen im Mapping deshalb nicht abgebildet werden. Dabei muss abgewogen werden, ob die dadurch verloren gegangene Komfortfunktionalität zum Nutzen im Verhältnis steht. Wenn die abhängigen Daten immer nachgeladen werden müssen, kann auch die Beziehung bestehen bleiben.

Für die Verwendung des Lazy-Loading in dem geschilderten Szenario ist es notwendig, auf statisches Weaving zurückzugreifen. Weaving erlaubt es den Byte-Code der Klassen dahingehend zu manipulieren, das auch die Parent-Child-Beziehungen erst auf Anforderung nachgeladen werden.

Bei der Verwendung von Join-Bedingungen in Abfragen muss Vorsicht walten gelassen werden. Dem Softwareentwickler muss klar sein, dass falsche Joins zu keinen Daten beziehungsweise zu großen Datenmengen führen. Eine Außensicht auf die verschiedenen Datenzustände ist dafür hilfreich.

Kontaktadresse:
Falko Steglich
Robotron Datenbank-Software GmbH
Stuttgarter Straße 29
01189 Dresden

Telefon:	+49 351 25859-2654
Fax:	+49 351 25859-3699
E-Mail	falko.steglich@robotron.de
Internet:	www.robotron.de