Criteria API:
Komplexe SQL Queries mit Eclipslink bauen

Thomas Haskes
Triestram & Partner GmbH
Bochum

Schlüsselworte

rapid.Java, EclipseLink, Oracle, Criteria API, JPA, Datenbank, SQL

Einleitung

In der Java Persistence API gab es bis zur Version 2.0 keine Unterstützung für den dynamischen Zusammenbau von Queries. War es bis dahin für eine datenbankbasierte Anwendung, die einen O/R Mapper verwendet, notwendig, Queries dynamisch zu erstellen, so musste auf proprietäre API des jeweiligen Persistence Providers wie EclipseLink oder Hibernate zurückgegriffen werden. Sollte eine solche Abhängigkeit nicht eingeführt, sondern eine „reine“ JPA Implementierung erzeugt werden, blieb nichts anderes übrig, als die dynamisch zu erzeugenden Queries „von Hand“ zu einem syntaktisch korrekten JPQL oder SQL String zusammenzusetzen. Letzteres Vorgehen hatte allerdings einen entscheidenden Nachteil: Die so erzeugten Queries waren erst zur Laufzeit auf ihre syntaktische Korrektheit überprüfbar, was die dynamische Erstellung von Queries sehr fehleranfällig machte. Unabhängig von der Anfälligkeit wird das dynamische Zusammenstellen eines korrekten Query-Strings mit zunehmender Komplexität sehr schwierig, man denke hier nur an die korrekte Klammerung von WHERE-Bedingungen, die mit dem OR Operator verknüpft sind.

Sind dynamisch erzeugte Queries notwendig?

Ein typischer Anwendungsfall für dynamisch erzeugte Queries sind Suchformulare, wie sie z.B. in der rapid.Java Software „lisa.lims“, einem Informationssystem von T&P, verwendet werden. In lisa.lims ist jedes Modul mit einem Suchformular versehen; je nach Anwendungsfall sind diese Suchformulare unterschiedlich komplex. Außerdem besteht in rapid.Java die Möglichkeit, die Suchformulare über Einstellungen zu konfigurieren. So können einem Suchformular per Konfiguration Felder hinzugefügt oder ausgeblendet werden. Aufgrund dieser Konfigurierbarkeit ist es nicht möglich, ein bestimmtes Suchformular mit einer festen Query zu versehen, deren Bedingungen lediglich über Parameter mit den in das Suchformular eingegebenen Werten gefüllt werden.

JPA 2.0

Mit dem Start der Entwicklung von lisa.lims in der Version 10 wurde die bis dato formsbasierte Software auf Java Technologien umgestellt, wobei schon von Beginn an auf den O/R Mapper EclipseLink gesetzt wurde. Damals befand die JPA sich noch in der Version 1.0. Die Erfahrung aus der Produktentwicklung hat gezeigt, dass das manuelle Erstellen dynamischer Queries mit wachsender Anzahl an Anforderungen an die Suchformulare zu komplex (und damit zu aufwändig)und fehleranfällig wurde. Mit der JPA in der Version 2.0 kam die Criteria API, mit der das dynamische Erzeugen von Queries stark vereinfacht wird, genau zur richtigen Zeit.

Anforderungen aus der Praxis

Folgender Screenshot zeigt das Suchformular für die Kundenverwaltung in lisa.lims.

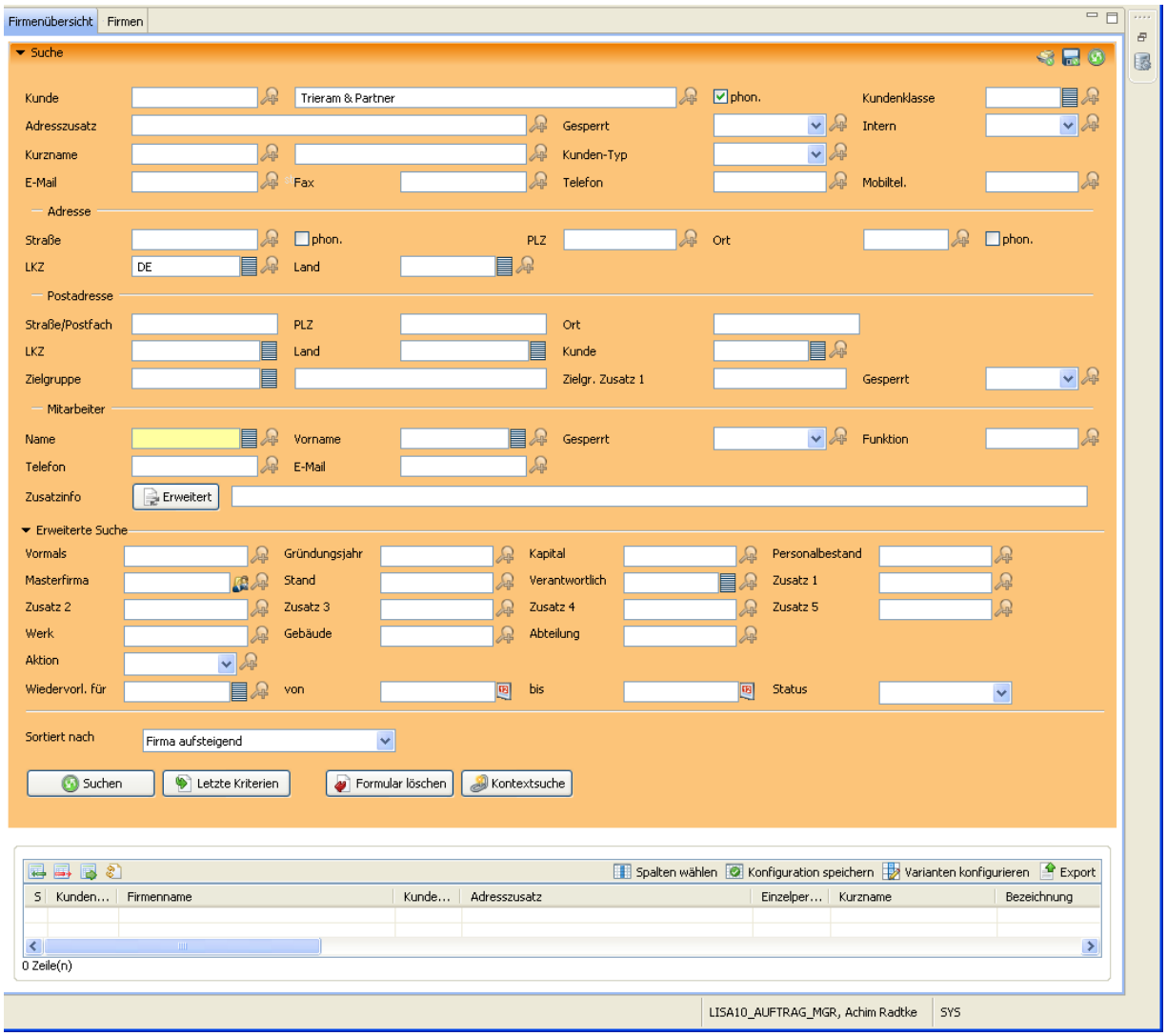

Abb. 1: Suchformular der Kundenverwaltung aus der Software "lisa.lims" von T&P

Allein die Anzahl Suchfelder zeigt schon, dass die für dieses Suchformular verwendete Query dynamisch zusammengesetzt werden muss, da eine statische Query, die für jedes Suchfeld einen Eingabeparameter enthält, der mit einer Wildcard vorbelegt ist, bei dieser Anzahl von Suchfeldern nicht mehr sinnvoll ist.

Dynamisches Hinzufügen von AND oder OR verknüpften WHERE-Bedingungen

Viele der Suchfelder können mit mehreren einschließenden oder auch ausschließenden Einzelkriterien oder Intervallen gefüllt werden. Dies geschieht durch einen Zusatzdialog, der auf einem Suchfeld geöffnet werden kann.

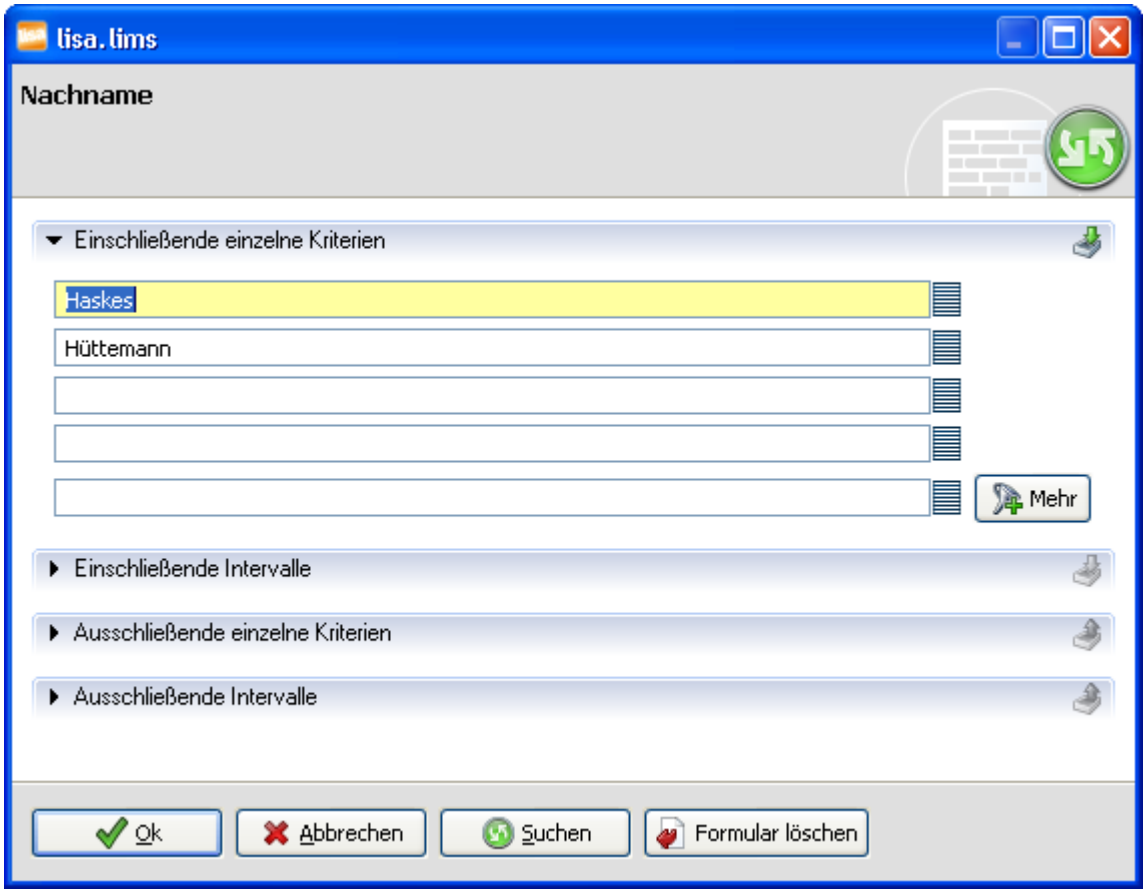

Abb. 2: Zusatzdialog zur Eingabe mehrerer einschließender bzw. ausschließender Kriterien

Für diese Funktionalität müssen in die Query dynamisch mehrere OR-verknüpfte Kriterien bzw. Intervall-Kriterien eingefügt werden.

Einfügen von Subqueries

Die in den Zusatzdialog eingetragenen Werte (Abb. 2) bedeuten, dass nach Firmen gesucht werden soll, die einen Mitarbeiter mit dem Nachnamen „Haskes“ oder „Hüttemann“ haben. Im Datenmodell von lisa.lims werden Mitarbeiter von Firmen in einer eigenen Tabelle gepflegt. Es muss in die Hauptquery also eine Subquery mit EXISTS oder IN eingefügt werden, die die Mitarbeiter jeder Firma durchsucht. Da das Einfügen einer Subquery Auswirkungen auf die Performance der gesamten Query hat, ist es sinnvoll, diese Subquery nur bei Bedarf einzufügen.

Aufruf einer Funktion

Im Screenshot des Suchformulars (Abb. 1) wird nach dem Wert „Trieram & Partner“ phonetisch gesucht (zu erkennen an der gehakten Checkbox „phon.“). Hier soll ein Kunde gesucht werden, dessen Name ähnlich dem eingegebenen klingt (z.B. „Triestram & Partner“). Für die phonetische Suche nach Kunden in der Datenbank wurde ein Algorithmus in PL/SQL programmiert. Dieser Suchalgorithmus kann so nicht oder nur sehr aufwändig mit Hilfe der Sprachkonstrukte, die SQL bietet, formuliert werden. Es ist daher notwendig, eine Query dynamisch erzeugen zu können, die- wenn benötigt – einen Funktionsaufruf enthält. Auch hier zeigt sich, dass die Vordefinition einer Query nicht sinnvoll ist, da sich das Aufrufen einer Funktion innerhalb der WHERE-Bedingungen nachteilig auf die

Performance auswirken kann. Wird die Query dynamisch erzeugt, wird der Aufruf nur dann in die Query eingefügt, wenn er zwingend notwendig ist.

Erweiterbarkeit

Die Suchformulare, die mit Hilfe des rapid.Java Frameworks erzeugt werden können, sind so stark konfigurierbar, dass damit nahezu jeder Standardfall abgedeckt werden kann. Trotzdem kommt es bei der Entwicklung einer Software wie lisa.lims vor, dass ein Suchformular einige ganz bestimmte, nur für einen bestimmten Anwendungsfall notwendige Sonderfunktionen benötigt, die zu spezifisch sind, als das es sinnvoll wäre, diese Funktionen als Standardfunktionalität anzubieten. So dürfen beispielsweise in lisa.lims bestimmte Daten nur für eine bestimmte Gruppe von Anwendern sichtbar sein. Für diesen Anwendungsfall muss es möglich sein, eine „standardmäßig" erzeugte Query noch weiter zu verfeinern. Da die Queries objektorientiert erzeugt werden, ist es relative leicht, die erzeugte Query unmittelbar vor deren Ausführung an einen Hook im Framework weiterzuleiten, innerhalb dessen die erstellte Query nochmal verändert werden kann (ähnlich der Verwendung des PREQUERY Triggers in Oracle Forms). Klassen, die bereits erzeugte Queries für einen bestimmten Anwendungsfall erweitern, werden im rapid.Java Framework „Query Customizer" genannt. Dieses Konzept wird ausführlich im Vortrag vorgestellt.

Vereinfachte API für Clients

Die Architektur von rapid.Java ist dreischichtig. Rapid.Java Clients haben keine direkte Abhängigkeit zur auf dem Server verwendeten Persistenzimplementierung, insbesondere soll sich im Quellcode der Clients kein SQL befinden, um die Clients möglichst frei von Business Logik zu halten. So werden die in ein Suchformular eingegebenen Werte lediglich an den rapid.Java Server auf der Mittelschicht in Form einer Map mit den Schlüsselwerten „Operator + Tabellenspalte" und den eigentlichen Kriterien (z.B. „Triestram%") an den Server weitergereicht. Aus diesen Informationen wird dann auf dem Server die entsprechende Query erzeugt. Diese vereinfachte API kann auf den Clients auch zur programmatischen Datenabfrage verwendet werden, wodurch die Programmierung erleichtert wird. Reicht die vereinfachte API für den Anwendungsfall nicht aus, kann vom Client aus der Aufruf eines Query-Customizers angestoßen werden, mit Hilfe dessen die benötigte Query vollständig ausformuliert werden kann.

Im Vortrag wird die oben erwähnte, vereinfachte API vorgestellt und gezeigt, wie die übergebenen Werte mit Hilfe der Criteria API von EclipseLink zu einer dynamisch erzeugten SQL Query verarbeitet werden. Dazu wird zu jedem oben genannten Anwendungsfall ein Beispiel implementiert.

Kontaktadresse:

Thomas Haskes

Triestram & Partner GmbH

Kohlenstraße 55

D-44795 Bochum

Telefon:	+49 (0) 234-94375-0
Fax:	+49 (0) 234-452206
E-Mail	t.haskes@t-p.com
Internet:	www.t-p.com