

# Datenbankzugriff aus Eclipse RCP-Anwendungen via EclipseLink

Johannes Michler, PROMATIS software GmbH

Zunächst von IBM und ab 2001 als Open-Source-Projekt unter der Führung der Eclipse Foundation entwickelt, war Eclipse lange Zeit vor allem als integrierte Entwicklungsumgebung für Java bekannt. Seit Juni 2009 ist nun die aktuelle Version 3.5 für alle verbreiteten Plattformen verfügbar (Windows, Linux, Mac). Noch immer gilt Eclipse als komfortable und weit verbreitete Java-Entwicklungsumgebung mit vielfältigen Möglichkeiten, beispielsweise zur Syntaxüberprüfung und -vervollständigung, aber auch für Restrukturierung oder Debugging.

Aufgrund des modularen Aufbaus gibt es für die Eclipse-Plattform eine große Anzahl von Erweiterungen, die im Eclipse-Umfeld „Plugins“ genannt werden. Dank dieser Erweiterungen eignet sich Eclipse außer für die Entwicklung von Anwendungen in der Java Standard Edition auch für die Enterprise Version (JEE) sowie für eine Vielzahl weiterer Programmiersprachen (beispielsweise PHP, C/C++). Darüber hinaus stehen Erweiterungen zur Modellierung (EMF) inklusive der Erstellung grafischer Editoren (GMF), für Reporting (BIRT), den Datenbank-Zugriff (EclipseLink) und vieles mehr zur Verfügung. Unter [www.eclipse.org](http://www.eclipse.org) kann man verschiedene Zusammenstellungen dieser Erweiterungen herunterladen. Neben den von der Eclipse Foundation selbst angebotenen Plugins gibt es eine große Anzahl durch Drittanbieter bereitgestellter Erweiterungen (siehe <http://eclipse-plugins.2y.net>).

## Eclipse im Datenbank-Umfeld

Für Datenbanken existieren zwei Arten von Eclipse Plugins: Erweiterungen zur Verwaltung und Entwicklung von Datenbanken sowie solche, die den Zugriff auf Datenbanken aus Java-Anwendungen heraus ermöglichen. Auf der einen Seite steht mit der Eclipse Data-Tools-Plattform (DTP) eine Umgebung für den Zugriff auf beliebige SQL-Datenbanken zur Verfügung. Dies ermöglicht es, über die jeweiligen JDBC-Treiber aus der Eclipse Entwicklungsumgebung heraus SQL zu entwickeln und gegen die Datenbank

auszuführen. Dabei steht auch eine kontextabhängige Syntax-Vervollständigung zur Verfügung. Die Ergebnisse werden, wie von ähnlichen Tools gewohnt, in einer komfortablen Tabledarstellung ausgegeben. Außerdem stehen Werkzeuge zur tabellarischen Bearbeitung von SQL-Tabellen zur Verfügung (siehe Abbildung 1).

Neben diesen generischen Eclipse-Werkzeugen gibt es mit jOra ein spezielles Eclipse-Plugin für die Verwaltung von Oracle Datenbanken. Mit diesem können über DTP hinaus – ähnlich wie im Oracle SQL Developer – vielfältige Datenbank-Aufgaben über eine grafische Oberfläche erledigt werden, angefangen von der Erstellung und Änderung von Tabellen, Views oder Prozeduren über die Darstellung von Explain-Plänen oder Daten (einschließlich Änderungen oder einfacher Filterung) bis hin zu einem komfortablen PL/SQL-Editor.

Dieser Artikel konzentriert sich auf die zweite Art von Eclipse Datenbank-Erweiterungen: den Zugriff von Eclipse-RCP-Anwendungen auf Datenbanken. Hier steht im Eclipse-Umfeld mit der Java-Persistence-API (JPA)-Implementierung „EclipseLink“ ein mächtiges Werkzeug zur Verfügung. Zu dessen Verständnis ist im nächsten Abschnitt die Entwicklung einer ersten eigenen Anwendung auf Basis von Eclipse-RCP beschrieben. Diese wird anschließend um eine Datenbank-Persistierung erweitert.

## PhoneBookRCP

Die Eclipse-Plattform bietet zwei grundsätzliche Möglichkeiten zur Anwendungserstellung. Entweder kann man ein Plugin in die zum Teil aus mehr als tausend weiteren Erweiterungen bestehende Eclipse-Entwicklungsumgebung selbst laden oder in Form

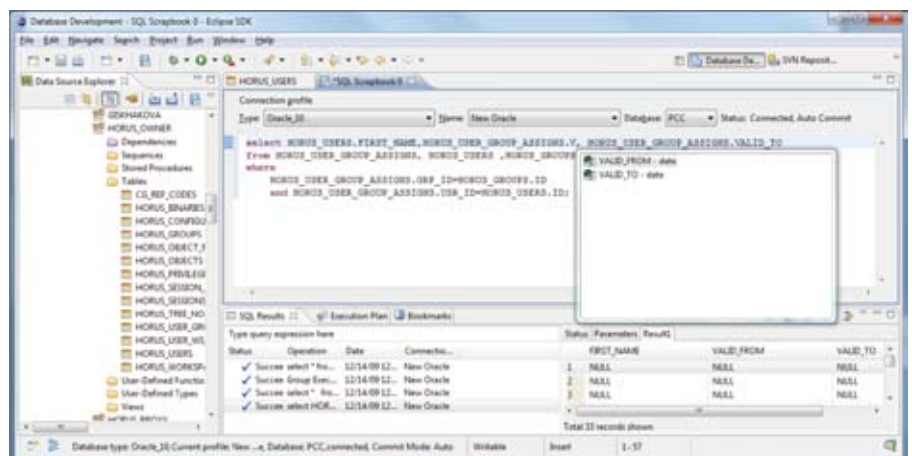


Abbildung 1: Eclipse Data-Tools-Plattform

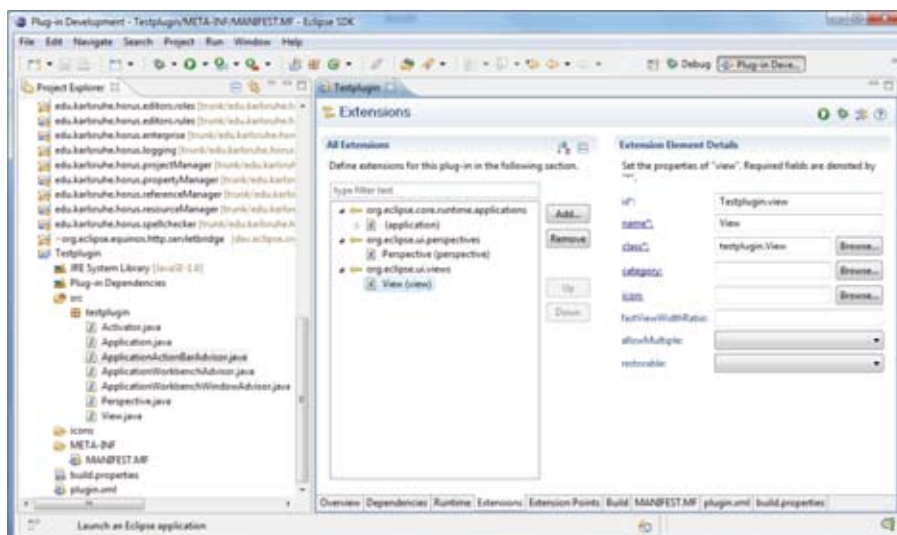


Abbildung 2: Plugin-Editor

einer Eclipse-RCP-Anwendung eine eigenständige Anwendung mit einer genau definierten Menge an zu ladenden Plugins implementieren. Der Aufbau der einzelnen Plugins unterscheidet sich in beiden Fällen nicht wesentlich und auch in einer eigenständigen RCP-Anwendung sind bestehende Eclipse-Erweiterungen problemlos nutzbar. Zur Entwicklung eigener Eclipse-Plugins bietet sich die Nutzung der „Eclipse for RCP/Plug-in Developers“-Variante an, die bereits alle nötigen Bestandteile enthält.

Der „Plug-in Project Wizard“ im Datei-Menü unterstützt die ersten Schritte zum eigenen Eclipse-Plugin. Er fragt zunächst Basisinformationen wie Name, Version und Laufzeitumgebung ab. Außerdem unterscheidet er zwischen RCP- und „reinen“ Plugin-Projekten und fragt, ob die Komponente zur grafischen Oberfläche beiträgt. Eine einfache grafische RCP-Anwendung lässt sich im letzten Schritt über die Vorlage „RCP Application with a view“ erzeugen. Die grafische Oberfläche von auf Eclipse basierenden Anwendungen besteht aus einer oder mehreren Perspektiven, wobei jede Perspektive eine initiale Menge an „Views“ und deren Aufteilung auf der Oberfläche umfasst. Eine einzelne View unterstützt jeweils eine bestimmte Aufgabe. In der Entwicklungsumgebung sind beispielsweise der Projekt-Explorer oder die Eigenschaftenanzeige als Views realisiert.

Die generierte Beispielanwendung besitzt genau eine Perspektive (Perspective.java), welche von der einzigen View (View.java) voll ausgefüllt wird. Diese View enthält zunächst eine einfache Liste mit drei Elementen, die mit den ebenfalls als Plugins bereitgestellten Basistechnologien (JFace und SWT) implementiert ist. Über den Eclipse-Erweiterungsmechanismus werden View und Perspektive in der Datei plugin.xml, die man mit dem in Abbildung 2 gezeigten Editor einfach editieren kann, im Basis-Plugin org.eclipse.ui registriert. Wie aus der Manifest-Beschreibung hervorgeht, hängt die Beispiel-RCP-Anwendung damit unter anderem von dieser Erweiterung ab.

## Die Anwendung ausführen und erweitern

Zur Ausführung der RCP-Anwendung müssen die zugehörigen Plugins in der Equinox-Laufzeitumgebung registriert und gestartet werden. Dies kann über „Launch an Eclipse Application“ auf der Übersichtsseite des in Abbildung 2 gezeigten Plugin-Editors erfolgen. Dabei entsteht automatisch eine Ausführungskonfiguration, die neben dem eigenen „Testplugin“ automatisch die von diesem Plugin in der Manifest.mf definierten abhängigen Erweiterungen enthält. Inklusive geschachtelter Abhängigkeiten besteht so selbst die minimale HalloWelt-Anwendung bereits aus 45 Einzelkomponenten.

Ausgehend von der HalloWelt-Anwendung des vorigen Abschnitts wird nun eine einfache Telefonbuch-Anwendung implementiert. Diese besitzt mehrere Telefonbücher, wobei jedes aus einer Menge von Personen besteht. Für jede Person können Attribute wie Name, E-Mail-Adresse und verschiedene Telefonnummern gepflegt werden. Abbildung 3 zeigt die fertige Anwendung.

## Die Programmierung

Der erste Schritt bei der Programmierung der Anwendung besteht in der Implementierung eines Modells für eine Telefonbuchsammlung „Phonegroups.java“, für ein einzelnes Telefon-

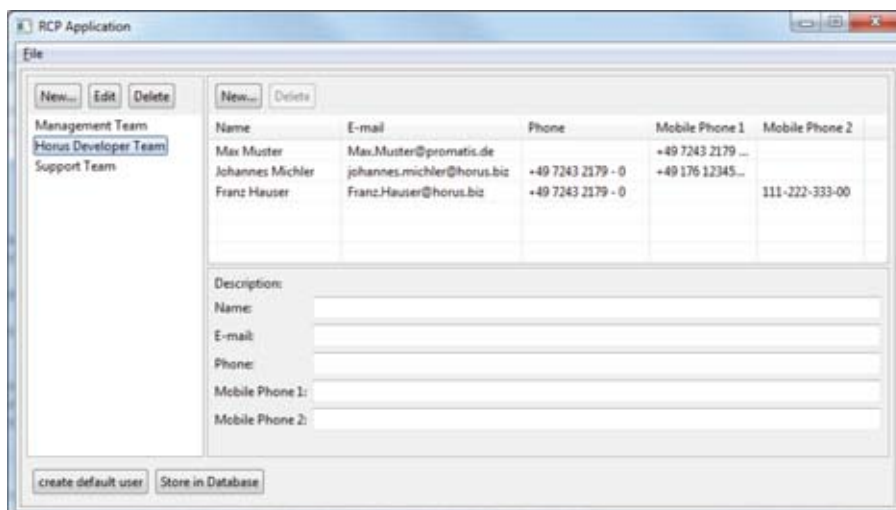


Abbildung 3: Fertige Telefonbuch-Anwendung

buch „Phonergroup.java“ sowie für die Personen in diesen Telefonbüchern „Person.java“. Die Klasse „Person“ besitzt dabei fünf String-Attribute für ihre Eigenschaften (samt zugehöriger get- und set-Methoden) sowie eine Variable vom Typ „Phonergroup“ zur Speicherung ihres „Besitzers“. Ein solches Telefonbuch besteht wiederum aus einer Liste von Personen sowie einem String, der den Namen des Telefonbuchs speichert. In der Klasse für die Telefonbuchsammlung sind mehrere solcher Telefonbücher schließlich in einer Liste zusammengefasst. Das komplette Datenmodell ist – wie alle anderen hier gezeigten Klassen – unter den Downloads zu diesem Artikel zu finden. Dort liegt auch eine RCP-View, welche die Darstellung des Modells übernimmt und die generierte Beispiel-View „View1.java“ ersetzt. Diese View ist mittels SWT und JFace realisiert. Eine solche grafische Oberfläche lässt sich entweder direkt von Hand in Java oder komfortabler mithilfe eines GUI-Designers wie dem kommerziellen SWT-Designer oder (mit einigen Abstrichen) über die Open-Source-Anwendung Eclipse Visual Editor (VE) implementieren. In der View wird mittels Eclipse-Databinding das Datenmodell an die grafische Darstellung gebunden. Änderungen an der grafischen Darstellung führen dabei unmittelbar zu Änderungen im zugrunde liegenden Datenmodell und umgekehrt.

### Datenbank-Zugriff aus Java-Anwendungen

Die im vorigen Abschnitt beschriebene Anwendung behält Telefonbücher und Kontakte nur während eines Programmlaufs im Arbeitsspeicher. Beim Beenden der Anwendung gehen die Daten verloren. Im Folgenden werden nun die Telefonbücher samt der Kontakte in einer relationalen Datenbank gespeichert.

Für den Zugriff einer Java-Anwendung auf eine Datenbank existieren verschiedene Möglichkeiten: Ein bekannter Weg ist es, über JDBC eine direkte Verbindung zur Datenbank herzustellen und über diese Verbindung übliche SQL-Select-, Update- und In-

sert-Anweisungen auszuführen. Dieser Weg bietet über den Aufruf beliebiger SQL-Anweisungen oder PL/SQL-Prozeduren eine hohe Flexibilität und bei richtiger Verwendung eine hohe Performance. Allerdings ist es oft mühsam, die SQL-Anweisungen manuell zu erstellen, ohne dabei SQL Injection oder anderen Angriffen ausgesetzt zu sein. Die SQL-Anweisungen werden dabei außerdem oft Datenbank-abhängig.

Objektrelationale Abbildungen bieten hingegen eine automatische Abbildung zwischen Java-Objekten und relationalen Tabellen. Im Java-Umfeld hat sich dabei die Java-Persistence-API (JPA), die neuerdings in Version 2.0 vorliegt, als offizielle Brückentechnologie etabliert. Der Standard besitzt mehrere Implementierungen, darunter Hibernate, Apache OpenJPA oder EclipseLink. Letzteres ging als Open-Source-Variante aus Oracle TopLink hervor und stellt auch die Referenz-Implementierung von JPA 2.0 dar. Der nächste Abschnitt geht auf JPA genauer ein.

### Java-Persistence-API

Zentraler Punkt der JPA-Spezifikation sind sogenannte „Entitäten“. Diese kennzeichnen eine spezielle Form von Klassen, nämlich solche, die persistente Instanzen besitzen können. Eine Klasse wird als Entität deklariert, indem man sie mit der Annotation „@Entity“ markiert. Solche Klassen müssen einen parameterlosen Standard-Konstruktor besitzen und dürfen nicht final sein. JPA verwendet an vielen Stellen das Prinzip „Configuration by exception“: Für viele Einstellungen sind Standardwerte voreingestellt, die bei Bedarf jedoch überschrieben werden können. So führt obige „@Entity“-Annotation automatisch zu einer Abbildung der Klasse auf eine Tabelle gleichen Namens, wobei die primitiven Eigenschaften auf gleichnamige Spalten in der Tabelle abgebildet sind. Über die Annotationen „@Table“ und „@Column“ lässt sich dieses Verhalten jedoch auch manuell beeinflussen. Eine Sonderrolle spielt eine mit „@Id“ versehene Eigenschaft, die als Primärschlüssel abgebildet wird.

Beziehungen zwischen Entitäten werden in Java meist über Referenzen oder Behälter-Typen wie Felder, Listen oder Mengen realisiert. Diese können in JPA mit den Annotationen „@OneToMany“ (für 1:1-Beziehungen), „@ManyToOne“ (n:1), „@OneToMany“ (1:n) oder „@ManyToMany“ (n:m) auf die Datenbank-Entsprechungen „Fremdschlüssel-Beziehung“ oder „Join-Tabelle“ abgebildet werden.

### Erweiterung des Telefonbuch-Datenmodells

Um die Anwendung prinzipiell datenbankfähig zu machen, sind zunächst einige vorbereitende Schritte notwendig. Dazu müssen die Komponenten „javax.persistence“ und „org.eclipse.persistence.\*“ in der Manifest.mf als benötigte Plugins deklariert sein. Außerdem muss der datenbankspezifische JDBC-Treiber in den Klassenpfad aufgenommen werden (beispielsweise ojdbc6.jar). Zur Erweiterung des Datenmodells kann anschließend die Klasse „Person“ mit der Annotation „@Entity“ versehen und dort ein Id-Feld für den Primärschlüssel hinzugefügt werden:

```
@Id @GeneratedValue private
long id;
```

Dieselben Anpassungen sind auch für die Klasse „PhoneGroup“ durchzuführen. Dort ist zusätzlich die Liste der Personen als „@PrivateOwned @OneToMany(cascade=CascadeType.ALL)“ zu kennzeichnen. Dadurch wird die Beziehung über einen Fremdschlüssel in der Personen-Tabelle abgebildet. Dank der Kaskadierung führt das Entfernen oder Hinzufügen einer Person zu dieser Liste automatisch zur Persistierung beziehungsweise Löschung der entsprechenden Person im EntityManager. Auch das erweiterte Datenmodell kann wieder bei den Downloads gefunden werden.

### Lebenszyklus und Auffinden von Entitäten

Ein sogenannter „EntityManager“ verwaltet Entitäten in JPA. Über diesen

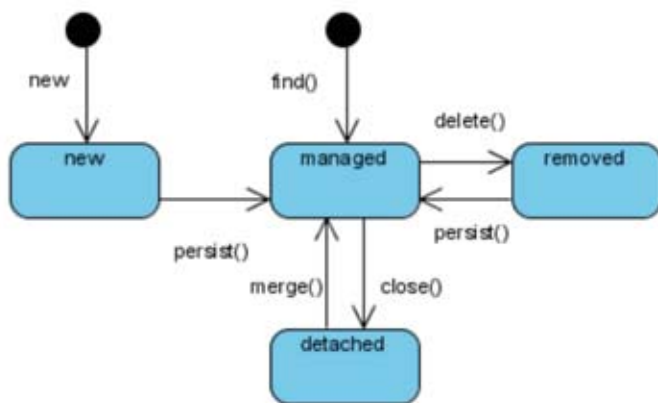


Abbildung 4: Lebenszyklus einer Entität

werden Entitäten gefunden, gespeichert oder gelöscht. Abbildung 4 zeigt den Lebenszyklus einer Entität.

Wird eine als Entität markierte Klasse ganz normal über ihren Konstruktor erstellt, so befindet sie sich zunächst im Status „new“. Erst durch Aufruf der persist-Methode des EntityManagers wird die Klasse zu einer verwalteten Entität, die dann auch in der Datenbank gespeichert ist. Bestehende Entitäten kann man über die Methode „find“ des EntityManagers anhand ihres Primärschlüssels finden. Mit der Java Persistence Query Language (JPQL) können auch komplexere Abfragen gestellt werden – die Syntax orientiert sich dabei an SQL.

Ein EntityManager stellt dabei eine Art Datenbank-Sitzung dar. Über ihn kann man auch einen transaktionalen Kontext aufbauen. Zur Erstellung eines EntityManagers dient die EntityManagerFactory. Diese existiert genau einmal pro Persistenz-Kontext und besitzt unter anderem einen Verbindungspool zur Datenbank sowie einen gemeinsamen Cache.

### Fertigstellung der Demo-Anwendung

Für die Telefonbuch-Anwendung muss man zunächst unter src/META-INF/persistence.xml einen Persistenz-Kontext definieren. Dieser besitzt einen eindeutigen Namen und enthält eine Liste seiner Entitäten (Person und PhoneGroup). Anschließend muss die Klasse „PhoneGroups“, die in der ursprünglichen Version alle Telefonbücher über

eine Liste verwaltet, folgendermaßen angepasst werden:

- Im Konstruktor der Klasse werden der JDBC-Treiber registriert und eine EntityManagerFactory für den Persistenz-Kontext erstellt. Unter anderem ist dabei festgelegt, wo und mit welchen Benutzerdaten die Datenbank zu erreichen ist. Diese Parameter können auch bereits statisch in der persistence.xml gesetzt sein. Anschließend können auf der Fabrik ein neuer EntityManager erstellt und eine neue Transaktion gestartet werden. Über die einfache Anfrage „select pg from PhoneGroup pg“ wird anschließend die Liste aller Telefonbücher abgerufen. Diese Liste kann man direkt der bereits früher implementierten lokalen Liste der Klasse zuweisen.
- Die Methoden „addGroup“ und „removeGroup“, die ein neues Telefonbuch hinzufügen beziehungsweise löschen, sind um einen persist- bzw. remove-Aufruf mit dem entsprechenden Telefonbuch auf dem EntityManager zu ergänzen.
- Schließlich muss man eine Methode „save“ implementieren (samt zugehörigem Button auf der GUI), die die bisherige Transaktion des EntityManagers abschließt (commit) und eine neue startet (begin).

Beim ersten Start der Anwendung werden automatisch die entsprechenden Datenbank-Tabellen generiert. Die Abfrage nach existierenden Datensätzen

liefert natürlich noch kein Ergebnis. Erfasst man diese jedoch in der Anwendung und speichert sie anschließend, so stehen die Daten auch bei zukünftigen Anwendungsstarts oder für andere Benutzer zur Verfügung.

### Erweiterte Anwendungsszenarien

Der komponentenorientierte Ansatz der Eclipse-Laufzeitumgebung ermöglicht eine gute Modularisierung von Anwendungen. Es hat sich dabei bewährt, die Datenbank-Zugriffsschicht als eigenständiges Plugin unter Zuhilfenahme von EclipseLink zu realisieren. Ein Anwendungsbeispiel ist die Implementierung der Eclipse-RCP-Anwendung „Horus Business Modeler“. Horus ist eine frei verfügbare Modellierungs- und Simulations-Software, die in einem Unternehmen oder einer vernetzten Business Community das verfügbare Organisationswissen nutzbar macht und Geschäftsprozesse abbildet (<http://www.horus.biz>). Bei der Implementierung ist es möglich, das für den Datenbank-Zugriff zuständige Plugin (ohne es neu zu übersetzen) sowohl direkt in der RCP-Anwendung für den lokalen Datenbankzugriff als auch – über die serverseitigen Eclipse-Komponenten – auf einem Server zu verwenden.

### Fazit

Insgesamt ermöglicht der komponentenorientierte Ansatz der Eclipse-Plattform eine agile und verteilte Entwicklung großer Rich-Client-Anwendungen. Über die freie JPA-2.0-Referenzimplementierung EclipseLink ist es mit moderatem Aufwand möglich, objektorientierte Datenmodelle in einer relationalen Datenbank zu speichern. Dabei kann von vielen sinnvollen Voreinstellungen profitiert werden – von Caching über Standard-Tabellennamen bis hin zur Abbildung von Attributen auf Tabellenspalten.

### Kontakt:

Johannes Michler  
johannes.michler@promatis.com