Martin–Luther–Universität Halle–Wittenberg

Institut für Informatik

Projektarbeit

Implementierung einer grafischen Nutzeroberfläche zur Administration und Verwaltung von Datenbankzugriffsrechten für ORACLE Virtual Private Database

Verfasser:

Karl Spies

19. Dezember 2005

Betreuer:

Prof. Dr. Stefan Brass, Dipl.–Inf. Matthias Lange

Universität Halle–Wittenberg
Institut für Informatik
Von–Seckendorff–Platz 1 , D–06120 Halle (Saale)
Germany

Vorwort

Die Projektarbeit entstand in Zusammenarbeit des IPK Gatersleben mit der Martin–Luther–Universität Halle. Sie wurde betreut von Seiten des IPK durch Matthias Lange. Die universitäre Schirmherrschaft übernahm Prof. Dr. Stefan Brass. Es sollte dabei ein Werkzeug für das IPK implementiert werden, welches zur Administration von Nutzerrechten verwendet werden kann.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Verzeichnis der Abkürzungen

DBMS	Datenbankmanagementsystem
DB	Datenbank
DBA	Datenbankadministrator
DBS	Datenbanksystem
HTML	Hyper Text Markup Language
IPK	Institut für Pflanzengenetik und Kulturpflanzenforschung
JDBC	Java Database Connectivity
JNLP	Java Network Launch Protokol
JRE	Java Runtime Environment
LDAP	Lightweight Directory Access Protocol
MVC	Model View Controller
OID	Oralce Internet Directory
PDW	Plant Data Warehouse
RDBMS	Relationales Datenbankmanagementsystem
VPD	Virtual Private Database

Kapitel 1

Einleitung

In der heutigen Zeit bekommen Sicherheit und Kontrolle einen immer größeren Stellenwert. Das ist bei Datenbanksystemen nicht anders. Die Komplexität und der zeitliche Aufwand steigen bei großen Nutzerzahlen und Datenbanken enorm. Außerdem wird eine immer feinere Abstufung der Kontroll– und Sicherheitsmechanismen notwendig. Oracle versucht diesen Ansprüchen mit ihrem Konzept der "Virtual Private Database" (VPD) gerecht zu werden.

Dabei wird der Nutzerzugriff auf Tabellen und Schemata nicht über Sichten (Views) und Benutzerrechte geregelt, sondern über in der Datenbank abgelegte Sicherheitsrichtlinien (Security Policies). Diese erweitern zur Laufzeit die SQL–Abfragen des Nutzers und garantieren so den Sicherheitsanspruch. Dabei ist die Adminstration ein wichtiger Aspekt, der sehr viel Zeit in Anspruch nimmt.

Ziel der Projektarbeit ist es, ein Werkzeug zur Verfügung zu stellen, das den Aufgaben der Verwaltung gerecht wird und dabei den zeitlichen Aufwand minimiert.

1.1 Gliederung der Arbeit

Die Arbeit gliedert sich wie folgt: Die Beschreibung der Aufgabe erfolgt in Kapitel 1. In Kapitel 2 werden die Grundlagen erarbeitet und Kapitel 3 beschreibt die programmiertechnischen Hintergründe. Wie das Programm benutzt wird und welche Aufgaben sich damit erledigen lassen wird in Kapitel 4 erklärt.

1.2 Motivation

Am IPK Gatersleben werden eine Vielzahl von Datenbanken auf Basis des RDBMS Oracle entwickelt. Für den Datenzugriff existieren ebenfalls verschiedene Werkzeuge. Der Datenschutz am IPK erfordert die Umsetzung von individuellen Zugriffsrechten auf Datenbankobjekte, wie etwa Tabellen. Zur Zeit wird von dem konventionellen Datenschutz über Schemanutzer verbunden mit propritärer Nutzer– und Rechteverwaltung gebrauch gemacht. Dieses Vorgehen kann den Anforderungen am IPK zum Datenschutz auf feingranularer Ebene mit zentraler Rechteverwaltung nicht mehr genügen. Aus diesem Grund wird eine Oracle–Technologie eingesetzt, die es erlaubt Nutzern und Gruppen Zugriffsrechte auf Datenbankobjekte für die Operationen `INSERT, UPDATE, DELETE, SELECT` zu geben. Diese Zugriffsrechte haben eine Granularität auf Tupelebene. D.h. es ist möglich, Zeilen einer bestimmten Tabelle und eines bestimmten Schema hinsichtlich der genannten Datenbankoperationen zu beschränken. Diese Technologie setzt voraus, dass eine zentrale Nutzerverwaltung in Form einer **LDAP** Implementation von Oracle namens Oracle Internet Directory (**OID**) existiert. Diese dort verwalteten Nutzer werden auf Datenbankschemanutzer abgebildet. Die Zugriffsrechte für jeden Nutzer werden in einem separaten Schema verwaltet und vom VPD genutzt.

1.3 Anforderungen

Das Werkzeug soll eine intuitive Verwaltung der Nutzerrechte in dem vorhanden Datenbankschema umsetzen. Es soll dezentral und plattformunabhänig umgesetzt werden. Für den Zugriffsschutz auf Tupelebene sollen geeignete Visualisierungsmethoden entwickelt werden, die mit großen Datenmengen umgehen können. Folgende Funktionen soll das Werkzeug beherrschen:

1. Darstellung der Tupelrechte pro Nutzer/ Nutzergruppen für gegebene Tabellen
2. Editieren einzelner Tuppelrechte
3. Editieren von Tuppelgruppen bzw.–bereichen, die bestimmten Kriterien genügen
4. Editieren von Nutzergruppen und Nutzerattributen

Kapitel 2

Grundlagen

In diesem Kapitel werden grundlegende Techniken und Designmodelle beschrieben. Jede dieser Ideen hat Einfluß auf den Entwicklungsprozess von WARM genommen und findet sich in Teilen oder auch volltändig im Quellcode von WARM wieder. Sollten sie mit diesen Technologien vertraut sein, können sie auch gleich mit Kapitel 3 fortfahren.

2.1 Model–View–Controller

Das MVC–Modell[1] ist ein Entwurfsdesign, bei dem eine Software in die drei von einander unabhänigen Einheiten Datenmodell, Präsentation und Programmsteuerung unterteilt wird. Das Ziel ist ein flexibles Programmkonzept, das spätere Änderungen und Erweiterungen vereinfacht und die Wiederverwendbarkeit von vorhandenen Komponenten ermöglicht. Durch die Reduktion der Komplexität werden große Anwendungen übersichtlicher und durch die Trennung können verschiedene Teile des Programmes gleichzeitig von verschiedenen Programmierern bearbeitet werden. Im Folgenden werden Model, View und Controller näher betrachtet und erläutert.

2.1.1 Model

Das Model bildet den Kern einer Applikation. Es enthält die Programmlogik und den Datenbestand. Es hat also Zugriff auf Informationssysteme, wie zum Beipsiel eine Daten-

[1] Das MVC Entwurfsdesign wurde erstmals 1979 von Trygve Reenskaug beschrieben. Es inspirierte viele Entwickler von GUI–Toolkits (z.B. SWING, NeXTSTEP, Cocoa). Für weitere Information sei auf [3], [7] oder [8] verwiesen.

bank oder ein XML–Dokument. Es ist außerdem völlig getrennt von View und Controller. Das Model hat keine Kenntnis über Darstellungsweise oder Interaktion mit dem in ihm enthaltenen Datenbestand. Änderungen am Datenbestand werden über Updatemechanismen vorgenommen, welche nur die Änderungen entgegen nehmen. In welcher Form die Änderungen gespeichert werden oder was sich im Detail ändert, bleibt den Benutzern des Model's verborgen. Im Allgemeinen existiert für eine Anwendung auch nur ein Model.

2.1.2 View

Die Darstellungsschicht repräsentiert eine Sicht auf den Zustand der Daten der Anwendung. Die View kann verschieden und mehrfach vorkommen. Zum Beispiel kann eine Menge von Zahlenwerten, als Tabelle (View 1) oder als Graph (View 2) dargestellt werden. Die View wird beim Model registriert, so dass sie auf Änderungen aufmerksam wird und sich dementsprechend ändern kann. Unterschieden wird dabei ob das Model eine aktive oder passive Rolle spielt. Muss die View Änderungen am Model erfragen, handelt es sich um ein passives Model. Das aktive Model teilt den registrierten Komponenten Veränderungen mit.

2.1.3 Controller

Die Steuerungsschicht ist die Schnittstelle zwischen Benutzer und Daten. Sie realisiert die eigentliche "Intelligenz" des Programmes und bildet mit Hilfe der anderen Module die anfallenden Prozesse ab. Sie steuert den Ablauf, verarbeitet Daten, entscheidet welche View aufgerufen wird, etc.

2.2 Standard Rechteverwaltung in Oracle

Die Standard Nuterverwaltung vom Oracle DBMS beruht darauf, das es verschiedene Nutzer (User) gibt, die auf die Datenbank zugreifen möchten. Jeder Nutzer hat bestimmte Rechte um Datenbankobjekte anzulegen. Diese Objekte "bilden ein Schema oder gehören zu einem Schema." [9, S.57] Aus diesem Zusammenhang ergibt sich die Namensgebung "Schemanutzer". Jeder Schemanutzer hat zuerst einmal nur Berechtigungen für die von ihm angelegten Objekte. Er hat also nur Zugriff auf sein Schema.

Weitere Rechte, z.B. Leserechte auf andere Schemata, können dem Nutzer per `GRANT` Befehl erteilt werden. Oracle unterscheidet dabei in System– und Objektrechte. Ein

Systemrecht erlaubt dem Besitzer Aktionen durchzuführen, die die gesamte Datenbank betreffen. Objektrechte beschränken sich ausschließlich auf die vom Benutzer angelegten Datenbankobjekte. Zusätzlich zu einzelnen Rechten kann einem Schemanutzer eine Rolle zugewiesen werden. Eine Rolle ist eine Menge von `GRANT` Befehlen. Verschiedene Schemanutzer dürfen die gleich Rolle besitzen, auch mehrfach Zuweisungen sind erlaubt. Beziehungen von Rolle zu Rolle sind zusätzlich möglich. Dadurch können Abhänigkeiten und ganze Hierachien von Rechten erstellt werden.

Die vorgestellte Rechtezuweisung beschränkt sich auf Datenbankobjekte. D.h. die Einschränkungen können bis zur Spaltenebene getätigt werden. Eine höhere Feinheit ist nicht möglich und genau hier setzt das VPD–Konzept an.

2.3 Virtual Private Database

Mit der Version Oracle 8.1.5 wurde in das DBMS ein neuer Mechanismus eingebaut, welcher die effiziente Implementierung der Zugriffssteuerung mit Granularität auf Tupelebenen erlaubt. Er ist unter den Folgenden Namen bekannt:

- Virtual Private Database
- Row Level Security
- Fine Grained Access Control

Es soll möglich sein, einem Benutzer nur die Tupel einer Tabelle zu zeigen, die für ihn freigebenen sind. D.h. zwei Nutzer mit unterschiedlichen Rechten sehen für ein und die selbe Tabelle verschiedene Tupel. Im Allgemeinen werden zur Lösung der Aufgabe für jeden Nutzer eine oder mehrere Views erstellt. Die View filtert die Tabelle mit vorgegebenen Parametern. Der Nutzer stellt seine Anfragen nicht mehr direkt an die Tabelle, sondern an die View. Der betriebende Aufwand ist sehr groß und erhöht sich, wenn sich die Paramerter mit der Zeit oft ändern. Genau hier setzt die Idee der Virtual Private Database an. Die Filterung wird zur Laufzeit der Anfrage bewerkstelligt, indem die Anfrage mit versteckten `WHERE` Bedingungen erweitert wird. Die Erweiterungen werden transparent vom DBMS durch Policy Funktionen erzeugt. Dazu wird eine PL/SQL Funktion über eine Policy mit der zu schützenden Tabelle verknüpft. Die Verknüpfung stellen Oracle Funktionen aus dem `DBMS_RLS Packages` zur Verfügung. Der PL/SQL Teil des Konstruktes aus Tabelle, Policy und PL/SQL erzeugt die ergänzenden `WHERE` Bedinungen und die Policy dient als Mittler zwischen Tabelle und PL/SQL Teil. Das folgende Beispiel verdeutlicht die Funktionsweise.

Beispiel 1:

Aufgabe: Der Zugriff auf die Tabelle `SCOTT.EMP` ist für alle Benutzer nur während der Geschäftszeit erlaubt.

Lösung mittels VPD: Es wird eine PL/SQL Funktion `GetWorkHoursPredicate` erzeugt, die folgende `WHERE` Klauseln in Form eines Text–Strings generiert:

- 1 = 1 zwischen 8:00 und 17:00
- 0 = 1 sonstige Systemzeit

Diese wird dann mit der Tabelle `SCOTT.EMP` verkünpft. Dazu wird ein Aufruf des `DBMS_RLS Packages` benötigt:

```
    object_schema   => 'SCOTT',
    object_name     => 'EMP' ,
    policy_name     => 'SAMPLE_POLICY',
    function_schema => 'SECOWNER',
    policy_function => upper('GetWorkHoursPredicate'),
    statement_type  => 'SELECT,INSERT,UPDATE,DELETE',
    update_check    => TRUE );
```

Listing 2.1: Definieren einer Policy

Bei jeder Anfrage an diese Tabelle wird nun die Policy aufgerufen. Sie besitzt mehrere Parameter. Interessant sind insbesondere `policy_function` und `statement_type`. Erstere verweist auf die PL/SQL Funktion und letzere gibt an, bei welcher Art von Anfrage die Policy reagiert.

Reaktionsweise: Der Nutzer führt um 8:00 Uhr den folgenden Befehl in der SQL–Konsole aus:

```
SQL> SELECT ename
     FROM scott.emp
     WHERE deptno = 10;

 ENAME
 --------
 KING
 CLARK
 MILLER
```

Listing 2.2: SQL Statement

Weil die Anfrage vom Typ `SELECT` ist, führt Oracle die Policy Funktion aus. Diese überprüft die Zeit mittels `GetWorkHoursPredicate` und liefert einen Wert zurück, der an die `SELECT` Anweisung als zusätzliche `WHERE` Bedingung angehangen wird. So sieht die SQL Anweisung intern aus:

```
SELECT ename
FROM scott.emp
WHERE deptno = 10 AND (1=1);
```

Listing 2.3: SQL Statement mit VPD Erweiterung

Das Sicherheitskonzept funktioniert völlig transparent, dynamisch und ohne weitere Eingriffe in die Datenbank. Es ist dem DBA überlassen, wie er die Policies und PL/SQL Funktionen implementiert. Es stehen ihm alle Möglichkeiten offen. Weitere Beispiele und mehr Informationen befinden sich in [2] und [1].

Die vorgestellte VPD Technologie ermöglicht erstmals einen auf Tupelebene wirkenden Zugriffsschutz.

2.4 VPD Umsetzung am IPK

Die Entwicklung der für die VPD nötigen Policies und den damit verbundenen Überprüfungsmechanismen wurde im Auftrag des IPK duch die B.I.M.–Consulting mbH durchgeführt. Ursprünglich wurde der feingranulare Zugriff für das PDW (Plant Data Warehouse) Projekt des IPK entworfen, mittlerweile ist er für alle Datenbanken des IPK verfügbar. Der Entwurf erweitert das VPD Konzept um eine Nutzer– und Rechteverwaltung. Diese Erweiterung ist in Form von Relationenschemata innerhalb eines eigenständigen Datenbankschema (`PDW_SECURE`) organisiert. “Das Datenbankschema zum Nutzer– und Rechtemanagement soll die Verwaltung sämtlicher Nutzer und Organisationseinheiten sowie deren feingranularen Zugriffsrechte (diese unter Nutzung der VPD–Technologie) auf den Datenbankschemata der Operativsysteme übernehmen.“ [9, S.61] Um die folgenden Erklärungen besser nachvollziehen zu können, wird auf die Abbildung 2.1 auf der nächsten Seite verwiesen.

2.4.1 Nutzerverwaltung

Die Nutzerverwaltung speichert Informationen zu einzelnen Nutzern und Organisationseinheiten (OE) (z.B. Arbeitsgruppen, Projekte). Es ist möglich, Nutzer zu Organisationseinheiten und OE zu anderen Organisationseinheiten zuzuordnen. Die für die

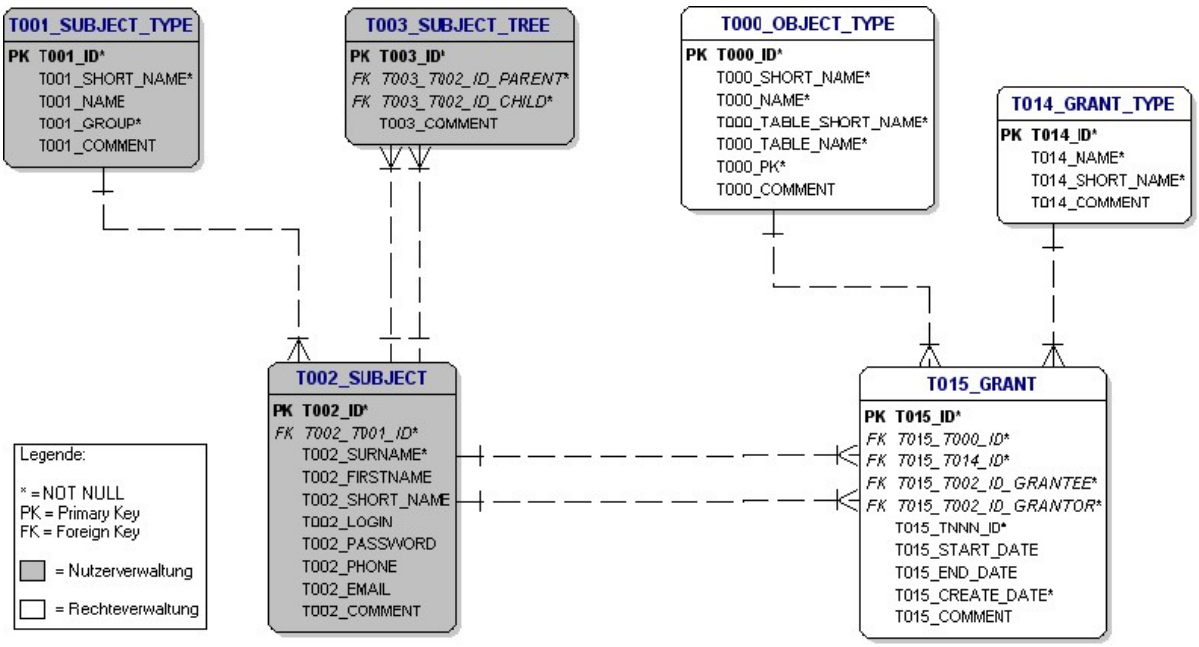


Abbildung 2.1: Relationenschemata für die VPD Umsetzung

Nutzerverwaltung relevanten Teile der Relationenschemata sind: `T001_SUBJECT_TYPE`, `T002_SUBJECT` und `T003_SUBJECT_TREE`. Sie sind in der Abbildung 2.1 grau hinterlegt.

Die Relation `r(T002_SUBJECT)` enthält Informationen über Nutzer und OE. Die Attribute der Relation sind selbsterklärend, wobei `t002_login` einzigartig bleiben muss.

Die Relation `r(T001_SUBJECT_TYPE)` bestimmt die Ausprägung der in `T002_SUBJECT` abgelegten Objekte und ist damit Fremdschlüssel in `T002_SUBJECT`. Ein Beispiel für einen Eintrag in `T001_NAME` ist “user“ oder “group“.

Abhänigkeiten zwischen den Objekten in `T002_SUBJECT` werden durch einen Selbstbezug oder rekursive Beziehung der Relation `r(T003_SUBJECT_TREE)` zu sich selbst erzeugt. Dafür werden zwei Fremdschlüssel, die auf die Relation `T002_SUBJECT` referenzieren, genutzt. Sie agieren nach dem “Vater–Kind–Prinzip“ und dürfen nicht NULL sein. Dadurch ergeben sich einige Schwierigkeiten bei der Bildung von Gruppen ohne Mitglieder und der Festlegung der Wurzelgruppe. Eine Wurzelgruppe zeichnet sich dadurch aus, dass sie niemals das Kind einer andern Gruppe ist. D.h. sie ist in der Spalte der Kinder nicht vorhanden. Dieses Merkmal ist nicht eindeutig und somit besteht die Möglichkeit mehrerer Wurzelgruppen.
Die weitere Schwierigkeit besteht darin, eine Gruppe ohne Mitglieder zu erstellen. Es

müßte einen Eintrag in T002_SUBJECT_TREE geben, dessen Kindeintrag NULL wäre. Wie eingangs erwähnt, verstößt das gegen die Integrationsbedinungen und kann nur mit "Dummynutzern" gelöst werden. Die Visualisierung der Abhänigkeiten muss entsprechend auf diese Eigenheiten reagieren können.

2.4.2 Rechteverwaltung

Die weiß hinterlegeten Relationen in Abbildung 2.1 modellieren die Rechteverwaltung der VPD Umsetzung am IPK. Sie speichern eine Vielzahl von Informationen ab. Es wird die Art des Schutzes gespeichert. Hinzu kommen noch Informationen welche Objekte geschützt werden sollen. Der wichtigste und größte Teil bildet die Relation T015_GRANT. Im folgenden werden die einzelnen Relationen benannt und ihre Funktion erläutert.

Die Relation r(T014_GRANT_TYPE) speichert alle Arten des Schutzes. "Bei einem Zugriffsschutz, der auf vier SQL–Operationen wie SELECT, INSERT, UPDATE und DELETE basieren soll,"[9, S.79] sind in der Relation nur diese vier Tupel vorhanden.

Informationen über die zu schützende Tabelle befinden sich in T000_OBJECT_TYPE. Wichtige Attribute sind der Name der Tabelle, der Name des Schema in der sich die Tabelle befindet und der Name des Primärschlüssels. Mit Hilfe des Primärschlüssels kann ermittelt werden, welche Tupel geschützt werden sollen. Nur Tabellen die in T000_OBJECT_TYPE eingetragen sind, stehen auch unter VPD Schutz. Ist dies nicht der Fall, gelten die in Abschnitt 2.2 beschriebenen Mechanismen.

Das zentrale Element der VPD Umsetzung ist die Relation r(T015_GRANT). Sie enthält alle nötigen Informationen für den VPD Schutz. Jeder Eintrag beschreibt die Ausprägung eines Tupels aus der in T000_OBJECT_TYPE eingetragenen Tabelle. Im folgenden werden die wichtigsten Elemente näher erläutert und geben einen ersten Eindruck über die Funktionsweise der VPD.

- T015_T000_ID verweist auf die Relation T000_OBJECT_TYPE und beschreibt, zu welcher Tabelle der Tupel gehört.
- T015_T014_ID verweist auf die Relation r(T014_GRANT_TYPE) und gibt Auskunft, für welche Operation der Schutz aktiv ist.
- T015_T002_ID_GRANTEE verweist auf die Relation T002_SUBJECT. Es enthält die Informationen für welchen Nutzer das vergebene Recht gilt. Der Nutzer "all" nimmt eine Sonderstellung ein. Er steht stellvertretend für alle im VPD eingetragenen Nutzer und Gruppen. Wird ein Recht für diesen User erstellt, so gilt es auch für alle anderen Nutzer im VPD.

- TO15_TO02_ID_GRANTOR verweist ebenfalls auf die Relation TO02_SUBJECT. Der hier eingetragene Nutzer hat das vergebene Recht erteilt. Im allgemeinen ist das die ID des Nutzers "admin". Er verwaltet das VPD und hat eine ähnliche Funktion wie der User "root" unter Linux. Genau wie für "root" gelten für "admin" keine Einschränkungen. D.h. die vergebenen Rechte haben keine Auswirkungen mehr auf die SQL–Abfragen.
- TO15_TNNN_ID enthält den Wert des Primärschlüssels des Tupels der geschützten Tabelle. Zu beachten ist, dass der Primärschlüssel numerisch sein muss. Außerdem gibt es hier zwei Sonderwerte. Wie weiter oben erläutert wurde, wird für jedes geschütze Tupel und jeden Nutzer ein Eintrag in TO15_GRANT angelegt. Bei großen Tabellen und Nutzerzahlen würde das zu sehr vielen Einträgen führen. Um dem entgegen zu wirken, wurden besondere Werte und Nutzer (siehe oben) eingeführt. Nimmt TO15_TNNN_ID den Wert $-1\mathcal{E}37$ an, so gilt das vergebene Recht für alle Tupel der geschützten Tabelle. Der Wert $-1\mathcal{E}36$ steht für das Recht, eine INSERT Operation auszuführen. Dieses Recht kann nur einmal pro Tabelle gegeben oder entzogen werden. Eine Ausprägung für jeden einzelnen Tupel würde keinen Sinn ergeben.

Durch die Vielfalt der Informationen in TO15_GRANT ist eine angemessene Feinheit vorhanden um den VPD Schutz zu erstellen. Die nötigen Mechanismen und das Zusammenspiel von VPD–Technolgie mit den im Schema PDW_SECURE enthaltenen Informationen wird im folgenden Abschnitt erklärt.

2.4.3 Funktionsweise des VPD

Die in Abschnitt 2.3 erläuterten Mechanismen bedienen sich der Informationen aus PDW_SECURE. Bei jeder Anfrage des Nutzers an die Datenbank wird geprüft ob ein Eintrag in der Tabelle TO15_GRANT vorhanden ist und anschließend verarbeitet. Damit die Mechanismen greifen, muss dem DBMS mitgeteilt werden welcher Nutzer jetzt mit der Datenbank verbunden ist. Das DBMS weiß nur welcher Schemanutzer angemeldet ist, aber nichts über den VPD Nutzer. Es gibt einen Unterschied ob Schemanutzer SCOTT nun als Mitarbeiter 1 oder Mitarbeiter 2 arbeitet. Über den Applikationskontext wird der VPD Nutzer mitgeteilt. Dazu muss nach der Anmeldung an das DBMS der Applikationskontext für den Nutzer gesetzt werden. Meldet sich der Nutzer über das Oracle Internet Directory (OID) an, so wird der Kontext automatisch gesetzt. Erfolgt die Anmeldung nicht über das OID, so muss der Applikationskontext vom Nutzer selbst mitgeteilt werden. Dazu genügt ein Aufruf der Funktion PDW_SECURE.check_Password('Nutzer','Passwort'). Wird nicht mitgeteilt welcher

Nutzer mit dem DBMS arbeitet, gelten die vergebenen Standardrechte. Sie sperren zunächst jeglichen Zugriff auf die Tabellen des Schemas. In den meisten Fällen wird dann eine beliebige SQL–Anfrage im Schemakontext eine leere Ergebnismenge zurückgeben.

Kapitel 3

WARM

3.1 Einführung

WARM steht für "WorkAssistent for Rights Management" und ist eine graphische Oberfläche zur Administration von Datenbankzugriffsrechten. Die ursprüngliche Verwaltung mit SQL–Scripten genügte nicht mehr den Ansprüchen des IPK. Der Wunsch nach einer intuitiv bedienbaren Software war der Grund für die Entstehung von WARM.

3.2 Auswahl der Programmiersprache

Für die Auswahl der Programmiersprache waren zwei Bedingungen ausschlaggebend. WARM sollte plattformunabhänig und dezentral sein. Unter diesen Bedingungen wurde Java als Programmiersprache gewählt. Zudem bietet Java die folgenden Vorteile:

1. Java bietet ausgereifte GUI–Toolkits.
2. Die Java Web Start Technologie ermöglicht eine einfache Installation von WARM.
3. Der Datanbankzugriff über die JDBC Treiber ist ohne Installation möglich.

3.3 Features

WARM bietet folgende Möglichkeiten für die Bearbeitung der Rechteverwaltung:

- Darstellen der Nutzerstruktur des VPD
- Veränderung der Abhänigkeiten in der Nutzerstruktur
- Anlegen neuer Nutzer und Gruppen
- Bearbeiten von Nutzer– und Gruppendetails
- Darstellen der Rechte für eine Tabelle
- Ändern, löschen und neu vergeben von Rechten für eine Tabelle
- Entfernen und hinzufügen von Tabellen zum VPD
- Seitenorientiertes Blättern durch große Tabellen
- Filteroperationen auf einzelnen Spalten
- Ausblenden von nicht literalen Datentypen wie BLOB, CLOB, usw

3.4 Installation mit Java Web Start

Java Web Start wurde von Sun Microsystems entwickelt, um Java Applikationen mit nur einem Klick über ein Netzwerk zu starten. “Im Unterschied zu Java–Applets benötigen Web Start Anwendungen keinen Browser“[6] zum Ausführen. Bei jedem Aufruf eines Web Start Programmes wird überprüft, ob auf dem System die aktuelle Version vorhanden ist. Ist dies nicht der Fall, wird die ältere Version durch die neue ersetzt. Dadurch ist es möglich, immer mit der neusten Programmversion zu arbeiten. Das gilt sowohl für das Programm selber als auch für die benötigten zusätzlichen Pakete, wie zum Beispiel JDBC Treiber. Dem Nutzer wird der Aufwand erspart, die Software zu pflegen.

Vorraussetzung für die Java Web Start Technologie ist, dass

- die Anwendung auf einem Server zum Download bereit steht und sie durch eine JNLP Datei beschrieben wird,
- der Server die JNLP Datei richtig interpretiert (Dazu muss er den MIME–Type `application/x-java-jnlp-file` kennen) und
- der Anwender eine JRE und Java Web Start installiert hat.

Ein Beispiel für eine JNLP Datei ist im Folgenden Listing 3.1 zu sehen.

```
<jnlp spec="1.0+" codebase="http://www.ipk-gatersleben.de">
    <information>
        <title>WARM</title>
        <vendor>IPK Gatersleben</vendor>
        <homepage href="http://www.ipk-gatersleben.de"/>
        <icon href="Icons/ipk_logo.jpg"/>
        <offline-allowed/>
    </information>
    <security>
        <all-permissions/>
    </security>
    <resources>
        <!-- Welche JRE muß vorhanden sein-->
        <j2se version="1.4+"/>
        <jar href="WARM.jar"/>
        <!-- Welche Pakete sollen noch geladen werden-->
        <jar href="log4j-1.2.9.jar"/>
        <jar href="ojdbc14.jar"/>
    </resources>
    <application-desc main-class="de.ipk.agbi.warm.main.Start"/>
</jnlp>
```

Listing 3.1: JNLP Beschreibung von WARM

3.5 Aufbau der grafischen Oberfläche

Einen ersten Eindruck von WARM verschafft die Abbildung 3.1 auf der nächsten Seite. Das Programm gliedert sich in drei Bereiche auf: die Nutzer–, die Rechte– und die Tabellenverwaltung. Das Fenster “User and DB–Schemes“ visualisiert die Nutzerverwaltung und listet die vorhandenen Schemanutzer der Datenbank auf.

“Visual Table Rights“ ist die graphische Darstellung des Inhalts der in “Tables“ ausgewählten Tabelle mit den in `T015_GRANT` vorhandenen Rechten. Zu sehen ist das in Abbildung 3.2(a) für ein Tabellenrecht und in 3.2(b) für ein Tupelrecht.

Die Idee, die Rechte dort zu erteilen wo sie auch wirken, beruht auf Erfahrungen mit ähnlichen Systemen, wie den Dateirechten bei UNIX oder Windows. Außerdem bietet es die verlangte intuitive Bediehnbarkeit durch das direkte Arbeiten mit den zu schützenden Tabellen. Für die bessere Lesbarkeit von großen Tabelleninhalten, ist ein seitenorientiertes Blättern (Paging) möglich. Dazu werden die Navigationspfeile im unteren Teil vom Fenster “Visual Table Rights“ verwendet.

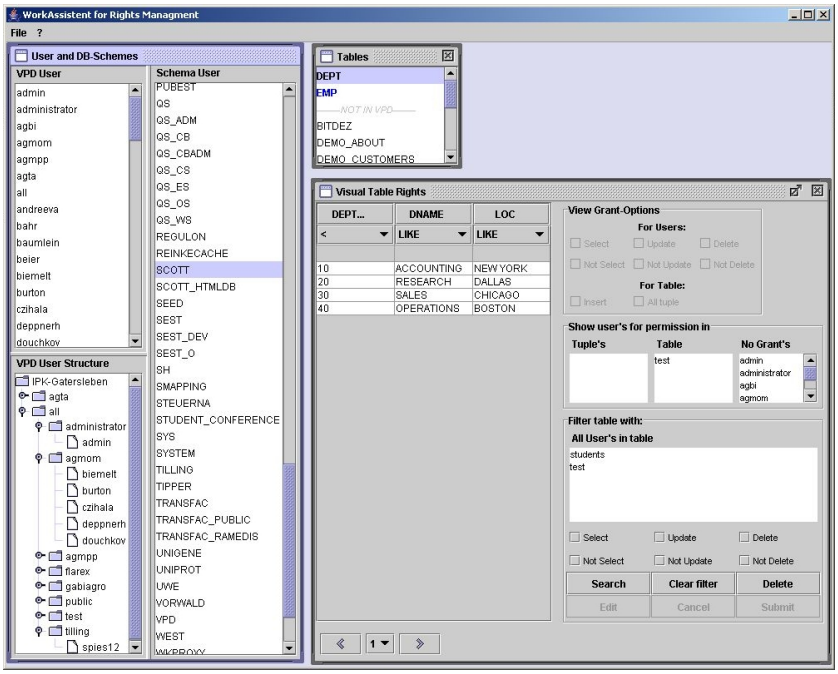
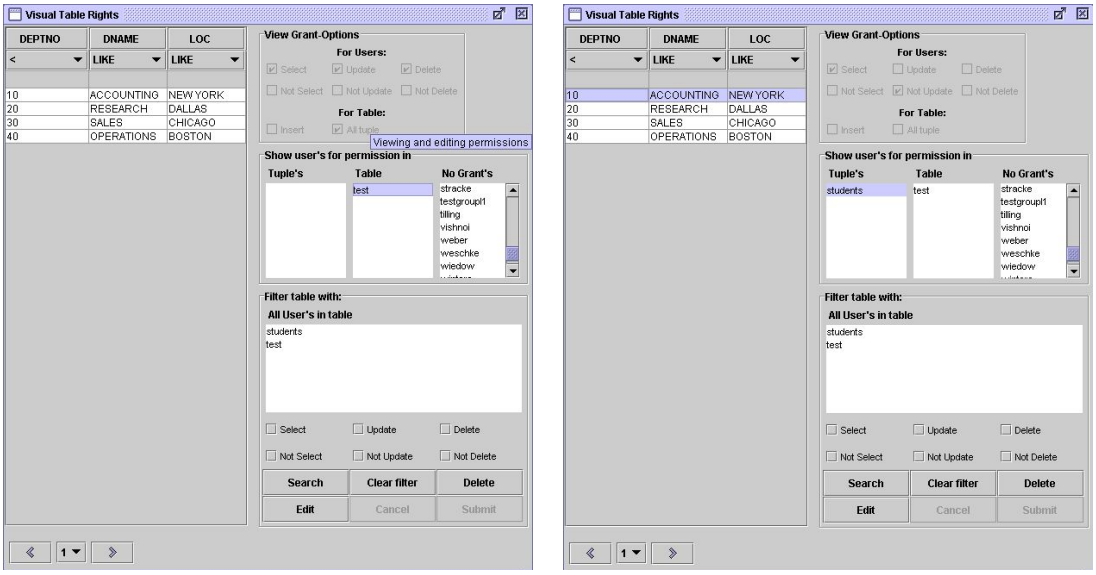


Abbildung 3.1: Screenshot der WARM Anwendung



(a) Tabellenrecht (b) Tupelrecht

Abbildung 3.2: Darstellung von VPD Rechten in WARM

3.6 Quellcode im Detail

Zur besseren Lesbarkeit und Strukturierung ist WARM vollständig in Pakete aufgeteilt. Die Pakete gliedern sich in Datenbankzugriff, GUI und Eventhandling - entsprechend dem MVC–Muster. Im Folgenden werden die einzelnen Programmteile näher beleuchtet.

3.6.1 WARM – Model

Folgende Pakete gehören dazu:

- de.ipk.agbi.warm.db.*
- de.ipk.agbi.warm.properties.*

Das Model von WARM teilt sich in zwei Klassen auf: *DBObject.java* und *Model.java*. Beide befinden sich im Paket **de.ipk.agbi.warm.db.***. *Model.java* stellt Methoden zur Datenmanipulation und zur Datenbeschaffung bereit. Die Klasse *DBObject.java* verwaltet die Datenbankverbindung und beinhaltet Methoden für das Absenden von SQL–Befehlen. Die Idee ist, dass der Datenbankzugriff soweit wie möglich abstrahiert wird. Das eigentliche Model soll sich nicht damit beschäftigen wie eine Anfrage an die Datenbank zu richten ist. Es stellt nur die SQL–Abfrage zusammen und übergibt sie *DBObject.java*. Dieses kümmert sich um eine Verbindung zur Datenbank und setzt den übergebenen Befehl ab. Als Rückgabewert stehen mehrere Möglichkeiten zur Verfügung. Es kann die Menge der verarbeiteten Zeilen oder das Ergebnis in Form eines **Java ResultSet** zurück gegeben werden. *DBObject.java* übernimmt außerdem die Fehlerbehandlung.

Tritt ein Fehler bei der Anfrage an das DBMS auf, wird ein Fehlercode generiert und an *Model.java* zurückgegeben. Listing 3.2 gibt einen Einblick.

```
public int queryUpdateDB(String q) {
    Statement stmt1 = null;
    int b = 0;
    try {
        if (stmt1 == null)
            stmt1 = con.createStatement();
        stmt1.executeUpdate(q);
        stmt1.close();
    } catch (SQLException e) {
        // GENERIERE FEHLERCODE
        return -1;
    }
    return b;
}
```

Listing 3.2: Erzeugen eines Fehlercodes

Eine weitere Möglickeit zur Interaktion mit der Datenbank sind vorgefertigte SQL–Abfragen. Sie stehen in einer Konfigurationsdatei (**Properties– Datei**) und können bei Bedarf geholt werden. Alle nötigen Anfragen zur Bearbeitung der Rechteverwaltung stehen in der Datei *db.properties*. Durch die Properties–Datei kann WARM flexibler auf Änderungen reagieren. Die SQL–Abfragen können verändert werden ohne in den Quellcode gehen zu müssen und damit ein erneutes Übersetzen zu erzwingen. Der allgemeine Aufbau ist eine **Schlüssel = Wert** Zuweisung. Ein Auszug ist in Listing 3.3 zu sehen.

```
#sql-statements
### For User List
userList    = Select t002_login, t002_id from t002_subject order by t002_login
### For Scheme List
dBSchemeList= Select username from ALL_Users order by username
### For Tree View
treeUser    = Select t002_id, t002_login from t002_subject
### Delete Statements
deleteUser  = Delete from t002_subject where t002_login = 'UNAME'
```

Listing 3.3: Konfigurationsdatei für SQL–Abfragen

Die Klasse *Model.java* bildet die Informationen im VPD Schema ab. Dazu werden die Daten über *DBObject* angefordert. Bei der anschliessenden Aufbereitung entstehen Java–Objekte. Die Klasse *TreeData.java* repräsentiert zum Beispiel die strukturelle Abhänigkeit der Nutzer zu Gruppen und umgekehrt. Dazu speichert *Model.java* das Ergebnis einer **Connect–By–Prior–Abfrage** in einem zweidimensionalen Stringarray ab. Die Connect–by–Prior–Abfrage (Listing 3.4) wird an die Tabelle `T003_SUBJECT_TREE` gestellt und findet mittels rekursiven `SELECT` Anfragen einen Abhänigkeitsgraphen. Die Ausgabe ist eine Tabelle, die in einzelne Level unterteilt ist. Der Begriff Level kann ähnlich den einzelnen Schritten einer BFS Suche auf dem Abhänigkeitsgraphen verstanden werden. Für weitere Informationen siehe [4].

```
SELECT distinct level, t003_t002_id_parent ,t003_t002_id_child
FROM t003_subject_tree
START WITH t003_t002_id_parent NOT IN
    (
    SELECT t003_t002_id_child
    FROM t003_subject_tree
    )
CONNECT BY PRIOR t003_t002_id_child = t003_t002_id_parent
ORDER BY level, t003_t002_id_parent
```

Listing 3.4: Connect–By–Prior

Ein weiteres Beipiel ist die Klasse *RightTableData.java*. Sie speichert alle nötigen Informationen für die Darstellung der Tabellenrechte ab. Den Zugang zu den Daten bilden

Getter und **Setter** Methoden. Die einzelnen Rechte werden in **Hashtabellen** gespeichert. Als Schlüssel dient entweder der Wert des Primärschlüssel bei Tupelrechten oder der Nutzername bei Tabellenrechten. Weitere Getter und Setter Methoden gibt es für

- die Filterkriterien der einzelnen Spalten.
- die Variablen des Paging.
- den Tabellennamen.
- den Tabelleninhalt.
- die Information welche Spalten einen nicht literalen Datentyp besitzen.

Die im Abschnitt 3.5 gezeigte Darstellungsweise der Tabellenrechte verlangt zum einen den Zugriff auf das ausgewählte Datenbankschema mit seinen Tabellen und zum anderen den Zugriff auf das Rechteschema `PDW_SECURE`. D.h. es sind immer zwei Datenbankverbindungen nötig, weil aus Sicherheitsgründen das Rechteschema von anderen Schemata abgegrenzt ist und umgekehrt. Aus diesem Grünt erzeugt *Model.java* zwei Objekte vom Typ *DBObject.java*. Das eine wird mit dem ausgewählten Schemanutzer initialisiert und das Andere mit den Verbindungsparametern für `PDW_SECURE`. Außerdem wird der Applikationskontext der Schemanutzerverbindung auf den VPD–User “admin“ gesetzt.

Die Klasse *Model.java* übernimmt zusätzlich zur Datenaufbereitung auch die Datenmanipulation. Dafür besitzt sie Methoden, die Veränderungen in `PDW_SECURE` vornehmen. Listing 3.5 zeigt einige Beispiele.

```
public int deleteEntryInVPD(String table, String grantee) {...}
public int insertEntryInVPD(String table, String[] pk, String option,
           String grantee, String grantor) {...}
public int deleteUserFromVPD(String uName) {...}
public int updateEntryInVPD(String table, String[] pk, String optnew,
           String optold, String grantee, String grantor) {...}
```

Listing 3.5: Manipulation der VPD Informationen

3.6.2 WARM – Controller

Folgende Pakete gehören dazu:

- de.ipk.agbi.warm.controller.*

- de.ipk.agbi.warm.handler.*

Der WARM–Controller ist in zwei Bereiche geteilt. Zum einen haben wir die zentrale Klasse *Controller.java* und zum anderen haben wir die verschiedenen Handler. Es gibt sie für jedes größere Steuerungselement (Listen, Kontextmenüs, Schalter, Auswahlboxen, usw). Sie nehmen Benutzereingaben wie Mausklicks oder Tastatureingaben entgegen und verarbeiten sie. Anschließend starten sie den gewünschten Prozess in WARM. Ein Prozess ist ein Methodenaufruf in der Klasse *Controller.java*. Dieser kann eine ganze Reihe von Verarbeitungschritten enthalten. Außerdem reagieret *Controller.java* auf die Fehlercodes aus dem Model. Zu sehen ist das in Listing 3.6. Hier wird eine Tabelle aus dem VPD entfernt. Sollte dies nicht möglich sein, liefert das Model mit

```
int e = refModel.removeTableFromVPD(table);
```

einen numerischen Fehlercode als Rückgabewert. Diese Zahl ist für den Anwender aber wenig informativ, also übersetzt es der Controller in eine verständlichere Fehlermeldung.

```
public void removeFromVPD(String table) {
    int control;
    control = JOptionPane.showConfirmDialog(refView.getMain(),
            "Do you really want remove? :" + table, "Confirm delete",
            JOptionPane.YES_NO_OPTION);
    int c2 = JOptionPane.showConfirmDialog(refView.getMain(),
            "Do want to remove triggers too? ", "Confirm delete",
            JOptionPane.YES_NO_OPTION);
    //wenn Sicherheitsabfrage bestätigt
    if (control == 0) {
        //führe Löschung durch
        int e = refModel.removeTableFromVPD(table);
        //falls Fehler
        if (e < 0) {
            //Zeigt einen Dialog als Fehlermeldung
            showError("Can't delete table from VPD!");
        } else{
            if(c2 == 0){
                 // lösche jetzt die Trigger
                 e = refModel.removeDefaultTrigger(table);
                 //falls Fehler
                 if(e == -1){
                     //Zeigt einen Dialog als Fehlermeldung
                     showError("Problem: Can't delete trigger!");
                 }
            }
        }
        fillTables();
    }
}
```

Listing 3.6: Beantwortung eines Fehlercodes

3.6.3 WARM – View

Folgende Pakete gehören dazu:

- de.ipk.agbi.warm.gui.*
- de.ipk.agbi.warm.gui.dialog.*
- de.ipk.agbi.warm.rightTable.*

WARM wurde mit dem GUI–Toolkit Java–Swing erstellt. Swing bietet gegenüber AWT eine bessere Designmöglichkeit und ein verbessertes Eventhandling. Die Handhabung ist sehr flexibel und es lässt sich leicht an eigene Bedürfnisse anpassen. Außerdem stehen eine Vielzahl von Komponenten zur Verfügung [5].

Aufbauend auf der Idee der Modularität ist jede GUI–Komponente von WARM als extra Klasse entstanden. Diese Klassen bieten ein auf WARM abgestimmtes Layout. Zum Beispiel sind alle Komponenten in eine `JScrollPane` gebettet und leiten sich von `JPanel` ab. Außerdem werden Font und Schriftgröße einheitlich gesetzt.

Die Klasse *View.java* fügt alle diese Komponenten zu einer grafischen Oberfläche zusammen. Das Erscheinungsbild aus Abbildung 3.1 auf Seite 15 ist nur ein Muster. Durch den modularen Aufbau kann das Layout beliebig verändert werden. Dazu muss lediglich der Quellcode der Methode

```
public void Layout()
```

verändert werden.

Das Paket **de.ipk.agbi.warm.rightTable** enthält alle Klassen die das Aussehen der Rechtetabelle bestimmen können. Es gibt dort speziell auf WARM angepaßte Celleditoren und –renderer. Sie können Spalten ausgrauen und Eingabe– und Auswahlfelder für den Tabelleninhaltsfilter erzeugen.

3.7 Probleme

Das verwendete MVC–Design bringt auch Probleme mit sich. Die starke Modularisierung und strikte Trennung der einzelnen Bereiche bewirkt einen großen Anstieg der Kommunikation zwischen den Komponenten. Das erhöht den programmiertechnischen Aufwand. Es muß deshalb entschieden werden, ob es für ein gegebenes Problem notwendig ist, auf diesen Modellansatz zu bauen. Es müssen Aufwand der Kommunikationsverarbeitung

und der Gewinn an Weiterverwendbarkeit und Erweiterbarkeit gegen einander abgewogen werden. Die Entscheidung bei großen Projekten wie WARM geht allerdings klar zugunsten des MVC–Modells.

Das große Problem bei der Rechtedarstellung sind die auftretenden Datenmengen. Dadurch kann es bei hoher Netzwerklast zu Wartezeiten kommen. Außerdem ist es nicht hilfreich, alle Daten auf einmal zu sehen. Deswegen wurde das Paging eingesetzt, was jedoch nur zum Teil eine Verbesserung gebracht hat. Die darzustellenden Größen der Tabellen waren im allgemeinen ein Problem. So dauert das Sortieren trotz Paging noch recht lange wenn die Tabellen mehrere tausend Einträge haben. Die Sortierung wird im Moment noch Datenbank–gestützt durchgeführt.

3.8 Ausblick

Zukünftige Versionen von WARM werden Verbesserungen der Geschwindigkeit des Sortierens bei großen Datenmengen und erweiterte Suchfunktionen enthalten. Die Sortierung des Tabelleninhalts wird dann auf Java basieren. Zudem soll es möglich sein, einen Nutzer auszuwählen und die zugehörigen Rechte über alle Tabellen und Datenbankschemata darzustellen. Außerdem wären erweiterte Sortierfunktionen, nach einzeln vergebenen Rechten hilfreich.

Kapitel 4

Anwendungsfälle

Dieses Kapitel ist eine Hilfestellung im Umgang mit WARM. Es werden typische Aufgaben gestellt und deren Lösung "Schritt–für–Schritt" erklärt.

4.1 Benutzerverwaltung

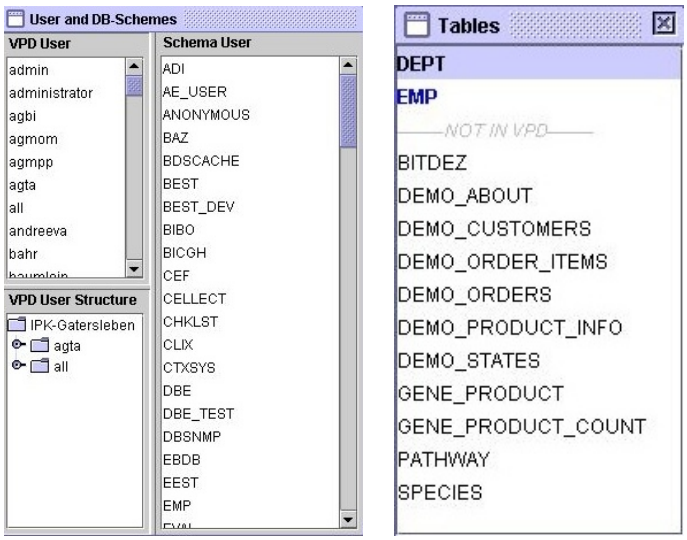
Alle nachfolgenden Anleitungen sind auch für Gruppen und andere Organisationseinheiten gültig. Der erste Schritt ist immer, WARM zu starten und das Passwort einzugeben. Danach sollte WARM sich ähnlich Abbildung 3.1 auf Seite 15 präsentieren. Abbildung 4.1(a) auf der nächsten Seite zeigt den für die Nutzerverwaltung relevanten Teil.

4.1.1 Fall 1: Wie erstelle, ändere oder lösche ich einen Nutzer?

Durch einen Rechtsklick auf die Liste mit der Überschrift "VPD User" öffnet sich ein Menü. Aus diesem kann die gewünschte Aktion ausgewählt werden.

4.1.2 Fall 2: Wie ändere ich den Nutzerbaum?

Auch hier genügt ein Rechtsklick auf den entsprechenden Eintrag im Nutzerbaum um diesen zu bearbeiten. Jedoch gibt es dabei ein paar Dinge zu beachten. Sie können nur eine neue Gruppe hinzufügen, wenn diese auch schon Mitglieder hat. Eine leere Gruppe hinzuzufügen ist nicht möglich. Dann ist noch zu erwähnen, dass der Menüeintrag "Add



(a) Nutzerverwaltung (b) Tabellenverwaltung

Abbildung 4.1: GUI Elemente

Existing Group“ nur Gruppen meint, die auch schon im Baum vorhanden sind. Also nicht eventuell neu angelegte Gruppen die noch keine Mitglieder haben.

4.2 Rechteverwaltung

Starten sie WARM und geben sie das Passwort ein. Damit sie die vergebenen Rechte bearbeiten können, sollten sie wissen in welchem Schema die zu bearbeitende Tabelle vorhanden ist. Wenn sie das wissen, melden sie sich bitte an diesem Schema an. Dazu klicken sie doppelt auf den entsprechenden Eintrag in der Liste “Schemes“. Diese befindet sich im gleichen Fenster wie die Liste der VPD–Nutzer (siehe Abbildung 4.1(a)).

4.2.1 Fall 1: Wie füge ich eine Tabelle zum VPD hinzu?

Sind die obrigen Schritte getan, öffnet sich ein weiteres Fenster (Abbildung 4.1(b)). Hier werden alle Tabellen angezeigt die zum ausgewählten Schema gehören. Die blau eingefärbten Einträge stehen schon unter VPD–Schutz, die restlichen nicht.

Möchten sie eine Tabelle unter VPD–Schutz stellen, klicken sie mit der rechten Maustaste auf den entsprechenden Eintrag. Es öffnet sich ein Menü und sie können “Add to

VPD“ auswählen. Daraufhin erscheint ein Dialogfenster. Sie können nun die Werte ihren Wünschen anpassen oder die Standardwerte übernehmen. Mit dem Klick auf “Submit“ schließen sie den Vorgang ab. Danach sollte es einen neuen “blauen“ Eintrag geben.

Wollen sie den umgekehrten Weg gehen und eine Tabelle aus dem VPD–Schutz herausnehmen, klicken sie wieder mit der rechten Maustaste auf den gewünschten Eintrag und wählen “Remove from VPD“. Es erscheint eine Sicherheitsabfrage. Wenn sie sich ihrer Sache sicher sind, können sie diese mit “Yes“ beantworten. Bitte beachten sie, dass sie damit auch alle erteilten Rechte für diese Tabelle ebenfalls löschen.

4.2.2 Fall 1: Wie vergebe ich ein Recht?

Zuerst wählen sie sich die gewünschte Tabelle aus der Liste aus. Mit einem Doppelklick öffnen sie sie. Nach einer kurzen Wartezeit öffnet sich ein Fenster. Es ähnelt dem in Abbildung 3.2(a) auf Seite 15. Auf der linken Seite sehen sie den Inhalt der Tabelle. Darunter befindt sich die Navigationsleiste. Mit dieser können sie sich die Tabelle Stück für Stück ansehen. Es werden jeweils 100 Tupel der Tabelle angezeigt. Der rechte Teil des Fensters sind Kontrollelemente zur Rechtevergabe.

Möchten sie einem Nutzer ein Recht für alle Tupel der Tabelle geben, wählen sie ihn aus der Liste “No Grant's“ aus. Sollte er dort nicht vorhanden sein, so schauen sie in der Liste “All User's in Table“ nach. Die anderen Listen sind für sie im Moment nicht von Belang. Wenn sie den Nutzer gefunden haben, wählen sie ihn durch einen Klick aus. Danach sollte ein Klick auf den Button “New“ die oberen Auswahlboxen aktivieren. Suchen sie die gewünschte Operation heraus und achten sie darauf, auch die Auswahlbox “all tuple“ zu aktivieren. Sie bestimmt, dass die eben ausgewählten Operationen für alle Tupel gelten. Zum Abschluß klicken sie auf den Button “Submit“. Damit bestätigen sie die Änderungen und weisen WARM an, diese zu speichern. Zur Kontrolle können sie auf den oben ausgewählten Nutzer in der Liste “Table“ klicken. Er sollte die eben erteilten Rechte haben.

Wenn sie Rechte nur für ein oder mehrere Tupel vergeben möchten, wählen sie die entsprechenden Einträge aus der Tabelle aus. Es folgt nun die gleiche Vorgehensweise wie eben erläutert, mit dem Unterschied das sie die Auswahlbox “all tuple“ nicht aktivieren.

4.2.3 Fall 2: Wie kann ich ein Recht verändern?

Suchen sie das betreffende Recht heraus. Für Tupelrechte klicken sie auf den entsprechenden Eintrag in der Tabelle und dann auf den Nutzer in der Liste “Tuple's“. Sie

sehen nun in “View Grant Options“ die aktuellen Rechte. Ein Klick auf “Edit“ aktiviert die Auswahlboxen und sie können ihre Änderungen vornehmen. Bestätigen sie sie mit “Submit“.

4.2.4 Fall 3: Wie kann ich ein Recht löschen?

Hier gilt die gleiche Ablauffolge wie in Fall 3. Nur klicken sie nicht auf “Edit“ sondern auf “Delete“.

4.2.5 Fall 4: Wie kann ich ein Recht suchen?

Möchten sie ein bestimmtes Recht in den angezeigten Tupeln suchen, klicken sie auf den gewünschten User in der Liste “All User's in table“. Die darunter aufgelisteten Operationen stehen ihnen als Suchkriterien zur Verfügung. Ein Klick auf “Search“ markiert die gefundenen Einträge in den angezeigten Tupeln.

Literaturverzeichnis

[1] Donald Burleson. Establish security policy with Oracle virtual private database. `http://builder.com.com/5100-6388_14-5062064.html`, 2003.

[2] Pete Finnigan. Oracle Row Level Security: Part 1. `http://www.securityfocus.com/infocus/1743`, 2003.

[3] Michael Lauer. *Python und GUI-Toolkits*. mitp, 2002.

[4] Kevin Loney. *Oracle database 10g : the complete reference*. Oracle Press. McGraw-Hill Education, 2004.

[5] Sun Microsystems. Creating a GUI with JFC/Swing. `http://java.sun.com/docs/books/tutorial/uiswing/index.html`, 1995-2005.

[6] Sun Microsystems. Java Web Start Technology. `http://java.sun.com/products/javawebstart/`, 2005.

[7] Model View Controller. `http://de.wikipedia.org/wiki/MVC`, 2005.

[8] Trygve M. H. Reenskaug. The original MVC. `http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html`, 1979-2003.

[9] Alexander Seitz. Konzept und Entwurf eines Vorgehensmodells zur Entwicklung von anwenderspezifischen Datenbanklösungen im Bereich Bioinformatik. Master's thesis, Universität Magdeburg, 2005.