Projektarbeit

Thema:

Konzeption und Implementation einer
plattformunabhängigen Importsoftware
für ORACLE Datenbanken

Betreuer: Dr. Uwe Scholz
Dipl.-Inf. Matthias Lange
Prof. Dr. Hardy Pundt

vorgelegt von: Thomas Rutkowski
Studiengang: pII
MatrikelNr.: 7553
E-Mail: uploader@arcor.de

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

Abkürzungen

CSV	Comma separated values (Synonym für ein Dateiformat)
DB	Datenbank
DBMS	Datenbank Managementsystem
DBS	Datenbanksystem
DML	Data Manipulation Language
ERD	Entity Relationship Diagram
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protokoll
IPK	Institut für Pflanzengenetik und Kulturpflanzenforschung
JDBC	Java Database Connectivity
JDK	Java Development Kit
JNLP	Java Network Launching Protocol
JRE	Java Runtime Enviroment
MS	Microsoft
SQL	Structured Query Language
UML	Unified Modeling Language

1 Problemstellung und Ausgangssituation

Im Rahmen von Experimenten fallen am IPK große Datenmengen an, die zur späteren Auswertung in Datenbanken gespeichert werden. Ein Grossteil dieser Daten wird durch spezielle Auswertungssoftware generiert und liegt in Form von MS-Excel® - Dateien vor. Teilweise existieren auch CSV-Dateien. Es wird nun eine Lösung gesucht, um diese Daten in eine Datenbank zu importieren.

Eine schnelle und einfache Lösung dieses Problems wäre ein Programm zu schreiben, dass eine Datei mit einer zuvor definierten Struktur einliest und die Daten in festgelegte Tabellen schreibt. Allerdings existieren verschiedene Quelldateien mit unterschiedlichem Inhalt. So enthalten z.B. einige Dateien Informationen über die Platzierung von Proben auf Trägerelementen, während andere Dateien Information über verwendete Materialien enthalten. Es würden folglich viele kleine Programme entstehen, die alle annähernd das Gleiche machen, aber nur auf einen bestimmten Quelldateityp zugeschnitten sind.
Oracle stellt mit dem SQL-Loader ein Programm zu Verfügung mit dem der Import von großen Datenmengen durchgeführt werden könnte. Eine solche Lösung scheidet jedoch von vornherein aus, da der Import auch von Mitarbeitern ohne Datenbankkenntnisse vorgenommen werden soll. Diese Mitarbeiter haben folglich keine Kenntnisse im Umgang mit Datenbanktools. Daraus resultierend kann auch keine speziell (Datenbank)Software vorausgesetzt werden.

Es wird daher ein Werkzeug benötigt, das vielfältig konfigurierbar ist, um sich so den unterschiedlichen Inhalten der Quelldateien anzupassen. Dazu gehört z.B.

- Zugriff auf Sequenzen
- Verteilen der Daten über mehrere Tabellen
- Behandlung von Datumsangaben
- Verwendung von Abfragen innerhalb der insert-Anweisungen zum sichern der Relationen

Andererseits muss beachtet werden, dass es für die Endanwender einfach und ohne Datenbankkenntnisse zu bedienen ist.

2 Grundlagen Datenbanken

Für ein besseres Verständnis der nachfolgenden Kapitel sollen in diesem Abschnitt einige Grundbegriffe zum Thema Datenbanken erläutert werden.

Grundstruktur

Eine Datenbank bietet die Möglichkeit große Datenmengen strukturiert abzulegen bzw. zu verwalten. Zur Verwaltung seiner Daten, die im Zusammenhang mit genetischen Experimenten anfallen, setzt das IPK das Datenbanksystem ORACLE ein. Jede Datenbank fasst Daten eines großen Komplexes zusammen – So werden z.B. in der FLAREX – Datenbank Informationen zu Hybridisierungsexperimenten gespeichert. Ein Anwender arbeitet i. d. R. nur mit einer Datenbank (siehe hierzu Abbildung 1).

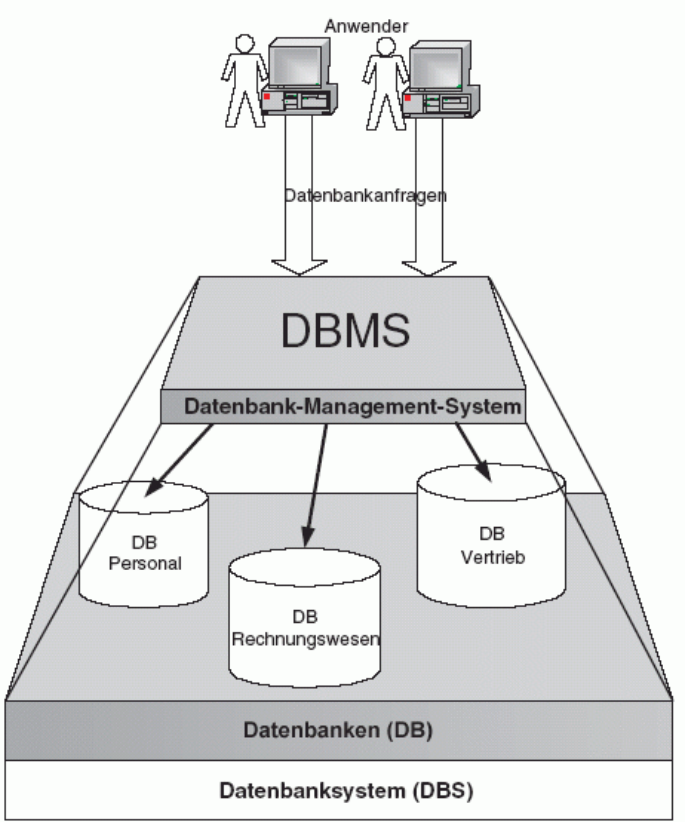

Abbildung 1 DBMS, DB und DBS [Cordts02]

Eine relationale Datenbank ist aus Tabellen aufgebaut. Jede Tabelle fasst bestimmte Informationen zusammen – Z. B. alle Adressen (Name, Vorname, Strasse …) von Zulieferern. Jeder Tabelle sind deshalb Spalten mit einem bestimmten Datentyp zugeordnet. So existieren beispielsweise spezielle Datentypen zum Speichern von Zeichenketten, Zahlen oder Datumsinformationen.

Die einzelnen Datensätze der Tabellen lassen sich nun untereinander in Beziehung setzen. Somit läst sich z.B. festlegen, dass eine Sonde genau einen Zulieferer haben muss und ein Zulieferer beliebig viele Sonden liefern kann (Vgl. Beispieldatenbank S. 36).
Eine der wichtigsten Aufgaben ist es dabei, die Daten widerspruchslos zu verwalten. Um dies zu sichern können CONSTRAINTS festgelegt werden. Damit lässt sich z.B. festlegen, dass eine Tabelle eine Spalte (oder eine Kombination aus mehreren) haben muss, die den gesamten Datensatz eindeutig identifiziert. Diese Spalte wird als Primärschlüssel bezeichnet. Spalten, die mit ihrem Wert auf eine andere Tabelle verweisen, werden als FOREIGN KEY gekennzeichnet. Dadurch wird die referentielle Integrität sichergestellt. Das heißt, das DBS kontrolliert, ob der Datensatz, auf den verwiesen wird, auch existiert. Außerdem können Daten, auf die verwiesen wird, nicht versehentlich gelöscht werden.

SQL

Um auf die Daten zu greifen zu können existiert die Sprache SQL. Diese wird in die Teile DML, DDL und Abfragen unterteilt. Während die *Data Definition Language* z.B. zum erstellen der Tabellen dient, können mit der *Data Manipulation Language* Daten eingefügt, gelöscht und geändert werden. Mit Hilfe der Abfragen können Daten wieder ausgelesen werden. So liefert die Abfrage aus CodeBsp 1 z.B. die Adresse des Lieferanten ‚Meier'. Mit Hilfe von CodeBsp 2 kann ein neuer Datensatz eingefügt werden (Siehe Beispieldatenbank S. 36).

```
SELECT FROM Providers WHERE Name = 'Meier'
```

CodeBsp 1 Select – Anweisung

```
INSERT INTO Libraries (Lib_ID, Name, Describtion) VALUES
(1, 'GAN', NULL)
```

CodeBsp 2 Insert – Anweisung

Commit und Rollback

Beim Einfügen vieler Datensätzen entsteht unter Umständen das Problem, dass erst nach dem Einfügen Mehrer festgestellt wird, dass alle Änderungen verworfen werden müssen. In diesem Fall muss die Transaktion, die alle Änderungen zusammenfasst, zurückgenommen werden.
"Gemäß dem SQL:1999 Standard beginnt eine Transaktion immer automatisch mit der

ersten SQL-Anweisung. … Beendet wird eine Transaktion durch die SQL-Anweisung ROLLBACK oder COMMIT. Die SQL-Anweisung ROLLBACK stellt den Ursprungszustand vor Beginn der Transaktion wieder her. Commit dagegen beendet eine Transaktion erfolgreich und schreibt die Änderungen unwiderruflich in die Datenbank.“ (siehe [Cordts02] S. 184).

Sequenzen

Sequenzen dienen zum Erzeugen von Zahlenfolgen. Es gibt verschiene Möglichkeiten eine Sequenz zu definieren. So kann die Zahlenfolge z.B. stets um einen konstanten Wert Ansteigen oder sich, nach erreichen eines Maximalwertes, wiederholen.
Sequenzen können eingesetzt werden um einen eindeutigen Primärschlüssel zu generieren.

Zu ausführlichen Erklärungen zum Thema Datenbankgrundlagen wird auf einsprechende Literatur verwiesen, z. B. [Cordts02].

3 Anforderungen an die Anwendung

Programmstruktur

Wie unter 1. erläutert muss das Programm zwar flexibel konfigurierbar, gleichzeitig aber einfach zur bedienen sein. Um dies zu gewährleisten, soll die Anwendung aus zwei Teilen bestehen. Der erste Teil dient zur Erstellungen einer Konfiguration für den Datenimport. Daher ist dieser Teil nur für Mitarbeiter, die genaue Kenntnisse über das entsprechende Datenbankschema besitzen anwendbar. Der zweite Teil dient dem eigentlichen Import der Daten. Er soll im wesentlich zwei Optionen bieten: Erstens das Laden der zuvor erstellte Konfiguration, zweitens das Auswählen der Datei mit den Daten, die in die Datenbank geladen werden sollen. Die folgenden UML UseCase - Diagramme sollen diesen Zusammenhang graphisch darstellen. (Allgemeine Informationen zum Thema UML siehe z.B. [Oestereich05])

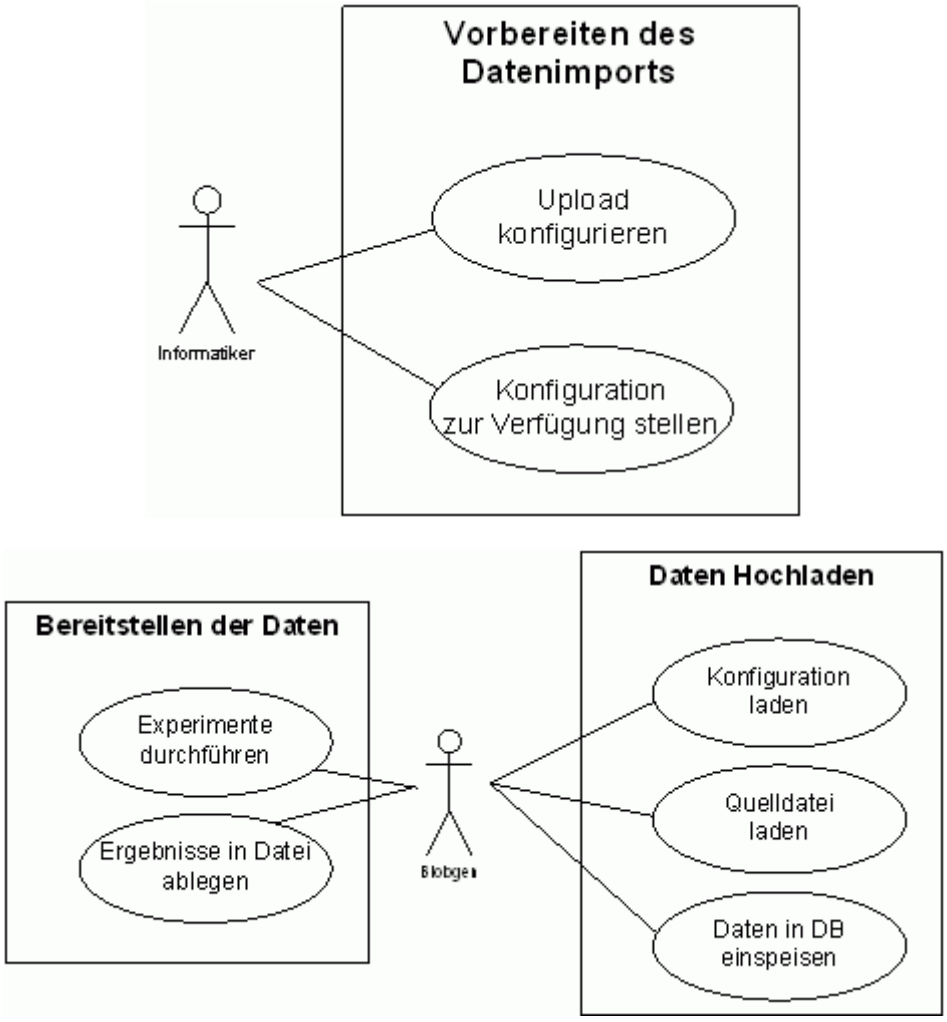

Abbildung 2 Anwendungsfalldiagramme

Allgemeines Funktionsprinzip

Jeder Spalte in der Quelldatei kann letztlich einer Spalte in einer Datenbanktabelle zugeordnet werden. Diese Zuordnung ist die wesentliche Aufgabe, welche der Konfigurationsteil der Anwendung erfüllen soll. Dazu werden die Datenbanktabellen samt Spalten aufgelistet. Der Anwender wählt eine Spalte aus und ordnet dieser eine Spalte in der Quelldatei zu.
Wird der Hochladevorgang gestartet, so wird für jede Tabelle, in die Werte eingespielt werden sollen, eine Insert – Anweisung vorbereitet. Die Quelldatei wird nun zeilenweise eingelesen. Nach jeder Zeile werden die eingelesenen Werte in die vorbereiteten Insert – Anweisungen eingesetzt und die Insert - Anweisungen werden ausgeführt. Für den Fall, dass in mehrere Tabellen geschrieben wird, kann die Reihenfolge der Inserts festgelegt werden.
Wenn während des gesamten Hochladevorgangs keine Fehler aufgetreten sind, wird am Ende ein ‚Commit' gesendet, um die Änderungen wirksam zu machen. Im Fehlerfall wird ein ‚Rollback' gesendet. D. h. eine Quelldatei wird entweder komplett oder gar nicht in die Datenbank eingespielt. Auf diese Weise soll Datenverlust oder gar Inkonsistenzen vorgebeugt werden.

Datenbanksystem

Das IPK setzt als DBS Oracle 9i in der Version 9.2 ein. Bei der Entwicklung der Anwendung darf also grundsätzlich davon ausgegangen werden, dass das Zielsystem von Oracle ist.
Grundsätzliches zum Thema Oracle soll hier nicht weiter abgehandelt werden. Sie hierzu z.B. [Abbey02].

Programmiersprache

Da es sich bei den Endanwendern in erster Linie um Nichtinformatiker handelt, darf neben speziellen Kenntnissen im Umgang mit Datenbanken auch keine spezielle Software vorausgesetzt werden. Des Weiteren werden am IPK unterschiedlichste Betriebssysteme eingesetzt, womit sich sehr schnell der Forderung ergibt, dass die Anwendung vollständig in JAVA umgesetzt werden muss. Der Datenbankzugriff erfolgt auf Grundlage des JDBC.
Damit ist die einzige Softwareanforderung, dass eine JRE 1.4.2 installiert ist. Allgemeine Informationen zu Java und JDBC sind z.B. in [Bonazzi02] zu finden.

Quelldateien

Als Datenquelle dienen CSV-Dateien oder Dateien im MS–Excel®-Format. Die Standartbibliothek von Java bietet keinerlei Möglichkeiten Excel®-Dateien zu lesen. Da die Anwendung natürlich auch unter Linux lauffähig sein muss, darf auch nicht davon ausgegangen werden, das MS–Excel® auf dem jeweiligen Rechner installiert ist. Um dennoch dieses Format lesen zu können wird auf das Jarkarta POI–Projekt zurückgegriffen – siehe hierzu [wwwPOI]. Dabei handelt es sich um eine vollständig Javabasierte Lösung zum bearbeiten von MS - Officedateien. Im vorliegenden Fall wird jedoch nur ein kleiner Teil des Projektes genutzt, da es hier nur notwendig ist Excel®-Dateien zu interpretieren.

Konfigurationsmöglichkeiten

Zu den Konfigurationsmöglichkeiten existieren folgende allgemeine Forderungen:

- Zur Generierung des Primärschlüssels werden i. d. R. Sequenzen eingesetzt, deren Abfrage damit möglich sein muss.

- Die Daten müssen über mehrere Tabellen, die eventuell untereinander in Beziehung stehen, verteilt werden können.

- Daten aus der Quelldatei müssen verarbeitet werden können.
  AnwendungsBsp 1: *In der Quelldatei steht neben anderen Daten der Name eines Zulieferers, der Zulieferer ist bereits in der DB erfasst. Deshalb soll nun ein Fremdschlüssel auf diesen Zulieferer erzeugt werden.*
  Die generierten INSERT - Anweisungen müssen also auch untergeordnete Abfragen enthalten können.

- An einigen Stellen werden in der Datenbank Datumsangaben benötigt. Dies ist problematisch, da das Datum in unterschiedlichen Schreibweisen angegeben werden kann. Die Datumssyntax muss also ggf. Berücksichtigung finden.

- Die erstellte Konfiguration muss gespeichert bzw. wieder geladen werden können.

4 Das UPLoadTool

Im Folgenden wird die erstellte Anwendung beschrieben. Einiges wird anhand von Anwendungsbeispielen erklärt. Diese beziehen sich auf die FLAREX - Datenbank. Das zugehörige ERD ist im Anhang B S.36 dargestellt, eine ausführliche Erklärung zu dieser Datenbank ist in [Rutkow04] zu finden.
Das Programm mit Quellcode und den benötigten Komponenten befindet sich auf der beiliegenden CD.

Login

Nach dem Start muss als erstes eine Verbindung zur Datenbank hergestellt werden. Voraussetzung hierfür ist eine Netzwerkanbindung zu einer Oracledatenbank. Alle notwendigen Eingaben können in einem Dialog, der nach dem Start automatisch erscheint, eingegeben werden. Abbildung 3 zeigt den Eingabedialog.

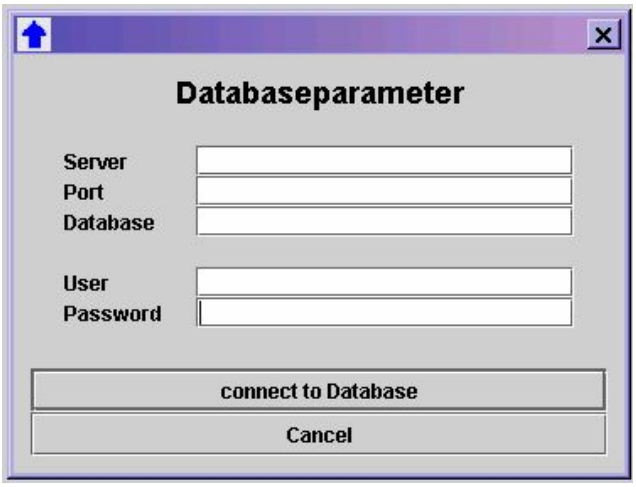

Abbildung 3 Eingabemaske Datenbankverbindung

Nach dem erfolgreichen Verbinden zur Datenbank kann mit Hilfe der Registerkarten „SETTINGS“ und „UPLOAD“ am linken unteren Rand zwischen 2 Modi umgeschaltet werden. Siehe hierzu Abbildung 23 und Abbildung 24 im Anhang A - Screenshots.
Die Karte „SETTINGS“ enthält alle Optionen zum Erstellen einer Konfiguration, während die Registerkarte „UPLOAD“ zum Hochladen der Daten dient.

4.1.1 Erstellung einer Upload-Konfiguration

Um den Datenimport zu definieren muss festgelegt werden, welche Werte in welche Datenbanktabelle geschrieben werden sollen. Dazu werden Spalten in der DB mit

Spalten in der Quelldatei in Beziehung gesetzt. Da jedoch nicht alle Informationen in der Quelldatei stehen (z.B. eine fortlaufende Nummerierung, die als Primärschlüssel dient), existieren weitere Optionen für die Datenquelle. Dazu zählen: Parameter, Sequenzabfrage, konstanter Wert, SQL-Abfrage. Diese Optionen und weitere Konfigurationsmöglichkeiten sollen im Folgenden erklärt werden.
Die gesamte Konfigurationserstellung findet auf der Registerkarte „SETTINGS“ statt. (Siehe Abbildung 24 S. 34)

4.1.1.1 Tabellen und Spalten auswählen

Der linke obere Block „- TABLES/COLUMNS IN DATABASE -“ enthält zwei Listen. In der Linken werden alle Tabellen der Datenbank angezeigt. Nach Anklicken eines Tabellennamens erscheinen in der rechten Liste die zugehörigen Spalten (siehe Abbildung 4). In diesem Block müssen alle Spalten ausgewählt werden, in die Daten geschrieben werden sollen. Dazu zählen also auch Schlüsselattribute.

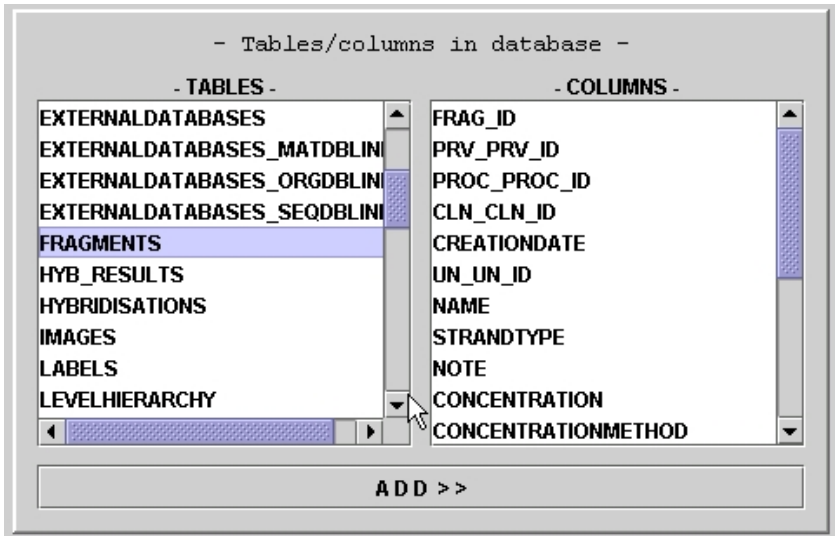

Abbildung 4 Auswahlmaske für Tabellen bzw. Spalten

Nach Auswahl einer Spalte und Betätigung des Buttons „ADD“ erscheint ein Eintrag im rechten oberen Block „- SELECTED TABLES/COLUMNS -“ (Vgl. Abbildung 5). Hier werden in einer Tabelle alle ausgewählten Tabellen/Spalten angezeigt. Die dritte Spalte der Tabelle enthält eine kurze Information zur Datenquelle (siehe nächster Abschnitt). Über die zwei Schaltflächen am unteren Rand können einzelne bzw. alle Einträge wieder gelöscht werden. Zur besseren Übersicht kann die Tabelle alphabetisch sortiert werden, dazu muss der jeweilige Tabellenkopf angeklickt werden. Zum sortieren wird der „TableSorter“ von Sun verwendet, siehe [wwwSun]. Die Reihenfolge, in der die

Spalten in Abbildung 5 angezeigt werden, hat keine Bedeutung für den Hochladevorgang.

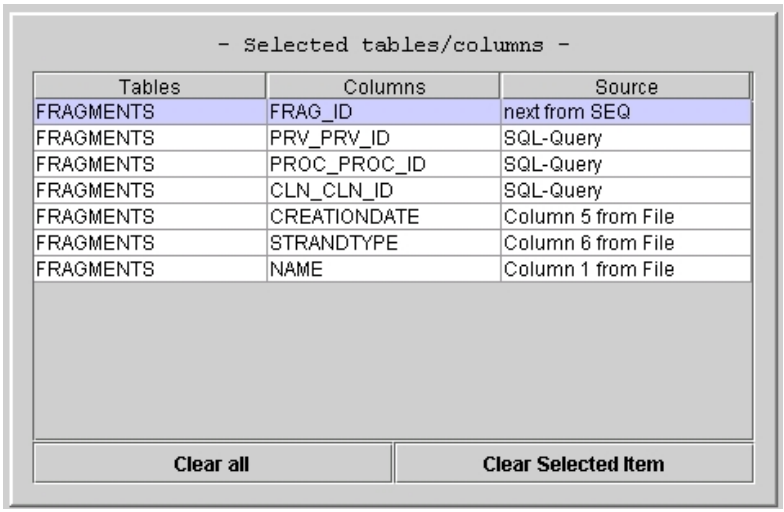

Abbildung 5 Übersicht über ausgewählte Spalten

4.1.1.2 Datenherkunft bestimmen

Nachdem die Spalten ausgewählt wurden muss festgelegt werden was in die jeweilige Spalte geschrieben werden soll bzw. woher die Daten genommen werden sollen. Im Block „- DATASOURCE -“ stehen dafür verschiedene Optionen zur Auswahl. Die Eingaben in diesem Block beziehen sich immer auf den unter „- SELECTED TABLES / COLUMNS -“ (Abbildung 5) markierten Eintrag.

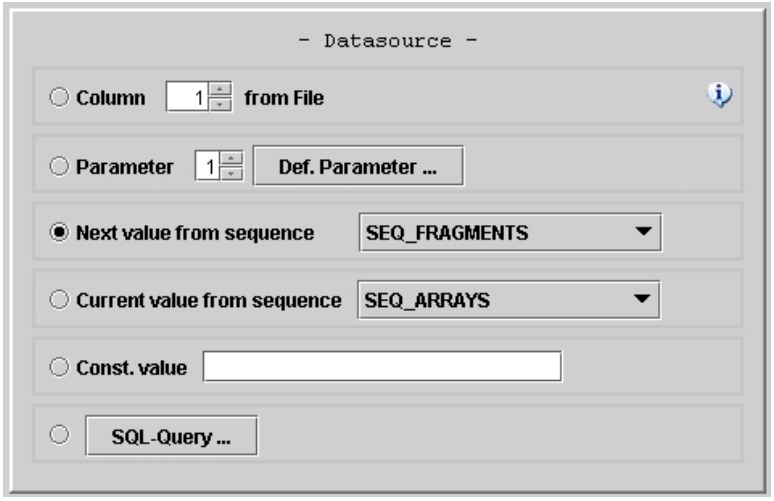

Abbildung 6 Eingabemaske für Datenquelle

Es stehen folgende Optionen zur Verfügung:

- COLUMN FROM FILE
  Dieser Eintrag bezieht sich auf die Quelldatei. Im zugehörigen Textfeld kann der Index der Spalte angegeben werden, aus der die Daten genommen werden sollen. Die erste Spalte hat den Index 1.

- PARAMETER
  Dieser Eintrag bezieht sich auf die verwendeten Parameter. Im zugehörigen Textfeld muss der Index von dem Parameter eingeben werden, dessen Wert in die ausgewählte Spalte geschrieben werden soll. Durch anklicken des Buttons ‚DEF. PARAMETER' wird ein Dialog geöffnet in dem die Parameter definiert werden können. Weiteren Informationen zur Verwendung von Parametern sind in *4.1.1.3 Verwendung von Parametern* zu finden.

- NEXT VALUE FROM SEQUENCE
  Bei Auswahl dieser Option wird der einzusetzende Wert aus eine Sequenzabfrage gewonnen. In der zugeordneten Auswahlbox kann die Sequenz ausgewählt werden. Die entsprechende Abfrage ist in CodeBsp 3 zu sehen.

```
SELECT Sequenzname.NEXTVAL FROM DUAL
```

CodeBsp 3 Nächsten Wert einer Sequenz abfragen

- CURRENT VALUE FROM SEQUENCE
  Bei Auswahl dieser Option wird der Wert ebenfalls aus der Abfrage einer Sequenz gewonnen. Die entsprechende Abfrage ist in CodeBsp 4 zu sehen. Die Sequenz kann wieder durch eine Combobox ausgewählt werden.

```
SELECT Sequenzname.CURRVAL FROM DUAL
```

CodeBsp 4 Aktuellen Wert einer Sequenz abfragen

- CONST. VALUE
  Hier kann ein Wert eingesetzt werden, der in allen Datensätzen gleich ist. Wird diese Option ausgewählt und in das Textfeld jedoch nichts eingetragen, so wird in die zugehörige Spalte NULL eingesetzt.

- QUERY
  Bei dieser Option ist der Wert, der in die Datenbank eingetragen wird, das Ergebnis einer Abfrage. Um die Abfrage zu definieren muss der Button „SQL-QUERY …" angeklickt werden, siehe hierzu *4.1.1.4 Verwendung von Abfragen*.

4.1.1.3 Verwendung von Parametern

In einigen Fällen stehen nicht alle Informationen, die für das Hochladen erforderlich sind, in der Quelldatei. Stattdessen sind sie dem Anwender, der die Daten einspielt, bekannt. Dies kann dann der Fall sein, wenn eine Information für alle Datensätze gleich ist.

AnwendungsBsp 2: *In die DB sollen SPOTS (DNA - Proben) für ein neues ARRAY (Trägerelement) eingespielt werden. Da alle Spots auf dem gleichen Array platziert werden, wird der Anwender die Bezeichnung des Arrays kennen.*

In solch einem Fall wäre es sinnvoll, wenn vor Beginn des Hochladens dieses Array vom Anwender ausgewählt, und damit allen Datensätzen zugewiesen, werden kann. Dafür dient die als Parameter bezeichnete Option.
Parameter können über die Menüleiste unter dem Eintrag Settings / Parameter eingestellt werden. Nach Auswahl dieses Menüpunktes öffnet sich ein Dialog, der die Konfiguration der Parameter vorsieht. Aus Gründen der Übersichtlichkeit wurde die Anzahl der Parameter auf max. 6 begrenzt. Das Konfigurationspanel für einen Parameter ist in Abbildung 7 zu sehen.

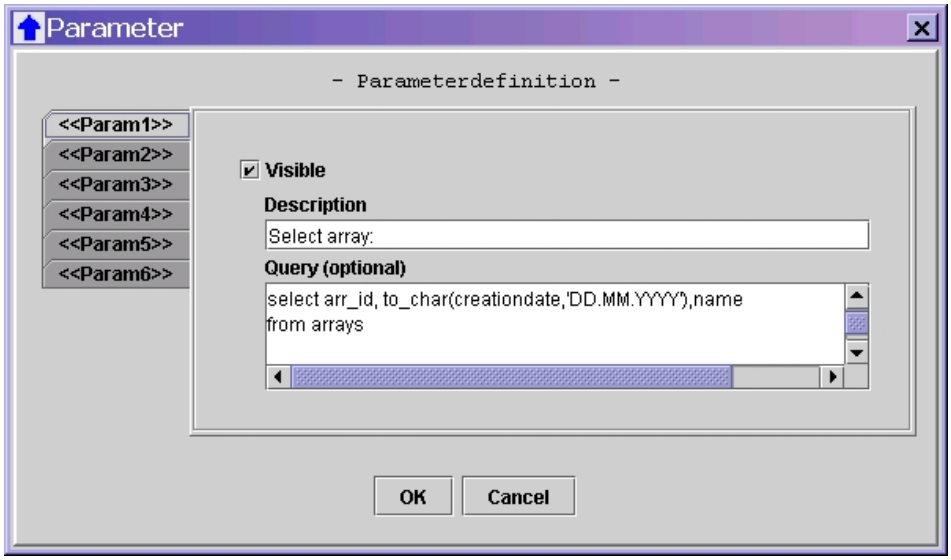

Abbildung 7 Eingabemaske zur Parameterdefinition

Mit Hilfe der Checkbox VISIBLE wird die Anzeige des Parameters aktiviert bzw. deaktiviert. Im Textfeld DESCRIPTION kann eine Eingabeaufforderung eingegeben werden.
Durch Aktivierung eines Parameters und Eingabe von *„Select Array:“* in das Feld DESCTIPTION entsteht auf der Hauptregisterkarte „Upload“ (siehe auch Abbildung 23 S.33) eine Eingabemöglichkeit wie in Abbildung 8 zusehen ist. Hier könnte der Anwender den Namen des Arrays eintragen.

Abbildung 8 Erstellte Eingabemaske für Parameter

Eine Eingabemöglichkeit über ein Textfeld zu gestallten wird i. d. R. zu Problemen führen – Bereits Fehler bei der Groß - bzw. Kleinschreibung könnten dazu führen, dass Datensätze nicht verknüpft werden können. Um diesem Fehler vorzubeugen, enthält die Eingabemaske (Abbildung 7) den Punkt QUERY. Hier kann eine SQL – Abfrage eingegeben werden. Daraufhin werden Eingaben in das Textfeld gesperrt, lediglich die Resultate der Abfrage stehen dem Anwender, in einer Auswahlbox, zur Wahl. Diese Auswahlbox öffnet sich nachdem auf den Button mit den drei Punkten geklickt wurde – siehe Abbildung 9. Auf diese Weise werden Fehleingaben von vornherein ausgeschlossen.
Die Abfrage muss immer min. 2 Spalten liefern. Die erste Spalte enthält die Werte, von denen einer ausgewählt werden soll bzw. mit dem das Programm weiter arbeiten soll. Bezogen auf das oben genannte Beispiel wird es sich hierbei also um den Primärschlüssel des ausgewählten Arrays handeln. Die Zweite, und ggf. nachfolgende Spalten, enthalten die Informationen, die dem Anwender angezeigt werden sollen. CodeBsp 5 zeigt ein Beispiel für die Abfrage, Abbildung 9 zeigt die entsprechende Eingabemaske für den Anwender.

```
SELECT arr_id, TO_CHAR(creationdate,'DD.MM.YYYY'), name
                FROM arrays
```

CodeBsp 5 SQL – Abfrage für Parameter

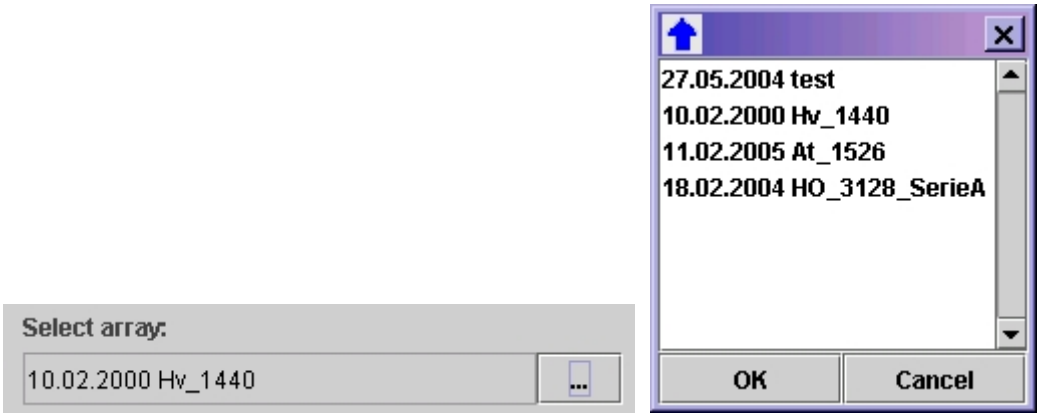

Abbildung 9 Erstellter Parameter mit Auswahlbox

Um den ausgewählten Primärschlüssel einer Spalte in der Datenbank zuzuordnen muss als Datenquelle der entsprechende Parameter ausgewählt werden. Siehe Abbildung 10.

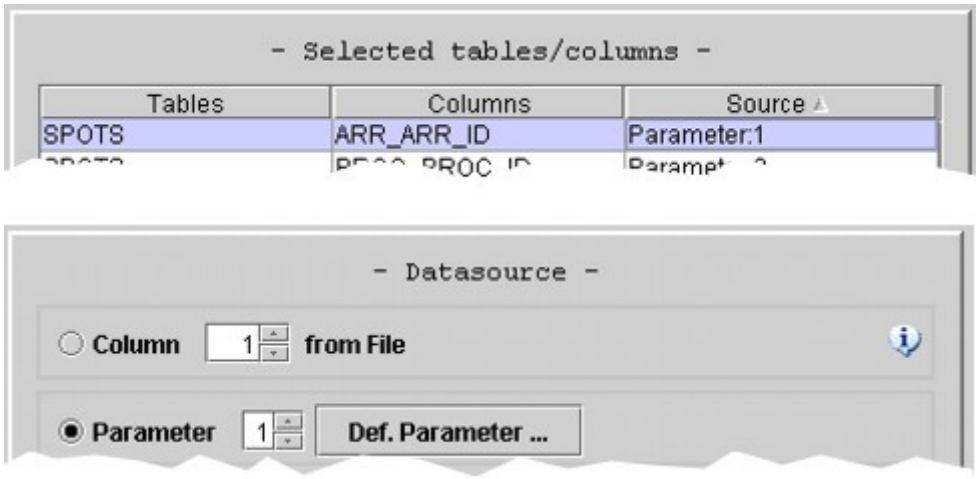

Abbildung 10 Parameterwert einer Spalte zuordnen

Weiter ist es Möglich den ausgewählten Wert in einer Abfrage weiterzuverarbeiten. Für Details siehe hierzu *4.1.1.2 – „Datenherkunft bestimmen“* bzw. *4.1.1.4 Verwendung von Abfragen.*

4.1.1.4 Verwendung von Abfragen

In Folge der Normalisierung sind Informationen in einer DB über viele Tabellen verteilt, die jedoch untereinander in Beziehung stehen. Diese Primär-Fremdschlüsselbeziehungen müssen beim Hochladen natürlich auch richtig gesetzt werden. Um die richtigen Schlüsselattribute zu bestimmen können Abfragen verwendet werden.
Abbildung 11 zeigt den Eingabedialog „OPTIONS FOR SQL-QUERY“. Die Abfrage kann im oberen Block des Dialogfensters eingegeben werden. Ihr Ergebnis darf nur eine Spalte und nur eine Zeile liefern, andernfalls wird der Datenimport abgebrochen.
Ein Anwendungsbeispiel könnte wie folgt aussehen:

AnwendungsBsp 3: *In die DB sollen Organismen eingespielt werden. Neben Cultivar (genaue Bezeichnung des Organismus) ist ein Verweis auf die Spezies (übergeordnete Bezeichnung) notwendig. Die Quelldatei wird also aus zwei Spalten bestehen:*
1 – Name des Organismus (CULTIVAR),
2 – Name der übergeordneten Spezies (SCIENTIFIC_NAME).

Um jedem Organismus die richtige Spezies zuzuordnen muss daher jedes Mal eine Abfrage durchgeführt werden, um den richtigen Schlüssel zu bestimmen. Grundlage für diesen richten Schlüssel ist der wissenschaftliche Name der Spezies aus der Quelldatei.

Die Fremdschlüsseleinträge für die Spalte SPC_SPC_ID in Tabelle ORGANISMS bestimmen sich also jeweils durch eine Abfrage wie in CodeBsp 6 zu finden ist.

```
SELECT spc_id FROM species WHERE scientific_name =
'<<FileVal2>>'
```

CodeBsp 6 Abfrage des Primärschlüssel

Wie zu sehen ist, kann mit Hilfe eines Platzhalters auf die Werte in der Quelldatei zugegriffen werden. Neben der Quelldatei können hier auch die Parameter verwendet werden. CodeBsp 7 zeigt die möglichen Platzhalter.

```
<<FileValX>> oder <<ParamY>>
   X – Spaltenindex [1..n]
   Y – Parameterindex [1..6]
```

CodeBsp 7 Mögliche Platzhalter

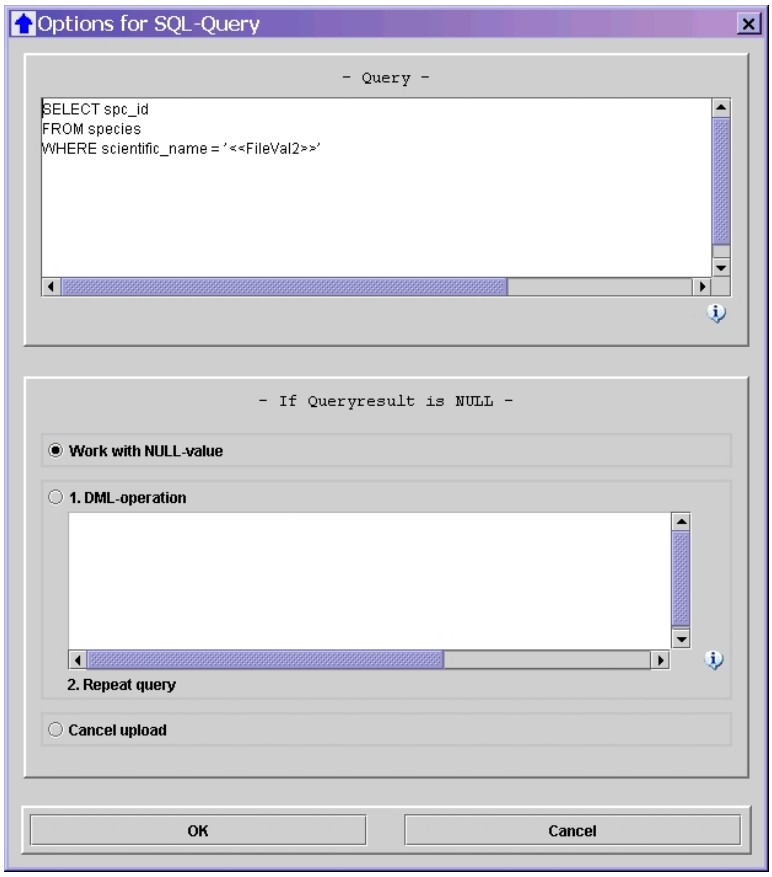

Abbildung 11 Eingabedialog für Abfragen

Der oben beschriebene Vorgang setzt voraus, dass die jeweilige Spezies auch in der Datenbank existiert. Ist das nicht der Fall, so liefert die Abfrage als Ergebnis NULL. Da

das Einsetzen von NULL in die Spalte SPC_SPC_ID nicht erlaubt ist, würde der Hochladevorgang mit der entsprechenden Fehlermeldung abgebrochen werden. Um solche Fälle zu vermeiden existieren im unteren Teil des Dialogfeldes (Abbildung 11) weitere Optionen. Hier kann festgelegt werden, ob das Ergebnis der Abfrage kontrolliert werden soll und wenn ja wie auf das Ergebnis NULL reagiert werden soll. Es stehen drei Optionen zu Auswahl:

1. Standardmäßig wird keine Kontrolle durchgeführt. In diesem Fall wird NULL als ein gültiges Ergebnis angesehen. – Diese Option kann auch gewählt werden, wenn NULL zwar nicht erwünscht ist, jedoch sicher ist, dass das Einfügen von NULL durch eine CONSTRAINT verhindert wird.

2. Der Upload wird vom Programm abgebrochen. In diesem Fall wird gar nicht versucht den Wert NULL in die Datenbank einzufügen.

3. Es wird eine DML Operation ausgeführt. Anschließend wird die Abfrage wiederholt. Bei der wiederholten Ausführung darf das Ergebnis nicht erneut NULL ergeben, da der Hochladervorgang dann ebenfalls abgebrochen wird.

Die dritte Option erlaubt es also den Datensatz, auf den verwiesen werden soll, einzufügen. Kommt die Spezies ein zweites mal in der Quelldatei vor, so wird die Abfrage sofort das Schlüsselattribut finden. Eine entsprechende Insert-Anweisung, bezogen auf AnwendungsBsp 3, ist in CodeBsp 8 zu sehen. Auch hier können Platzhalter (CodeBsp 7 S.15) verwendet werden.

```
INSERT INTO species (spc_id, scientific_name) VALUES
        (seq_species.NEXTVAL, '<<FileVal2>>')
```

CodeBsp 8 Einfügen eines neuen Lieferanten

4.1.1.5 Weitere Optionen

Saveorder

Wenn Daten über mehrere Tabellen verteilt werden, die untereinander in Beziehung stehen, muss beachtet werden, in welche Reihenfolge die Datensätze in die Tabellen eingefügt werden. Durch markieren eines Tabellennamens und Betätigung der Pfeilbuttons kann die Reihenfolge geändert werden. Siehe Abbildung 12.

Abbildung 12 Eingabemaske ‚Saveorder'

Separator

Als Datenquelle dient entweder eine MS-Excel® - oder eine CSV-Datei. Bei letzterer kann in diesem Optionsfeld (Abbildung 13) das Trennzeichen eingegeben werden. Als Standardzeichen dient der Tabulator. Wird ein anderes Zeichen verwendet kann dies im zugehörigen Textfeld eingegeben werden.

Abbildung 13 Eingabemaske ‚Separator'

Date Syntax

Ein Datum kann in verschiedenen Schreibweisen angegeben werden – z.B. 14.12.2004, 2004-12-14, 14-12-04 usw. Für den Fall, dass sich in der Quelldatei Datumsangaben befinden, kann hier die erwartete Syntax eingegeben werden (siehe Abbildung 14). Die Eingabemöglichkeiten in dieser Option entsprechen den Parametern der Oraclefunktion TO_DATE('*datum*', '*syntax*'). Beispiele hierfür sind in Tabelle 1 zu finden.

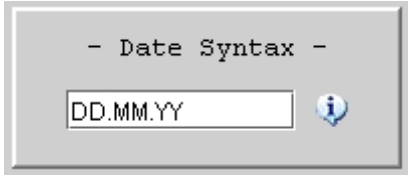

Abbildung 14 Eingabemaske ‚Date Syntax'

Tabelle 1 Beispiele Datumssyntax [Abbey02]

Syntax	**Bedeutung**
DD	Tag im Monat
MM	Monat [1..12]
Y oder YY oder YYY	Die letzte Ziffer, die letzten beiden oder die letzten drei Ziffern der Jahresangabe
YEAR	Das Jahr ausgeschrieben

Expected columns

Nachdem der Anwender (auf der Registerkarte „UPLOAD“) eine Quelldatei ausgewählt hat, wird im unteren Bereich ein Teil dieser als Vorschau angezeigt. Siehe hierzu Abbildung 23 S.33. Durch die Option „- EXPECTED COLUMNS -“ können die Spaltenüberschriften bei dieser Vorschau bestimmt werden.
Durch diesen Punkt entsteht für den Anwender eine leichte Kontrollmöglichkeit, ob die Quelldatei die erwartete Struktur hat. Diese Option wirkt sich also nicht auf den eigentlichen Hochladeprozess aus. Wird hier nichts eingetragen, so wird als Spaltenüberschrift nur „Column X“ angezeigt, wobei X der Spaltennummer entspricht.
Abbildung 15 zeigt die entsprechende Eingabemaske. Durch ein Betätigen der ENTER-Taste in der letzten Zeile wird eine neue hinzugefügt, Markieren und Betätigen der ENTF-Taste löscht eine Zeile.

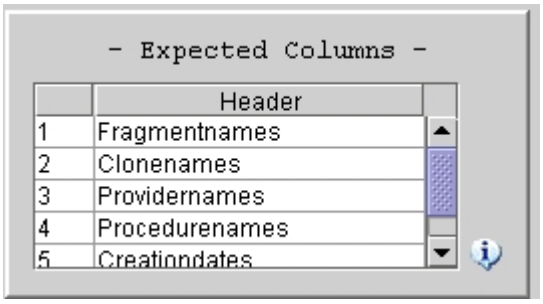

Abbildung 15 Eingabemaske ‚Expected Columns’

Rows to Skip

Standardmäßig wird davon ausgegangen, dass die Quelldatei von der ersten bis zur letzten Zeile Daten enthält, die in die Datenbank geladen werden sollen. Für den Fall, dass die erste Zeile z.B. Spaltenüberschriften enthält, kann hier (siehe Abbildung 16) angegeben werden, wie viele Zeilen am Anfang überlesen werden sollen. Diese Angabe

ist unabhängig davon, ob eine CSV- oder Excel-Datei ausgewählt wurde. Die zu überlesenden Zeilen werden in der Dateivorschau mit einer anderen Farbe dargestellt (siehe Abbildung 23 S. 33).

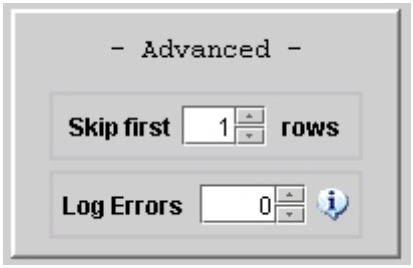

Abbildung 16 Eingabemaske ‚Advanced'

Log Errors

In diesem Optionspunkt kann festgelegt werden, wie die Anwendung auf Fehler reagieren soll. Standardmäßig ist hier der Eintag ‚0' gewählt. Somit wird beim Auftreten eines Fehlers ein ‚ROLLBACK' gesendet und der Hochladevorgang sofort abgebrochen. Diese Option ist sinnvoll, wenn Anwender ohne Datenbankkenntnisse Daten einspielen sollen.
Einige Anwender wünschten sich jedoch eine Möglichkeit großen Dateien auf Fehler prüfen zu können. Wird der Wert im Feld ‚Log Errors' auf ‚-1' gesetzt, so führen Fehler nicht sofort zum Abbruch des Datenimports. Stattdessen wird die fehlerhafte Zeile mit der entstandenen Fehlermeldung vermerkt und der Hochladevorgang in der nächsten Zeile fortgesetzt. So wird letztlich die gesamte Quelldatei durchlaufen und alle problematischen Zeilen werden erfasst. Diese fehlerhaften Zeilen werden im Statusdialog (siehe Abbildung 25 S.35) angezeigt und können zur späteren Bearbeitung gespeichert werden (siehe auch *4.1.2 - Hochladen von Daten S. 21*). Alternativ kann ein Eintrag größer 0 gewählt werden. In diesem Fall wird der Import beim erreichen der entsprechenden Fehlerzahl abgebrochen.
Für den Fall, dass das Einspielen der Daten in jeder Zeile erfolgreich war (also keine Fehlermeldungen aufgetreten sind), wird am Ende ein ‚COMMIT' gesendet. War nur eine Zeile fehlerhaft wird ein ‚ROLLBACK' gesendet.

4.1.1.6 Zusammenfassung Konfigurationserstellung

Abschließend soll Abbildung 17 den Abschnitt 4.1.1 - *Erstellung einer Upload-Konfiguration* in Form eines UML – Aktivitätsdiagramm zusammenfassen. Als

Aktivitätsbeschreibung wurde jeweils die Überschrift der Option, die abgearbeitet werden muss, gewählt.

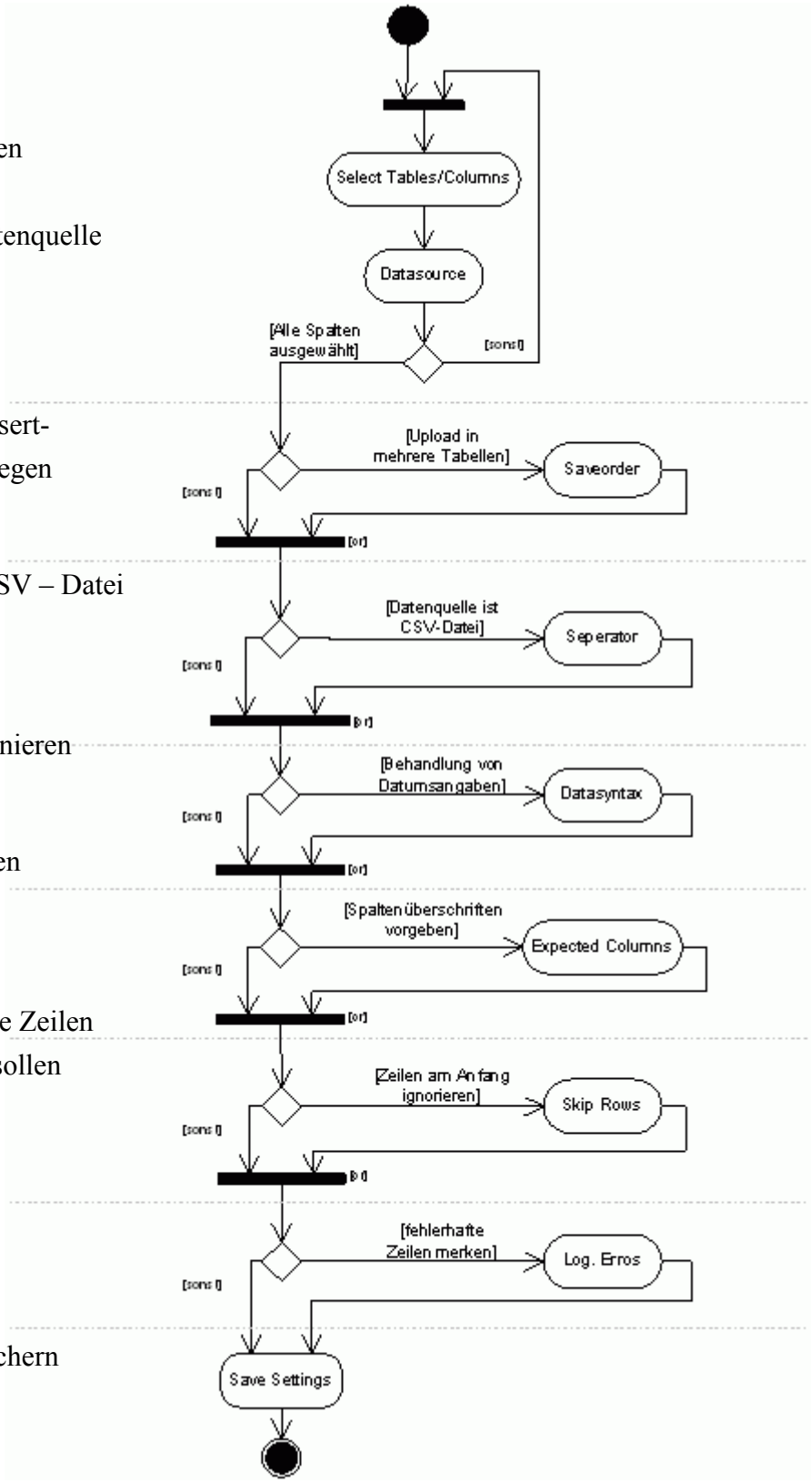

Abbildung 17 Aktivitätsdiagramm: Konfiguration erstellen

4.1.2 Hochladen von Daten

In diesem Punkt wird davon ausgegangen, dass bereits eine Konfiguration erstellt wurde und diese als Datei vorliegt. Alle, für das Hochladen von Daten notwendigen, Elemente befinden sich auf der Registerkarte „UPLOAD“ (siehe Abbildung 23 S. 33).

Zum Hochladen der Daten sind folgende Schritte notwendig:

1. Auswahl der gespeicherten Konfiguration.
   Durch Anklicken des Buttons „SELECT SETTINGS ...“ öffnet sich ein Dateiauswahldialog. Die Konfigurationsdateien tragen die Erweiterung „.upl“ und werden mit dem Programmicon dargestellt (siehe Abbildung 18).

2. Ggf. müssen Parameter eingegeben bzw. ausgewählt werden.

3. Mit Hilfe des Buttons „SELECT SOURCEFILE …“ kann die Quelldatei ausgewählt werden.

4. Kontrolle, ob die Spaltenüberschriften zu den Spalteninhalten passen.

5. Anklicken des Buttons „START UPLOAD“

6. Nach dem Hochladen den Statusdialog mit Hilfe des OK – Buttons schließen.

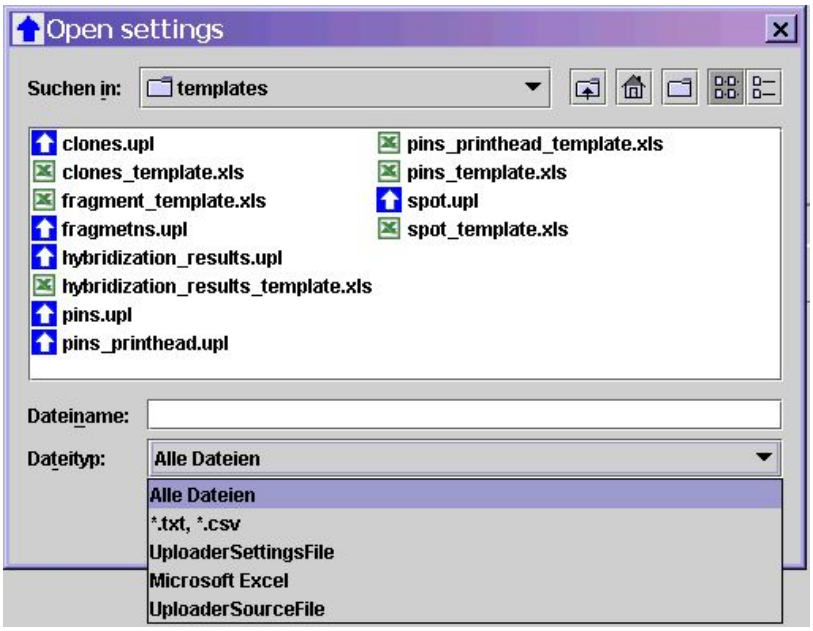

Abbildung 18 Dateiauswahldialog

Während des Hochladens wird im Statusdialog (Abbildung 25 S. 35) der Fortschritt angezeigt. Dazu befindet sich im oberen Bereich ein Fortschrittsbalken, darunter wird die Anzahl der Datensätze angezeigt, die schon verarbeitet wurden. Diese Anzeige bezieht sich auf die Zeilen in der Quelldatei.
Im mittleren Teil kann zwischen zwei Ansichtsfenstern umgeschaltet werden. In einen

werden allgemeine Informationen zum Programmstatus, im anderen (Abbildung 19) fehlerhafte Datensätze angezeigt (Zum Punkt fehlerhafte Datensätze siehe „*Log Errors*“ S. 19). Der Inhalt eines jeden Fensters kann mit Hilfe des Buttons „SAVE LOGFILE …“ gespeichert werden.

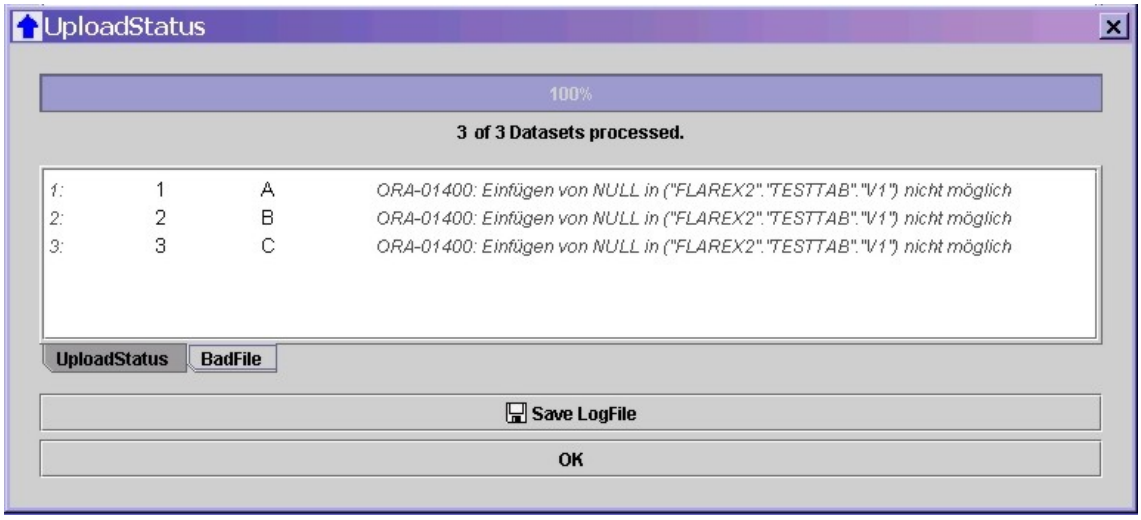

Abbildung 19 Anzeige der Fehlerhaften Datensätze

Am unteren Rand befindet sich ein Button, der je nach Programmstatus, seine Beschriftung ändert. Solange der Hochladevorgang noch nicht abgeschlossen ist, trägt er die Beschriftung „CANCEL“. Hiermit kann, nach einer Sicherheitsabfrage, der Vorgang abgebrochen werden. In diesem Fall wird ein ROLLBACK gesendet und keine der Daten wurden in die Datenbank übernommen.
Nach dem Beenden bzw. Abbruch des Hochladens trägt der Button die Beschriftung „OK“ und dient nur noch dazu, den Statusdialog zu schließen.

4.1.3 Einbetten in eine anderen Java- Applikation

So wie die Abwendung bis jetzt beschrieben wurde bringt sie noch folgende Nachteile mit sich:

- Bei einer großen Anzahl von Upload – Konfigurationen besteht die Gefahr, dass die Anwender versehentlich die falsche wählen.
- Der “UPLoader“ dient lediglich zum Einspielen großer Datenmengen. Für eine Datenbank existiert aber eventuell schon ein Eingabetool. In diesem Fall würden also zwei separate Programme existieren, was nicht anwenderfreundlich wäre.

Um diese Nachteile zu beseitigen, existiert die Möglichkeit den “UPLoader“ in eine andere Applikation einzubetten. Durch diese Vorgehensweise wird erreicht, dass der

Anwender weder eine Datenbankverbindung aufbauen noch die Settings - Datei auswählen muss. Außerdem werden auch einige Eingaben gesperrt. Dies betrifft, die gesamte Registerkarte Settings, die Menüleiste sowie den Button „SELECT SETTINGS …“.
Der eingebettete UPLoader dient somit ausschließlich dem Hochladen von Daten, Änderungen an den Einstellungen sind nicht möglich.

Dazu sind in der Anwendung, in die der UPLoader integriert werden soll, folgende Schritte notwendig.

- Es muss eine Datenbankverbindung existieren bzw. aufgebaut werden.
- Ein übergeordnetes Fenster muss festgelegt werden.
  (Dieses Objekt, der Klasse JFRAME, dient während des Hochladen als übergeordnetes Fenster für den Statusdialog - Abbildung 25 S.35)
- Die Position einer UPLoader - Settingsdatei (*.upl) muss bekannt sein.
  (Angabe muss Protokoll enthalten: HTTP:\\ …)
- Die oben genannten Objekte werden dem Konstruktor der Klasse UPLCONFIGURATION übergeben. Das zurückgegebene Objekt dient wiederum als Übergabeparameter beim Start des UPLoaders.

CodeBsp 9 zeigt den notwendigen Java - Quellcode dazu.

```
Connection dbCon = ...;
JFrame owner = ...;
String url="http:\\ ... \beispielKonfig.upl";

UplConfiguration uplConfig = new UplConfiguration(url, owner);
new UploaderStart(uplConfig, dbCon);
```

CodeBsp 9 Einbetten des UPLoaders in eine andere Anwendung

Über die Klasse UPLCONFIGURATION können dem UPLoader auch Werte für die Parameter (siehe 4.1.1.3 Verwendung von Parametern, S. 11) übergeben werden. Soll der Anwender einen Parameterwert nicht mehr ändern können, so muss die Sichtbarkeit dieses Parameters deaktiviert sein. Tabelle 2 zeigt die Methoden der Klasse `UplConfiguration`.

Tabelle 2 Methoden der Klasse UPLCONFIGURATION

`public UplConfiguration (String url,JFrame dialogOwner)`	*url* – Position der Settingsdatei *dialogOwner* – übergeordnetes Fenster
`public final void setParameter (int index, Object value, Object`	Weist einem Parameter einen Wert zu. *index* – Index des Parameters [1..6]

visibleValue)	*value* – Wert, mit dem der UPLoader arbeiten soll *visibleValue* – Wert, der dem Anwender angezeigt werden soll
public final Object getParameter (**int** index)	Diese Methoden werden von Programm genutzt und dienen dem Auslesen der gesetzten Informationen.
public final JFrame getDialogOwner()	
public final String getUrl()	

4.2 Spezifikationen

4.2.1 Generierte Insert – Anweisungen

Vor Beginn des Imports wird für jede Tabelle eine Insert-Anweisung generiert. Diese wird in Form eines „Prepared Statement“ angelegt. Bei einem Prepared Statement handelt es sich um eine Anweisung, die mit Platzhaltern versehen ist. An Stelle der Platzhalter werden später die Werte aus der Datei eingesetzt. Die Verwendung dieses Statements bringt einige Vorteile mit sich. So wird der eigentliche Befehl nur einmal übertragen, im weiteren Verlauf werden den Platzhaltern nur die Werte zugewiesen. Bei einer großen Anzahl von Datensätzen reduziert dies bereits deutlich den notwendigen Datendurchsatz. Der größte Vorteil ist aber, dass die Datenbank die Anweisung nur einmal auf korrekte Syntax prüft und anschließend auf die Ausführung einer solchen Anweisung vorbereitet ist. Ausführliche Informationen zu diesem Thema sind z.B. in [Bonazzi02] und [Krüger02] zu finden.
Die generierten Insert-Anweisungen haben die Struktur, wie sie in CodeBsp 10 zu sehen ist.

```
INSERT INTO Tabellenname (Spalte1, …, SpalteN) VALUES (
                SpaltenWert1, …, SpaltenWertN)
```

CodeBsp 10 Struktur Insert – Anweisung

Was für $SpaltenWert_n$ eingesetzt wird, hängt davon ab, was für eine Datenquelle $Spalte_n$ zugeordnet wurde (Vgl. 4.1.1.2 - Datenherkunft bestimmen, S.10). Es kommen folgende Fälle in Frage:

- Soll in `Spalte`$_n$ ein Wert aus der Datei oder ein konstanter Wert eingesetzt werden, so wird der eingelesene bzw. eingegebene Wert ohne Veränderung eingesetzt.

- Für den Fall, dass der Spalte ein Parameter zugeordnet wurde, wird der vom Anwender ausgewählte Wert eingesetzt (erste Spalte der Abfrage, Details siehe 4.1.1.3 - Verwendung von Abfragen S. 14).

- Soll das Ergebnis eine Sequenzabfrage eingesetzt werden, so wird *Sequenzename*.CURRVAL bzw. *Sequenzename*.NEXTVAL eingesetzt, je nachdem ob der aktuelle oder der nächste Wert abgefragt werden soll.

- Falls in `Spalte`$_n$ das Ergebnis einer Abfrage eingesetzt werden soll, wird standardmäßig die Abfrage an der Stelle von *SpaltenWert*$_n$ eingesetzt.
  Soll das Ergebnis der Abfrage gegen das Ergebnis NULL geprüft werden, so kann die Abfrage jedoch nicht eingesetzt. Stattdessen wird für die Abfrage ein eigenes Prepared Statement angelegt. Dieses wird dann vor dem Insertbefehl ausgeführt und geprüft. Das Ergebnis wird dann an Stelle von *SpaltenWert*$_n$ eingesetzt.

Handelt es sich bei *Spalte*$_n$ um eine Spalte vom Typ DATE, so wird für *SpaltenWert*$_n$ ein Ausdruck wie in CodeBsp 11 zu sehen ist eingesetzt.
Ob es sich bei Spalten um eine Datumsspalte handelt braucht nicht extra angegeben werden, sondern wird aus der Datenbank ausgelesen.

```
TO_DATE('SpaltenWert_n', 'DatumsSyntax ' )
```

CodeBsp 11 Behandlung von Datumsspalten

4.2.2 Anforderungen in die Quelldatei

Als Importquelle können CSV- Dateien oder MS - Excel® - Dateien verwendet werden. Die Quelldatei darf i.d.R. keine Leerzeilen enthalten, da diese wie eine Spalte mit dem Wert NULL interpretiert werden.

CSV

„Comma separated values“ – Dateien sind Textdateien, die Zeilenweise aufgebaut sind. Jede Zeile entspricht einem Datensatz. Innerhalb einer Zeile sind die Spalten durch ein bestimmtes Trennzeichen voneinander getrennt (meistens ein Komma). Wenn dieses Zeichen in einer Spalte vorkommt, so muss die gesamte Spalte in Anführungszeichen gesetzt werden. Kommt das Anführungszeichen im Spaltentext vor, so wird es durch ein weiteres Anführungszeichen maskiert. (Vergleich [Donnermeyer03])

MS-EXCEL®

Neben CSV - Dateien können MS - Excel® – Dateien als Quelle dienen. Um diese Dateien einlesen zu können werden Klassen des POI Projektes verwendet. *„POI bedeutet »Poor Obfuscation Implementation« (Schlechte, verschleiernde Implementierung).“* [wwwPOIde]. Dabei handelt es sich um eine rein Java-basierte Lösung zum Verarbeiten von MS – Office – Dateien. Das Projekt besteht aus mehreren Komponenten, entsprechend den verschiedenen Office-Anwendungen. In der vorliegenden Anwendung wird jedoch nur die Komponente „HSSF“ zum Einlesen der Exceltabellen genutzt.
Die verwendete Version (2.5.1-final-20040804) kann Dateien in den Versionen „Excel '97(-2002)“ lesen. Bei Test wurde festgestellt, dass es zu Fehlern kommen kann, wenn die Dateien mit Anwendungen wie Sun – Staroffice® bearbeitet werden.
Ein weiteres Problem bei Umgang mit Exceldateien ist, das sich eingegebene und angezeigte Werte unterscheiden. Je nach Spaltendefinition werden z. B. Zahlen auf 2 Dezimalstellen gerundet angezeigt, obwohl mehr Stellen eingegeben wurden. Dies kann zu Missverständnissen und letztlich zu Fehlern führen, da in diesem Fall nicht eindeutig ist welcher Wert in die Datenbank geschrieben werden soll. Pauschal lässt sich sagen, dass ein genauerer Wert auch besser ist. Beim Einspielen in die DB kann jedoch genau dies zu Fehlern führen, nämlich wenn die entsprechende Spalte die Genauigkeit nicht unterstützt. Um dieses Problem zu lösen, wurde beschlossen, dass alle Zellen in der Exceldatei als „Standard“ oder „Text“ definiert sein müssen. In diesen Fällen wird genau das angezeigt, was auch eingegeben wurde. Infolgedessen kann es zu keinen falschen Deutungen der Daten kommen.
Wird eine Datei als Datenquelle ausgewählt, die diese Konvention nicht einhält, so wird sie mit einer entsprechenden Fehlermeldung abgewiesen.

4.3 Implementierung

Im Folgenden soll ein Überblick über die Packagestruktur bzw. den Klassen des Programms gegeben werden. Abbildung 20 verdeutlicht den Programmaufbau.

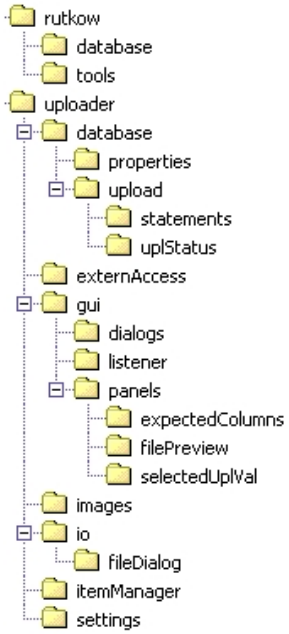

Abbildung 20 Packagestruktur

rutkow

Einige Klassen werden auch in einem anderen Projekt wieder verwendet und wurden deshalb in einem separaten Package abgelegt. Dies betrifft die Klassen, die für die Datenbankverbindung zuständig sind. Siehe Tabelle 3.

Tabelle 3 Klassen aus dem Package rutkow

Klassenname	**Bedeutung**
DBCONNECTION	Kapselt die das JAVA.SQL.CONNECTION – Object und stellt einige Hilfsmethoden zur Verfügung.
DATABASEPARAMETER	Dialogfenster zur Eingabe der Verbindungsparameter Vgl. Abbildung 3 S.8
SPLASHSCREEN	Dialog, der während des Verbindungsaufbaus angezeigt wird.
MYTOOLKIT	Hilfsklasse, stellt einige statische Methoden zur Verfügung. Z.B. um Fenster auf Bildschirm zu zentrieren.

uploader

Alle Klassen und untergeordneten Packages die zum „Uploader“ gehören sind in diesem Package zusammengefasst. Außerdem befindet sich hier die Klasse UPLOADERSTART, welche die MAIN – Methode zum starten des Programms enthält

uploader.database.properties

Mit Hilfe der Klasse DATABASEPROPERTIES, in diesem Package werden folgende Informationen über die Datenbank gespeichert: Tabellennamen und zugehörige Attribute mit Datentyp sowie Sequenznamen.

uploader.database.upload

Dieses Package enthält die Klassen, die für den Datenimport notwendig sind. Wie unter 4.2.1 erläutert wird für jede Tabelle, in die Daten eingefügt werden sollen, eine Insert-Anweisung in Form eines PREPAREDSTATEMENT erstellt. Dies wird von der Klasse UPLSTMTFACTORY durchgeführt. Die Klasse PREPAREDSTRING ist lediglich eine Hilfsklasse und außerhalb des Package nicht sichtbar. Die generierten Objekte sind vom Typ ABSTRACTUPLSTMT bzw. der daraus abgeleiteten Klassen, welche die eigentlichen STATEMENT-Objekte kapseln. Hier werden Methoden zur Verfügung gestellt, um die Platzhalter durch die richtigen Werte aus der Quelldatei bzw. Ergebnisse der Abfragen zu ersetzen. Um die richtige Zuordnung zu bestimmen wird die Klasse PLACEHOLDERASSIGMENT verwendet. In den daraus erstellen Objekten wird vermerkt, ob ein Platzhalter durch einen Wert aus der Datei oder durch das Ergebnis einer Abfrage ersetzt werden soll. Im ersteren Fall wird zusätzlich die Spalte vermerkt, im letzteren die Abfrage. Siehe hierzu Abbildung 21.
UPLOADPROCESS führt den eigentlichen Datenimport durch.

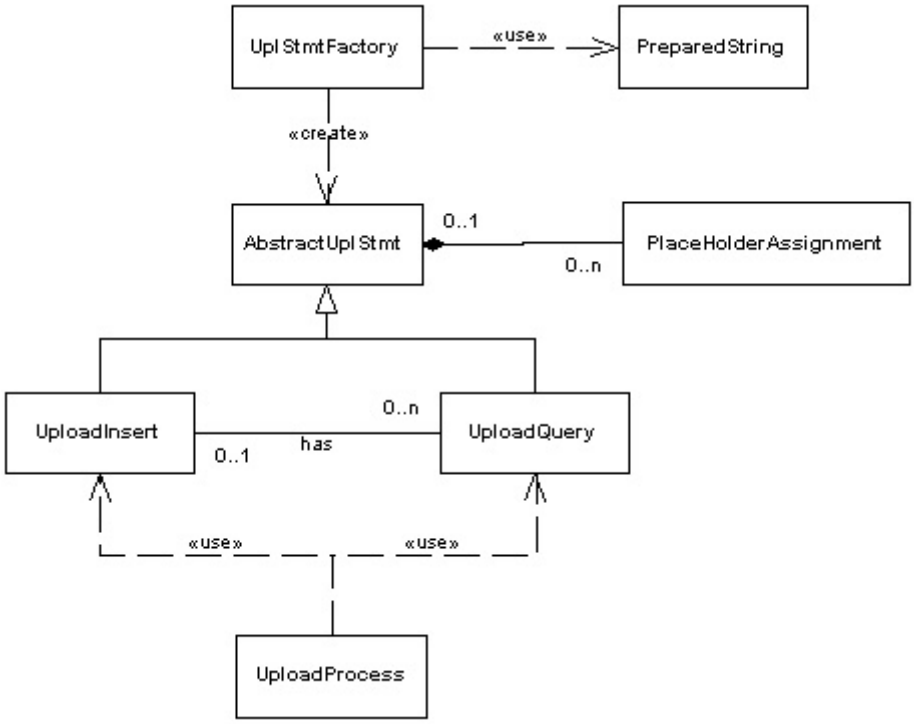

Abbildung 21 Klassen für den Datenimport

upoader.externAccess

Dieses Package enthält lediglich die Klasse UPLCONFIGURATION. Sie wird benötigt, wenn der UPLoader in eine andere Anwendung eingebettet wird. Siehe hierzu Abschnitt 4.1.3 - *Einbetten in eine anderen Java- Applikation* auf Seite 22.

upoader.gui

In diesem Package werden alle Klassen, die für die graphische Ein- bzw. Ausgabe notwendig sind zusammengefasst.
Im untergeordneten Package *panels* sind Klassen enthalten, welche die einzelnen Blöcke (Datasource, Date Syntax usw.) im Programm repräsentieren. Sie sind alle aus JPANEL abgeleitet und enden mit „GUI“. Da in einigen Fällen Hilfsklassen, wie z.B. erweiterte Tabellenmodelle, notwendig sind, wurden diese mit der zugehörigen GUI – Klassen abermals in ein untergeordnetes Package gelegt.
Zu jeder GUI – Klassen existiert im Package *listener* eine gleichnamige Klasse die jedoch mit „LST“ endet. Diese Klassen haben ein Listenerinterface implementiert und übernehmen so die Ereignisbehandlung der zugehörigen GUI – Klasse.
In *dialogs* befinden sich Klassen, die aus JDIALOG abgeleitet sind und die notwendigen

Listenerinterfaces direkt implementiert haben. Dazu zählen z.B. der „About“ – Dialog oder das Fenster zur Eingabe einer SQL-Abfrage.

upoader.images

IMAGES enthält alle Bilder, die vom Programm dargestellt werden. Des Weiteren befindet sich darin die Klassen IMAGEMANAGER mit der statischen Methode GETIMAGEICON(INT INDEX). Diese liefert die Bilder in Form eines JAVAX.SWING.IMAGEICON - Objektes. Zur Identifikation der Bilder dienen öffentliche Konstanten der Klasse IMAGEMANAGER.

upoader.io

Das Package *io* enthält Klassen, die für den Umgang mit Dateien notwendig sind. Dazu gehören neben einer Klasse zum Lesen der Quelldateien, Klassen zur Dateiauswahl. Die damit verbundenen Klassen (z.B. (Datei-)Filterklassen) sind im untergeordneten Package FILEDIALOG abgelegt.

upoader.itemManager

Die einzelnen Listenerklassen müssen z.T. auf mehrere GUI – Klassen zugreifen, da sich die Änderung in einen Eingabeblock auf einen anderen auswirkt. Zum Zugriff auf ein Objekt ist ein Java eine Referenz auf das jeweilige Objekt notwendig. Das heißt, dass ein Objekt alle anderen kennen muss, mit denen es in Kontakt treten will. Um dies zu gewährleisten existiert die Klasse ITEMMANAGER. Ihr Konzept ist wie folgt. Bei der Erzeugung eines neuen (GUI-)Objektes registriert sich dieses mit Hilfe eine entsprechende SETXXX – Methode. Benötigt ein anderes Objekt die Referenz, so kann diese mit einer zugehörigen GETXXX – Methode wieder abgefragt werden. Auf diese Weise wird erreicht, dass nicht jedes Objekt die Referenzen auf alle anderen besitzen muss. Stattdessen wird nur die Referenz auf ein Objekt der Klasse ITEMMANAGER benötigt. Dies wird durch die statische Methode ITEMMANAGER.GETINSTANCE() geliefert.

upoader.settings

Die Klasse SETTINGS im gleichnamigen Package dient zum zentralen speichern, der erstellten Einstellungen. Beim speichern der Konfiguration werden letztlich die Informationen aus dieser Klasse in eine Textdatei geschrieben. Abbildung 22 zeigt den Zusammenhang mit anderen Klassen in Gleichnamigen Package.
Objekte der Klasse UPLOADVAL ordnet einer Tabellen- Spaltenkombination eine Datenquelle (aus Datei, konstanter Wert usw.) zu. Da die Datenquelle auch eine Abfrage sein kann, kann ein UPLOADVAL-Objekt auch ein SUBQUERY-Objekt beinhalten. Dort wird die SQL-Abfrage und ggf. die DML-Anweisung gespeichert. In der Klasse UPLPARAMETER wird gespeichert, wie die Eingabeaufforderung und die optionale Abfrage lauten, sowie ob der Parameter sichtbar sein soll.

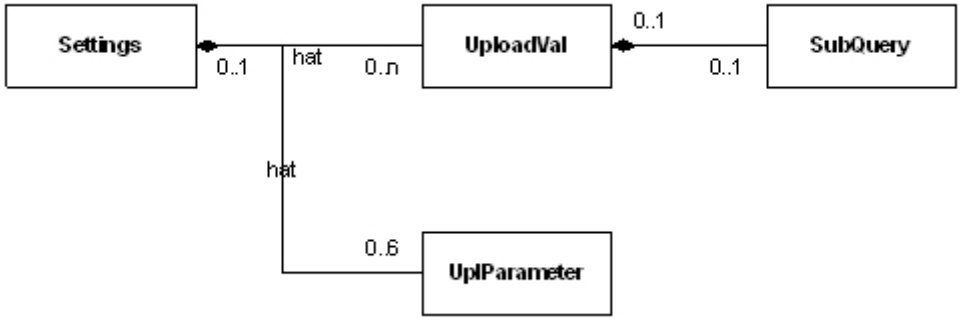

Abbildung 22 Klassen zum Speichern der Konfiguration

5 Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der Entwicklung einer plattformunabhängigen Applikation zum Import von CSV- und MS – Excel® - Dateien in eine Oracle Datenbank.
Da das Programm primär von Anwendern ohne Datenbankkenntnisse angewandt werden soll, war das oberste Ziel einen Kompromiss zwischen hoher Konfigurierbarkeit und einer einfachen Bedienung zu finden. Daraus resultierend besteht die Anwendung aus zwei Teilen. Im Ersten wird, von Anwendern mit DB – Kenntnissen, der Datenimport definiert. Die erstellte Konfiguration kann gespeichert werde. Der Zweite Teil der Anwendung dient dem eigentlichen Import. Hier kann der Endanwender eine Quelldatei und entsprechende Konfiguration auswählen, um dann die Daten zu importieren.
Um eine hohe Benutzerfreundlichkeit zu gewährleisten besteht die Möglichkeit das Programm in eine bestehende JAVA™ – Applikation zu integrieren. Dabei wird es möglich, dem Endanwender eine bestimmte Konfiguration vorzuschreiben und eine Änderungsmöglichkeit zu verbieten. Weiter kann eine Konfiguration auch über das HTTP – Protokoll geladen werden. Da die Anwendung auch als JAVA™ Web Start – Applikation ausgeführt werden kann besteht somit die Sicherheit, dass die Endanwender zum einen immer mit der aktuellen Programmversion und weiter mit den aktuellen Konfigurationsdateien arbeiten.

A Screenshots

A.1 Registerkarte „Upload“

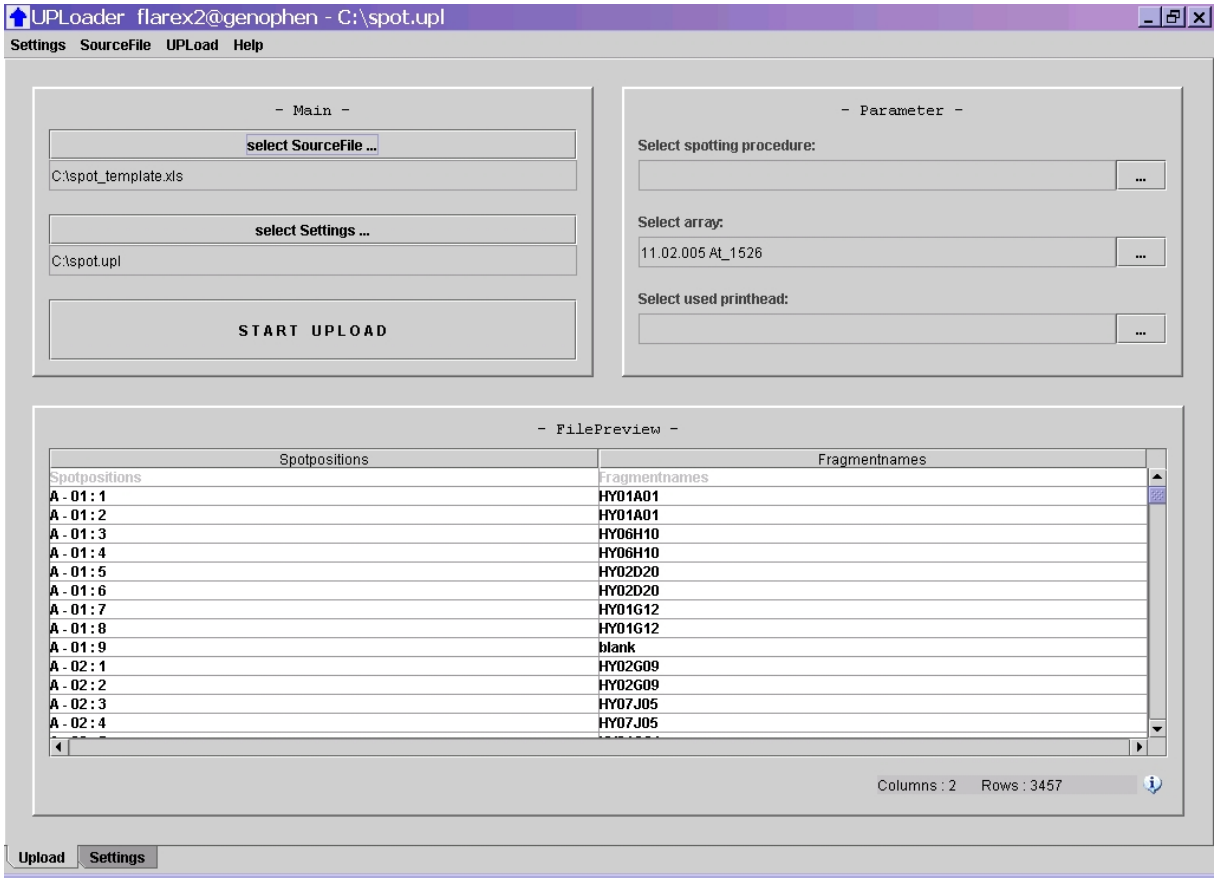

Abbildung 23 Registerkarte ‚Upload’

A.2 Registerkarte „Settings“

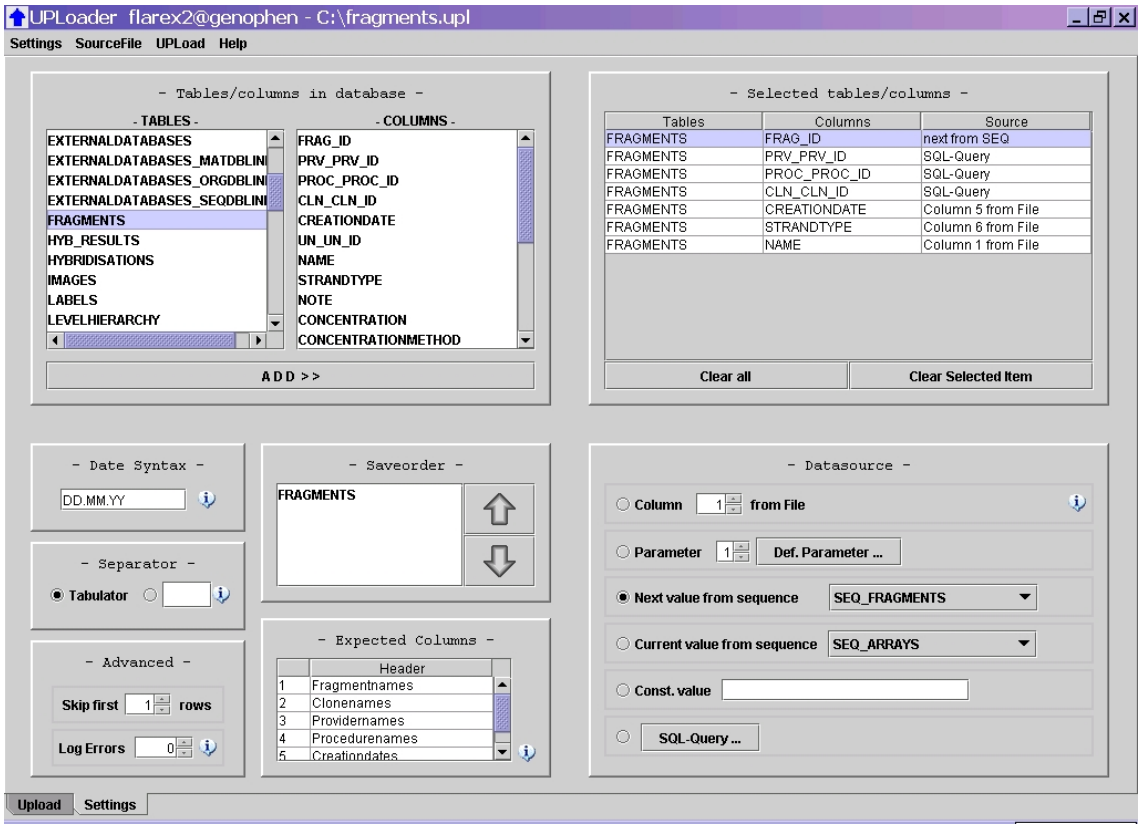

Abbildung 24 Registerkarte ‚Settings’

A.3 Dialogfenster „UploadStatus“

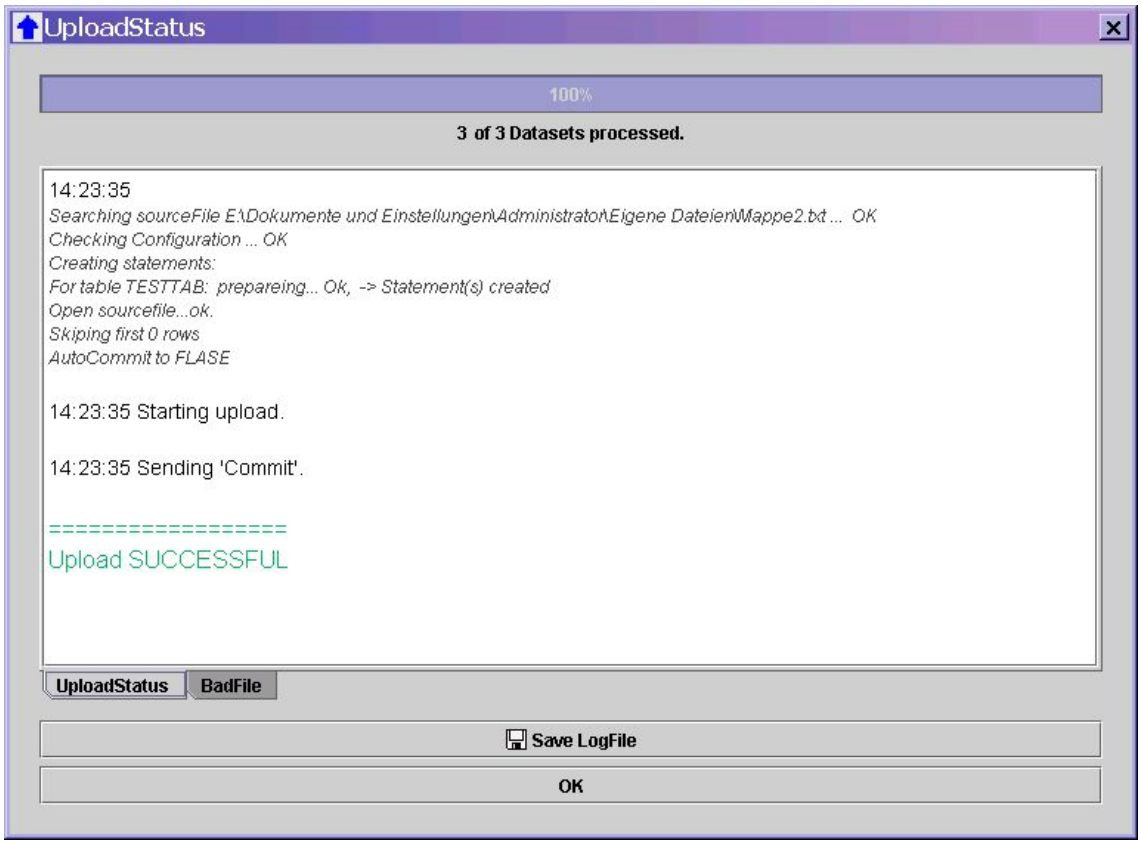

Abbildung 25 Dialogfenster ‚UploadStatus’

B Beispieldatenbank

Das ERD der FLAREX - Datenbank dient als Grundlage für einige Beispiele. Es befindet sind als pdf-Datei auf der beiliegenden CD. Hintergründe, bzw. Bedeutung der Tabellen, sind in [Rutkow04] zu finden.

Quellenangaben

[Abbey02] *„Oracle 8i Für Einsteiger“*
M. Abbey, M. J. Corey, I. Abramson
Hanser Verlag 2002
195

[Bonazzi02] *„Oracle und Java“*
E. Bonazzi, G. Stokol
Markt und Technik 2002

[Cordts02] *„Datenbankkonzepte in der Praxis“*
Sönke Cordts
Addison-Wesley 2002

[Donnermeyer03] *„Java Codebook“*
Donnermeyer M., Rusch B., Brodersen D., Wiederstein M., Skulschus M
Addison-Wesley 2003

[Krüger02] *„Handbuch der Javaprogrammierung“* (3.Auflage)
G. Krüger
Addison-Wesley 2002

[Oestereich05] *„Objektorientierte Softwareentwicklung – Analyse und Design mit der UML 2“* (7. Auflage)
B. Oestereich
R. Oldenbourg Wissenschaftsverlag GmbH 2005

[Rutkow04] Praktikumsbericht *„Weiterentwicklung der Flarexdatenbank, Erstellung einer Benutzerschnittstelle“*
T. Rutkowski
HS-Harz, Jan 2004

[wwwPOI] *„Jakarta POI - Java API To Access Microsoft Format Files“*
http://jakarta.apache.org/poi/
30.08.2004

[wwwPOIde] *„Jakarta POI“*
http://jakarta.apache.org/poi/trans/de/index.html
30.08.2004

[wwwSun] „How to Use Tables“
http://java.sun.com/docs/books/tutorial/uiswing/components/table.html
01.03.05