DBENGINE

Ein System zum Verwalten und Benutzen von Datenbanken über die Web-Schnittstelle

Detlef Dürr

DBENGINE: Ein System zum Verwalten und Benutzen von Datenbanken über die Web-Schnittstelle

Detlef Dürr

3.0.1

Table of Contents

List of Figures

List of Tables

Vorbemerkung

1. Einleitung

Das PostgreSQL[1]-Hilfstool DBENGINE ist ein Werkzeug zur Bedienung und Administration von Datenbanken unter PostgreSQL über die Web-Schnittstelle. Mit Hilfe dieses Tools können existierende Datenbanken in einem Intranet oder auch über das Internet genutzt und verwaltet werden. Eines ist dabei allerdings zu beachten: Z. Z. ist das Tool nur ohne Verschlüsselung, also nur mit dem Protokoll http und nicht mit https, getestet.

Dieses Handbuch wendet sich vorrangig an Administratoren, da weniger die Bedienung einer Datenbank als die Konfiguration seiner Bedienschnittstelle beschrieben wird. In einem ersten Teil wird dennoch kurz auf das Aussehen der Benutzerschnittstelle eingegangen, da eine sinnvolle Konfiguration nur möglich ist, wenn man weiß, unter welchen Anwendergesichtspunkten die Schnittstelle grundsätzlich vordefiniert ist. Die Kenntnis von PostgreSQL und der Sprache SQL wird vorausgesetzt und Textstellen, die sich auf dieses Datenbanksystem beziehen werden nicht weiter erläutert.

Im zweiten Teil erfolgt dann eine vollständige Beschreibung der Hilfsmittel zur Konfiguration von Datenbanksystemen, die dem Anwender zur Verfügung gestellt werden sollen. Die gesamte Konfiguration geschieht über die PostgreSQL-Datenbank dbengine_info_db, die selbst auch mit dem Tool konfiguriert werden kann.

Das Tool geht von einer Client-Server-Architektur aus und ist vollständig in Perl geschrieben. Die eigentliche Bearbeitung findet auf dem Server statt, während der Client als ein Thinclient nur zur Anzeige und Entgegennahmen der Daten dient.

Zur Benutzung des Tools ist daher serverseitig ein Rechner mit einem Perlinterpreter nötig. Zur Anzeige der Daten und zu ihrem Handling wird die Browserschnittstelle des Internet benutzt. Daher ist auf der Serverseite das Vorhandensein eines Webservers (z.B. Apache) nötig. Die Perlskripte laufen als CGI-Skript des Webservers.

Als Schnittstelle zu Datenbanken wird die DBI-Schnittstelle von Perl benutzt. Daher kann das Datenbanksystem auf einem anderen Rechner laufen und muss nicht auf dem des Webservers installiert sein. Soll allerdings von BLOBs[2] Gebrauch gemacht werden, so müssen der Webserver und der Datenbankserver auf dem gleichen System liegen. Vom Datenbanksystem wird nur die Datenbankclientschnittstelle benötigt. Auch können grundsätzlich andere Datenbanksysteme genutzt werden. Allerdings ist dann der Perl-Code an einigen Stellen zu ändern, da auch spezielle Funktionen vom Datenbankmoduln DBD::PG[3] benutzt werden, und da an einigen Stellen Besonderheiten von PostgreSQL genutzt werden. Auf einem PostgreSQL-Server muss außerdem eine Konfigurationsdatenbank dbengine_info_db existieren. Zwar läuft das System auch ohne jegliche zusätzliche Konfiguration, aber dann nur in einer nicht optimalen und eingeschränkten Form, da einige Funktionen (z.B. Berichte) nur nach entsprechender Konfiguration lauffähig sind. Auch Querverweise von einer Tabelle zu einer anderen sind nur nach einer entsprechenden Konfiguration möglich.

Auf der Clientseite wird lediglich ein Webbrowser (wie z.B. Firefox benötigt, da auf ihr – wie oben schon gesagt – nur die Anzeige und Entgegennahme von Eingabedaten erfolgt. Da normalerweise auf allen heutigen Systemen ein Webbrowser vorhanden ist, ist auf der Clientseite keine weitere Installation nötig.

[1] Mehr über dieses Datenbankmanagementsystem ist in der Internetseite für das Datenbanksystem PostgreSQL [http://www.postgresql.org/] zu finden

[2] Gemeint sind „Binary Large Objects“

[3] Sofern ein Perl-System auf dem Rechner installiert ist, sind die wichtigsten Bibliotheken auch installiert. Dazu gehören auch die Bibliotheken DBI und DBD::PG. Sollten sie nicht installiert sein, so können sie mit CPAN [http://www.cpan.org/] aus dem Internet besorgt werden.

Das Tool gibt seine Texte sprachabhängig aus. Welche Sprache es benutzt, wird anhand der vorhandenen Sprachen (z.Z. Deutsch und Englisch) und der Browsereinstellung entschieden. Bei den hier gezeigten Abbildungen war stets die Sprache Deutsch (de) als Vorzugssprache im Browser eingestellt.

Da die Webschnittstelle zustandsfrei arbeitet, empfiehlt es sich, bei jeder Tabelle einer Datenbank, die von mehreren Benutzern geändert werden kann, ein zusätzliches Feld mit dem Feldnamen tmin einzuführen und diesem Feld den Datentyp timestamp zu geben. Über dieses Feld, sofern vorhanden, wird überprüft, ob der Datensatz nach dem Einlesen durch einen Benutzer von einem anderen geändert wurde und deswegen von dem überprüften Benutzer nicht geändert oder gelöscht werden darf.

Mit Hilfe des Tools kann jede `PostgreSQL`-Datenbank den Benutzern über das Web zur Verfügung gestellt werden. Dazu bedarf es keiner Datenbank spezifischen Konfiguration. Diese aber ist möglich, um damit das Aussehen auf dem Bildschirm den Erfordernissen besser anpassen zu können. In der Konfiguration kann auch festgelegt werden, wer welche Datenbanken bearbeiten darf.

Das Tool DBENGINE ist freie Software und unterliegt der Lizenz nach den Richtlinien der LGPL (s. Lizenz im Glossary). Diese Lizenzvereinbarung ist im Anhang abgedruckt.

2. Einige typografische Konventionen

Zum Abschluss sollen hier noch einige typografische Besonderheiten erwähnt werden. diese Besonderheiten sollen der besseren Lesbarkeit des Textes dienen. Im wesentlichen werden folgende typografische Konventionen verwendet:

‹Ersetzungstext›	Platzhalter, an dessen Stelle ein bestimmter Wert, z. B. ein Kommando-Parameter, eingetragen werden muss
`Schreibmaschinenschrift`	In dieser form werden Dateinamen, Datenbanknamen, Elementnamen, Konstanten, Quellcode u. Ä. gekennzeichnet
`Schreibmaschinenschrift, kursiv`	Mit dieser Topografie werden Bildschirmmenues oder einzelne Befehle aus Bildschirmmenues gekennzeichnet.
[Knopf]	So werden Knöpfe, die der Benutzer betätigen muss, im Text gekennzeichnet.
`Schreibmaschinenscrift, fett`	So werden im Text einzugende Kommandos gekennzeichnet.

Kapitel 1. Vorbereitung

1.1. Installation

1.1.1. Installationsvorgehen

Bevor das eigentliche Tool DBENGINE installiert werden kann, müssen für das Datenbanksystem `postgreSQL` einige Einstellungen vorgenommen werden:

1. Initialisierung des Datenbanksystems, sofern das nicht schon geschehen ist. Dazu gehört auch deas Kreieren der nötigen Datenbanken, wobei darauf zu achten ist, dass sie mit einem verträgliche kodierungsschema versehen werden. Am besten ist es, man nimmt für alle Datenbanken das Schema `SQL_ASCII`:

```
createdb --encoding SQL_ASCII ‹Datenbankname›
```

2. Eintragen der Benutzer mit ihrem Passwort im Datenbanksystem. Es müssen mindestens der Administrator und als Benutzer der Benutzer eingetragen werden, unter dem der zu benutzende Web-Server läuft. Administrator wird sehr häufig `postgres` sein und der Web-Server läuft im Allgemeinen unter dem Namen `wwwrun`. Es ist dann also folgende Befehlsfolge einzu geben:

```
# Eintragen des Administrators mit allen Rechten eines Superuser
createuser --superuser --createdb --pwprompt postgres
# Eintragen des Web-Server-users
createuser --pwprompt wwwrun
# Eintragen der übrigen Benutzer
createuser --pwprompt ‹useri›
…
…
…
```

Mit diesen Werten ist dann das System für die Datenbanken zu nutzbar.

Für die Installation des DBENGINE wird ein gepacktes **`tar`**-File mit allen notwendigen Dateien geliefert. Als erstes muss diese Datei mit dem Kommando **`gunzip`** dekomprimiert und anschließend mit dem Kommando **`tar -xvf`** in einem zusätzlichen Verzeichnis entpackt werden. Dabei entstehen zwei Unterverzeichnisse:

- Das eine, `dbengine-<version>`, enthält alle Dateien der DBENGINE-Maschine.
- Das andere, `contrib`, enthält die Version 0.39 von `Math-Currency`.

Nach dem Entpacken kann die Installation erfolgen. Zur Installation ist es nötig als `root` im System zu arbeiten. Die Installation kann auf zwei Weisen geschehen:

- Durch Aufruf des Kommandos **`install.sh`** im Unterverzeichnis `dbengine-<version>` oder
- durch manuelles Ausführen von Kommandos zur Installation.

Die Installationsroutine **`install.sh`** erfragt acht wesentliche Merkmale – sofern sie nicht als Parameter mitgegeben wurden –, um dann die Dateien an die richtigen Stellen zu kopieren:

1. Parameter '-c': das Verzeichnis für das CGI-Script (default: /usr/local/httpd/cgi-bin)

2. Parameter '-d': das DBENGINE-Dateien Verzeichnis (default: /usr/local/dbengine)
3. Parameter '-n': der Name des postgreSQL Datenbankservers (default: localhost)
4. Parameter '-p': der Port des postgreSQL Datenbankservers (default: 5432)
5. Parameter '-a': der Name des postgreSQL-Administrators, i.e. Hauptbenutzer für die dbengine_info_db Datenbank (default: `postgres`).
6. Parameter '-t': der Name des Verzeichnisses in dem `BLOB`s zwischengespeichert werden (default: /srv/www/htdocs/blobs/)
7. Parameter '-o': der Name unter dem der Webserver im System läuft (default: wwwrun)
8. Parameter '-g:' der Name der Gruppe, unter dem der wbserver läuft (default: www)

Diese Eingaben werden von der Installationsroutine verarbeitet und daraus auch das eigentliche Startscript `dbengine.cgi` erstellt. Bei einer manuellen Installation ist auch dieses Script manuell zu erstellen. Es empfiehlt sich daher, die Installationsroutine zu benutzen.

Nach dem Ablauf von **`install.sh`** sind in der Regel noch einige Einträge in der dbengine_info_db-Datenbank zu korrigieren. Zur Benutzung der Datenbanken müssen die vorhandenen default-Werte geändert werden. Da dann das neue Tool noch nicht einsatzfähig ist, müssen diese Änderungen über das Kommando **`psql dbengine_info_db`** durchgeführt werden. Gleiches gilt für die Datenbank dbengine_test_db aus dem Verzeichnis `test`, sofern sie benutzt werden soll. Zusätzlich ist im Verzeichnis `test` noch das Schema einer ereignisse Datenbank, für die spezielle Aktionen programmiert wurden, vorhanden.

Als erstes muss die Tabelle `configs` dem aktuellen System angepasst werden, um die Liste der Benutzernamen, die als `admin` arbeiten dürfen zu setzen bzw. zu korrigieren. Um die Liste der `admin`-User zu erhalten, kann das Kommando

```
SELECT * FROM configs WHERE index = 'admins';
```

genutzt werden. Normalerweise trägt das Installationsskript den vorgegebenen Administrator in die Datenbank ein. Ist es eine leere Liste oder fehlt der richtige Administrator, so sind die aktuellen `admins` mit dem **`INSERT`**- sonst mit dem **`UPDATE`**-Kommando einzutragen. Um die Liste der `admin`-User zu ändern, kann das Kommando

```
UPDATE configs SET values='‹list of comma-separated-user-
names›' WHERE index='admins';
```

Ebenso muss die Tabelle `dbengine_user` geändert werden, um so die für die jeweiligen Datenbanken zulässigen Benutzer anzugeben. Wenn für eine Datenbank kein Benutzer eingetragen ist, so kann grundsätzlich jeder Benutzer (aus Sicht des Tools) diese Datenbank benutzen. Die Nutzungseinschränkungen liegen dann aber immer noch auf der Ebene von PostgreSQL. In jedem Fall aber werden die Datenbanken dem Benutzer angezeigt. Die Änderung geschieht analog der für die `admins`. Im folgenden ein kleines Beispiel dafür:

```
SELECT * FROM dbengine_user WHERE dbasename='‹name of the
database›';
```

So werden die vorhandenen User einer Datenbank angezeigt. Diese können dann mittels eines 'UPDATE' geändert werden, oder, falls kein Eintrag vorhanden ist, kann mit **`INSERT`** ein entsprechender angelegt werden. Das **`UPDATE`**-Kommando würde dann lauten:

```
UPDATE dbengine_user SET values='username' WHERE
dbasename='‹name of the database›';
```

Normalerweise ist wenigstens ein Satz schon in der Tabelle vorhanden, der nur für die `admins` den Zugriff auf die Datenbank `dbengine_info_db` gestattet.

Sollte das Verzeichnis für BLOBs vom Defaultwert abweichen, so muss auch dieses bekanntgegeben werden. Der Eintrag für diesen Wert muss dann ebenfalls in der Tabelle configs geändert werden:

```
UPDATE configs SET usrSaveBlob='‹name of Place for Blob-
files (Directory and base-url joind by :::)'›
```

Dabei steht der gleiche Wert auch in der Datei dbengine.pl und muss dort in gleicher Weise geändert werden. Als Beispiel für einen Eintrag sei hier der Default-Wert aus der Datei dbengine.cgi wiedergegeben (es ist eine Zeile des Eintrags in ein Hash-Array):

"usrSaveBlob"=> "/srv/www/htdocs/blobs:::blobs", # Place for Blob-files (Directory and base-url)

1.1.2. Resourcen

1.1.2.1. Perl-Resourcen

Zum Lauf des DBENGINE-Tools wird folgendes PERL-System benötigt:

1. PERL > Version 5.00
2. PERL-MODULES:
   - Crypt::CBC
   - MIME::Base64
   - DBI (nur für DBD::Pg getestet)
   - DBD::Pg
   - CGI..Carp
   - CGI

   Fehlt das eine oder andere Modul im System, so ist es aus der CPAN-Bibliothek nach zu laden.

1.1.2.2. Sonstige Resourcen

Zur Nutzung des DBENGINE-Tools müssen noch folgende zusätzliche Bedingungen erfüllt sein:

- Es muss ein System mit einem Webserver (z.B. der Apache-Server) vorhanden sein, auf dem das Tool als CGI-Programm laufen kann.
- Es muss ein PostgreSQL-Server (Version >= 8.0) von diesem System aus ansprechbar sein.
- Dem PosgreSQL-Server müssen alle Nutzer des Tools mit ihrem Benutzernamen und dem Passwort bekannt und in der Konfigurationsdatei pg_hba.conf muss zumindest folgende Erlaubnis – sofern der PostgreSQL-Server auf dem gleichen Rechner läuft, wie der Web-Server – eingetragen sein:

```
# TYPE  DATABASE    USER        CIDR-ADDRESS          METHOD
# "local" is for Unix domain socket connections only
local   all         all                               trust
# IPv4 local connections:
host    all         all         127.0.0.1/32          trust
# IPv6 local connections:
host    all         all         ::1/128               trust
```

PostgreSQL kann unterschiedliche Zeichensätze benutzen. Das DBENGINE-System stellt dabei für alle Datenbanken stets die Datenbankvariable client-encoding auf LATIN9 ein, unabhängig davon, mit welcher Codierung der Server läuft (festgelegt bei der Kreierung der Datenbank).

Sollen im Datenbanksystem eine oder mehrere Funktionen eingebracht werden (s. auch Abschnitt 2.2.5.1, „Die Datenbankfunktion htmltext“), so ist dafür zu sorgen, dass das Datenbanksystem die benutzte Sprache

kennt, in diesem Falle die Sprache `PERL`. Die Prozedur `htmltext` z. B. ist in PERL geschrieben. Um also derartige Prozeduren nutzen zu können, muss vorher für PostgreSQL und die benötigten Datenbanken die Sprache `PERL` kreiert werden. Dazu wird das Biblitheksmodul `libperl.so` benötigt und muss im `ld.so.cache` liegen, muss also in `/etc/ld.so.conf` eintragbar sein. Dies erreicht man am besten durch die Kreation einer Datei `pgperl.conf` mit dem Inhalt: `/usr/lib/perl5/5.10.0/i586-linux-thread-multi/CORE/`, dem Verzeichnis also, in dem die notwendige Datei liegt.

1.1.2.3. Der Webbrowser

Zur Nutzung des Systems durch den Anwender ist ein Webbrowser nötig, der „XHTML 1.0 strict“ analysieren und anzeigen kann (z.B. Firefox oder Konquerer). Zu beachten ist, das der Browser als erste Position für die Sprache *Deutsch de* besitzen sollte, damit bei der Sprachauswahl im System diese Sprache gefunden wird. Die bevorzugte Browsersprache nämlich dient zur Auswahl der Sprache im DBENGINE-System.

1.1.2.4. Sonstige Einstellungen

Bevor das DBENGINE-System dann gestartet werden kann, muss noch ein Bereich eingerichtet weren, in dem das Tool temporäre Dateien speichern kann. Dafür ist ein Verzeichnis `dbengine` unter dem Systembereich `/var/cache` einzurichten. In diesem Verzeichnis muss der Benutzer unter dem der Web-Server läuft (also meist `wwwrun`) Dateien kreieren, verändern und löschen dürfen.

Sofern in einer Datenbank `BLOB`s genutzt werden sollen, muss auch ein temporäres Verzeichnis für die Dateien dieser binären Objecte angelegt werden. Es muss zwei Bedingungen gehorchen:

- Der Inhalt dieses Verzeichnisses muss als URL zugreifbar sein, damit beim Auslesen einer `BLOB`-Datei der Webbrowser auf diesen zugreifen kann.
- Das Verzeichnis muss als Eigentümer der Benutzer eingetragen sein, unter dem der Webserver läuft, ebenso wie für das Verzeichnis für andere temporäre Dateien (s. Sonstige Einstellungen 1. Absatz [4]). Die Default-einstellung ist `/srv/www/htdocs/blobs/`, sie kann bei der Installation geändert werden, muss dann aber auch in der Datenbank `dbengine_info_db` in der Tabelle `configs` geändert werden (s. BLOB in der Tabelle configs [3]).

1.2. Test des Systems

Bei Erweiterungen oder auch spezifischen Einträgen in der Konfigurationsdatenbank in Form von Perl-Prozeduren ist es meist nötig, diese zu testen. Der Test in seiner Ausführlichkeit wird über einen Debuglevel in der Variablen `$debug` im Kopf des Moduls `dbengine.pl` gesteuert:

Tabelle 1.1. Wirkung der verschiedenen DEBUG-Level

Level	Wirkung
0	Keine Testausschriebe
1	Testausschriebe erfolgen in jedem Fall im Text der erzeugten HTML-Seite. Ein Beispiel dafür sind die SQL-Kommando-Ausschriebe beim Erzeugen eines Listings am Anfang des Listings.
2-10	Erhöhung der Intensität der Ausschriebe, je höher der Level, um so mehr Ausschriebe werden erzeugt. Durch Korrektur der Variablen '$debug_co' im Kopf des Moduls 'dbengine.pl' kann gesteuert werden, ob die Ausschriebe direkt erfolgen oder als Kommentar.
11	Höchster Level, dient auch zum Test der Apachekopplung ..., Alle Ausschriebe werden in eine Testdatei '/tmp/..db_debug.$$' geschrieben.

Zur Erzeugung von Testausschrieben stehen Prozeduren, die als 1. Parameter den DEBUG-Level bekommen und auswerten, zur Verfügung. Eine Liste, der vom Benutzer anwendbaren Prozeduren zeigt die folgende Tabelle, die den Prozedurnamen, die Parameter der Prozedur und als letztes eine kurze Beschreibung der Aufgabe der Prozedur enthält.

Tabelle 1.2. Tabelle der DEBUG-Prozeduren

Name	Aufrufparameter	Bedeutung
dbprint	(debuglevel, format, par1, par2, ...)	Format ist ein Perl-Format, das mit Parametern par1, ... versorgt werden kann.
dbprint_cmd_binds	(debuglevel, cmd, bv1, bv2, ...)	Ausgabe des SQL-commands cmd mit den zugehörigen Bind-values bv1, ...
dbprint_array	(debuglevel, name, rarray)	Ausgabe eines Arrays mit der Referenz 'rarray' unter dem Namen name.
dbprint_hash	(debuglevel, name, rhash)	Ausgabe eines Hash-Arrays mit der Referenz 'rhash' unter dem Namen name.
dbprint_array_hash	(debuglevel, name, rarray)	Ausgabe von Hash-Arrays, die unter der Array-Referenz 'name' gefunden werden.
dbprint_hash_hash	(debuglevel, name, rhash)	Ausgabe von Hash-Arrays, die unter den Schlüsseln des Hash-Arrays gefunden werden.

Kapitel 2. Der Aufbau des Bildschirms

In diesem Kapitel wird das grundsätzliche Aussehen der Clientseite beschrieben. Alle Daten, die vom Server kommen, werden auf einem Fenster des Webbrowsers dargestellt. Hier werden die verschiedenen Fenster beschrieben.

2.1. Das Startfenster

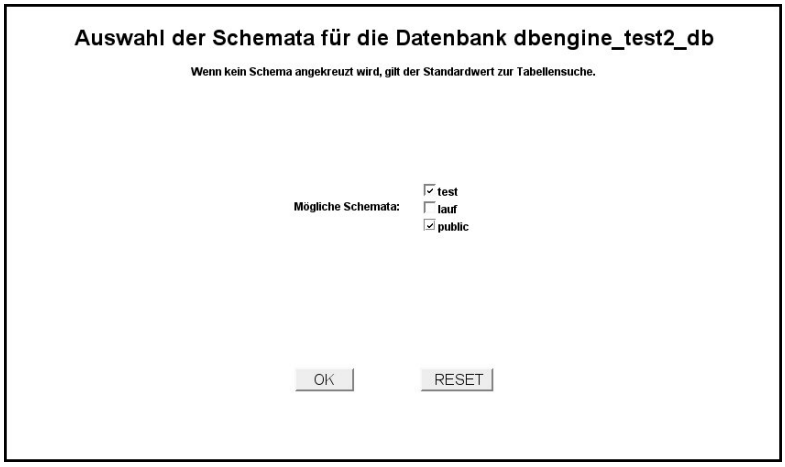
Um das Tool DBENGINE zu starten, muss auf der Clientseite im Webbrowser in der Regel die URL **`http://‹Servername‹/cgi-bin/dbengine.cgi`** angegeben werden. Das Perlmodul `dbengine.cgi` auf dem Server dient für alle Aufrufe als Verbindung zum Datenbanksystem. Der Server antwortet dann mit dem Startfenster. Sein Aussehen ist in Abbildung 2.1, „Startfenster von DBENGINE“ dargestellt.

Abbildung 2.1. Startfenster von DBENGINE

Vom Benutzer werden die Eingabe seines Zugangsnamens und seines Passwortes, unter dem der Benutzer Zugang zur gewünschten Datenbank hat, erwartet. Außerdem kann er aus einer Liste (dem Popupmenu) die Datenbank, die er bearbeiten möchte, auswählen. Welche Datenbank(en) er bearbeiten darf, kann der Administrator in der Tabelle `configs` der Konfigurationsdatenbank einstellen. Mit dem Drücken des OK-Buttons startet der Anwender das Datenbanksystem. Da das System nahezu zustandsfrei arbeitet, ist ein Ausloggen nicht nötig, wird aber empfohlen. Nach einiger Zeit der Nichtbenutzung (einstellbar ebenfalls über die Tabelle `configs`) werden die noch vorhandenen Anwenderdaten aus dieser Sitzung auf dem Server automatisch gelöscht.

Ist eine Datenbank mit mehreren Schemata angewählt, so muss durch den Administrator in der Datenbankspezifischen Konfigurationsdatei `databaseconfigs` unter der Bezeichnung `search_pathes` die Menge der Schemata angegeben werden, die der Benutzer bearbeiten darf. Ist keine Eingabe erfolgt, so gilt – sofern der Benutzer die Datenbank aufrufen darf – der Schemastandardwert: `public, $user`, wie er in dem Datenbanksystem `PostgreSQL` als `default`-Wert eingetragen ist. Andernfalls erscheint ein Fenster zur Auswahl der Schemata, die der Benutzer in dieser Sitzung anwenden kann. In der Abbildung 2.2, „Auswahl der Schemata“ sind zwei Schemata markiert: `test` und `public`.

Abbildung 2.2. Auswahl der Schemata



Dabei kann auch mehr als ein Schema ausgewählt werden. Wird keines gewählt, so gilt, wie oben gesagt, die Voreinstellung. Da in allen Schemata die gleichen Tabellenbezeichnungen auftreten können, dient bei den Aufrufen einer Tabelle, wenn mehr als ein Schema ausgewählt wurde, der Schemaname als Namensergänzung für die Tabelle.

2.2. Das Arbeitsfenster

Nach der Eingabe der Benutzerdaten antwortet das System mit dem Arbeitsfenster. Dieses ist in zwei Bereiche aufgeteilt: die Steuerleiste und den Arbeitsbereich. Die Steuerleiste dient der Auswahl der zu bearbeitenden Tabelle der gewählten Datenbank und auf dem Arbeitsbereich werden die Daten der ausgewählten Funktion dargestellt.

2.2.1. Die Steuerleiste

Die Steuerleiste nimmt als eigener Bereich stets die linke Seite des Arbeitsfensters ein und ist selbst wiederum in mehrere Bereiche unterteilt. Zunächst werden alle zugreifbaren Tabellen – unter Umständen in mehreren Gruppen – aufgelistet. Für Tabellen, die nicht zum Schema `public` gehören, wird als Tabellenname die übliche Kopplung von Schemaname und Tabellenname genutzt. Sind Tabellen über die Basiskonfiguration in der Konfigurationsdatenbank definiert, so werden sie in vorgegebenen Gruppen einsortiert und, sofern gewünscht, mit dem vorgegebenen – für den Benutzer besser assoziierbaren Titel versehen – eingetragen.

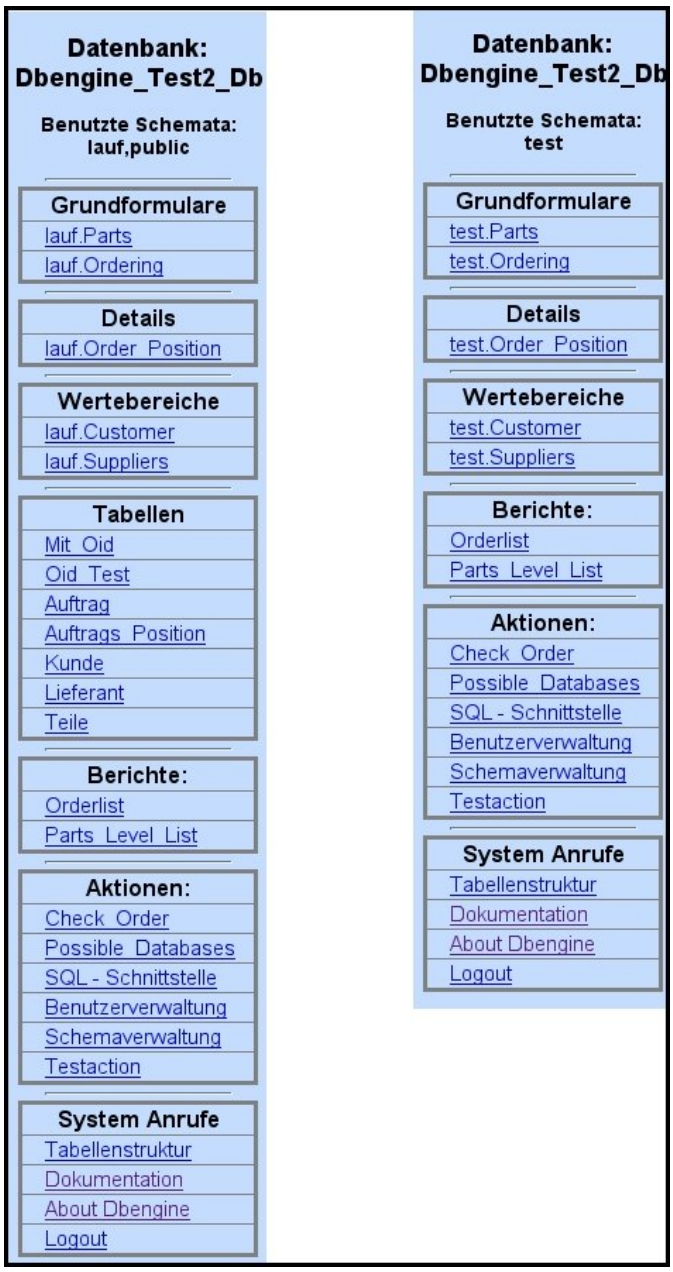
In der Abbildung 2.3, „Beispiele von Steuerleisten“ sind zwei Aufteilungen der Steuerleiste für die Beispieldatenbank `dbengine_test2_db` mit unterschiedlicher Auswahl der zu benutzenden Schemata abgebildet. Man sieht, dass die Tabellen in beiden Fällen in unterschiedlich viele Bereiche unterteilt sind, wie die folgende Liste für die gleiche Datenbank, aber mit unterschiedlichen Schemata, zeigt:

- **Fall 1:**
  - Grundformulare, Details und Wertebereiche
- **Fall 2:**
  - Grundformulare, Details, Wertebereiche
  - und zusätzlich Tabellen

In beiden Fällen gibt es die Zuordnung von Tabellen zu *`Grundformulare`*, *`Details`* und *`Wertebereiche`*. Im zweiten fall kommt noch die Gruppe *`Tabellen`* hinzu, was anzeigt, dass in diesem Fall nicht alle Tabellen der Datenbank über DBENGINE konfiguriert sind. Die Daten der Konfiguration

von Tabellen werden in der Tabelle `Basedesign` (siehe auch Abschnitt 3.1.1, „Die Tabelle basedesign (Tabellendefinition)“) der Konfigurationsdatenbank eingetragen. Alle Tabellen einer Datenbank, die nicht in der Konfiguration eingetragen sind, werden schließlich unter der zusätzlichen Rubrik, **`Tabellen`**, mit ihrem Tabellennamen, gegebenenfalls inklusive dem Schemanamen, eingetragen. Die Unterscheidung der Gruppen dient nur der Übersichtlichkeit, intern werden alle Tabellen gleich behandelt.

Abbildung 2.3. Beispiele von Steuerleisten



Im unteren Teil der Steuerleiste sind noch drei weitere Bereiche vorhanden. Es sind

- Aktionen und
- Berichte, sowie
- System Anrufe

Zusätzlich zu den Tabellen werden in diesen Gruppen die dem Benutzer zur Verfügung stehenden Datenbankaktionen und Berichte aufgeführt. Sie können über die Konfigurationsdatenbank für die Benutzer unterschiedlich freigegeben sein. Die Trennung in Aktionen und Berichte ist wie die Aufteilung der Tabellen eine Aufgabe der Konfiguration. Die Methode der Bearbeitung ist in beiden Fällen gleich: es werden durch die Konfiguration festgelegte Funktionen gestartet. Die Konfiguration geschieht über die Tabelle 'Actions' der Konfigurationsdatenbank. Zu den Aktionen gehören stets die sogenannten „Standardaktionen“:

- *`SQL Schnittstelle`*: Bei dieser Aktion wird dem Benutzer eine vollständige SQL Benutzung ermöglicht.
- *`Benutzerverwaltung`*: Mit dieser Aktion kann der Benutzer weiteren Benutzer Erlaubnisse erteilen, oder Erlaubnisse ändern.
- *`Schemaverwaltung`*: Mit dieser Aktion kann der Eigentümer der Datenbank (oder der Administrator) weitere Schemata in seiner Datenbank einrichten.

In dem letzten Bereich sind Aufrufmöglichkeiten für Systemfunktionen untergebracht:

- *`Tabellenstruktur`*: Eine Ausgabe der Struktur der Tabellen mit ihren Spalten kann angefordert werden.
- *`Dokumentation`*: Eine Dokumentation (dieses Handbuch) kann im HTML-Format ausgegeben werden.
- *`About Dbengine`*: Eine kleine Ausgabe (in einem extra Fenster) mit der Angabe der Versionen der Hauptbestandteile des DBENGINE-Systems kann angefordert werden.
- *`Logout`*: Mit Anklicken von *`Logout`* kann ein Beenden des Laufs erzwungen werden. Es erscheint danach wieder der Startbildschirm. Diese Aktion ist nicht unbedingt nötig, aber doch empfehlenswert.

Die ersten beiden Aufrufe erscheinen im bisherigen Arbeitsfenster. Es empfiehlt sich, dafür ein erneutes Fenster zu öffnen, damit man diese Angaben stets zur Verfügung hat.

2.2.2. Die Eingabemaske

Wählt der Anwender aus der Steuerleiste eine Tabelle aus, so erscheint die Eingabemaske. Sie dient drei Aufgaben:

- Der Suche nach Sätzen in der gewählten Tabelle,
- Der Neueintragung von Sätzen in der Tabelle und
- Der Änderung einzelner Sätze der Tabelle.

Die Eingabemaske ist dazu in vier Bereiche unterteilt, die Eingabefelder, die Funktionsknöpfe, einen Querverweisbereich Abschnitt 2.2.2.3, „Der Querverweisbereich“ auf andere Tabellen und ein Eingabefeld für einen eigenen speziellen **`SELECT`**-Befehl. Diese Bereiche werden nun im Einzelnen kurz erläutert. Weitergehende Erläuterungen über die Bedeutung der Bereiche ergibt sich auch aus den Masken der Beispieldatenbank dbengine_test2_db.

2.2.2.1. Die Eingabefelder

Die Eingabefelder dienen verschiedenen Aufgaben: der Suche nach Sätzen aus der Datenbank (Suchmaske) oder dem Eintrag neuer bzw. der Änderung vorhandener Datensätze (Ergebnismaske).

Für jedes Tabellenfeld der ausgewählten Tabelle ist in der Regel ein Eingabefeld vorhanden. Die Anordnung und die Art der Bearbeitung der Felder geschieht über die Konfigurationstabelle `desinginfo` (s. Die Tabelle designinfo). Sind Felder dort nicht definiert, so werden sie an das Ende der Eingabefelder gestellt und dort entsprechend ihres Datentyps für Eingaben vorgesehen.

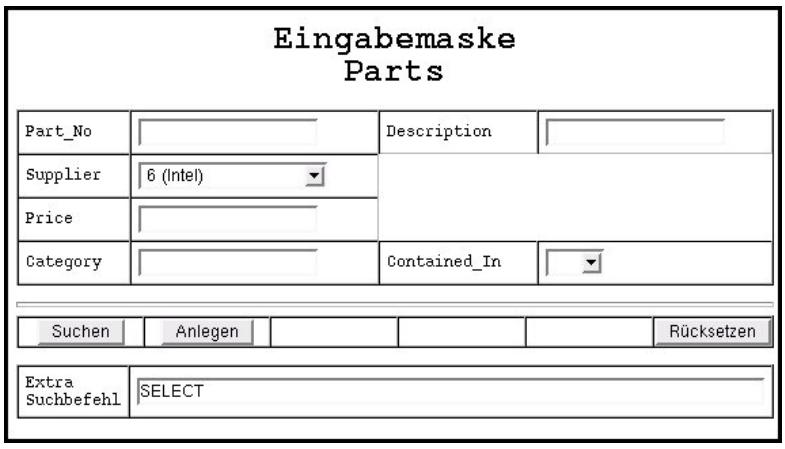
Neben den Tabellenfeldern können in diesem Bereich auch berechnete, virtuelle, Felder erzeugt werden. Ihr Aussehen und ihre Berechnung erfolgt über die Konfigurationsdatei `virtual` (s. Abschnitt 3.2.2,

„Die Tabelle virtual (Definition berechneter Felder)“) zusammen mit der Beschreibung in `designinfo`. In letzterer Datei haben diese Felder stets den Datentyp `VIRTUAL`.

Sollen einzelne Felder nicht ausgegeben werden, so bekommen sie in 'designinfo' den Datentyp `NONE`.

Die anfängliche Eingabemaske enthält normalerweise noch keine Daten (eine Ausnahme können durch die Konfiguration festgelegte `Default`-Werte sein). Es erscheint also ein Bild der Eingabefelder mit leeren Eingabemasken. Auch die virtuellen Felder enthalten noch keine Daten, da Werte für diese Felder normalerweise in Abhängigkeit von den Werten eines Datensatzes errechnet werden. In der Abbildung 2.4, „Beispiel einer Suchmaske“ ist eine Suchmaske für die Beispieldatenbank dargestellt. Im Popupfeld für das Feld `Supplier` ist schon ein Wert aus der vorhandenen Liste möglicher Lieferanten aus der Tabelle Supplier ausgewählt. Für das Popupfeld wurde aus der Liste die Zahl 6 und in Klammern den Wert `Intel` ausgewählt. Dies geschieht durch die Verklammerung von Wert und Beschreibung in der Konfiguration über die Konfigurationstabelle `designinfo`.

Abbildung 2.4. Beispiel einer Suchmaske



Wird in dieser Maske nun der Button *[Suchen]* betätigt, so wird in der Datenbank nach Sätzen gesucht, die das Suchmuster befriedigen. Dabei können in einfachen Eingabefeldern neben den Werten auch Bedingungsoperatoren angeben werden. Welche Bedingungsoperatoren angegeben werden können, ist Datentyp spezifisch. Die möglichen Operatoren und die zugehörigen Datentypen sind in der Tabelle 2.1, „Die Bedingungsoperatoren“ aufgelistet:

Tabelle 2.1. Die Bedingungsoperatoren

Operator	**Symbol**	**Zulässige Datentypen**
Gleich	=	Char, Text, Int, Float, Numeric, Date
Ungleich	<=>	Char, Text, Int, Float, Numeric, Date
Kleiner	<	Int, Float, Numeric, Date
Kleiner oder Gleich	<=	Int, Float, Numeric, Date
Größer	>	Int, Float, Numeric, Date
Größer oder Gleich	>=	Int, Float, Numeric, Date
Match case-sensitiv (regular expression)	~	Char, Text, Int, Float, Numeric, Date
Match case-insensitiv (regular expression)	~*	Char, Text
Not Match case-sensitiv (regular expression)	!~	Char, Text

Operator	Symbol	Zulässige Datentypen
Not Match case-insensitiv (regular expression)	!~*	Char, Text
Entsprechend dem LIKE-Operator	~~	Char, Text
Entsprechend dem LIKE-Operator	!~~	Char, Text

Die Bedingungsoperatoren können nur in Feldern eingegeben werden, die eine normale Eingabe zulassen. Nach der Eingabe der Bedingungsoperatoren können beliebig viele Leerstellen vor der eigentlichen Bedingung folgen. Die Bedingung selbst kann, wie bei SQL in vielen Fällen nötig, in „'“ eingeschlossen sein; dies ist aber nicht erforderlich. Nach einem Operator werden die einschließenden „'“ entfernt. Für einige Datentypen gibt es noch Besonderheiten zu beachten:

- Datumsfelder sind 10 Zeichen breit. Damit trotzdem Bedingungsoperatoren eingegeben werden können, können zum einen 25 Stellen eingegeben werden – nötig für eine Bereichseingabe – und außerdem kann die Jahreszahl zweistellig angegeben werden. Dann bedeuten Werte über 45 Jahre ab 1900 und Werte unter 46 sind Jahre ab 2000. Außerdem können in Datumsfeldern spezielle Datenbereiche angegeben werden. Diese Möglichkeiten zeigt die Tabelle 2.2, „Mögliche Datumsbereichangaben“:

Tabelle 2.2. Mögliche Datumsbereichangaben

Angabe	Bereichswerte	Beispiel Eingabe	Beispiel Wertebereich
mm.jj oder mm.jjjj (1>=mm>=12)	Monat mm des Jahres jj	06.02	>= 01.06.2002 AND > 01.07.2002
Qx.jj oder Qx.jjjj (1>=x>=4)	Quartal x des Jahres jj	Q2.98	>= 01.04.1998 AND < 01.07.1998
Hy.jj oder Hy.jjjj (1>=y>=2)	Halbjahr y des Jahres jj	H2.99	>= 01.07.1999 AND < 01.01.2000
Jj oder jjjj	Jahr jj	96	>= 01.01.1996 AND < 01.01.1997
[d1 – d2]	Zeitraum von Datum d1 bis Datum d2	[01.02.78-31.10.79]	>=01.02.1978 AND <= 31.10.1979

- In Character- oder Textfeldern eingegebene Werte werden standardmäßig als „Case-insensitiv-regular-expression“ aufgefasst. Ist allerdings in dem Feld ein „%“ oder ein „_“ eingegeben, so wird die Eingabe eines SQL-Jokerzeichens vermutet und der Vergleich mit dem Operator `LIKE` durchgeführt. Soll eine Identitätsprüfung erfolgen, so muss als erstes Zeichen der Identitätsoperater „=“ mit eingegeben werden. Wird eine Auswahl aus einem *`Multiplepopupfenster`* (s. multiplepopup) als Suchbedingung gewählt, so werden alle ausgewählten Einträge als eine Oder-Bedingung aufgefasst und daher im Select-Statement mit `OR` verknüpft. Eine Angabe von Bedingungen in mehreren Feldern hingegen wird als Und-Bedingung aufgefasst, und dementsprechend werden die einzelnen Suchkriterien im **`Select`**-Statement mit `AND` verbunden.
- Wird eine Auswahl aus einem *`Multiplepopupfenster`* als Suchbedingung gewählt, so werden alle ausgewählten Einträge als eine Oder-Bedingung aufgefasst und daher im **`Select`**-Statement mit `OR` verknüpft. Eine Angabe von Bedingungen in mehreren Feldern hingegen wird als Und-Bedingung aufgefasst, und dementsprechend werden die einzelnen Suchkriterien im **`Select`**-Statement mit `AND` verbunden.

Sollen Bedingungen zu Suche angegeben werden, die durch Eingabe in den einzelnen Feldern nicht erfüllt werden können, so kann im Feld *`Extra Suchbefehl`* ein SQL-SELECT-Statement angegeben werden. Dies unterliegt einigen Einschränkungen: im `FROM`-Teil der Anweisung muss als erste Tabelle die Tabelle auftreten, deren Maske angewählt wurde. Auch darf kein „Subselect“ in der Anweisung benutzt werden.

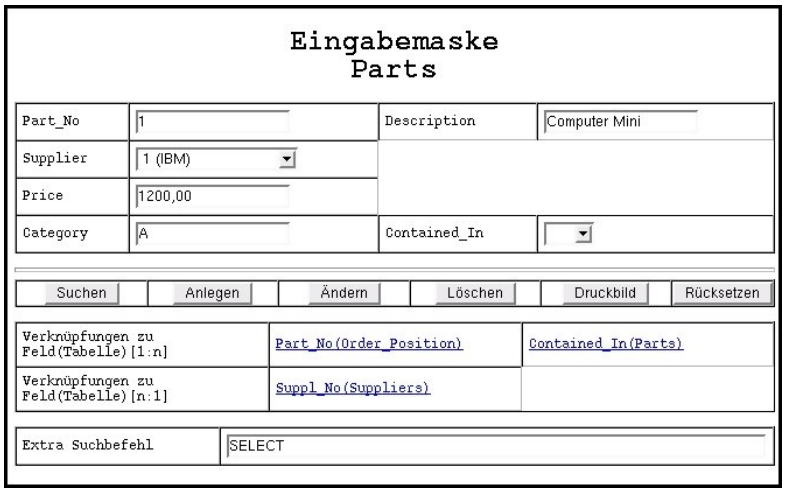
Gleichgültig, wie die Suche erfolgen soll, ob über die Eingabe in den Eingabefeldern oder über einen gezielten SQL-SELECT: Ein Betätigen des *[Suchen]* – Buttons bewirkt dann eine Suchen nach Datensätzen, die dann entweder als Liste von Tabellensätzen (s. Abbildung 2.6, „Ergebnisliste einer Suche“ in Abschnitt 2.2.2.2, „Die Funktionsknöpfe“) oder als Ergebnismaske (s Abbildung 2.5, „Beispiel einer Ergebnismaske“) erscheinen. Werden mehrere Sätze gefunden, so wird eine Liste von Tabellensätzen (s. Abbildung 3.1, „Liste der Tabellenzuordnung von dbengine_info_db“) erzeugt. Wird in der Datenbank nur 1 Satz gefunden, der die Suchbedingen erfüllt, so erscheint er in der Ergebnismaske. Dies kann für eine Datenbank dadurch verhindert werden, dass für diese Datenbank in der Konfiguration der Wert `single_sentence` auf `true` gesetzt wird.

Will man keine Sätze suchen, sondern nur neue anlegen, so kann dies durch Eintragen der Werte in die Tabellenfelder geschehen und durch anschließendes Betätigen des *[Anlegen]*-Buttons. Die Ergebnismaske hat im ersten Teil den gleichen Aufbau wie die Suchmaske. Nur sind jetzt die Eingabefelder ausgefüllt, soweit Daten in der Datenbank für diesen Satz vorhanden sind. Auch erlaubt die Ergebnismaske mehr Operationen. Aber die Felder können wie bei der Suchmaske beliebig verändert werden und für eine erneute Suche genutzt werden.

In der Ergebnismaske sind häufig auch berechnete Felder vorhanden, Felder vom Konfigurationstyp `VIRTUEL`, die erst für die Darstellung auf dem Bildschirm berechnet werden. Diese Felder können nicht manipuliert werden.

Das gleiche gilt für Felder, die zwar Bestandteil der Datenbank sind, aber dort aus anderen Werten berechnet werden. Für diese kann in der Konfiguration angegeben werden, dass sie nur gelesen, nicht aber verändert werden dürfen.

Abbildung 2.5. Beispiel einer Ergebnismaske



2.2.2.2. Die Funktionsknöpfe

In der Suchmaske sind nur 3 unterschiedliche Button zur Auswahl der durchzuführenden Aktion angegeben: *[Suchen]*, *[Anlegen]* und *[Rücksetzen]*.

In der Ergebnismaske hingegen kommen noch 3 weitere Button hinzu: *[Ändern]*, *[Löschen]* und *[Druckbild]*. Bei dieser Maske liegt ja ein spezifischer Datensatz aus der angewählten Tabelle vor. In den Eingabefeldern können nun Änderungen vorgenommen werden und durch Betätigen des *[Ändern]*-Buttons in die Datenbank eingebracht werden, bzw. über den Anlegen – Button als neuer Satz in die Datenbank eingetragen werden. Ein Datensatz kann über den *[Löschen]*-Button aus der Datenbank entfernt werden. Soll ein Datensatz ausgedruckt werden, so kann mit Hilfe des *[Druckbild]*-Button eine Web-Seite erstellt werden, die im A4-Format ausgedruckt werden kann (ohne Datenverlust). Der *[Rücksetzen]*-Button dient

dazu, alle in den Eingabefeldern vorgenommenen Eingaben rückgängig zu machen und den Zustand wieder herzustellen ,der vom Server an den Browser übermittelt wurde.

2.2.2.3. Der Querverweisbereich

Der Querverweisbereich ist nur in der Ergebnismaske (s. Abbildung 2.5, „Beispiel einer Ergebnismaske“) vorhanden, da dieser Bereich Verweise von einem speziellen Datensatz der angewählten Tabelle auf andere Datensätze in anderen Tabellen enthält. Die Inhalte dieses Bereiches werden gesteuert durch die Konfigurationstabelle `relations` (s. Abschnitt 3.2.1, „Die Tabelle relation (Definition von Tabellenbeziehungen)“), in der die Beziehungen zwischen Tabellen eingetragen werden können. Der Bereich wird auch in der Ergebnismaske nur ausgegeben, sofern Beziehungen zwischen der angewählten und anderen Tabellen bestehen.

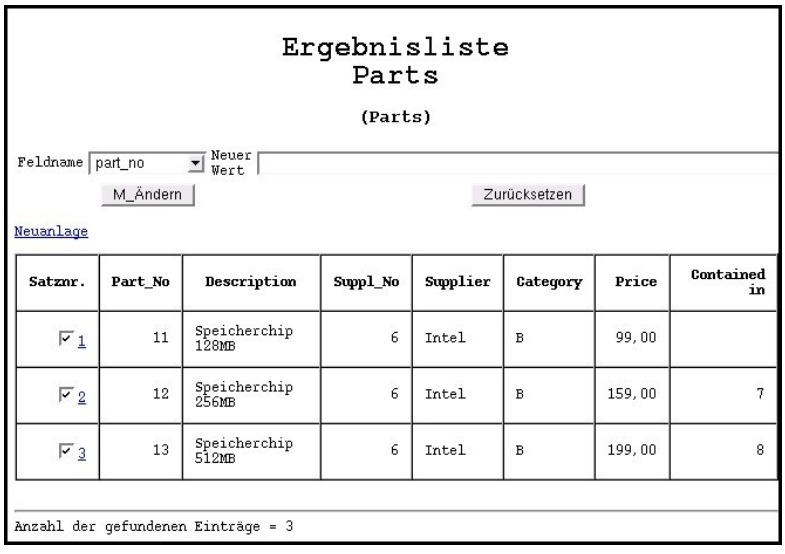
Der Bereich ist in zwei Unterbereiche unterteilt:

- Im ersten Bereich werden die Beziehungen zu den Tabellen ausgegeben, die die aktuelle Tabelle referenzieren (die aktuelle Tabelle ist der Vater der anderen Tabellen, bzw. es herrscht eine 1:n Beziehung zwischen den Tabellen).
- Im zweiten Bereich werden die reversen Beziehungen ausgegeben, d.h. Die Beziehungen, bei denen die aktuelle Tabelle die anderen referenziert (die aktuelle Beziehung ist Kind der anderen Tabelle, bzw. es herrscht eine n:1 Beziehung zwischen den Tabellen).

Durch ein Anklicken einer der aufgeführten Beziehungen wird dann ein Suchvorgang in der entsprechenden Tabelle mit dem aktuellen Wert im Beziehungsfeld durchgeführt. Werden Datensätze in der anderen Tabelle gefunden, so werden diese angezeigt wie nach einer Suche über die Suchmaske.

2.2.3. Die Liste von Tabellensätzen

Abbildung 2.6. Ergebnisliste einer Suche



Für das Anzeigen der Sätze selbst gibt es zwei durch die Konfiguration der Tabelle vorgebbare Varianten (s. Abbildung 2.6, „Ergebnisliste einer Suche“):

- Mehrere Sätze der Tabelle dürfen auf einmal geändert werden (s. **`multiple update`** bzw. *`modifyablelist`* Tabelle 3.8, „Die Felder der Tabelle tabledesign“) oder

- Nur einzelne angewählte Sätze dürfen geändert werden.

Im ersten Fall sind vor den eigentlichen Sätzen zwei Felder angelegt, in denen das zu ändernde Feld und der neue einzutragende Wert angegeben werden können. Geändert werden alle Sätze, die über die Satznummer-Kontrollkästchen ausgewählt wurden.

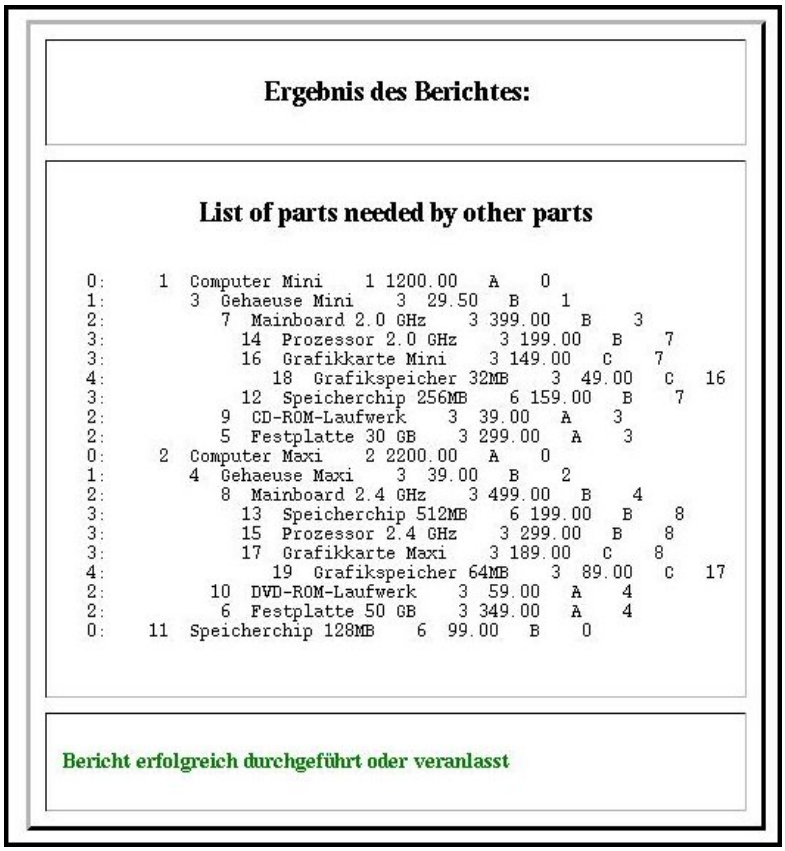
Die Liste der Sätze enthält alle ausgewählten Felder. Werden keine konfiguriert, so werden alle Felder angezeigt. Sollen auch berechnete Felder angezeigt werden, so kann dass über die Konfigurationsdatenbank in den Tabellen `virtual` und `tabledesign` vorgegeben werden (s. Abschnitt 3.1.3, „Die Tabelle tabledesign (Listingdesign)“ und Abschnitt 3.2.2, „Die Tabelle virtual (Definition berechneter Felder)“). Auch die Anordnung und Listenüberschrift kann explizit konfiguriert werden.

In der ersten Ausgabespalte stehen neben dem Marker für die Mehrfachveränderung (*`multiple update`*) eine Ordnungsnummer. Normalerweise ist diese als Hyperlink gestaltet, um den Satz in eine Eingabemaske verändern zu können. Für den Hyperlink dient, sofern vorhanden, der primary key. Ist er nicht vorhanden, so wird bei `PostgreSQL` versucht, das `oid`-Feld zu nutzen. Ist auch dieses nicht vorhanden, so wird kein Hyperlink erzeugt.

2.2.4. Aktionen und Berichte

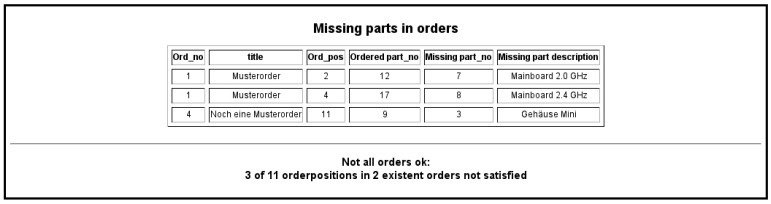
In der Steuerleiste gibt es zwei weitere Bereiche: den Bereich *`Aktionen`* und den Bereich *`Berichte`*. Diese beiden Bereiche haben Systemintern die gleiche Bedeutung, sie geben nur dem Anwender Hinweise darüber, was sich hinter den angezeigten Möglichkeiten verbirgt.

Abbildung 2.7. Beispiel eines Berichtes



Ein Bericht oder eine Aktionen für eine Datenbank muss in der Konfigurationsdatenbank als Perl-Funktionen eingetragen sein. Wenn ein Bericht beispielsweise sofort erzeugt werden kann, so kann er auch unmittelbar im Berichtsfenster (s. Abbildung 2.7, „Beispiel eines Berichtes“) ausgegeben werden. Wird die Anzeige des Ergebnisses vollständig in der Perl-Funktion in der Konfigurationsdatenbank gesteuert, so ist für diese Aktion, diesen Bericht, der Wert für das Feld `ext_result` auf `true` zu setzen. Dann wird der Rahmen nicht angezeigt (s. Abbildung 2.8, „Beispiel eines extern definierten Berichtes“). Es kann aber – dies gilt vor allem für Hintergrundsaktionen – auch geschehen, dass solche Aktionen nur angestoßen werden. Dann erfolgt eine Ausgabe, dass die entsprechende Aktion veranlasst wurde.

Abbildung 2.8. Beispiel eines extern definierten Berichtes

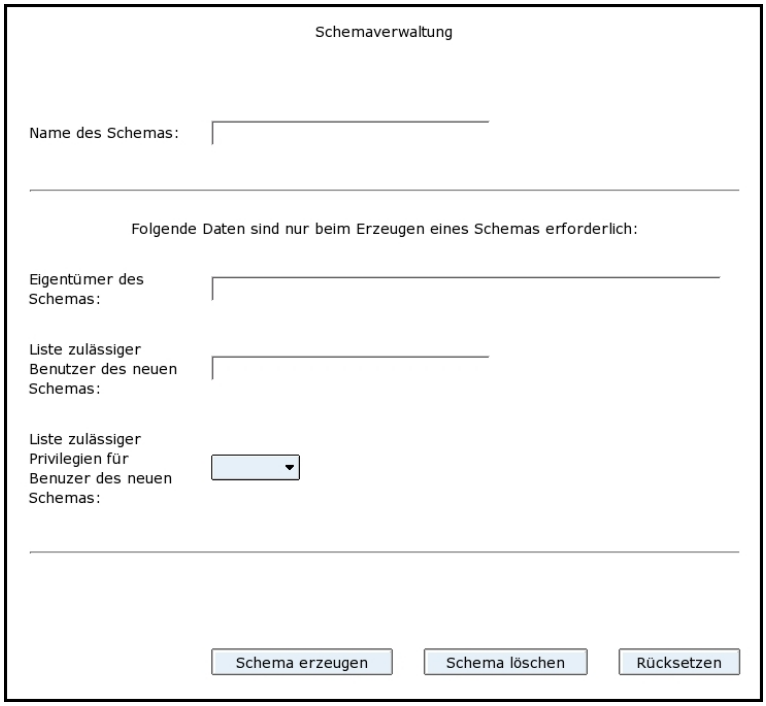


Neben den datenbankspezifischen Aktionen gibt es auch Standardaktionen (s. Abschnitt 3.1.4, „Die Tabelle action (Aktions-/Berichtsdefinition)“), die i. a. für alle Datenbanken gelten. In der Konfigurationsdatenbank sind z. Z. 4 Standardaktionen enthalten:

- Die Datenbankverwaltung, sie dient der Erzeugung und Löschung von Datenbanken und darf daher nur vom Administrator benutzt werden.

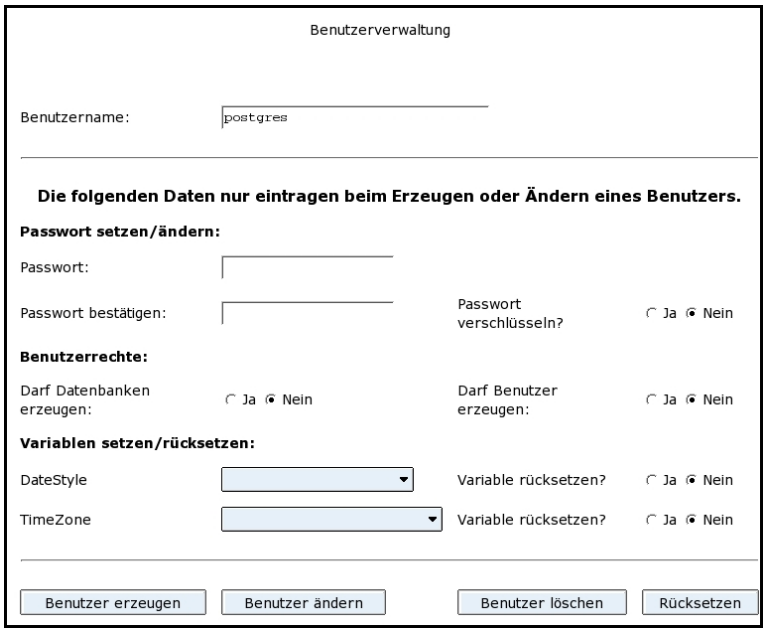
- Die Schemaverwaltung, sie dient der Erzeugung und Löschung von Schemata in der aktuell angewählten Datenbank und darf daher von dem Administrator und dem Besitzer der Datenbank aufgerufen werden. Ein Beispiel für einen solchen Aufruf zeigt die Abbildung 2.9, „Standardaktion Schemaverwaltung“

Abbildung 2.9. Standardaktion Schemaverwaltung



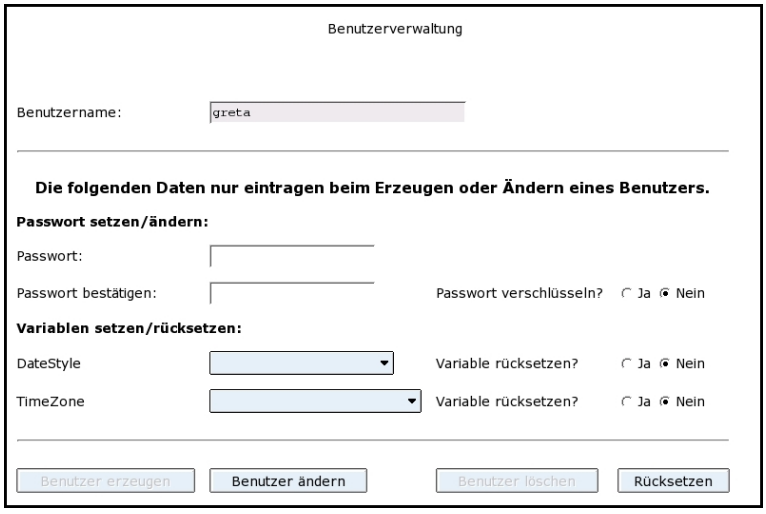
- Die Benutzerverwaltung: mit dieser Standardaktion (s. Abbildung 2.10, „Maske für die Benutzerverwaltung durch Administratoren“ können vom Administrator neue Benutzer angelegt und bereits vorhandene Benutzer verwaltet werden. Jeder Benutzer kann außerdem seine eigenen Daten mit dieser Standardaktion verwalten.

Abbildung 2.10. Maske für die Benutzerverwaltung durch Administratoren



Ist der aktuelle Benutzer kein Administrator, so sieht die Maske entsprechen der Abbildung 2.11, „Maske für die Benutzerverwaltung durch Benutzer“ aus. Zum einen ist der Benutzername nicht veränderbar – zu sehen an dem grau unterlegten Feld für den Namen –, zum anderen fehlen Eingabemöglichkeiten, die nur der Administrator vergeben darf.

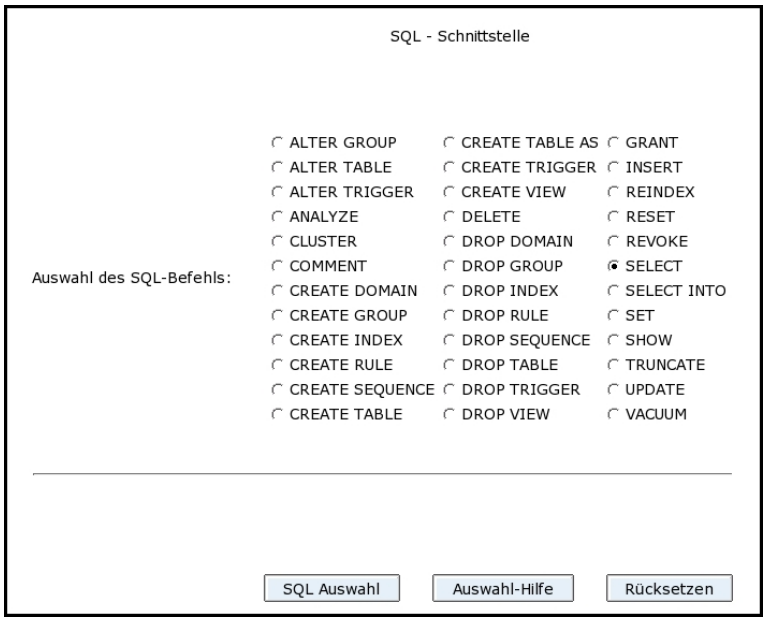
Abbildung 2.11. Maske für die Benutzerverwaltung durch Benutzer



- Die SQL-Schnittstelle: mit dieser Standardaktion wird allen zugelassenen Benutzern die generelle SQL-Schnittstelle mit Ausnahme der SQL-Befehle, die durch die anderen drei Standardaktionen abgedeckt

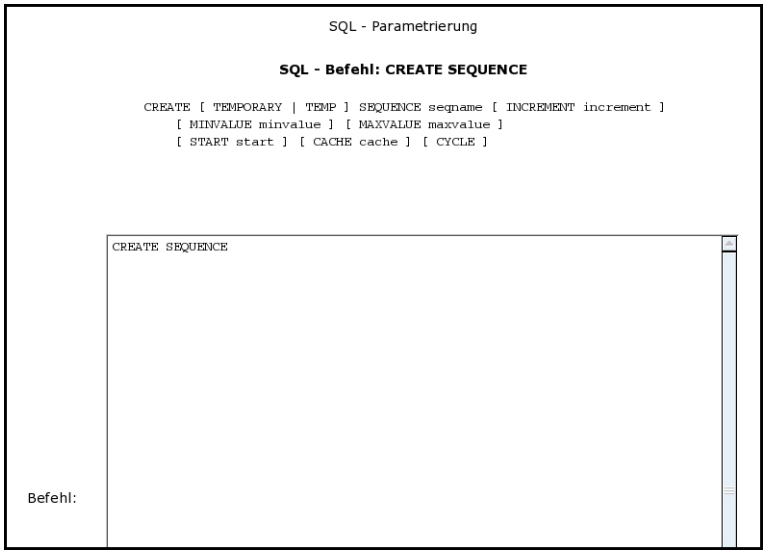
werden, freigegeben. Mit der Freigabe sollte vorsichtig umgegangen werden, aber in jedem Fall kann der Eigentümer der aktuellen Datenbank und der Administrator das Recht bekommen.

Abbildung 2.12. Befehlsauswahlmaske der SQL-Schnittstelle



Ein Beispiel für die SQL-Schnittstelle zeigt die Abbildung 2.12, „Befehlsauswahlmaske der SQL-Schnittstelle“ Es werden alle SQL-Befehle angezeigt, die ausgewählt werden können und zur Auswahl stehen zwei Buttons zu Verfügung: der erste, *[SQL-Auswahl]*, gibt die Möglichkeit zur Befehlseingabe ohne Unterstützung, der zweite, *[Auswahl-Hilfe]*, gibt auf der Folgemaske außerdem als Unterstützung die Syntax des Befehls aus. Dennoch sollte der Anwender dieser Schnittstelle mit SQL vertraut sein, denn die Syntax allein gibt natürlich noch keine oder nicht viele Hinweise auf die Semantik eines Befehls.

Abbildung 2.13. Beispiel für einen SQL-Befehl mit Hilfeangabe



In der Abbildung 2.13, „Beispiel für einen SQL-Befehl mit Hilfeangabe“ ist der Befehl **CREATE SEQUENCE** mit Angabe der Syntax ausgewählt worden. Diese Syntaxbeschreibung kann manchmal sehr umfangreich sein. Sie ist aus den Hilfemodi des postgtreSQL-Befehls **psql** entnommen.

2.2.5. Formatierungshilfen für große Textfelder

2.2.5.1. Die Datenbankfunktion `htmltext`

Die Funktion stellt eine Menge von Abkürzenden Verfahren zur Verfügung, um damit eine große Zahl der Textausgaben in XHTML-Format ausgeben zu können. Einige werden direkt umgesetzt, d.h. es wird nur eine verkürzte Schreibweise angeboten. Dabei handelt es sich um folgende Abkürzungen:

Tabelle 2.3. Tag Abkürzungen für TEXTAREAs

Abkürzung	erzeugte Dastellung	Bedeutung
` `	` `	Aus einem Zeilenwechsel für HTML wird der für XHTML notwendige erzeugt. Ein Endetag ist nicht erlaubt.
`<ep>`	`  `[5*]	Es wird ein Zeilenwechsel mit Einzug von fünf Leerstellen ( ) erzeugt. Ein Endetag ist nicht erlaubt.
`<pt>`	`<p class="text">`	Es wird ein Absatz mit Einzug erzeugt, wie er in den Cascadet- Stylesheet-Definitionen (s. Abschnitt 2.2.5.2, „Cascadet Stylesheet von DBENGINE“eingetragen ist. Als Endetag ist hierfür ein </pt> nötig.
`<bq>`	`<blockquote>`	Das abkürzende Tag `bq` ist eine reine Abkürzung für das offizielle Tag `blockquote`. Als Endetag ist hierfür ein </bq> nötig.
-	`—`	Aus einem einfachen Minuszeichen wird ein langer Gedankenstrich erzeugt, sofern das Minuszeichen in Leerstellen eingeschlossen ist.
...	`…`	Für „...“ wird das HTML-Symbol erzeugt, das diese Punkte verstärkt und etwas zusammengezogen, also als 1 Zeichen darstellt.
`&#xA`	`  `	Ein Zeilenvorschub, gegeben von der Tastatur, wird durch einen XHTML-Zeilenvorschub ersetzt. Voraussetzung dafür ist, dass dieser Zeilenvorschub nicht innerhalb eines Tabellen- oder Listen-Tags erfolgt. Mehrfache Leerstellen vor dem eingegebenen Zeilenwechsel werden durch eine einfache Leerstelle gefolgt von entsprechend vielen gebundenen Leerstellen ( ) ersetzt.

Neben diesen reinen Abkürzungen gibt es insbesondere für Listen und Tabellen noch vereinfachende Schreibweisen:

- **Tabellen:** Innerhalb von Tabellen, gekennzeichnet durch <table> und </table>, werden Zeilenwechsel ersetzt durch eine neue Tabellenzeile (<row> ... </row>). Der Sprung von einer Tabellenspalte zur nächsten hingegen kann durch ein Tabulatorzeichen erzeugt werden
- **Listen:** Bei Listen werden zwei Verfahren unterschieden:

- Die Definitionsliste, sie wird abgebildet auf und
- Die geordnete bzw. die ungeordnete Liste.

In jedem der beiden Fälle dient der Zeilenwechsel zur Kennzeichnung eines neuen Listenelementes. Bei der Definitionsliste wird durch die eingabe eines Tabulatorzeichens zusätzlich die Listenelement-bezeichnung getrennt von der Definition dieses Listenelementes.

Diese Funktion muss in der Datenbank, in der sie für große Textfelder benutzt werden soll, definiert werden (s. textarea)

2.2.5.2. Cascadet Stylesheet von DBENGINE

Zusätzlich zu der eben genannten Routine können auch die Formatierungen durch das vorhandene CSS (Cascadet Style-Sheet) genutzt werden. Zur leichteren Benutzung sei hier die aktuelle Version abgedruckt:

```
<style type="text/css">
 div.actionErg {line-height: 1.4em;
   text-align: center;
   padding: 0.5em;
   border:solid 3px;}
 div.retVal {line-height: 1.4em;
   text-align: left;
   padding: 0.5em;}
 div.actinh {text-align: left; line-height: 1.2em}
 div.texdl {text-align: left;
   margin: 0.5em 0 0.5em 1em;}
 div.texdd {text-align: left;
   text-indent: 1em;
   margin-left: inherit;
   padding: 0 0 0 1em;}
 #navi {display: block;}
 div#navi {float: left;
   width:185px;
   height: 100%;
   overflow: auto;
   background-color: $$rconfigs{'bgmcol'};
 }
 div.menu {  margin-left: 5%;
   margin-right: 5%;
   text-align: center;
 }

 div#inhalt {  margin-left: 19px;
   overflow: auto;
   background-color: $$rconfigs{'bgcol'};
 }
 div.tabs {  width: 700px;
   margin-left: 5%;
   margin-right: 5%;
   text-align: center;
   font-size: 1.25em;
 }
 div.ctab {text-align: center;}
 div.werte { text-align: left;}
```

```
div#about {  width: 350px;
  margin-left: 5%;
  margin-right: 5%;
  text-align: center;
}

q:lang(de) {quotes: '»' '«' '"' '"' '"' '"';}

table.texfield {  margin: 0 0 0 2em;
  padding: 0.5em 1em 0.5 em 1em;
  border-spacing: 4px;
  border: solid 1px black;}
th.texfield, td.texfield { padding: 4px; vertical-align: top;}

table.listing { margin: 0;
  padding: 6;
  width: 100%;
  border: solid 1px black;
  margin: 0;}

table.menu {font-size: 1.25em;
  width: 100%;
  padding: 4;
  border: solid 3px gray;
  text-align: left;}
th.menu {font-size: 1.1em;
  text-align: center;}
td.menu {margin-left: 5%;
  margin-right: 5%;
  text-indent: 1em;}
table.mr {text-align: left;
  padding: 4;
  border: solid 1px black;
  margin:0;}
table.or {text-align: left;
  padding: 4;
  border: solid 0px black;
  width: 100%;
  margin:0;}
th.left { text-align: left;}

p.text {text-indent: 2em;}
p.txt2 {text-indent: 2em; margin: 0;}
p.txt3 {margin: 0;}
p.actErg {font-size: 1.25em;
        font-weight: bold;
   text-align: center;
   line-height: 120%;}
p.acterg {font-size: 1.25em;
        font-weight: bold;
   text-align: center;
   line-height: 120%;
   margin: 0;}
p.fehler {color: red; font-size: 1.25em;}
```

```
p.warnung {color: #BDB76B;}
p.erfolg {color: green; font-size: 1.25em;}
p.zentriert {text-align: center;}
p.center2 {text-align: center; margin: 0;}

h1, h2, h3, h4, h5, h6, h7 {text-align: center;}
h1.left, h2.left, h3.left, h4.left, h5.left, h6.left, h7.left {text-align: left;}
h2#actionFehler {color:red;}
h2#actErfolg {color:green;}
h4.left {text-align: left;}
h4.menu {margin: 1.33em 1.33em 1.0em 10%;
text-align: left;}
hr.menu {width: 80%; align: left;}
li.menu {margin-left: -1em;}
ul.menu {margin: 1.2em 1.0em 1.0em -1em}

@media print { #navi {display: none;} }
</style>
```

Diese CSS-Anweisungen stehen in dem Modul `dbengine.css` im Verzeichnis `src`. Dieses Modul kann jederzeit durch weitere Anweisungen den eigenen Bedürfnissen angepasst werden. Zu beachten ist dabei nur, dass die Klassen `menu` und `navi` für den grundsätzlichen Aufbau der Seite genutzt werden.

2.3. Besonderheiten bei BLOBs

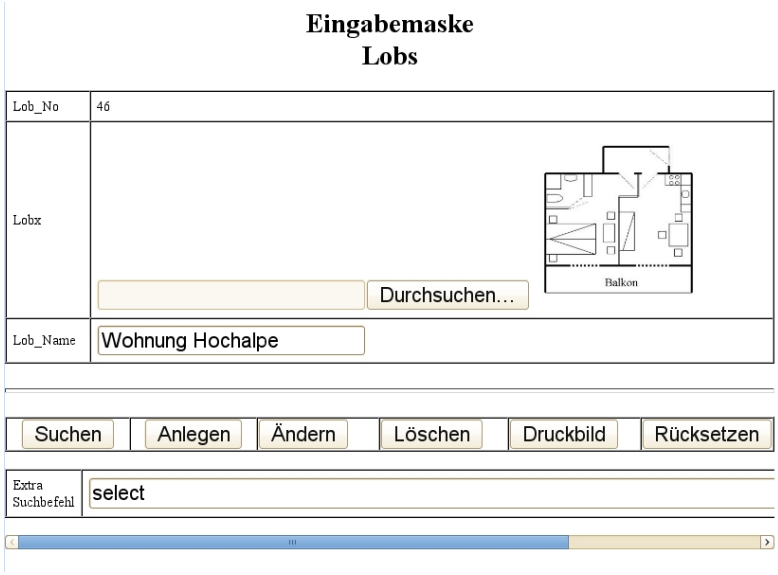
Für `Blob`s müssen einige Besonderheiten beachtet werden. Da es sich bei `Blob`s um große Dateien handelt, die binäre Daten enthalten können, muss ihre Eingabe gezielt und gesondert durchgeführt werden. Die bisherige Implementierung der `BLOB`s geht davon aus, dass diese in einer eigenen Tabelle gespeichert werden. Als Beispiel kann die Tabelle `Lobs` aus der Datenbank `dbengine_test_db` genutzt werden. Diese Tabelle hat folgende Felder:

Tabelle 2.4. Felder der Tabelle LOBS

Feldname	Feldtyp	Bedeutung
`lob_no`	`SERIAL`	Nummer des `BLOB`, wird gleichzeitig als `primary key` genutzt
`lob_name`	`TEXT`	Beschreibung des `BLOB`
`blobx`	`oid`	Verweis auf das Binärobjekt

In der Tabelle `lobs` ist das binäre Objekt also als Typ `OID` eingetragen. Um es aber von einem allgemeinen `OID`-Objekt unterscheiden zu können, muss es in der Konfigurationsdatenbank als Typ `BLOB` eingetragen sein. Im Kapitel „Die Tabelle designinfo“ ist im Bereich der Beschreibung „Das Feld displayinfo“ auch die Konfiguration von BLOBs beschrieben und auch ein Listing aus der Datenbank `dbengine_test_db` (s. Abbildung 3.4, „Darstellung binärer Objekte in Listings“) angegeben. Hier soll die Abbildung 2.14, „Darstellung binärer Objekte in einer Eingabemaske“ verdeutlichen, wie binäre Objekte in eine Datenbank eingebracht werden können.

Abbildung 2.14. Darstellung binärer Objekte in einer Eingabemaske



Listing aus der Datenbank `dbengine_test_db`

Im Eingabefeld für das binäre Objekt ist zum einen das Objekt selbst dargestellt (sofern eines vorhanden ist und es sich um ein Bild handelt), zum anderen ist als Eingabemöglichtkeit die Eingabe eines Dateinamens (inklusive Pfadangabe) möglich. Diese Datei wird dann vom Webbrowser als temporäre Datei an den Server übertragen und von dort in die Datenbank übernommen.

Kapitel 3. Die Konfiguration einer Datenbank

Zur einfachen Gestaltung der Nutzung bedient sich das Tool DBENGINE selbst der PostgreSQL-Datenbank `dbengine_info_db`, in der alle wichtigen Werte zur Erzeugung von Formularen (Eingabemasken) und Tabellenübersichten bzw. Berichten gespeichert sind. Einige Werte sind auch im Hauptprogramm `dbengine.pl` gespeichert (als Vorbelegung in dem Hash-Array `configs`). Dabei handelt es sich aber lediglich um Daten, die benötigt werden, um überhaupt eine Datenbank anzusprechen. Die Tabellen der Konfigurationsdatenbank `dbengine_info_db` enthalten Konfigurationsvorschriften für jede anzusprechende Datenbank. Doch ist eine Datenbank auch ohne jede weitere Konfigurationsvorschrift bedienbar. Die Konfigurationen gestatten aber eine Definition der Bediener-Oberfläche, der Definition von Zugriffsberechtigungen und noch etliche andere Möglichkeiten.

Tabelle 3.1. Liste der Konfigurationstabellen

Tabellenname	Tabellenart	Bedeutung
`basedesign`	*`Haupttabelle`*	Abschnitt 3.1.1, „Die Tabelle basedesign (Tabellendefinition)“ enthält für jede Tabelle einer Anwender-Datenbank die wichtigsten Anordnungsdaten.
`designinfo`	*`Haupttabelle`*	Die Tabelle designinfo enthält alle Angaben, die zur Gestaltung der Benutzeroberfläche für eine Maske zur Dateneingabe, Datenänderung oder Datensuche in einer einzelnen Tabelle nötig sind.
`tabledesign`	*`Haupttabelle`*	Abschnitt 3.1.3, „Die Tabelle tabledesign (Listingdesign)“ enthält alle Angaben, die zur Ausgabe einfacher Listen aus der Tabelle nötig sind. Diese Listen werden bei Suchvorgängen erstellt.
`action`	*`Haupttabelle`*	Abschnitt 3.1.4, „Die Tabelle action (Aktions-/Berichtsdefinition)“ enthält entweder Perlprozeduren oder eine URL zum Durchführen bestimmter Aktionen oder zum Erstellen von Berichten.
`relation`	*`Detailtabelle`*	Abschnitt 3.2.1, „Die Tabelle relation (Definition von Tabellenbeziehungen)“ enthält die Beziehungen der Tabellen einer Datenbank untereinander. Es wird für jede Tabelle angegeben, ob sie ich auf eine andere bezieht, oder ob eine andere Tabelle sich auf diese bezieht
`virtual`	*`Detailtabelle`*	Abschnitt 3.2.2, „Die Tabelle virtual (Definition berechneter Felder)“ enthält für Masken oder Listen, die berechnete Felder enthalten sollen, die Berechnungsvorschriften für diese Felder angegeben.
`dbengine_user`	*`Detailtabelle`*	Abschnitt 3.2.3, „Die Tabelle dbengine_user (Definition zulässiger Datenbankbenutzer)“ enthält die für die aktuelle Datenbank zulässigen Benutzer.

Tabellenname	Tabellenart	Bedeutung
equation	*Detailtabelle*	Abschnitt 3.2.4, „Die Tabelle equation (Definition von Unterprogrammen)“ enthält Perl-Prozeduren, die global in dieser Datenbank genutzt werden können.
databaseconfigs	*Wertebereiche*	Abschnitt 3.3.1, „Die Tabelle databaseconfigs (Datenbankspez. Konfigurationen)“ enthält Konfigurationen, die systemspezifische Konfigurationen datenbankspezifisch überschreiben oder als zusätzliche gesetzt werden.
valuelist	*Wertebereiche*	Abschnitt 3.3.2, „Die Tabelle valuelist (Wertebereichsdefinition)“ enthält Datenbank spezifische Wertelisten, die für den Maskenaufbau genutzt werden können.
langtexts	*Wertebereiche*	Abschnitt 3.3.3, „Die Tabelle langtexts (Sprachspezifische Texte)“ enthält die sprachabhängigen Texte für das Datenbanksystem DBENGINE. Diese Texte können somit leicht dem eigenen Bedarf angepasst werden und auch für weitere Sprachen definiert werden.
configs	*Wertebereiche*	Abschnitt 3.3.4, „Die Tabelle configs (Konfigurationswerte)“ enthält die System spezifischen Konfigurationsparameter für DBENGINE. Jeder Parameter enthält auch eine Kommentierung, die die eigene Konfiguration erleichtert. Neben den vorhandenen Parametern können weitere definiert werden, die dann in eigenen Prozeduren abgefragt werden können.
helpfile	*Wertebereiche*	Abschnitt 3.3.5, „Die Tabelle helpfile (Hilfetexte)“ enthält sprachspezifische Hilfetexte, die für bestimmte Aktionen vorgegeben werden können, um die Benutzer zu unterstützen.

Die Tabellen der Konfigurationsdatenbank sind eingeteilt in *Haupttabellen*, *Detailtabellen* und *Wertebereiche*. Die Haupttabellen definieren die Schnittstelle zum Benutzer, die Detailtabellen sind dabei als Hilfen notwendig und die Wertebereiche sind Tabellen, die auch unabhängig von der einzelnen Datenbank Wertebereiche für die Nutzung von Dbengine festlegen. Die Tabellen (s. Tabelle 3.1, „Liste der Konfigurationstabellen“) der Konfigurationsdatenbank werden nachfolgend im Einzelnen beschrieben. Jedesmal, wenn in einer der Konfigurationstabellen ein Tabellenname angefordert wird, kann er in zwei Arten gegeben werden: mit oder ohne zugehörigem Schemanamen (Tabellen aus dem Schema `public` werden stets ohne Schemanamen angegeben. Für eine Tabelle aus einem anderen Schema (z.B. Schema `lauf` in `dbengine_test2_db`) wird in der Konfigurationstabelle zunächst nach dem vollen Namen gesucht. Ist dieser nicht vorhanden, wird zusätzlich nach einer Konfiguration für die Tabelle ohne Schemanamen gesucht. Dies soll dazu dienen, bei zwei Schemata, die gleiche Tabellen besitzen (z.B. `test` und `lauf` in `dbengine_test2_db`), die Konfiguration nur einmal angeben zu müssen.

3.1. Die Haupttabellen

3.1.1. Die Tabelle basedesign (*Tabellendefinition*)

Diese Tabelle enthält die grundlegenden Angaben für eine Datenbanktabelle. Über die Einträge dieser Tabelle wird das Aussehen der Steuerleiste definiert (s. Abschnitt 2.2.1, „Die Steuerleiste“). Folgende Felder sind in der Tabelle basedesign vorhanden:

Tabelle 3.2. Die Felder der Tabelle basedesign

Feldname	Anzeigenname	Bedeutung
dbasename	*Datenbank*	Name der Anwender-Datenbank
tablename	*Tabelle*	Name der Anwendertabelle
displayname	*Anzeigename*	Name der Tabelle in der Anzeige in der Steuerleiste ...
tableart	*Tabellenart*	Art der Anwendertabelle. Es gibt dafür drei verschiedene Möglichkeiten: • Haupttabellen (eingetragen als maintables) • Detailtabellen (eingetragen als detailtables) • Wertebereiche (eingetragen als valueranges)
tableorder	*Reihenfolge*	Nummer der Tabelle als Reihenfolge der Tabellen in der Tabellenart.
username	*Zulässige Benutzer*	Durch Komma getrennte Angabe der zulässigen Benutzer. Mit dem Wort ALL bekommen alle Benutzer Zugriffsrecht
description	*Beschreibung*	Kurze Beschreibung der Aufgabe der Tabelle.

Abbildung 3.1. Liste der Tabellenzuordnung von dbengine_info_db

Satznr.	Dbasename	Tablename	Displayname	Tableart	Tableorder	Username	Description
1	dbengine_info_db	relation	lang(r_relation)	detailtables	1		
2	dbengine_info_db	virtual	lang(r_virtual)	detailtables	2		
3	dbengine_info_db	dbengine_user	lang(r_dbengine_user)	detailtables	3		
4	dbengine_info_db	equation	lang(r_equation)	detailtables	4		
5	dbengine_info_db	basedesign	lang(r_basedesign)	maintables	1		
6	dbengine_info_db	designinfo	lang(r_designinfo)	maintables	2		
7	dbengine_info_db	tabledesign	lang(r_tabledesign)	maintables	3		
8	dbengine_info_db	action	lang(r_action)	maintables	4		
9	dbengine_info_db	databaseconfigs	lang(r_databaseconfigs)	valueranges	1		
10	dbengine_info_db	valuelist	lang(r_valuelist)	valueranges	2		
11	dbengine_info_db	configs	lang(r_configs)	valueranges	3		
12	dbengine_info_db	langtexts	lang(r_langtexts)	valueranges	4		
13	dbengine_info_db	helpfile	lang(helpfile)	valueranges	5		Table for helptexts for actions.

Sind für eine Datenbank hier keine oder nicht alle Tabellen eingetragen, so werden diese Tabellen alphabetisch in der Steuerleiste unter der Bezeichnung *Tabellen* angezeigt. Ein Beispiel für die Aufteilung

von Tabellen einer Datenbank ist die Konfigurationsdatenbank selbst. Einen Ausschnitt einer Liste für Tabellen-Konfigurationen einer Datenbank, wie sie durch den Suchbefehl

```
SELECT a.* FROM "basedesign" a
    WHERE "dbasename" LIKE 'dbengine_i%'
    ORDER BY dbasename, tableart, tableorder
```

– eingegeben im Feld des Extrasuchbefehls – gefunden wird, zeigt die Abbildung 3.1, „Liste der Tabellenzuordnung von dbengine_info_db“ für die Zuordnung aller Tabellen der Konfigurationsdatenbank.

3.1.2. Die Tabelle `designinfo` (*`Maskendefiniton`*)

Diese Tabelle beschreibt den Formularaufbau für eine bestimmte Anwendertabelle. Sie benötigt je Tabellenzeile dabei eine Reihe von Informationen, die in den einzelnen Tabellenfeldern eingetragen werden. Nicht alle Informationen sind nötig, da viele aus den Anwendertabellen-Feldern automatisch ermittelt werden können. Sie sind nur auszufüllen, sofern von den Standardwerten abgewichen werden soll.

Folgende Felder sind in der Tabelle `designinfo` vorhanden:

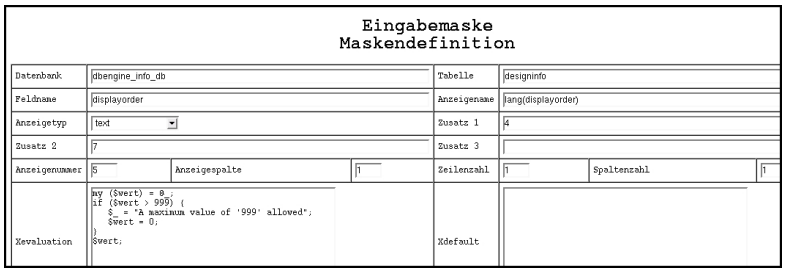
Tabelle 3.3. Die Felder der Tabelle `designinfo`

Feldname	Anzeigenname	Bedeutung
`dbasename`	*`Datenbank`*	Name der Anwender-Datenbank
`tablename`	*`Tabelle`*	Name der Anwendertabelle
`fieldname`	*`Feldname`*	Name des Feldes
`displaystring`	*`Anzeigename`*	Name des Feldes das definiert wird
`displayinfo`	*`Anzeigetyp`*	Art der auszugebenden Information im Feld (s. Das Feld displayinfo).
`disptable`	*`Zusatz 1`*	Zusatzfeld für *`displayinfo`* (s. Das Feld displayinfo).
`dispvfield`	*`Zusatz 2`*	Zusatzfeld für Displayinfo (s. Das Feld displayinfo)
`dispdfield`	*`Zusatz 3`*	Zusatzfeld für Displayinfo (s. Das Feld displayinfo)
`displaystring`	*`Anzeigename`*	Im Formular auszugebende Feldbezeichnung; wird als Angabe ein Aufruf der Funktion `lang` eingegeben, so wird der Funktionswert als Index für die Tabelle `langtexts` genutzt und dort sprachabhängig der eigentliche Anzeigename ermittelt (s. Abbildung 3.2, „Beispiel einer Designinfo-Maske“ und Abschnitt 3.3.3, „Die Tabelle langtexts (Sprachspezifische Texte)“). Fehlt diese Angabe so wird der echte Feldname der Tabelle ausgegeben, wobei der 1. Buchstabe groß geschrieben wird.
`displayorder`	*`Anzeigenummer`*	Formularzeile, in der das Feld stehen soll. Dabei wird nur eine größer/kleiner Beziehung berücksichtigt. Eine Reihe ist dabei immer durch die Felder in ihr definiert.
`displaycolumn`	*`Anzeigespalte`*	Spalte, in der das Feld stehen soll. Dabei werden Felder gezählt, nicht Zeichen und ein Feld besteht immer aus seiner Bezeichnung und dem Wert.

Feldname	Anzeigenname	Bedeutung
rowspan	*Zeilenzahl*	Anzahl der Zeilen, über die sich das Feld erstrecken soll.
colspan	*Spaltenzahl*	Anzahl der Spalten, die das Feld einnehmen soll.
xevaluation	*Xevaluation*	Prüf- oder Berechnungsvorschrift (s. Abbildung 3.2, „Beispiel einer Designinfo-Maske“) für in das Feld eingegebene Daten: es muss sich dabei um ein mit `eval` ausführbares Perl-Programm handeln.
xdefault	*Xdefault*	Default-Wert für ein Feld. Der Inhalt von *xdefault* ist ein mit `eval` zu berechnendes Perl-Programm. Der Wert wird nur berechnet, wenn eine leere Maske, eine Suchmaske, ausgegeben werden soll.

Für alle Felder sind in der Datenbank `dbengine_info_db` bzw. der Testdatenbank Beispiele vorhanden.

Abbildung 3.2. Beispiel einer Designinfo-Maske



Das Feld *displayinfo*. Von besonderer Bedeutung ist das Feld *displayinfo*, da über dieses Feld eine Korrektur der typgebundenen Ausgabe erfolgen kann. Auch Verbindungen zu anderen Tabellen erfolgen über dieses Feld. Hier daher die Bedeutung der verschiedenen *displayinfo* Einträge:

- **NONE:** Dieses Feld wird nicht ausgegeben und nicht berechnet.
- **COMPUTATION:** Diese Felder werden, wenn ein Wert vorhanden ist, ausgegeben, aber es kann kein Wert dafür eingegeben werden. Es handelt sich um vom Datenbanksystem berechnete Felder (s. Feld `price` in Tabelle *order_position*).
- **SERIAL:** Dieses Feld wird, wenn Daten vorhanden angezeigt, aber nicht bearbeitet, d.h. es werden keine neuen Daten für **`insert`** und **`update`** berechnet. Diese Feldart dient zur Ausgabe des PostgreSQL - Datentyps `serial`.
- **NUMERIC:** Sollen `DECIMAL(6,2)`- oder `NUMERIC(6,2)`-Felder als Währung ausgegeben werden, so kann dies gegenüber der schematischen Ausgabe durch das Datenbanksystem mit dem Eintrag `NUMERIC` in dem Feld *displayinfo* geschehen. Dieser Eintrag bietet für die Ein- und Ausgabe von Daten für diesen Typ einige Besonderheiten. Zunächst können bei der Eingabe ein Währungssymbol oder auch „Tausender“-Markierungen mit angegeben werden, sofern in der Tabelle `configs` (s. Abschnitt 3.3.4, „Die Tabelle configs (Konfigurationswerte)“) ein Wert für `currency_‹Landeskennzeichen›` angegeben worden ist. Alternativ zu dem Eintrag in der Tabelle `configs` oder auch in der Datenbankspezifischen Konfigurationstabelle `databaseconfigs` (s. Abschnitt 3.3.1, „Die Tabelle databaseconfigs (Datenbankspez. Konfigurationen)“) können die Zusatzfelder `Zusatz 1` und `Zusatz 2` genutzt werden, um die Konfiguration tabellenspezifisch anzugeben.

Tabelle 3.4. Die Bedeutung der Zusatzfelder bei `NUMERIC`

Feldname	Zusatzinformation
Zusatz 1	Angaben für das Währungssymbol `WS`, den Dezimaltrenner `DT` und den Tausendertrenner `TT`. Die drei Werte werden durch einen `:` getrennt. Die Angabe muss also so aussehen, wie es in der Abbildung 3.18, „Standardkonfigurationswerte“ unter dem `Index currency_de` gezeigt ist: `€:.:,`. Also `‹WS›:‹DT›:‹TT›`.
Zusatz 2	In diesem Feld kann dann angegeben werden, ob in der Spaltenüberschrift das Symbol ausgegeben wird — dann erfolgt die Ausgabe im Einzelfeld ohne Währungsangabe — oder ob die Währungsangabe bei jedem Einzelfeld erfolgen soll. Im ersteren Fall muss in diesem Feld die Art der Währungsangabe in der Kopfzeile eingegeben werden (z. B. als `[€]`, um das Währungssymbol in Klammern aus zu geben).

Die Auswirkungen der beiden Möglichkeiten zur Währungsangabe sind in der Abbildung 3.3, „Währungsdarstellung in Listings“ im Beispiel 1 mit der Währungsangabe im Feld `Price` und im Beispiel 2 mit der Währungsangabe im Kopf von dem Feld `Price [€]` dargestellt (in beiden Fällen mit dem Tausendertrennzeichen).

Abbildung 3.3. Währungsdarstellung in Listings

Satznr.	Part_No	Description	Suppl_No	Supplier	Category	Price	Contained in
1	1	Computer Mini	1	ATM	A	1.202,00 €	
2	20	Notebook	1	ATM	A	3.000,00 €	
3	21	Mil-Notebook	1	ATM	A	4.000,00 €	
4	27	Communicationbox	1	ATM	A	300,00 €	21

Beispiel 1: Listing aus der Datenbank `dbengine_test_db`

Satznr.	Part_No	Description	Suppl_No	Supplier	Category	Price [€]	Contained in
1	1	Computer Mini	1		A	1.200,00	
2	20	Notebook	1		A	3.000,00	
3	21	Mil-Notebook	1		A	4.000,00	
4	27	Communicationbox	1		A	300,00	21
5	28	Combo Box	1		B	100,00	

Beispiel 2: Listing aus der Datenbank `dbengine_test2_db` und dem Schema `lauf`

- **CURRENCY:** Sollen Gleitpunktfelder als Währung ausgegeben werden (statt mit dem SQL-gerechteren Format DECIMAL(6,2) bzw. NUMERICAL(6,2)), so kann dies mit dem Eintrag `CURRENCY` geschehen. Alle dazu nötigen Konfigurationsangaben entsprechen denen vom Typ `NUMERIC` (s. NUMERIC). Dieser Hilfstyp ist veraltet und kommt noch aus einer Zeit, als postgreSQL den Datentyp `NUMERIC` nicht kannte. Er kann jedoch weiterhin genutzt werden.

- **VIRTUAL:** Felder mit dieser Angabe im Feld `displayinfo` sind berechnete Felder, die nur ausgegeben werden, aber nie in der Datenbank abgespeichert werden. Diese Felder werden aktuell über das kleine Perl-Programm in `xequation` (s. Abschnitt 3.2.2, „Die Tabelle virtual (Definition berechneter Felder)“) berechnet. Zur Berechnung stehen die Werte der anderen Felder dieser Tabellenzeile in `$rvalues` mit den Feldnamen als Keys zur Verfügung. Tritt während der Berechnung ein Fehler auf, der als solcher zurückgemeldet werden kann, so kann dies mit der Anweisung **die("‹Fehlertext›")** geschehen, da die Funktion `eval` dann diesen Fehler zurückmeldet.
- **BLOB:** Felder mit dieser Angabe im Feld `displayinfo` sind Felder zur Aufnahme von `BLOB`s. Im Konfigurationsfeld `disptable` (Zusatz 1) kann eine Breite für Bilderdarstellungen angegeben werden. Dieser Eintrag hat nur für die Ausgabe eine Wirkung: Sofern das binäre Objekt ein Bild ist, wird es zur Darstellung auf diese Pixelbreite skaliert. Ist im Feld `Zusatz 1` kein Eintrag vorhanden, so werden Bilder stets in voller Größe angezeigt.

Abbildung 3.4. Darstellung binärer Objekte in Listings

Satznr.	Lob_No	Lob_Name	Lobx
1	59	Bildbeispiel	
2	56	Wörterbuch Testblob	41694 (UTF-8 Unicode text)
3	57	Anne-Guide als rtf-File	41695 (Rich Text Format data)
4	46	Wohnung Hochalpe	
5	48	Weihnachtsoratorium 2009, Beschreibung in der Zeitschrift 'Takt' der SWP	41686 (PDF document)
6	51	testbild	
7	55	Weihnachtsoratorium 2009, Beschreibung in der Zeitschrift 'Takt' der SWP	41693 (PDF document)

Anzahl der gefundenen Einträge = 7

Listing aus der Datenbank `dbengine_test_db`

Ein Beispiel einer Eingabemaske für `BLOB`s ist im Kapitel „Besonderheiten bei BLOB“ in der Abbildung 2.14, „Darstellung binärer Objekte in einer Eingabemaske“ gegeben.

- **popup:** Für Felder dieser Art wird ein Popup-Menu aus der dbengine_info_db-Datenbank aus der Datei `valuelist` generiert. Die Auswahl in der Tabelle `valuelist` (s. Abschnitt 3.3.3, „Die Tabelle langtexts (Sprachspezifische Texte)“) geschieht über den Datenbanknamen und den Feldnamen (in `valuelist` das Attribut `name`, während das Attribut `display` den Popup-Wert enthält.
- **relationpopup:** Bei diesem Feld wird ein Popup-Fenster erzeugt, aus dem ein Wert ausgegeben werden kann. Der aktuelle Wert steht dabei an erster Stelle in der Liste und wird beim Aufruf auch angezeigt. Für diesen Typ werden in einigen Feldern Zusatzinformationen erwartet:

Tabelle 3.5. Die Bedeutung der Zusatzfelder bei `relationpopup`

Feldname	Zusatzinformation
`Zusatz 1`	Tabelle, in der die Werte für das Popup-Feld stehen. In der Regel handelt es sich dabei um Wertebereichs-Dateien.
`Zusatz 2`	In diese Feld wird der Feldname aus der Popup-Tabelle eingetragen, in dem die zulässigen Werte stehen. Soll die Auswahl aus der Tabelle eingeschränkt werden, so können hier zwei weitere Felder (durch Leerzei-

Feldname	Zusatzinformation
	chen getrennt) eingetragen werden, die das Testfeld und den Testwert in der Tabelle enthalten.
Zusatz 3	Falls erforderlich kann in dem Popup-Feld noch eine zusätzliche Beschreibung angezeigt werden, die als Erläuterung für den Wert dienen kann (ist der Wert nur eine Schlüsselzahl, so kann mit diesem Feld z. B. eine zugehörige Bezeichnung ausgegeben werden.

Ein Beispiel soll das verdeutlichen. Der Inhalt der Felder sei:

Zusatz 1: maler
Zusatz 2: pa_no ort berlin
Zusatz 3: name

Bei diesen Werten werden die Werte des Popup-Fensters mit folgendem Kommando ermittelt:

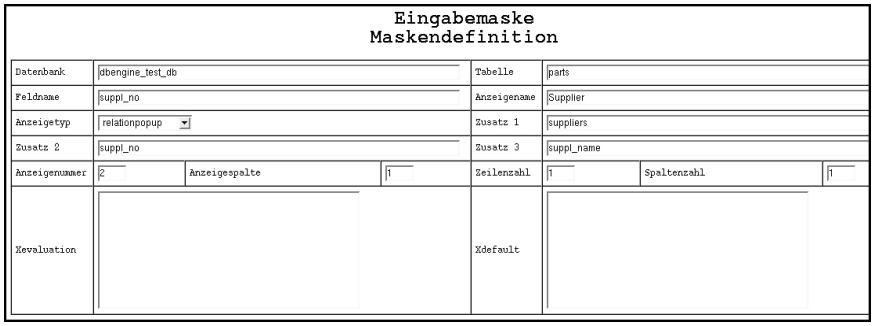
SELECT pa_no,name FROM maler WHERE ort='berlin';

Das Popup-Fenster selbst hat dann für jeden einzelnen Wert folgendes Aussehen:

<Wert von pa_no>(<name>)

In der Testdatenbank ist dafür ein Beispiel für die Tabelle `parts` enthalten.

Abbildung 3.5. Beispiel für die Anwendung von `relationpopup`



Deutlich ist in der Abbildung 3.5, „Beispiel für die Anwendung von relationpopup“ zu sehen, dass die Einträge im Feld *suppl_no* aus der Tabelle `suppliers` zu nehmen sind. Zum besseren Erkennen, welcher Wert zu nehmen ist, wird gleichzeitig der Wert aus *suppl_name* mit angezeigt. Beim Benutzen der Testdatenbank ergibt sich für die Tabelle `parts` dann eine Maske mit Werten für die *suppl_no* wie in der Abbildung 2.4, „Beispiel einer Suchmaske“ dargestellt.

- **multiplepopup:** Nur bei `varchar`- bzw. `text`-Feldern erlaubt. Dieser Typ bewirkt, dass eine Popup-Liste erzeugt wird, aus der mehrere Werte ausgewählt werden können. Gespeichert werden die Werte in der Datenbank, indem die Einzelwerte durch die Zeichenfolge `':::'` miteinander verbunden werden. Die Auswahl der darzustellenden Listeneinträge geschieht über die Zusatzfelder in gleicher Weise wie bei dem Eintrag `relationpopup` im relationpopup.
- **text:** Diese Feldart ist nur dann nötig, wenn Einschränkungen für die Feldlänge und die maximale Datenlänge gemacht werden sollen. Ein Tabellen-Feld des Typs VARCHAR(40) kann z. B. durch Zusatzangaben auf eine maximale Formularfeldlänge von 20 Zeichen und eine maximale Länge von 30 Zeichen eingeschränkt werden. Diese Zusatzangaben stehen in den Feldern (s. Abbildung 3.2, „Beispiel einer Designinfo-Maske“):

Tabelle 3.6. Die Bedeutung der Zusatzfelder bei `text`

Feldname	Zusatzinformation
Zusatz 1	Länge des Formularfeldes (z. B. 20)
Zusatz 2	Maximale Textlänge für die Datei (z. B. 30)

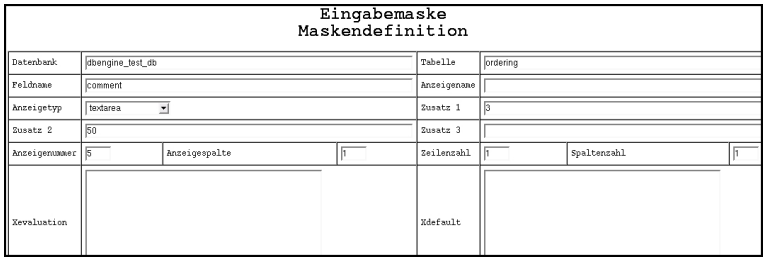
- **textarea:** Soll die Eingabe mehrzeiliger Texte in ein Tabellenfeld ermöglicht werden, so muss die Angabe `textarea` gemacht werden. Als Zusatzangaben werden die Zeilenzahl und Zeichenzahl je Zeile benötigt. Diese Werte stehen in den Zusatzfeldern:

Tabelle 3.7. Die Bedeutung der Zusatzfelder bei `textarea`

Feldname	Zusatzinformation
Zusatz 1	Anzahl der Zeilen des Formularfeldes (z. B. 20)
Zusatz 2	Anzahl der Zeichen je Zeile (z. B. 30)
Zusatz 3	Eintragen des Wertes `pre`, wenn Ausgaben vorformatiert

Ein Beispiel für diese Art der Feldbeschreibung ist in der Testdatenbank vorhanden (s. Abbildung 3.6, „Beispiel einer Anwendung des Typs textarea“):

Abbildung 3.6. Beispiel einer Anwendung des Typs `textarea`



Die Tabelle `ordering` enthält ein Kommentarfeld `comment`, das mehrzeilige Eingaben verträgt, da ein Kommentar ausführlich sein sollte. Ist eine Ausgabe nicht vorformatiert, sondern mit HTML-Formatierungen versehen, so können bei Benutzung der `PostgreSQL`-Perlfunktion `htmltext` für die Ausgabe Tabellen oder Listen erzeugt werden, indem der auszugebende Text sowohl mit den üblichen XHTML-Tags wie auch mit abgekürzten Tags versehen werden kann. Zum Beispiel kann eine Tabelle mit dem üblichen HTML-Tags `<table>` und Listen können mit den Tags `<dl>`, `<ol>` oder `<ul>`, gegebenenfalls auch mit ihren möglichen Attributen, und den entsprechenden Endetags eingegrenzt werden (diese Funktion steht in dem Verzeichnis `db_functions` der Quellen zur Verfügung und kann mit dem Kommando

psql ‹dbname› < textbau.pl

in die Datenbank eingebracht werden). Jede Zeile einer Tabelle bzw. jedes definierende Tag oder jedes Element einer Liste wird dann automatisch erzeugt durch Eingabe eines Zeilenwechsels, jeder Feldwechsel durch ein Tabulatorzeichen. In diesem Fall muss das Feld in der Listing-Beschreibung als `html(‹feldname›)` angegeben werden. Eine ausführliche Beschreibung dieser Funktion ist im Abschnitt 2.2.5.1, „Die Datenbankfunktion htmltext“ zu finden.

Im Übrigen gilt, dass in berechneten Feldern (`xdefault` und `xevaluation`) Variablen-Bereiche und Prozeduren zur Verfügung stehen, wie im Abschnitt 3.2.4, „Die Tabelle equation (Definition von Unterprogrammen)“ beschrieben. Die Werte dieser Variablen können damit jederzeit in den `eval`-Prozeduren

benutzt werden. Wenn für die Prozeduren Eingangsparameter nötig sind, so werden sie stets in der Variablen $_ zur Verfügung gestellt.

3.1.3. Die Tabelle `tabledesign` (*`Listingdesign`*)

Diese Tabelle beschreibt den Aufbau eines Listings von mehreren Sätzen aus einer Anwendertabelle. Ein Listing wird aufgerufen durch Suchen über die Suchmaske oder durch die Angabe eines *`Extra Suchbefehl`*.

Zur Definition eines solchen Listing sind in der Tabelle `tabledesign` folgende Felder definiert:

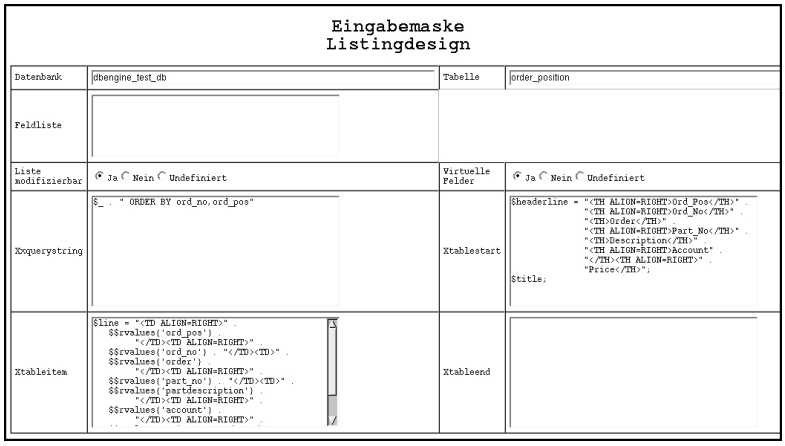
Tabelle 3.8. Die Felder der Tabelle `tabledesign`

Feldname	Anzeigename	Bedeutung
`dbasename`	*`Datenbank`*	Name der Anwender-Datenbank
`tablename`	*`Tabelle`*	Name der Anwendertabelle
`tablefields`	*`Feldliste`*	Liste der Felder, die ausgegeben werden sollen. Damit wird gleichzeitig die Reihenfolge der Felder im Listing festgelegt. Ist dieses Feld leer, so werden alle Felder in der Reihenfolge ausgegeben, in der sie von der Datenbank geliefert werden.
`modifyable-list`	*`Liste modifizierbar`*	Boolesches Feld, das angibt, ob eine Modifikation eines Feldes für mehrere Sätze zugleich durchgeführt werden darf (*`multiple update`*). In diesem Fall werden in dem Satznummernfeld Kontrollkästchen angelegt (s. Abbildung 2.6, „Ergebnisliste einer Suche“). Sätze mit markierten Kontrollkästchen werden geändert, alle anderen nicht. Defaultmäßig sind alle Kontrollkästchen markiert. Im Listing werden außerdem zwei weitere Felder und zwei weitere Knöpfe angelegt, deren Bedeutung in Abschnitt 2.2.3, „Die Liste von Tabellensätzen“ beschrieben ist.
`needsvirtuals`	*`Virtuelle Felder`*	Boolesches Feld, das angibt, ob im Listing virtuelle Felder vorkommen (s. Abschnitt 3.2.2, „Die Tabelle virtual (Definition berechneter Felder)“). Ist dies der Fall so müssen in der Regel auch die Felder *`xtablestart`* und *`xtableitem`* ausgefüllt werden. Berechnet werden alle Felder, die für diese Tabelle in der Konfigurationstabelle `virtual` definiert sind (s. Abbildung 3.11, „Beispiel einer virtual-Felddefinition“).
`xxquerystring`	*`Xxquerystring`*	Dieses Feld dient der zusätzlichen Angabe von Suchbedingungen. In der Regel werden hier Sortiervorschriften angegeben. Dies Feld wird mit der Perl-Funktion `eval` ausgewertet. Es können also umfangreiche Berechnungen erfolgen, sofern nötig. Beim Aufruf der `eval`-Funktion steht in der Perl-Variablen $_ der vollständige bis dahin aufgebaute Suchstring (inkl. Der WHERE-Bedingung). Es genügt also eine Stringkonkatenation

Feldname	Anzeigename	Bedeutung
		($_ . "ORDER BY ‹bedingung›") mit der ORDER BY Angabe, um die Sortierung zu veranlassen.
xtablestart	*Xtablestart*	Auch dieses Feld wird über die `eval`-Funktion ausgewertet, und zwar vor der Listenausgabe. In diesem Feld kann daher ein Titel für das Listing und eine Kopfzeile definiert werden, die vor den Zeilen der einzelnen Sätze ausgegeben werden sollen (s. Abbildung 3.7, „Beispiel für eine komplexe Listingdefinition“ und Abbildung 3.12, „Beispiel eines Listings mit berechneten Feldern (virtual)“). Hier sind die Variablen `$headerline` für die Kopfzeile und `$title` für die Überschrift von Bedeutung.
xtableitem	*Xtableitem*	Über dieses Feld (ebenfalls ein `eval`-Feld) kann dann die Ausgabe einer einzelnen Satzzeile gesteuert werden. Die Inhalte aller gelesenen und berechneten Felder befinden sich in der Referenz `$rvalues` auf ein Hash-Array, wobei der Key der interne Feldname ist. Die Ausgabe eines Satzes wird in die Variable `$line` eingetragen. Ein Beispiel für eine solche satzweise Ausgabe ist in der Abbildung 3.12, „Beispiel eines Listings mit berechneten Feldern (virtual)“ zu sehen.
xtableend	*Xtableend*	Dieses `eval`-Feld kann benutzt werden, um am Ende des Listings spezielle Ausgaben zu tätigen. Standardmäßig steht am Ende eines Listings die Anzahl der gefundenen Sätze, doch kann diese Ausgabe durch eine Perl-Prozedur verändert werden. Die Anzahl der gefundenen Sätze steht in der Variablen `$ntuples` zur Verfügung.

Für die evaluierbaren Felder gilt die gleiche Werteversorgung wie sie im Abschnitt 3.2.4, „Die Tabelle equation (Definition von Unterprogrammen)“ in der Tabelle 3.14, „Die Felder der Tabelle equation“ und Tabelle 3.15, „Für eine Berechnung mit eval zur Verfügung stehende Werte“ beschrieben sind. Zusätzlich sind einige Variable vorhanden, die gesetzt werden können. Die jeweils wesentlichen sind bei der Beschreibung der einzelnen Felder mit beschrieben. Die Abbildung 3.7, „Beispiel für eine komplexe Listingdefinition“ zeigt eine der Standardeinstellungen für die Testdatenbank. Es wird die Reihenfolge der Sätze angegeben, eine Tabellenkopfzeile definiert (der Titel aber nicht geändert) und eine Ausgabeprozedur für jede Zeile angegeben. Weitere Beispiele für die Definition von Listings sind sowohl für die Tabellen jeder Konfigurationsdatenbank als auch für die Testdatenbank im System vorhanden.

Abbildung 3.7. Beispiel für eine komplexe Listingdefinition



3.1.4. Die Tabelle `action` (*`Aktions-/Berichtsdefinition`*)

In dieser Tabelle werden die Perl-Prozeduren für die Datenbankspezifischen Aktionen und Berichte definiert. Es werden dabei unterschieden:

- Berichte und
- Aktionen. Bei den letzteren wird weiter unterschieden zwischen
  - Datenbankspezifischen Aktionen und
  - Standardaktionen. Als Standardaktionen sind z.Z. folgende Aktionen definiert:
    - Datenbankverwaltung (*`__STD_dbcreate`*): nur für den Administrator aus der angewählten Datenbank dbengine_info_db zulässig.
    - Schemaverwaltung (*`__STD_schema_manage`*): ist in allen Datenbanken möglich, aber nur für den Administrator und den Eigentümer der aktuell aufgerufenen Datenbank (siehe Abbildung 2.9, „Standardaktion Schemaverwaltung“).
    - Benutzerverwaltung (*`__STD_manage_user`*): in in allen Datenbanken möglich und auch für alle Benutzer, bei Benutzern, die nicht gleichzeitig Administratoren sind allerdings nur eingeschränkt: nur für sich selbst darf der Benutzer einige Daten ändern (siehe Abbildung 2.10, „Maske für die Benutzerverwaltung durch Administratoren“ und Abbildung 2.11, „Maske für die Benutzerverwaltung durch Benutzer“).
    - SQL-Schnittstelle (*`__STD_global_SQL`*): Diese Standardaktion kann für jeden Benutzer freigegeben werden, sofern ihm wenigstens das aktuell angewählte Schema gehört (siehe Abbildung 2.12, „Befehlsauswahlmaske der SQL-Schnittstelle“ und Abbildung 2.13, „Beispiel für einen SQL-Befehl mit Hilfeangabe“).

Die Tabelle enthält folgende Felder:

Tabelle 3.9. Die Felder der Tabelle `action`

Feldname	Anzeigename	Bedeutung
`dbasename`	*`Datenbank`*	Name der Anwender-Datenbank. Für Standardaktionen kann hier auch der Wert `__ALL` eingetragen werden: dann ist die Aktion für alle Datenbanken zulässig.
`actionname`	*`Aktion`*	Name der Aktion oder des Berichtes, unter dem sie auf der Steuerleiste erscheinen sollen. Namen von

Feldname	Anzeigename	Bedeutung
		Standardaktionen beginnen stets mit der Zeichenfolge „__STD_“
actionorder	*Reihenfolge*	Reihenfolge der Aktion oder des Berichtes auf der Steuerleiste. Die Nummern größer 1000 sind für Standardaktionen vorbehalten.
displayname	*Anzeigename*	Anzeigename der Aktion
report	*Bericht*	Boolesches Feld zur Angabe, ob es sich um einen Bericht oder eine Aktion handelt. Behandelt werden beide Arten intern gleich, der Unterschied besteht in der Anordnung auf der Steuerleiste: es gibt für beide Arten extra Bereiche, damit die Anwender schneller erkennen, um welche Art es sich logisch handelt.
ext_result	*Ergebnis extern*	Boolesches Feld zur Angabe, ob für einen Bericht oder eine Aktion der Erfolg oder Misserfolg angezeigt werden soll, oder ob die durchgeführte Perl-Prozedur dieses selbst durchführt.
in_list	*Anzeigen*	Boolesches Feld zur Angabe, ob diese Aktion bzw. dieser Bericht in der Steuerleiste angezeigt werden soll, oder ob sie/er nur intern benutzt wird. Beispiele für den letzteren Fall sind in der Datenbank ereignisse vorhanden.
usernames	*Zulässige Benutzer*	Angabe einer Benutzerliste. Diese Benutzer dürfen die Aktion oder den Bericht starten. Ist die Benutzung für alle zulässig, so kann statt dessen der Wert `ALL` eingetragen werden. Mit den Namen `DB_OWNER` bzw. `SCHEM_OWNER` kann die Erlaubnis gegeben werden für den jeweiligen Datenbankbesitzer bzw. den Eigentümer des aktuell genutzten Schemas in der Datenbank. Nur bei den zugelassenen Benutzern erscheinen die Aktionen bzw. Berichte in der Steuerleiste.
actiontext	*Anweisungen*	Angabe der Perl-Prozedur, die beim Start der Aktion oder des Berichtes durchgeführt werden soll.

In der Testdatenbank dbengine_test_db sind drei Beispiele für Berichte bzw. zwei für datenbankspezifische Aktionen enthalten (s. Abbildung 2.3, „Beispiele von Steuerleisten“). Die Perl-Prozeduren können sehr umfangreich sein. Um das zu verringern, ist es auch möglich in einer getrennten Tabelle (s. Abschnitt 3.2.4, „Die Tabelle equation (Definition von Unterprogrammen)“) Prozeduren zu definieren und diese in dem Perlcode aufzurufen. Dies ist vor allem dann nützlich, wenn diese Prozeduren von verschiedenen Aktionen aufgerufen werden. Ein Beispiel dafür ist in dem Bericht *Parts_level_list* zu finden (s. Abbildung 3.8, „Beispiel einer Aktionsdefinition“). Das Ergebnis dieses Berichtes ist zu finden in der Abbildung 2.7, „Beispiel eines Berichtes“. Da die auszugebenden Texte nicht direkt mit dem Befehl **`print`** ausgegeben werden, sondern als Ergebnis des Codestückes zurückgeliefert werden (der Wert des letzten Statements ist das Ergebnis einer **`eval`**-Prozedur, in diesem Fall die Variable `$ergline`), wird der Bericht im umrandeten Fenster ausgegeben.

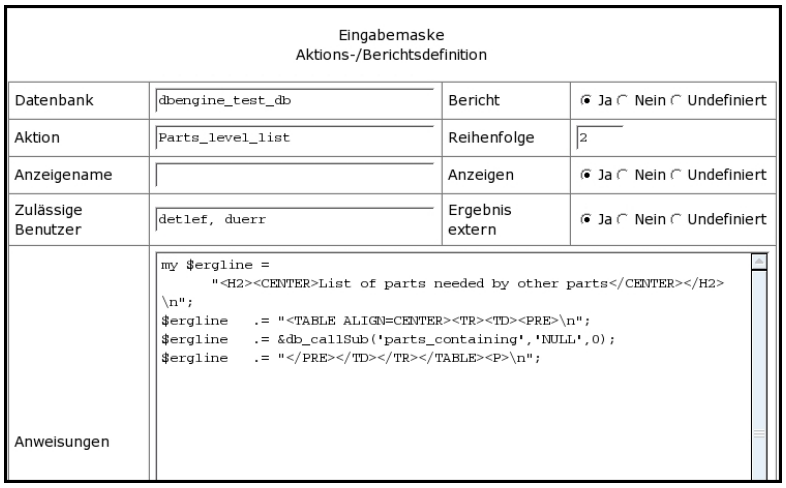
Außerdem gibt es die Möglichkeit, viele Prozeduren aufzurufen, die in dem DBENGINE-Tool schon definiert sind. Die wesentlichen Prozeduren sind in der Tabelle 3.10, „Die wichtigsten aufrufbaren Perl-Prozeduren von DBENGINE“ zusammengefasst.

Tabelle 3.10. Die wichtigsten aufrufbaren Perl-Prozeduren von DBENGINE

Prozedurname	Parameter	Ergebnis	Funktion
`db_db_get`	(`dbase`, ‹SELECT-Kommando›, ‹Bindvalues›)	DB-Handle	Aus der geöffneten Datenbank werden mit dem SELECT-cmd und den SELECT-Parametern (Bindvalues) die gewünschten Sätze gelesen, zurückgeliefert wird ein DB-Handle, über das dann die einzelnen Sätze angefordert werden können. Dies kann z.B. mit der Prozedur **`&db_db_hashref(‹DB-Handle›`** oder mit **`‹DB-Handle›->fetch`** geschehen.
`db_db_get_nrows`	(`dbase`, ‹SELECT-cmd›, Bindvalues)	(Anzahl Zeilen, DB-Handle)	Diese Prozedur arbeitet genauso, wie die Prozedur `db_db_get`. Zusätzlich liefert sie die Anzahl der gefundenen Sätze zurück.
`db_db_hashref`	(DB-Handle)	Ref-Hash-Array	Diese Prozedur liefert eine Referenz auf ein Hash-Array zurück. Der Key für den Zugriff auf die Werte ist jeweils der im früheren SELECT-Statement definierte Feldname.
`db_get_row`	(Tabelle, OID) (s. Glossar OID)	-	Diese Prozedur liest aus der gewünschten Tabelle den Satz mit der OID aus und schreibt die Werte in das Hash-Array `%values`, auf das über die Referenz `$rvalues` zugegriffen werden kann. Der Key ist jeweils der Feldname.
`db_SQL2value`	(Feldname, Wert)	Darstellung des Wertes	Diese Prozedur ermittelt aus der Darstellung des Wertes aus der Datenbank den Wert, wie er im Browser angezeigt werden soll. Der Wertetyp wird aus den Feldbeschreibungen ermittelt. Der Key ist jeweils der Feldname.
`db_callSub`	(Prozedur, par1, par2, ...)	Proz-Erg	Über diese Prozedur können in der Tabelle `equation` definierte Prozeduren mit den Parametern (par1, par2, ...) aufgerufen werden. Sie liefert das Ergebnis der Prozedur zurück.
`gtext`	(Index)	Langtext	Diese Prozedur liefert zu einem Index den sprachspezifischen Text aus der Tabelle `langtexts`(s. Abschnitt 3.3.3, „Die Tabelle langtexts (Sprachspezifische Texte)“). Ausgewählt wird der Text je nach Browsereinstellung beim Anwender und vorhandenen Sprachen für diesen Index.
`dbWarnText`	(Format, wert1, wert2, ...)	Warnhinweis	Diese Prozedur erzeugt aus dem Format und den Werten (wert1, wert2, ...) einen Warnhinweis, der dann mit dem **`print`**-Statement direkt ausgegeben werden kann.

Prozedurname	Parameter	Ergebnis	Funktion
`dbWarning`	(Format, wert1, wert2, ...)	-	Diese Prozedur erzeugt einen Warnhinweis wie die Prozedur `dbWarnText` und gibt diesen Warnhinweis sofort aus.
`dbError`	(Format, wert1, wert2, ...)	-	Diese Prozedur erzeugt eine Fehlermeldung beim Anwender, wobei der erzeugte Text auf die gleiche Weise entsteht, wie bei `dbWarning`. Doch kehrt die Prozedur nicht zur aufrufenden Stelle zurück, sondern beendet das Programm DBENGINE
`pr_get_parms`	(Name)	Parameter	Diese Prozedur liefert den Parameterwert aus dem Parameterfeld, einem Hasharray, der unter dem Namen dort gesetzt ist.
`pr_set_parms`	(Name, Wert)	-	Diese Prozedur setzt im Hasharray der Parameter den Parameter mit dem angegebenen Namen mit dem angegebenen Wert.
`pr_get_parmfields`	(Name, Anzeigename, Datentyp)	Ergebniswert	Von den vom Browser geleiferten Daten werden die für den Parameternamen Name geholt und gegebenenfalls in ein einziges Datum des angegebenen Datentyps geändert. Diese prozedur ist vor allem dann wichtig, wenn Werte aus einem Multiplepopup-Feld geholt werden müssen.

Abbildung 3.8. Beispiel einer Aktionsdefinition



Ein anderes Beispiel ist die in der Steuerleiste (Sie ist in der Abbildung 2.3, „Beispiele von Steuerleisten“ zu sehen) angezeigte Aktion *`Check_Order`* in der Testdatenbank (s. Abbildung 2.7, „Beispiel eines Berichtes“). In dieser umfangreicheren Aktion werden die wesentlichen Daten direkt ausgegeben und lediglich eine positive bzw. negative Erfolgsmeldung als Rückmeldung aus der `eval`-Prozedur gemeldet (ein Teil ihres Codes ist in dem folgenden Listing zu sehen).

```
if(@mis_parts) {
 print "<h2><center>Missing parts in orders</center></h2>\n";
 print "<table border cellspacing=5 align=center>\n<tr>" .
       "<th>Ord_no</th><th>title</th>" .
       "<th>Ord_pos</th><th>Ordered part_no</th>" .
       "<th>Missing part_no</th>" .
       "<th>Missing part description</th></tr>\n";
 foreach my $mispart (@mis_parts) {
        my ($ordert, $part_ord, $partd) = split(/;;;/,$mispart,3);
        my ($ord_no, $titel) = split(/:::/,$ordert,2);
        my ($part_no, $ord_pos) = split(/:::/,$part_ord,2);
        my ($partn, $descr) = split(/:::/,$partd,2);
        print "<tr><td>$ord_no</td><td>$titel</td><td>$ord_pos</td>" .
       "<td>$part_no</td><td>$partn</td><td>$descr</td></tr>\n";
 }
 print "</table><br />\n";
 my $missorders = @mis_parts;
 $erg = "Not all orders ok: $missorders of $orderpno orderpositions" .
 " in $orderno existent orders not satisfied";
```

Das Ergebnis des Aufrufs dieser Aktion zeigt (s. Abbildung 2.8, „Beispiel eines extern definierten Berichtes“) dann eine Tabelle ohne den Ergebnisbericht.

Wie aus den beiden Beispielen zu ersehen ist, erscheint die gerahmte HTML-Tabelle nicht in jedem Fall. Es ist eine einspaltige Tabelle. Dabei ist in der zweiten Zeile (unter der Überschrift) entweder der gesamte Bericht (s. Abbildung 2.7, „Beispiel eines Berichtes“) oder nur der Erfolg bzw. Misserfolg (gewollt durch ein gezieltes **`die`**-Kommando oder ungewollt durch fehlerhaften Ablauf der angeforderten Perl-Prozedur) zu sehen. Ein Beispiel dafür gibt die Abbildung 3.9, „Beispiel einer Aktionsdefinition“ .

Abbildung 3.9. Beispiel einer Aktionsdefinition

Die dritte Möglichkeit ist, dass die Erfolgsmeldung ganz weggelassen wird. Dann muss das Feld *`ext_result`* auf `false` gesetzt werden, wie es für den Bericht, der durch den Aufruf von *`Order-list`* aus der Steuerleiste (s. Abbildung 2.3, „Beispiele von Steuerleisten“) erzeugt wird, geschehen ist (s. Abbildung 2.8, „Beispiel eines extern definierten Berichtes“).

3.2. Die Detailtabellen

3.2.1. Die Tabelle `relation` (*Definition von Tabellenbeziehungen*)

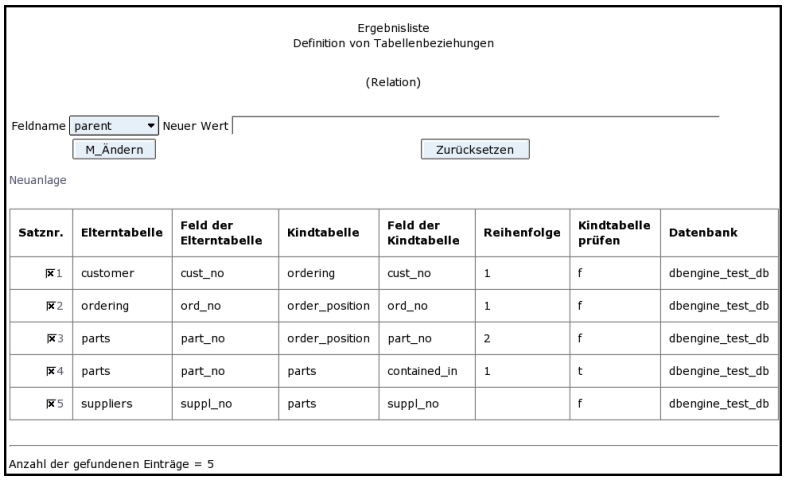
In dieser Tabelle werden die Beziehungen der Tabellen einer Datenbank untereinander (s. Glossery über die Relationalen Beziehungen einer relationalen Datenbank Querverweis), also die Querverweise (s. Abschnitt 2.2.2.3, „Der Querverweisbereich“) festgehalten. Sie definiert, welche Tabelle welcher anderen über- oder untergeordnet ist. Diese Tabelle enthält folgende Felder:

Tabelle 3.11. Die Felder der Tabelle `relation`

Feldname	Anzeigename	Bedeutung
dbasename	*Datenbank*	Name der Anwender-Datenbank
parent	*Elterntabelle*	Name der übergeordneten Tabelle.
parentfield	*Feld der Elterntabelle*	Feldname in der übergeordneten Tabelle, die die Relation definiert.
child	*Kindtabelle*	Name der abhängigen Tabelle.
childfield	*Feld der Kindtabelle*	Name des Feldes in der abhängigen Tabelle, durch das die Abhängigkeit spezifiziert wird.
childorder	*Reihenfolge*	Reihenfolge, in der diese Referenzen in der entsprechenden Ergebnismaske erscheinen sollen.
childcheck	*Kindtabelle*	Boolesches Feld, das angibt, ob bei **`UPDATE`** oder **`DELETE`** in der Elterntabelle überprüft werden soll, ob in der Kindtabelle noch Einträge vorhanden sind. In diesem Fall wird die angestoßene Aktion nicht durchgeführt.

Die Eintragungen in dieser Tabelle dienen im wesentlichen zur Erzeugung von Querverweisen (s. Abschnitt 2.2.2.3, „Der Querverweisbereich“). In der Testdatenbank beispielsweise existieren 5 Relationen (s. Abbildung 3.10, „Beispiel von Definitionen in der Tabelle relation“), die dann auch in den Querverweisen (s. Abbildung 2.5, „Beispiel einer Ergebnismaske“) nach Bedarf wieder auftauchen.

Abbildung 3.10. Beispiel von Definitionen in der Tabelle `relation`



In dem Eintrag von Satznr. 4 der Abbildung 3.10, „Beispiel von Definitionen in der Tabelle relation“ ist für *childcheck* `t` eingetragen. In diesem Fall wird beim Ändern oder Löschen eines Satzes in der `parts`-Tabelle geprüft, ob dieses Element Bestandteil eines anderen ist. Ist das der Fall, so wird das Löschen oder Ändern nicht durchgeführt und der Anwender gewarnt, dass erst die anderen, die abhängigen, Sätze geändert werden müssen.

3.2.2. Die Tabelle `virtual` (*Definition berechneter Felder*)

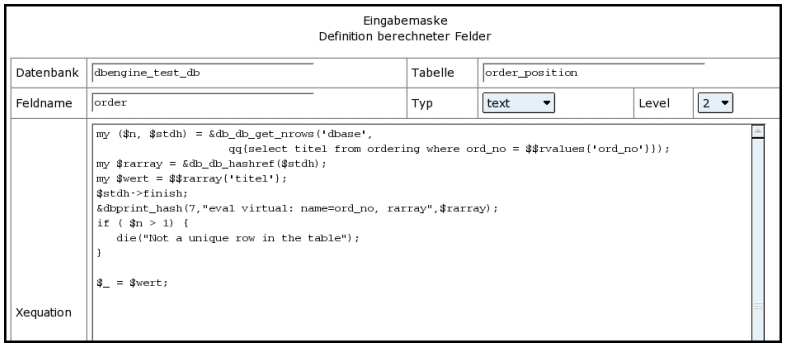
Werden Felder einer Ausgabe, sei es im Listing oder in einer Ergebnismaske, vor der Anzeige berechnet und können nicht eingegeben werden, so wird in dieser Tabelle die Berechnungsvorschrift für diese Felder hinterlegt. Die Tabelle `virtual` enthält folgende Felder:

Tabelle 3.12. Die Felder der Tabelle `virtual`

Feldname	Anzeigename	Bedeutung
dbasename	*Datenbank*	Name der Anwender-Datenbank
tablename	*Tabelle*	Name der geforderten Tabelle.
fieldname	*Feldname*	Feldname für das berechnete Feld.
fieldtype	*Typ*	Typ des berechneten Feldes.
level	*Level*	Level als Ordnungskriterium (Reihenfolge der Felder).
xequation	*Xequation*	Berechnungsprozedur für das Feld (Perlcode).

Die Berechnungsvorschriften gelten sowohl für virtuelle Felder einer Ergebnismaske, die über `designinfo` (s. Die Tabelle designinfo) konfiguriert werden, als auch für berechnete Felder in einem Listing, die über 'tabledesign' (s. Abschnitt 3.1.3, „Die Tabelle tabledesign (Listingdesign)“) konfiguriert werden. Die Abbildung 3.11, „Beispiel einer virtual-Felddefinition“ zeigt eine Ergebnismaske für die `virtual`-Konfiguration eines Feldes in der Testdatenbank.

Abbildung 3.11. Beispiel einer `virtual`-Felddefinition



Mit Hilfe dieser Definition ist im Beispielslisting (Abbildung 3.12, „Beispiel eines Listings mit berechneten Feldern (virtual)“) der Name (`title`) für das Feld `Order` ermittelt worden. Analoges gilt für das Feld `Description`.

Abbildung 3.12. Beispiel eines Listings mit berechneten Feldern (`virtual`)

Satznr.	Ord_Pos	Ord_No	Order	Part_No	Description	Account	Price
☑ 1	1	1	Musterorder	3	Gehaeuse Mini	1	29,50
☑ 2	2	1	Musterorder	12	Speicherchip 256MB	2	318,00
☑ 3	3	1	Musterorder	9	CD-ROM-Laufwerk	1	39,00

In der Testdatenbanktabelle `Ordering` sind weitere Beispiele zu finden, auch mit einem anderen Zieldatentyp.

3.2.3. Die Tabelle dbengine_user (*Definition zulässiger Datenbankbenutzer*)

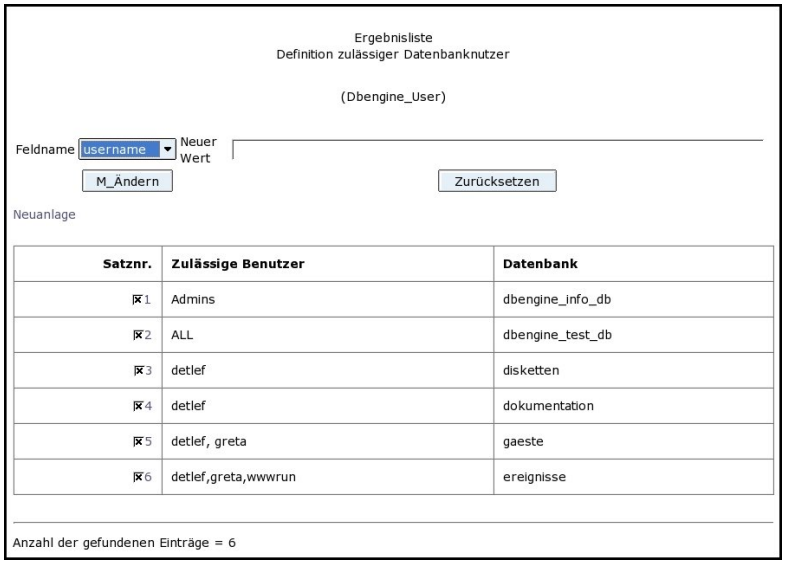
Diese Tabelle dient der Festlegung, welche Datenbank von welchen Benutzern genutzt werden darf. Nur die Tabellen der Datenbanken, die hier für den aktuellen Benutzer gefunden werden, werden in der Steuerleiste (s. Abschnitt 2.2.1, „Die Steuerleiste“) aufgeführt. Diese Tabelle besitzt folgende Felder:

Tabelle 3.13. Die Felder der Tabelle `dbengine_user`

Feldname	Anzeigename	Bedeutung
dbasename	*Datenbank*	Name der Anwender-Datenbank
username	*Zulässige Benutzer*	Liste von Benutzernamen, die diese Datenbank aufrufen dürfen.

In der Abbildung 3.13, „Beispiele für Einträge in die Tabelle dbengine_user)“ sind die Beispiele für unterschiedliche Einträge vorhanden. Der Eintrag `ALL` im Feld `username` bedeutet, dass jeder Benutzer diese Datenbank anfragen kann. Es gibt noch einen weiteren Eintrag mit Sonderbedeutung, das ist der Eintrag `Admins`. Dieser Eintrag heißt, dass nur die Administratoren (s. Abschnitt 3.3.4, „Die Tabelle configs (Konfigurationswerte)“) diese Datenbank benutzen dürfen. Das gilt insbesondere für die Konfigurationsdatenbank selbst.

Abbildung 3.13. Beispiele für Einträge in die Tabelle `dbengine_user`)



Ergebnisliste
Definition zulässiger Datenbanknutzer

(Dbengine_User)

Feldname username Neuer Wert

M_Ändern Zurücksetzen

Neuanlage

Satznr.	Zulässige Benutzer	Datenbank
1	Admins	dbengine_info_db
2	ALL	dbengine_test_db
3	detlef	disketten
4	detlef	dokumentation
5	detlef, greta	gaeste
6	detlef,greta,wwwrun	ereignisse

Anzahl der gefundenen Einträge = 6

Die weiteren Einträge in dieser Tabelle sind Beispiele für Benutzernamen, um den üblichen Gebrauch zu kennzeichnen. Wie aus der Tabelle zu sehen ist, können mehrere Namen als Liste, getrennt durch Kommata, angegeben werden. In diesem Fall darf jeder der angegebenen Benutzer die entsprechende Datenbank aufrufen. Unabhängig von diesen Eintragungen sind natürlich die bei den einzelnen Tabellen einer Datenbank durch das **`GRANT`**-Statement angegebenen Benutzereinschränkungen gültig.

3.2.4. Die Tabelle `equation` (*`Definition von Unterprogrammen`*)

Tritt in einzelnen eval-Prozeduren immer wieder die gleiche Abfolge von Statements auf, so kann diese Folge als Prozedur in der Tabelle `equation` eingetragen werden. In dieser Tabelle gibt es folgende Felder:

Tabelle 3.14. Die Felder der Tabelle `equation`

Feldname	Anzeigename	Bedeutung
`dbasename`	*`Datenbank`*	Name der Anwender-Datenbank
`eqname`	*`Bezeichner`*	Bezeichner der Perl-Routine, unter dem sie aufgerufen wird.
`content`	*`Inhalt`*	Perl-Statements

Die Anwendung einer solchen Prozedur geschieht über den Aufruf von `&db_callSub`. Dabei wird als erster Parameter der `eqname` der Prozedur übergeben. Alle weiteren Parameter stellen dann die Parameter für die Subroutine bzw. Funktion (falls sie einen Wert zurück liefert) dar. Für die Benutzung in Prozeduren stehen die Variablen, wie sie in der Tabelle 3.15, „Für eine Berechnung mit eval zur Verfügung stehende Werte“ beschrieben sind, zur Verfügung:

Tabelle 3.15. Für eine Berechnung mit `eval` zur Verfügung stehende Werte

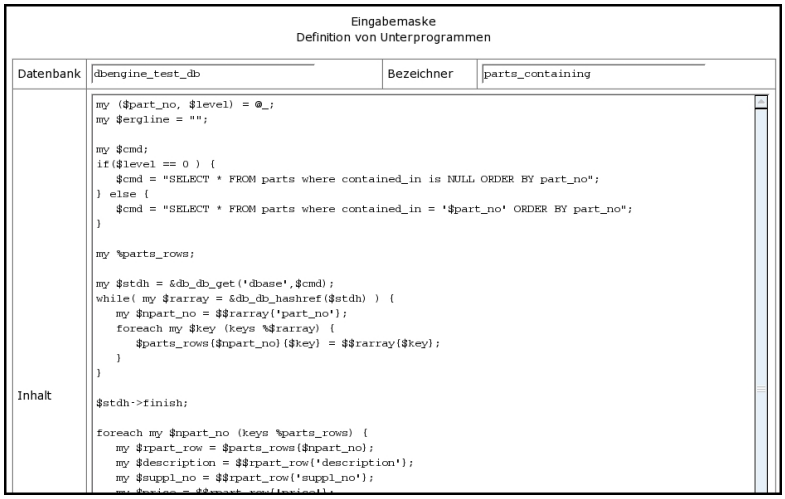
Name der Variablen	Bedeutung
`$rfielddesc`	Referenz auf ein Hash-Array, mit dem Feldnamen als Key, das als Wert ein Hash-Array mit allen notwendigen Angaben enthält. Es sind dies die

<table>
<tr><th>Name der Variablen</th><th>Bedeutung</th></tr>
<tr><td></td><td>Daten für das Feld aus der Tabelle und die Beschreibungen des Feldes aus <code>designinfo</code>. Es handelt sich dabei je Feld um folgende Informationen:
<table>
<tr><th>TYPE</th><th>Datentyp des Feldes aus der Datenbanktabelle</th></tr>
<tr><td>DISPLAY</td><td>Wert von <code>displayinfo</code></td></tr>
<tr><td>DISPTABLE</td><td>Wert von <code>disptable</code></td></tr>
<tr><td>DISPVFIELD</td><td>Wert von <code>dispvfield</code></td></tr>
<tr><td>DISPDFIELD</td><td>Wert von <code>dispdfield</code></td></tr>
<tr><td>DISPSTR</td><td>Wert von <code>displaystring</code> bzw. über <code>lang(..)</code> ermittelter Wert</td></tr>
<tr><td>XDEFAULT</td><td>Perlprozedur für den default-Wert</td></tr>
<tr><td>EVAL</td><td>Perlprozedur <code>xwvaluation</code></td></tr>
<tr><td>ROWS</td><td>Wert von <code>rowspan</code></td></tr>
<tr><td>COLS</td><td>echter Wert ermittelt über <code>colspan</code></td></tr>
</table></td></tr>
<tr><td><code>$rvalue</code></td><td>Referenz auf ein Hash-Array mit den Feldnamen als Key und den Werten der Felder aus der Datenbank als Inhalt.</td></tr>
<tr><td><code>$rparms</code></td><td>Referenz auf ein Hash-Array mit allen Parametern, die entweder berechnet oder aus dem Aufruf von <code>dbengine.cgi</code> kommen.</td></tr>
<tr><td><code>$rconfigs</code></td><td>Referenz auf ein Hash-Array mit allen Konfigurationen aus der Konfigurationstabelle (s. Abschnitt 3.3.4, „Die Tabelle configs (Konfigurationswerte)“)</td></tr>
</table>

Außerdem gibt es die Möglichkeit, viele Prozeduren aufzurufen, die in dem DBENGINE-Tool schon definiert sind. Die wesentlichen Prozeduren sind in Abschnitt 3.1.4, „Die Tabelle action (Aktions-/Berichtsdefinition)“ in der Tabelle 3.10, „Die wichtigsten aufrufbaren Perl-Prozeduren von DBENGINE“ zusammengefasst.

In der Abbildung 3.14, „Beispiel einer Anwenderprozedur“ ist als Beispiel der Ausschnitt einer selbst definierten Prozedur zu sehen. Sie wird benutzt in dem Bericht in Abbildung 3.9, „Beispiel einer Aktionsdefinition“ in Abschnitt 3.1.4, „Die Tabelle action (Aktions-/Berichtsdefinition)“. Es ist eine rekursive Prozedur, die sich selbst wieder aufruft. Sie macht ausgiebigen Gebrauch von Datenbankaufrufen und liefert die gesamte Liste, wie sie in Abbildung 2.7, „Beispiel eines Berichtes“ in Abschnitt 2.2.4, „Aktionen und Berichte“ gezeigt ist.

Abbildung 3.14. Beispiel einer Anwenderprozedur



3.3. Die Wertebereichstabellen

3.3.1. Die Tabelle `databaseconfigs` (*`Datenbankspez. Konfigurationen`*)

Im Abschnitt 3.3.4, „Die Tabelle configs (Konfigurationswerte)“ werden die generellen Konfigurationsmöglichkeiten des Systems beschrieben. Sie gelten für alle Datenbanken. Die Tabelle `databaseconfigs` hingegen gestattet es diese generellen Konfigurationen für jede einzelne Datenbank speziell zu setzen. Damit werden die generellen Konfigurationen dann für die spezifizierte Datenbank überschrieben. Dazu enthält die Tabelle die folgenden Felder:

Tabelle 3.16. Die Felder der Tabelle `databaseconfigs`

Feldname	Anzeigename	Bedeutung
`dbasename`	`Datenbank`	Name der Anwender-Datenbank
`configname`	`Name`	Name der Konfigurationsvariablen, für die der Wert dieser Tabelle genutzt werden soll.
`configvalue`	`Wert`	Wert der benutzt werden soll.

Im Feld `Name` wird der Name eingetragen, den die entsprechende Variable gegebenenfalls auch in der Tabelle `Configs` besitzt. Außer diesen Werten sind aber auch weitere, eigene Konfigurationsvariable eintragbar, die dann in Aktionen, Berichten oder auch sonstigen Listings auswertbar sind. Drei spezielle Werte werden vom System mit ausgenutzt (s. auch Abbildung 3.15, „Einige Einträge aus der Tabelle Databaseconfigs“):

- **start_mode:** Unter diesem Namen kann für den sofortigen Start einer speziellen Seite der angeforderte `mode` angegeben werden. Möglich sind die Werte `plain` und `action`. Bei dem Wert `plain` kann sofort eine Tabelle und ein Wert daraus ausgewählt werden, deren Inhalt angezeigt werden soll; unter dem Wert `action` kann eine Aktion oder ein Bericht ausgewählt werden, die/der beim Start sofort auszuführen ist.
- **start_fname:** Der Wert in dieser Konfigurationsvariablen bestimmt den Namen eines Parameters, der in der URL zum Aufruf der Startseite gesetzt werden soll.

- **start_fvalue:** Diese Konfigurationsvariable definiert den Wert des Parameters, der unter `start_fname` definiert wurde.

Abbildung 3.15. Einige Einträge aus der Tabelle `Databaseconfigs`

Satznr.	Datenbank	Name	Wert
1	dbengine_info_db	single_sentence	false
2	dbengine_test_db	single_sentence	false
3	ereignisse	single_sentence	true
4	dbengine_test2_db	comment	Test Datenbank zu Behandlung von Schemata
5	ereignisse	start_mode	plain
6	ereignisse	start_fname	table
7	ereignisse	start_fvalue	ereignis
8	ereignisse	std_actions	detlef:__STD_global_SQL
9	dbengine_test2_db	std_actions	detlef:__STD_global_SQL,__STD_global_SQL_do,__STD_global_SQL_execute
10	ereignisse	debug	
11	disketten	debug	
12	dbengine_test_db	search_pathes	Admins:ALL;ALL:public

In dem Listing in Abbildung 3.15, „Einige Einträge aus der Tabelle Databaseconfigs“ ist ein Beispiel für die Startvariablen angeführt. Die Datenbank `ereignisse` startet beim Aufruf stets mit der Tabelle `ereignis`. Dies könnte aber durch eine entsprechende Änderung umgesetzt werden auf die Aktion `Ereignis suchen`. Ein Listing dieser Aktion ist in der Tabelle `Actions` (s Abschnitt 3.1.4, „Die Tabelle action (Aktions-/Berichtsdefinition)“) zu finden. In dem Listing sind außerdem Beispiele für die Änderung der globalen Konfigurationsvariablen `single_sentence` (s. Abschnitt 3.3.4, „Die Tabelle configs (Konfigurationswerte)“). In den Datenbanken `dbengine_info_db` und `dbengine_test_db` sollen Suchergebnisse, bei denen nur ein Satz gefunden wurde sofort in der *`plain_mode`*-Darstellung ausgegeben werden, während bei der Datenbank `ereignisse` auch bei nur einem Satz die Listingausgabe bevorzugt wird.

3.3.2. Die Tabelle `valuelist` (*`Wertebereichsdefinition`*)

Im Abschnitt 3.1.4, „Die Tabelle action (Aktions-/Berichtsdefinition)“ wird für den Fall eines *popup*-Feldes schon auf diese Tabelle hingewiesen. Sie enthält die Werte für eine Auswahl. Zur Definition der Werte enthält die Tabelle folgende Felder:

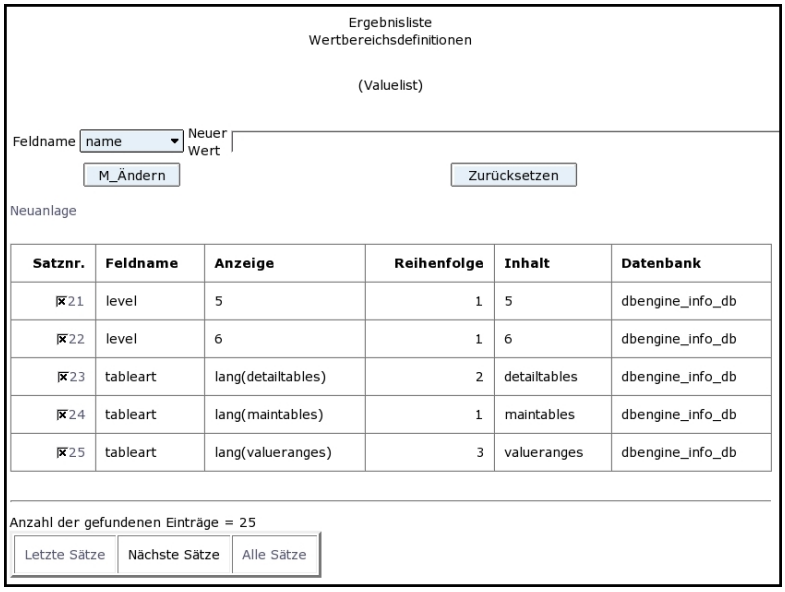
Tabelle 3.17. Die Felder der Tabelle `valuelist`

Feldname	Anzeigename	Bedeutung
`dbasename`	`Datenbank`	Name der Anwender-Datenbank
`name`	`Feldname`	Name des Popup-Feldes für das dieser Wert der Tabelle genutzt werden soll.
`content`	`Inhalt`	Wert der benutzt werden soll.
`display`	`Anzeige`	Wert der im Popup-Fenster angezeigt werden soll, um dann für die Datenbank den Wert aus `content` bereitzustellen. Für diesen Wert gilt das gleiche, wie schon in der Tabelle `designinfo`(s. Die Tabelle designinfo) für das Feld `displaystring` (s. Tabelle 3.3, „Die Felder der Tabelle desi-

Feldname	Anzeigename	Bedeutung
		gninfo“), wenn ein Anzeigewert als Funktion `lang(...)` angegeben ist.
`orderno`	`Reihenfolge`	Dieser Wert dient der Sortierung der Werte im Popup-Fenster, um so wichtige Werte an den Anfang stellen zu können.

Die Abbildung 3.16, „Ausschnitt aus einem Listing der Tabelle valuelist“ zeigt einen Ausschnitt aus einem Listing der Tabelle `valuelist`. Es sind im wesentlichen die Werte für das Feld `tableart` der Tabelle `basedesign`(s. Abschnitt 3.1.1, „Die Tabelle basedesign (Tabellendefinition)“). Hier ist zu sehen, dass der Anzeigewert sich sprachspezifisch von dem Wert, der zum Eintrag in die Datenbank ausgewählt wird unterschieden ist.

Abbildung 3.16. Ausschnitt aus einem Listing der Tabelle `valuelist`



Die Abbildung zeigt außerdem, dass das Gesamtlisting mehr als 1 Bildschirm ist, weshalb am Ende der Abbildung die Button zur Auswahl der Bildschirmseite angegeben sind.

3.3.3. Die Tabelle `langtexts` (*Sprachspezifische Texte*)

In dieser Tabelle werden die sprach bezogenen Texte hinterlegt. Es wird dabei unterschieden in Button-Texte und alle übrigen Texte wie Hinweise, Überschriften usw. Gesucht wird stets über einen Index, der zusammen mit der Sprache den primären Schlüssel der Tabelle bildet. Die Tabelle enthält folgende Felder:

Tabelle 3.18. Die Felder der Tabelle `langtexts`

Feldname	Anzeigename	Bedeutung
`indextext`	`Index`	
		Textueller Index, über den der sprach bezogene Text gesucht wird.

Feldname	Anzeigename	Bedeutung
translate_text	Ausgabetext	Text, der für den Index ausgegeben wird.
button	Button	Boolesches Feld, gibt an, ob der Text für einen Button benötigt wird, oder ob es sich um einen Standardtext handelt.
language	Sprache	Sprache, für die der Ausgabetext gilt.

Das Feld language (s. Tabelle 3.18, „Die Felder der Tabelle langtexts“) enthält die Kennzeichnungen für die Sprache in der Form, wie sie auch in den Browsern angegeben werden. Also steht de für *Deutsch*, en für *Englisch* usw. (s. Abbildung 3.17, „Beispiel einer sprachspezifischen Textdefinition“).

Das Feld translate_text (s. Tabelle 3.18, „Die Felder der Tabelle langtexts“) kann in der Form eines Formates für die Perl-Prozedur **print** angegeben werden. Beim Aufruf der Prozedur **gtext**(s. Tabelle 3.10, „Die wichtigsten aufrufbaren Perl-Prozeduren von DBENGINE“). müssen dann die im Format benötigten Werte als weitere Parameter mitgegeben werden. Die Eingabe- oder Ergebnismaske für die Tabelle langtexts ist in der Abbildung 3.17, „Beispiel einer sprachspezifischen Textdefinition“ gezeigt.

Abbildung 3.17. Beispiel einer sprachspezifischen Textdefinition

3.3.4. Die Tabelle configs (*Konfigurationswerte*)

In der Tabelle configs werden alle Konfigurationsabhängigkeiten hinterlegt. Sie hat folgenden Aufbau:

Tabelle 3.19. Die Felder der Tabelle configs

Feldname	Anzeigename	Bedeutung
index	Index	Textueller Index, über den der sprach bezogene Text gesucht wird und unter dem er in dem Hash-Array '%configs' abgelegt wird. Zugreifbar sind die Werte dann in eigenen Prozeduren über die Referenz auf das hash-Array $rconfigs(s. Tabelle 3.15, „Für eine Berechnung mit eval zur Verfügung stehende Werte“).
values	Wert	Konfigurationsparameter, der über den Index gefunden wird.
comment	Bemerkung	Beschreibung des Konfigurationsparameters, dient nur der besseren Handhabung.

Mit Hilfe dieser Tabelle können also beliebige aber Datenbank unabhängige Konfigurationen vorgegeben werden. Diese Tabelle kann jederzeit systemspezifisch erweitert werden. In der Abbildung 3.18, „Standardkonfigurationswerte“ sind die Standardwerte mit ihren Kommentaren eingetragen. Der Wert für den

Schlüssel zum verkrypten der Daten, die mit dem **GET**-Kommando vom und an den Server geliefert werden, allerdings wurde gelöscht..

Abbildung 3.18. Standardkonfigurationswerte

Index	Wert	Bemerkung
admins	postgres	List of usernames of people working as dbengine-admins separated by commas
allowed_dbs	()	names of only allowed databases to choose for using with dbengine
bgcol	#FFFFFF	background color for our generated pages
bgmcol	#C4DDFF	background color for our generated menus.
currency_de	€;,:.	Monetory-Symbol for currencies possible are: €/EUR, $/USD, £/GBP, JPY. Additional there must be the sign for decimalpoint and for 10^^3-parts. All three parts have to be divided by colon (:)
currency_gb	£:.:,	Monetory-Symbol for currencies possible are: €/EUR, $/USD, £/GBP, JPY. Additional there must be the sign for decimalpoint and for 10^^3-parts. All three parts have to be divided by colon (:)
currency_us	$:.:,	Monetory-Symbol for currencies possible are: €/EUR, $/USD, £/GBP, JPY. Additional ther must be the sign for decimalpoint and for 10^^3-parts. All three parts have to be divided by colon (:)
debug	7	Debuglevel for all Databases
exclude_dbs	"template0", "template1"	names of databases not allowed to choose for using with dbengine separated by commas
keypart	**********	part of the serverspecific key for encryption
maxinpwidth	75	Maximum chars in an Inputfield
maxrecno	20	Maximum number of records shown in a table-listing
mreltuplelines	4	Lines of a multipleselectfield
reltupleanz	25	More than these tuples for a relationPopup gives a field of reltuplelines
reltuplelines	2	Lines of a great Selectfield
shortenings	"pc","hw","sw","atm","IGM"	list of shorts to give all characters in uppercase separated by comma. Only for headlines and menus.
single_sentence	true	If Tablelisting for ntuples==1 should be done: set true. If for ntuples==1 a designed form should be displayd set false.
std_actions	Admins:ALL;DB_OWNER:__STD_global_SQL.*,__STD_.*user.*,__STD_.*sch.*	Definition of global actions for specific user. This global action defines for every database some standard actions for every user, who is a member of the Admins-group or the owner of the actual database. The action gives the user the possibility to define SQL-Statements which are allowed in the PostgreSQL - DBMS.
templ	/usr/local/dbengine/src/templates	Directory for predefined formulas for tablemasks
undefbool	f	if boolean value from reading the table undefined, set boolean to false(f) or undef(u)
upfirst	1	a true value means first character of word-parts in uppercase mode

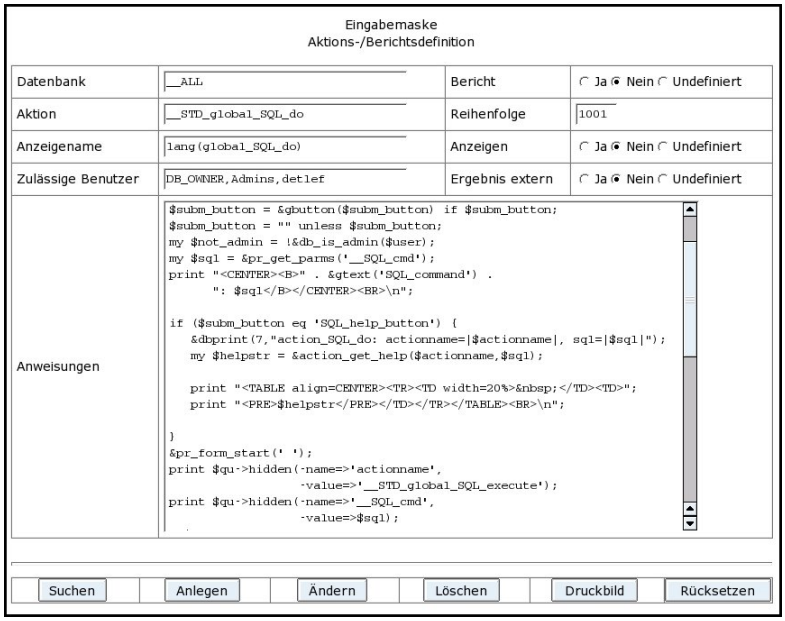
3.3.5. Die Tabelle `helpfile` (*`Hilfetexte`*)

Diese Tabelle dient der Bereitstellung von Hilfetexten für Aktionen. Ihre Anwendung wird aus einer Aktion mit dem Aufruf **`&action_get_help`** aufgerufen. Diese Prozedur hat 2 Parameter:

- Den Aktionsnamen und
- den Namen der Hilfegruppe (des Hilfetextes).

Ein Beispiel eines solchen Aufrufes findet sich in der Standardaktion zum Anwenden der allgemeinen SQL-Schnittstelle (s. Abbildung 3.19, „Beispiel eines Aufrufs von Hilfetexten“).

Abbildung 3.19. Beispiel eines Aufrufs von Hilfetexten



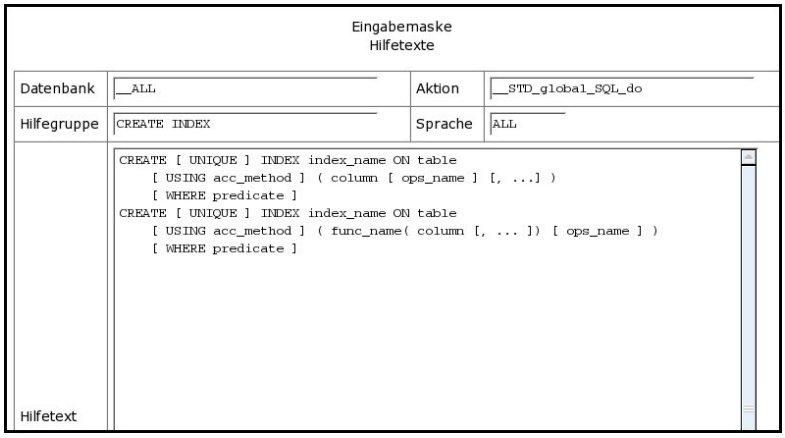
Die Tabelle `helpfile` besitzt zur Festlegung der Hilfetexte folgende Felder:

Tabelle 3.20. Die Felder der Tabelle `helpfile`

Feldname	Anzeigename	Bedeutung
`dbasename`	`Datenbank`	Name der Datenbank, für die dieser Hilfetext gilt.
`actionname`	`Aktion`	Name der Aktion, für die dieser Hilfetext gilt.
`helpname`	`Hilfegruppe`	Name der Hilfegruppe, .für die dieser Hilfetext gilt.
`languages`	`Sprache`	Angabe der Sprache, für die dieser Hilfetext gilt.
`helptext`	`Hilfetext`	Angabe eines vorformatierten Hilfetextes. Die Ausgabe dieses Textes erfolgt mit den HTML-Tags **`<pre>`** und **`</pre>`**.

Im Aufruf **`my $helpstr = &action_get_help($actionname,$sql);`** in Abbildung 3.19, „Beispiel eines Aufrufs von Hilfetexten“ werden nur zwei Parameter – wie oben angegeben – mitgegeben, denn die Datenbank ist die aktuell geöffnete und die Sprache wird aus der aktuellen Browsereinstellung entnommen.

Abbildung 3.20. Beispiel einer Hilfetextdefinition



Ein Beispiel für die Definition eines Hilfetextes ist in der Abbildung 3.20, „Beispiel einer Hilfetextdefinition“ zu sehen. An dieser Definition kann man erkennen, dass als Sprache auch das Wort ALL eingegeben werden kann, sofern dieser Hilfetext sprachunabhängig ist. Soll der Hilfetext auch für alle Datenbanken gelten, so kann dies mit dem __ALL angegeben werden, wie in der Abbildung 3.20, „Beispiel einer Hilfetextdefinition“ zu sehen ist.

Ein Beispiel für die Anzeige eines Hilfetextes ist in der Abbildung 2.13, „Beispiel für einen SQL-Befehl mit Hilfeangabe“ schon gezeigt. Schaut man den Quelltext dieser Ausgabe an (im folgenden ist ein kleiner, leicht umformatierter Ausschnitt davon abgedruckt), so kann man die Umschließung des Hilfetextes mit den, hier hervorgehobenen, HTML-Tags **<pre>** und **</pre>** erkennen:

```
<div class="actionErg">
 <!-- SQL - Parametrierung --><h1>SQL - Parametrierung</h1>
 <form method=post action="dbengine.cgi">
 <table align="center"><tr><td width=20%> </td><td>
 <div class="actinh">
 <pre>CREATE [ UNIQUE ] INDEX index_name ON table
     [ USING acc_method ] ( column [ ops_name ] [, ...] )
       [ WHERE predicate ]
    CREATE [ UNIQUE ] INDEX index_name ON table
     [ USING acc_method ] ( func_name( column [, ... ]) [ ops_name ] )
       [ WHERE predicate ]
  </pre></div></td></tr></table>
```

Auf diese Weise kann beim Festlegen der Hilfetexte das Aussehen auf dem Bildschirm vorgegeben werden.

Anhang A. GNU Lesser General Public License version 3

Version 3, 29 June 2007

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

a. under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

b. under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a. Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

b. Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

a. Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

b. Accompany the Combined Work with a copy of the GNU GPL and this license document.

c. For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

d. Do one of the following:

   1. Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

   2. Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e. Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a. Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b. Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy’s public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

Glossar

Bildschirmmaske	Eine Bildschirmmaske ist ein mit einzelnen Eingabefeldern versehenes Formular, das auf dem Bildschirm erscheint und dort ausgefüllt werden kann. Wir unterscheiden hier Eingabe- und Suchmasken. Während die ersteren ausgefüllt werden, um in einer Datenbank neue Sätze zu kreieren, oder vorhandene ab zu ändern, dienen die Suchmasken zum Suchen von Sätzen in einer Datenbank, wobei in den Feldern Suchwerte vorgegeben werden können.
Binary Large Object (BLOB)	Mit „Binary Large Object“ bezeichnet man ein Datenbankobject, dass die üblichen Regeln für solche sprengt, sowohl von der Größe her, als auch vom Inhalt. BLOBs bei PostgreSQL verhalten sich ähnlich wie in SQL 2003.
Client-Server	Als eine Client-Server-Architektur bezeichnet man eine Systemarchitektur, in der auf einem System die Ansprachen erfolgen, auf Grund derer ein anderes System eine bestimmte Arbeit verrichten soll. Ein Beispiel für eine solche Architektur stellt das Internet dar: während der Nutzer des Internet seine Anfragen über den Browser (den Client) stellt, werden diese Anfragen von dem angewählten Webserver (dem Server also) erarbeitet und dem Browser zur Ausgabe wieder zur Verfügung gestellt. Ein Client-Server-System muss nicht auf getrennten Rechnern laufen, es kann auch auf dem selben Rechnersystem zur Verfügung gestellt werden.
Datei	Eine einzeln stehende Ansammlung von Daten. Hier aber im allgemeinen gebraucht als Bezeichnung für eine Relation einer Datenbank Siehe auch Datenbank, Tabelle.
Datenbank	Ein System, dass zum Sammeln und Verknüpfen von Daten dient. Die gebräuchlichsten Datenbanken sind heute relationale Datenbanken. Datenbanken bestehen im allgemeinen aus mehreren Relationen (oder auch Dateien) mit mehreren Attributen (auch und insbesondere hier Felder genannt). Verknüpft werden zwei Relationen über die Gleichheit von Werten von bestimmten Attributen dieser Relationen.
Dictionary	Ein Dictionary ist eine vorgegebene Menge von einzelnen Daten, die eine Beschränkung der möglichen Werte für ein bestimmtes Attribut einer bestimmten Relation einer Datenbank angeben. Siehe auch Datenbank.
Eingabefeld	Ein Eingabefeld ist ein Feld einer Bildschirmmaske , in das Daten eingegeben werden können, um Daten für eine Bearbeitung ein zu geben. Siehe auch Bildschirmmaske.
Eingabemaske	Siehe Bildschirmmaske.
Elterntabelle	Siehe qvwg.
Feld	Mit dem Begriff Feld wird hier ein Attribut einer Datenbankrelation bezeichnet. Ein anderer Ausdruck für Feld ist häufig auch der Begriff Spalte. In der Regel wird der Begriff Feld genutzt, wenn es um das Attribut eines einzelnen Datenbanksatzes geht, während der Begriff Spalte genutzt wird, um das gleiche Attribut in einer Menge von Datenbanksätzen zu benennen. Siehe auch Datenbank.
HTTP(S)	Im Internet bzw. einem Intranet werden für das Anzeigen von Webseiten in der Regel zwei Protokolle benutzt: `http` und `https`. Der Unterschied zwischen die-

	sen beiden Protokollen besteht in der Sicherheit gegen Missbrauch. Das Protokoll `http` überträgt alle Daten im Klartext, während das Protokoll `https` die Daten vor dem Versenden verschlüsselt und so gegen Missbrauch schützt.
Hyperlink	Ein Hyperlink ist ein Verweis von einer Dokumentstelle auf eine andere im gleichen oder in einem anderen Dokument.
Kindtabelle	Siehe qvwg.
Lizenz	Auch Open-Source-Produkte besitzen eine Lizenz, die das benutzen nicht verbietet, aber je nach ihrer Art, bestimmte Einschränkungen, vor allem bei der Weitergabe, macht. Die bekannteste Form der offenen Lizenz ist die GNU General Public License (GPL). Häufig, so auch in diesem Fall wird die weniger strenge GNU Lesser General Public License (LGPL) genutzt.
OID	Eine OID (**O**bject **Id**entifier) ist ein Wert, der einen Datenbanksatz eindeutig beschreibt. Die Bezeichnung OID stammt von der Datenbank PostgreSQL, die diese Werte automatisch einer Relation zuweisen kann, wenn dies in der Datenbankdefinition angegeben ist. In anderen Fällen ist es der Wert eines `primary key`, der auch mehrere Spalten umfassen kann.
Perl	Perl ist eine weithin bekannte Skriptsprache und wird sehr häufig, so auch hier, zur Programmierung umfangreicher Hilfstools im Internetserverbereich genutzt. Für Perl gibt es eine umfangreiche Liste von Bibliotheken, die bei Bedarf aus dem Internet über `CPAN` geladen werden können.
Plain Mode	Unter Plain Mode wird hier die Darstellung von Dateneingabe für Satzsuche oder das Ergebnis einer Satzsuche, sofern nur ein Satz gefunden wurde, in Form einer Such- oder Ergebnismaske verstanden. Siehe auch Bildschirmmaske.
Popupfeld	Ein Popupfeld ist ein Eingabefeld, in dem aus einer Menge von Daten ausgewählt werden kann. Diese Daten werden im allgemeinen in einer anderen Datei oder einfach in Form von einzelnen Werten (s. Abschnitt 3.3.2, „Die Tabelle valuelist (Wertebereichsdefinition)“) als „Dictionary“abgelegt.
PostgreSQL	PostgreSQL ist ein spezielles Datenbankmanagementsystem, das gegenüber relationalen Datenbanksystemen einige Erweiterungen, wie zum Beispiel Vererbung, besitzt.
public	Das `Schema public` hat seinen Namen daher, dass in der Regel alle Datenbanktabellen und -sichten in diesem `Schema` für *alle* Benutzer, die überhaupt Zugriff zu der Datenbank haben, sichtbar sind.
Querverweis	Als Querverweis wird hier die Beziehung einer Tabelle einer Datenbank zu einer anderen Tabelle der Datenbank bezeichnet. Diese Beziehung ist durch die Inhalte passender Felder gegeben. Häufig ist ein Feld x1 einer Relation y1 bezogen auf ein Feld x2 einer andren Relation y2. Ist x1 gleichzeitig ein Primärschlüssel von y1, so handelt es sich um eine 1:n Beziehung von y1 auf y2 (y1 ist Elterntabelle von y2, y2 ist Kindtabelle von y1).
Satz	Siehe Zeile.
Schema	Ein Schema ist eine Unterteilung einer Datenbank, die es ermöglicht, unterschiedliche Tabellen oder Sichten von Tabellen für verschiedene Nutzer zugänglich zu machen. Prinzipiell besitzt jede Datenbank das `Schemapublic` (s. Begriff

	public). In einer Datenbank können weitere Schemata deklariert werden, die dann vom Anwender speziell ausgewählt werden können. Am häufigsten wird als Schema der Name des Anwenders genommen, insbesondere bei Datenbanken, in denen viele verschiedene Anwender Tabellen anlegen und bearbeiten dürfen. Für diesen Fall ist schon für den Standardfall als Schemapfad der Name des Anwenders unter der Abkürzung `$user` zusätzlich ausgewählt.
Schemapfad	Der Schemapfad definiert die Schemata, unter denen Tabellen bei ihrer Benutzung gesucht werden. Unabhängig davon kann jede Tabelle bei ihrer Benutzung auch mit dem Schemanamen benutzt werden.
Single Sentence	Die Definition von `single_sentence` ist eine Definition dessen, was bei einer Suche von Sätzen in einer Datenbank, bei der nur ein einziger Satz gefunden wurde, zu geschehen hat: entweder geschieht eine Ausgabe in Form einer Ergebnismaske, oder sie erfolgt in Form eines Listings.
Spalte	Siehe Feld.
SQL	Eine Programmiersprache zum ansprechen von relationalen Datenbanken. Am häufigsten wird sie heutzutage gebraucht in der Version von 1992 (SQL92), manchmal mit einigen neueren Erweiterungen.
Suchmaske	Siehe Bildschirmmaske.
Tabelle	Mit dem Wort Tabelle wird eine Datenbankrelation bezeichnet. Es ist hier im Sprachgebrauch gleich zu setzen mit dem Wort Datei. Eine Tabelle besteht aus vielen Sätzen oder auch Zeilen, die sich in einzelne gleichartige Felder (Attribute) unterteilen. Siehe auch Datenbank.
URL	Eine URL („Unified Resource Locator“) ist im Internet gleichbedeutend mit der „Web-Adresse“ eine Webseite, verbunden mit der Angabe des Protokolls, in der Regel `http`. Siehe auch HTTP(S).
Virtuelles Feld	Ein virtuelles Feld einer Bildschirmmaske ist ein Feld, in das keine Eingaben erfolgen können, sondern das nur errechnete Daten anzeigt. Siehe auch Bildschirmmaske.
Zeile	Mit Zeile wird hier ein Satz oder auch Tupel aus einer Datenbanktabelle bezeichnet. Die Zeile gliedert sich in die Attribute dieser Datenbanktabelle (Relation). Siehe auch Datenbank.

Index