FACHHOCHSCHULE OFFENBURG
HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT

STUDIENARBEIT

DATENBANKANBINDUNG VON WEB-SERVERN IN JAVA

Hendrik Saly
St. Martin-Str. 71/01/02
77652 Offenburg

Medien und Informationswesen
- Labor Medieninformatik -
Semester MI 4

Betreuer:
Prof. Dr. Tom Rüdebusch

Datum:
16.08.1999

INHALTSVERZEICHNIS

Thema:

Datenbankanbindung von Web-Servern in Java

Beschreibung:
Im Rahmen der Arbeit ist zunächst das Prinzip der serverseitigen Anbindung von Datenbanken im WWW zu dokumentieren. Anschließend sollen die verfügbaren Technologien für eine solche Anbindung kurz dargestellt und verglichen werden.

Auf die Datenbankanbindung unter Verwendung von serverseitigem Java ist dann detaillierter einzugehen. Für diese Technologie soll zunächst das Konzept beschrieben und danach eine generische Datenbankanbindung in Java realisiert werden. Diese grundlegende Implementierung soll dann für eine einfache, konkrete Anwendung "Angebot und Nachfrage von Tutorien" demonstriert werden.

Für die Realisierung soll freie Software (z.B. Apache Web-Server, mSQL, JDBC, Jserv, JSDK) verwendet werden.

Insgesamt ist auf eine verständliche, schrittweise aufbauende und kompakte Darstellung zu achten, da die Ergebnisse der Studienarbeit evtl. in einen Laborversuch eingehen sollen.

1 Datenbankanbindung an das WWW

1.1 Prinzip der serverseitigen Datenbankanbindung

Die zentralen Komponenten, die benötigt werden, um Daten aus Datenbanken im WWW verfügbar zu machen, sind zum einen die Datenbank selbst, der Web-Server und eine Erweiterung für den Web-Server bzw. ein eigenständiger Application Server, der mit dem Web-Server interagiert.

Der Web-Server, der als eigenständiger Prozeß auf der physikalischen Server Maschine läuft, hat die Aufgabe vom Client eingehende HTTP-Anfragen zu beantworten. Dies ist im HTTP-Standard 1.0 bzw. 1.1 festgelegt.
Im statischen Fall wird vom Client eine HTML-Seite angefragt und der Web-Server liefert diese an den Browser des Clients zurück. Soll jedoch die zurückgelieferte Seite dynamisch sein, d.h. je nach Anfrage des Clients anders aussehen, so muß eine weitere Komponente auf dem Server installiert sein. Mit „dynamisch“ sind nicht zwingend Datenbankabfragen gemeint, sondern auch andere Aktionen, die serverseitig ablaufen. Dies kann z.B. auch das Verschicken einer E-Mail sein oder eine andere direkte Interaktion mit dem Betriebssystem des Servers.

Zumeist jedoch liegen gerade im E-Commerce-Bereich oder bei anderen Geschäftsprozessen Datenbanken im Hintergrund. Was braucht man also, um mit einer Datenbank zu kommunizieren? In den meisten Fällen ist es so, daß der Web-Server entweder von sich aus oder durch geeignete Erweiterungen (Module) die Fähigkeit hat, eine dynamische Komponente anzusprechen.
Dies kann z.B. ein Java- oder PHP-Interpreter sein. Innerhalb dieser Skriptsprachen ist es nun möglich SQL-Befehle an die Datenbank zu schicken und die Inhalte in geeigneter Form über den Web-Server an den Client weiterzugeben.

Somit ergibt sich folgendes Schaubild:

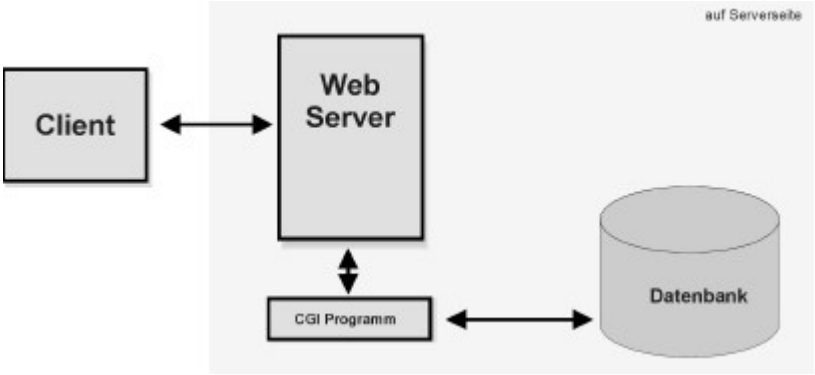


Abb.1 CGI-Aufruf

Bsp.: PHP3 in Verbindung mit dem Apache Web-Server und mySQL Datenbank („lamp“)

Damit der Apache die PHP3-Befehle, die unmittelbar im HTML-Quelltext stehen, auswerten kann, läßt er sich mit dem Modul mod_php3 erweitern. Um nur diejenigen HTML-Seiten durch den PHP- (PHP Hypertext Präprozessor) Interpreter zu schicken, die auch wirklich PHP-Code enthalten, werden diese mit einer speziellen Dateiendung versehen (typischerweise `.php3` oder `.phtml`).
Innerhalb des PHP-Skripts kann nun eine Datenbankverbindung hergestellt werden und eine SQL-Anfrage gesendet werden. Das Ergebnis geht nun über den Apache an den Client zurück.

Wegen der notwendigen Analyse der Webseite heißt das gesamte Konzept auch "server parsed HTML", eine seiner frühesten Ausprägungen fand es im Rahmen von Server Side Includes (SSI).

Eine weitere Möglichkeit Datenbanken verfügbar zu machen ist die CGI-Schnittstelle.
Hier werden vom Client meist ausführbare Programme als URL (oft mit Parametern) aufgerufen. Diese Programme liegen auf dem Server und sind typischerweise entweder Skripte, die interpretiert werden (Perl-, Shellskripte), oder kompilierte Programme (C/C++). Aus diesen CGI-Programmen heraus lassen sich mit den richtigen Befehlen dieser Sprache ebenfalls Datenbanken ansprechen und manipulieren. Diese funktionieren weitgehend unabhängig vom Web-Server, der jedoch für den CGI-Einsatz konfiguriert sein muß, da alle Client Anfragen über ihn laufen.

1.2 Die CGI-Schnittstelle

Grundlage fast jeder Dynamik im WWW ist die CGI Schnittstelle (Common Gateway Interface). Darüber ruft der Web-Server externe Programme auf, an die er Parameter übergeben kann. Außerdem werden Umgebungsvariablen (Environment) zur Verfügung gestellt.
So werden z.B. die IP-Adresse, der Hostname und andere Variablen des Clients an den Server übertragen und lassen sich mittels `REMOTE_ADDR` bzw. `REMOTE_HOST` auslesen.

Die externen Programme oder Skripte verarbeiten diese Informationen und generieren als Ergebnis HTML-Code, der dann über den Web-Server an den Client zurückgegeben wird.

Den Client erreichen so nur die bereits ausgewerteten Informationen, meist in Form von „normalem“ HTML-Code bzw. auch Grafiken.

Ohne die CGI-Schnittstelle wäre es auch nicht ohne weiteres möglich Datenbanken einzusetzen. Denn die Information, welche Manipulation der Client an der Datenbank durchführen möchte, wird meist in dem URL oder im Socket codiert und so an den Server übertragen. Der Web-Server stellt nun diese Daten dem System als globale Umgebungsvariable zur Verfügung (vgl. 2.2).

Mittlerweile befindet sich die CGI Spezifikation bei Version 1.1.

1.3 Was ist nötig ?

Hier soll nun ein kurzer Überblick über die einzelnen Komponenten gegeben werden, die nötig sind, um datenbankgestützte Informationen im WWW nutzbar zu machen.
Dieser Abschnitt ist bewußt allgemein gehalten, weil es in Anbetracht der Menge der einzelnen Systeme und deren Kombinationsmöglichkeiten nicht sinnvoll wäre, hier ein bestimmtes System zu beschreiben.

Der Web-Server

Diese Komponente ist natürlich unentbehrlich, da er die zentrale Instanz ist, die alle eingehenden HTTP-Anfragen beantwortet. Außerdem delegiert er Aufgaben, die er nicht selbst ausführen kann, an externe Programme. Ein Web-Server, der sehr weit verbreitet ist, ist der Apache Web-Server, der sich unter anderem dadurch auszeichnet, daß er auf fast allen Plattformen verfügbar ist und außerdem der Quelltext offen liegt (Open Source). Er läßt sich mit verschiedenen Modulen sehr gut erweitern.

Die Datenbank

Selbstverständlich muß auf dem Server auch ein Datenbanksystem installiert sein.
Es ist zwischen hierarchischen, relationalen und objektorientierten Datenbanken zu unterscheiden. Die meisten Datenbanken, die im WWW verwendet werden, sind relational. Diese Art Datenbank legt die Daten in Tabellen ab, die untereinander verknüpft werden können. Sie stellen eine effiziente Möglichkeit dar, hohe Datenmengen zu speichern und zu strukturieren. Um nun die Daten abzufragen oder neue einzufügen, existiert die standardisierte Sprache SQL (Structured Query Language). Sie ist datenbankunabhängig und bietet so eine abstrakte Möglichkeit unterschiedliche Datenbanken zu manipulieren. Nicht alle Datenbanken unterstützen SQL Befehle, aber die meisten und am weitverbreitetsten sind mit diesem Standard kompatibel. Eine sehr schnelle Datenbank ist mySQL, die ebenfalls als Open Source kostenlos zu bekommen ist (http://www.mysql.com).

Interpreter & Compiler

Sie stellen, ähnlich wie der Application Server, die Verbindung zwischen Web-Server und Datenbank (bzw. allgemein anderen Programmen oder dem Betriebssystem) her. Interpretierte Sprachen sind meistens Skriptsprachen und haben den Nachteil, daß sie langsam ablaufen (z.B. Perl). Kompilierte Programme (z.B. C/C++) laufen dagegen meist schneller, werden aber trotzdem eher selten eingesetzt.
All diese Sprachen enthalten Befehle, mit denen sich Datenbanken ansprechen lassen. Zudem können auch die CGI-Umgebungsvariablen, die der Web-Server bereitstellt, abgefragt und manipuliert werden. Der Web-Server muß für den Aufruf entsprechender Programme oder Interpreter konfiguriert sein.

Application Server

Der APS ist ein externes Programm, welches vom Web-Server bei Bedarf kontaktiert wird.
Der Application Server hat eigene Routinen, um mit Datenbanken zu kommunizieren.

Ein Beispiel ist Cold Fusion von Allaire. Alle HTML-Files, die Cold Fusion-Code enthalten, werden mit der Endung .cfm abgespeichert. Wird ein solches File aufgerufen, erkennt der entsprechend eingerichtet Web-Server dies und übergibt die Daten an den Application Server, der diese auswertet und das Ergebnis zurückliefert.
APS sind meist recht teure Zusatzsoftware, die dafür aber einfach und unkompliziert sind.

1.4 JDBC/ODBC

Um mit Datenbanken kommunizieren zu können, ist es zunächst einmal erforderlich eine Art Treiber zu installieren. In der Windows-Welt ist dies ODBC (Open Database Connectivity), das eine Art Standard Treibersammlung für sämtliche Datenbanken darstellt.
Da ODBC in C geschrieben ist, mußte es also für die Anbindung von Datenbanken unter Java an das objektorientierte Konzept angepaßt werden. So entstand JDBC (Java Database Connectivity). Damit ist es nun möglich jede Datenbank (für die ein JDBC Treiber existiert) anzusprechen. Falls die Datenbank einmal gewechselt werden muß, ist es nicht erforderlich den ganzen Code umzuschreiben, sondern nur die Stellen, an denen der Datenbanktreiber steht.

JDBC ist seit dem JDK 1.1 Bestandteil der Standarddistribution von Java und wird im Paket java.sql bereitgestellt. Wichtige Klassen und Methoden sind im Kapitel 2.8 näher erläutert.
Eine Liste der aktuell verfügbaren JDBC Treiber findet man hier:
http://java.sun.com/products/jdbc/jdbc.drivers.html

Um auch Datenbanken nutzen zu können, für die noch kein JDBC, wohl aber ein ODBC Treiber verfügbar ist, wurde die sogenannte JDBC-ODBC Bridge geschaffen, die als Art ODBC-Emulator dient.

Sun unterteilt die JDBC-Treiber in 4 Typen:

Typ 1: JDBC-ODBC Bridge

In diesem Sinne kein eigener Treiber, sondern stellt im wesentlichen die Kompatibilität zu ODBC-basierten Dankenbanken her. Es wird immer noch ein ODBC Treiber benötigt!

Typ 2:

Treiber von diesem Typ arbeiten mit proprietären Datenbank-Schnittstellen auf der Client-Seite. Der JDBC-Treiber ist also nicht in reinem Java-Code geschrieben und enthält plattformabhängigen Binärcode.

Typ 3

Treiber vom Typ 3 arbeiten mit einer Drei-Schichten-Architektur: Zwischen dem JDBC-Client und dem Datenbanksystem liegt eine dritte Ebene. Der JDBC-Client kommuniziert über ein

datenbankunabhängiges Protokoll mit dieser Schicht, die auch die Befehle an die Datenbanksysteme abschickt.

Typ 4

Treiber vom Typ 4 beschränken sich konzeptionell auf Verbindungen zu genau einem Datenbank-System. Der JDBC-Client ist hier in reinem Java-Code geschrieben und damit auf allen Plattformen einsetzbar. Er entspricht dem Typ 2 mit Plattformunabhängigkeit.

1.5 Structured Query Language (SQL)

Wer sich mit relationalen Datenbanken beschäftigt, wird früher oder später auf das Akronym „SQL“ stoßen. Es bedeutet soviel wie „Structured Query Language“. Es ist eine datenbankunabhängige Sprache um Datenbanken zu manipulieren. Mit den SQL-Befehlen können sowohl Daten ausgelesen werden, wie auch neue Daten in die Datenbank eingefügt werden. Dies alles ist in beliebig komplexer Form möglich.
Das liegt daran, daß SQL eine Sprache der „4. Generation“ ist, in der beschrieben wird, *was* mit den Daten geschehen soll und nicht *wie*.
SQL wurde ursprünglich von IBM zur Bearbeitung des Datenbanksystems DB2 Anfang der achtziger Jahre entwickelt. Heute unterstützen fast alle relationalen Datenbanken diesen Standard.

Ferner wird innerhalb der Sprache SQL zwischen zwei Teilbereichen unterschieden:
Anweisungen innerhalb der DDL (Data Definition Language) und der DML (Data Manipulation Language). Innerhalb der DDL Anweisungen werden z.B. neue Tabellen erstellt, bestehende Daten verknüpft, alles was dem strukturellen Aufbau einer Datenbank dient. In der DML dagegen werden Daten manipuliert, also Daten ein- und ausgelesen. Falls sich im Laufe einer Datenbankanwendung im WWW die Struktur einer Datenbank nicht ändert, sind im Programm selbst nur DML Anweisungen zu finden. Die Datenbank wird vor Beginn der Testphase manuell angelegt. SQL-Kommandos können natürlich auch über einen sogenannten SQL-Interpreter eingegeben werden. Dies geschieht typischerweise in der Shell und dient zum Aufbau einer Struktur oder zum Testen von SQL Anweisungen.

Wichtige SQL-Befehle werden im Kapitel 2.7 näher beschrieben.

Anmerkung:

Die großen Datenbanksysteme wie z.B. Oracle, Sybase, Infomix usw. haben zur Datenmanipulation meist auch einen eigenen Sprachstandard, der dann allerdings datenbankabhängig ist. Der Vorteil dieser spezifischen Sprache ist die Geschwindigkeit. In Systemen in denen es auf hohe Geschwindigkeit und schnelle Verfügbarkeit der Daten ankommt, wird das doch langsamere SQL zumeist nicht eingesetzt. Der Nachteil dieser eigenen Abfragesprachen ist zum einen die nicht mehr vorhandene Datenbankunabhängigkeit (der Code muß also zu großen Teilen umgeschrieben werden, falls sich die Datenbank im Hintergrund ändert), zum anderen die meist kryptische und schwer erlernbare Sprachsyntax.

1.6 Vergleich unterschiedlicher Technologien

In diesem Kapitel möchte ich sechs verschiedene Konzepte vorstellen und vergleichen, mit denen sich eine Datenbankanbindung an das WWW realisieren läßt:

- **CGI-Programme in Perl**
- **PHP3**
- **Cold Fusion**
- **Java Servlets**
- **Java SSI (Server Side Includes)**
- **Java Server Pages**

CGI-Programme in Perl (Practical Extraction and Report Language)

Dies ist wohl die typischste, aber auch die älteste Art und Weise, das WWW dynamisch zu machen. Perl ist eine interpretierte Sprache, die besonders mächtige Funktionen zur Text- und Zeichenkettenmanipulation enthält. Ursprünglich wurde sie von Larry Wall als Skriptsprache zur Vereinfachung von systemadministrativen Aufgaben unter UNIX entwickelt. Dabei ist es allerdings nicht geblieben.

Perl befindet sich zur Zeit bei Version 5 und ist mittlerweile sogar objektorientiert.

Die enormen Vorteile sind die freie Verfügbarkeit und die häufige Verbreitung. Der Perl-Interpreter existiert für fast alle Plattformen und harmoniert ebenso mit den meisten Web-Servern. Außerdem kann Perl noch für viele andere Aufgaben benutzt werden und ist auf den meisten Unix-Systemen sowieso schon installiert.

Erhebliche Nachteile finden sich in der langsamen Verarbeitungsgeschwindigkeit und in der (Un-)Lesbarkeit des Codes.

In Perl gibt es viele Module, deren Funktionen man in seinen Programmen nutzen kann (ähnlich wie die Bibliotheken in C). Eines davon, das DBI-Modul, erlaubt es Datenbanken anzusprechen. DBI steht für „Database Independent„ und ist, wie der Name schon sagt, datenbankunabhängig. Mit diesem Modul können alle unterstützten Datenbanken mit den gleichen Befehlen (und natürlich SQL-Anweisungen) angesprochen werden.

Perl-Skripte werden in einen bestimmten Ordner auf der Festplatte des Servers abgelegt (meist /cgi-bin) und mit Ausführungsrechten versehen. Dann kann man diese Files wie eine HTML Seite aufrufen und bei Bedarf natürlich auch Parameter übergeben.

PHP3

PHP (PHP Hypertext Präprozessor) ist eine weitere Möglichkeit das WWW interessanter zu gestalten. Der Code steht diesmal mitten im HTML-Quelltext und wird durch eine Folge spezieller Zeichen „escaped“, also vom sonstigen HTML-Text abgehoben (z.B. `<?php ...?>`). All diese HTML-Seiten die PHP-Code enthalten, müssen mit einer speziellen Dateiendung, die abhängig von der Konfiguration des Web-Servers ist, versehen werden (`.php3,.phtml`). Trifft der Web-Server auf PHP-Code innerhalb dieser Seiten, gibt er diesen an den PHP-Interpreter weiter. Für den Apache gibt

es diesen als Modulerweiterung. Außerdem steht der PHP-Interpreter auch als CGI-Programm zur Verfügung. Dieser verarbeitet den Code und schickt lediglich HTML an den Web-Server zurück, der diesen an die Stelle einsetzt, an der ursprünglich der PHP-Code stand.
Populär und weit verbreitet ist PHP in Verbindung mit dem Betriebsystem Linux, dem Apache Web-Server und der Datenbank mySQL („lamp"= linux, apache, mysql, php).
Innerhalb des PHP-Sprachstandards gibt es viele Funktionen, mit denen sich Datenbanken manipulieren lassen.
Vorteile sind die freie Verfügbarkeit und die bessere Übersicht, da der Code in HTML integriert ist.
Nachteile sind die bisher geringe Verbreitung und die Ausführungsgeschwindigkeit.

Cold Fusion

Als Vertreter kommerzieller Produkte will ich hier Cold Fusion von der Firma Allaire nennen. Ebenso wie PHP findet sich der Cold Fusion-Code direkt im HTML-Quelltext wieder, deren Dateiendung dann .cfm lauten muß. Die Cold Fusion- (CF-) Tags, die sehr an HTML angelehnt sind, bieten eine gute Übersicht. Als Interpreter tritt ein eigener Application Server in Aktion, der mit dem Web-Server interagiert.
Sonst funktioniert das System ähnlich wie PHP. Die Vorteile sind die generelle Einfachheit des Systems, die Wartung und Konfiguration über den Browser und die einfache Art und Weise Datenbanken anzubinden.
Nachteile: Der hohe Preis (ca. 4000-6000 DM) und die Tatsache, daß Cold Fusion derzeit nur unter Windows NT und Solaris läuft sowie die generelle Unflexibilität.

Java Servlets

Selbstverständlich ist die Java Technologie nicht nur auf clientseitige Verwendung (Applets) beschränkt, sondern bietet mit den Java Servlets die Möglichkeit, die Vorteile von Java auch für komplexe und dynamische Aufgaben innerhalb des WWW nutzbar zu machen. Java Servlets lassen sich mit einem normalen URL-Aufruf, ggf. mit Parametern, ansprechen. Falls der Web-Server nicht von vorne herein sowieso schon Java Servlets unterstützt, muß er mit den entsprechenden Modulen bzw. der Java Servlet Engine dafür konfiguriert werden.

Datenbanken lassen sich mit Hilfe von JDBC und entsprechenden Funktionen sehr gut anbinden.

Die Vorteile neben den generellen Vorteilen von Java, sind Konstrukte wie Persistenz, Session Tracking und Multithreading. Dies bringt einen erheblichen Geschwindigkeitsvorteil.
Nachteile zeigen sich in der geringen Verbreitung und in der mangelnden direkten Unterstützung populärer Web-Server (Apache, IIS, Netscape Server).
Das Thema Datenbankanbindung mit Java Servlets wird in den folgenden Kapiteln detaillierter behandelt.

Java SSI

SSI steht hier für Server Side Includes und beschreibt einen Mechanismus, der wiederum darauf beruht, daß der Web-Server die HTML-Datei nach bestimmten Tags (hier das `<servlet>`-Tag) durchsucht. Dann wird das Servlet, das mit dem Servlet-Tag referenziert ist, ausgeführt und die Rückgabe genau an die Stelle eingefügt, an der das Servlet-Tag stand. Damit nicht jede Datei zeitaufwendig nach dem Servlet-Tag durchsucht werden muß, erhalten die Dateien die SSI enthalten eine bestimmte Dateiendung, meist .shmtl. (vgl. auch Kapitel 2.3)

Java Server Pages

Sie funktionieren im Prinzip ähnlich wie PHP, auch hier wird der Java-Code direkt in den HTML-Quelltext geschrieben und durch spezielle Escape-Zeichenfolgen gekennzeichnet (`<% ...%>`). Außerdem werden die Seiten mit der Endung `.jsp` versehen. Damit wird dem Web-Server mitgeteilt, daß diese Seite dynamischen Code enthält und erst noch geparsed werden muß.
(vgl. auch Kapitel 2.4)

Hier soll noch mal ein genereller Überblick über die diskutierten Systeme gegeben werden.

	CGI (mit Perl)	**PHP3**	**Cold Fusion**	**Java Servlets**
Einfachheit der Sprache	kryptische aber sehr mächtige Sprache, ab Version 5 objekt-orientiert, nicht so schwierig wie C	angelehnt an Perl und C bzw. Java (*), aber bessere Lesbarkeit, steht im HTML-Quelltext	einfach zu lernen, gute Dokumentation, an HTML-Syntax angelehnt	verwendet Java-Syntax und -Konstrukte, daher einfach
Portabilität	möglich, aber aufwendig	möglich, aber aufwendig	sehr gut, da Schnittstellen-konzept	gut
Plattformunabhängigkeit	sehr gut, da Perl-Interpreter für fast alle Plattformen erhältlich	sehr gut, da PHP-Interpreter für fast alle Plattformen erhältlich	schlecht, da der CF-Server derzeit nur unter NT und Solaris läuft	sehr gut, da Java für fast alle Plattformen erhältlich
Sicherheitsaspekt	hängt vom Programmierer ab, völlige Transparenz (vor allem unter UNIX)	hängt vom Programmierer ab, völlige Transparenz (vor allem unter UNIX)	nach Aussagen des Herstellers sehr sicher, aber intransparent	hängt vom Programmierer ab, völlige Transparenz (vor allem unter UNIX)
Preis	kostenlos	kostenlos	ca. 4000-6000 DM	kostenlos
Offenheit des Systems	uneingeschränkt gegeben	gut, aber dennoch auf einen bestimmten Verwendungs-zweck hin entwickelt	begrenzt (keine Interaktion mit dem Betriebssystem, geringer Sprachumfang, prozedural)	uneingeschränkt gegeben

Tabelle 1 Vergleich unterschiedlicher Technologien

** für Java Server Pages*

2 Datenbankanbindung mit Java

2.1 Vorteile von Java

Java als serverseitige Sprache zur Anbindung von Datenbanken bzw. zur generellen Erzeugung dynamischer Inhalte im WWW bietet folgende Vorteile:

Plattformunabhängigkeit

Da das WWW ein heterogenes Netzwerk darstellt, ist gerade diese Eigenschaft sehr wichtig. Es ist ohne großen Aufwand möglich, Servlets z.B. bei einem Plattformwechsel sofort auf dem neuen System einzusetzen.
Java-Programme laufen in einer eigenen Umgebung, der sogenannten Java Virtual Machine (JVM).

Reduzierung der Entwicklungskosten

Unmittelbar aus der Plattformunabhängigkeit folgt ein weiterer Vorteil:
Es muß nur noch für eine Plattform entwickelt werden, was eine erhebliche Reduzierung der Kosten bedeutet.

Sicherheit

Auch das Sicherheits- bzw. Sandboxkonzept von Java stellt gerade im WWW einen weiteren Vorteil dar. Die sonst üblichen CGI-Skripts bieten im Gegensatz zu Java lange nicht soviel Schutz gegen feindliche Angriffe aus dem Internet.

Netzwerkfähigkeit

Java bietet bereits integrierte Netzwerkfunktionalitäten, die den Einsatz im Inter- oder Intranet unterstützen.

2.2 Aufbau und Konzept eines Servlets

Servlets sind in Java geschriebene und kompilierte Programme, die entweder durch einen Link oder durch das Form-/Action-Tag aufgerufen werden.

Es besteht aber auch die Möglichkeit Java-Servlets als sogenannte Server-Side-Includes zu benutzen. Um dies zu realisieren, wird das Servlet-Tag verwendet.
Dieses Tag ist kein Bestandteil von HTML, sondern wird durch den Web-Server ausgewertet.

Um die zugrundeliegende Struktur eines Servlets zu veranschaulichen, wird der nun folgende einfache Quellcode analysiert: (Es werden hier lediglich Servlets vom Typ HTTP betrachtet, aber Servlets sind im Prinzip protokollunabhängig.)

```
1:    import java.io.*;
2:    import javax.servlet.*;
3:    import javax.servlet.http.*;


4:    public class HelloWorld extends HttpServlet
5:    {
6:      public void doGet(HttpServletRequest req, HttpServletResponse res)
7:          throws ServletException, IOException
8:      {
9:          res.setContentType("text/html");
10:         PrintWriter out = res.getWriter();
11:
12:         out.println("<HTML><HEAD><TITLE> Hello World ! ");
13:         out.println("</TITLE></HEAD>");
14:         out.println("<BODY>");
15:         out.println("<h1> Hello World </h1>");
16:         out.println("</BODY></HTML>");
17:     }
18:   }
```

Jedes HTTP Servlet muß die Klasse *javax.servlet.http.** importieren, in der sämtliche HTTP-Funktionen implementiert sind (Zeilen 1-3).
Außerdem muß jedes HTTP-Servlet eine Subklasse von *HttpServlet* sein, also von *HttpServlet* abgeleitet werden (Zeile 4).

Servlets besitzen anstatt einer Main Methode eine sogenannte Service Methode.
Im speziellen Fall von HTTP-Servlets, besteht diese Service Methode entweder in der *doGet* oder in der *doPost* Anweisung. Damit werden die Übergabeparameter die von der Seite kommen, die das Servlet aufgerufen hat, übergeben (z.B. Auswertung von Formfelder, Zeilen 6 und 7).
Außerdem findet eine Ausnahmebehandlung statt.

GET oder POST

Ob nun *doGet* oder *doPost* verwendet werden muß, hängt vom Aufruf des Servlets ab. Betrachten wir hier einen Ausschnitt einer HTML-Seite, die Formfelder enthält und diese Daten an das Servlet *doCalculate* schicken möchte:

```
<HTML>
<HEAD>
<TITLE>Calculate something</TITLE>
</HEAD>
<BODY>
<FORM METHOD=GET ACTION="/servlet/doCalculate">
<INPUT TYPE=TEXT NAME="feld1"><p>
...
<INPUT TYPE=SUBMIT>
</FORM>
</BODY>
</HTML>
```

In Zeile 6 wird als Methode GET vereinbart. Deshalb muß im Ziel-Servlet auch die Methode doGet überschrieben werden. Wird als Methode POST verwendet, so muß doPost als Methode überschrieben werden. Der Unterschied liegt in der Art der Parameterübergabe.

Wird GET verwendet, so wird die gesamte Information in der URL-Zeile codiert und übertragen. Dies kann z.B. so aussehen:

```
http://rc5stats.distributed.net/rc5-64/psearch?st=linux%40medienmaus
```

Für Sonderzeichen und Leerzeichen werden Escape-Sequenzen benutzt. (hier: @ wird zu %40). Der Parameter hat hier den Namen *st* und den Wert *linux@medienmaus.*
Vorteil der GET Methode: Es lassen sich auch auf komplexe URLs Bookmarks setzten.
Nachteil: Es können max. 240 Zeichen übertragen werden. Außerdem ist die URL manipulierbar.

Die POST Methode dagegen schickt ihre Daten direkt über die Socket-Verbindung an den Server (im HTTP request body). Dies bleibt für den Benutzer im Gegensatz zur GET Methode unsichtbar.
Der Vorteil liegt hier in der unbeschränkten Menge der zu übertragenden Zeichen (im Extremfall können dies auch mehrere Megabyte sein). Der Nachteil hingegen ist, daß keine Bookmarks gesetzt werden können, da sich die URL nicht ändert. Ein weiterer Vorteil ist, daß die URL nicht manipuliert werden kann.

In der Regel sollte POST verwendet werden, da man hierdurch Probleme durch überlange URLs sowie beabsichtigte Manipulation oder Einsicht interner Strukturen und Daten vermeiden kann.

Grundlegendes zu Java Servlets findet sich in der Arbeit von Wolfram Jopp (http://info.mi.fh-offenburg.de/wolfram).

Anmerkung:
Problematisch wird die Entwicklung von Servlets, wenn man nicht weiß, ob die Daten per GET oder POST geschickt werden. In diesem Fall muß man beide Versionen implementieren und vorher eine geeignete Abfrage durchführen.

In der Zeile 9 wird der „Inhaltstyp“ (Content-Type) des Response Objekts auf `text/html` gesetzt. Damit wird dem Browser mitgeteilt, wie er die nun folgenden Daten interpretieren soll. Es wäre ja auch möglich, daß z.B. ein Bild im gif-Format dargestellt werden soll (`z.B. contentType("image/gif")`). In diesem Fall erwartet den Browser lediglich ganz normalen ASCII-Text, zusammengesetzt zu einem HTML-Dokument.

Die Methode setContentType() wird auf das Response Objekt `res` angewendet. Unter diesem Objekt ist all das zu verstehen, was an den Browser als Antwort zurückgesendet wird.

In Zeile 10 wird die `getWriter()`Methode auf das Response Objekt `res` angewendet.
Diese Methode erzeugt ein PrintWriter Objekt `out;` auf diese kann nun die Methode println() angwendet werden. Diese gibt dann eine Zeichenkette und einen Zeilenvorschub aus.

In den folgendes Zeilen wird auf das out Objekt jeweils die Methode println angewendet, die den Text in den Klammern direkt an den Browser des Clients schickt.

2.3 Java Server Side Includes (SSI)

SSI werden im Gegensatz zu Servlets nicht durch einen Link aufgerufen, sondern durch das Servlet-Tag, der im HTML-Quelltext steht. Die Datei wird mit der Endung .shtml abgespeichert, und der Web-Server durchsucht die so gekennzeichneten Files nach dem Servlet-Tag. Er führt das dort referenzierte Servlet aus und ersetzt den Servlet-Tag durch die Ausgabe des Servlets.

SSI sollte dann verwendet werden, wenn es viel statischen Anteil an einer HTML-Seite und wenig dynamische Elemente gibt. Ein sinnvolles Bespiel wäre das Darstellen des aktuellen Datums auf einer oder mehreren Seiten. An der Stelle, an der später das Datum stehen soll, wird das Servlet-Tag gesetzt, welches nun ein Servlet aufruft, daß das aktuelle Datum ausgibt.

Das Prinzip verdeutlicht das nachfolgende Schaubild

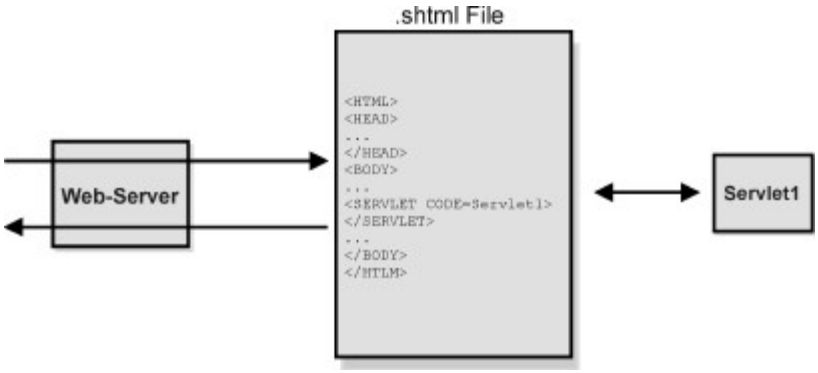


Abb.2 SSI Prinzip

2.4 Java Server Pages (JSP)

Bei Java Server Pages hingegen steht der eigentliche Quelltext in der HTML-Datei selbst.
Es wird also kein externes Servlet bzw. File aufgerufen. Der Web-Server, dem durch die Endung .jsp mitgeteilt wird, daß es sich um eine Java Server Page handelt, interpretiert den Code direkt. Der Server muß für den Einsatz von Java Server Page entsprechend konfiguriert sein.

Folgendes Beispiel gibt ein *Hallo* und dann den Wert der Variable "name" aus, falls diese übergeben wurde. Ist "name" nicht definiert, wird *Hallo Welt* ausgegeben:

```
<HTML>
<HEAD><TITLE>Test-JSP</TITLE>
<BODY>

<% if (request.getParameter("name" == null) {

    out.println("Hallo Welt");
    }
    else {
    out.println("Hallo "+ request.getParameter("name"));
    }
%>


</BODY>
</HTML>
```

Die Java Server Pages funktionieren im Prinzip wie die Active Server Pages von Microsoft (ASP). Den Vorteil, den ich bei den JSP sehe, ist die einfachere Aktualisier- und Wartbarkeit, da nur ein File geändert und die JSP nicht jedesmal kompiliert werden muß. Nachteile sind der begrenzte Funktionsumfang und mangelnde Unterstützung durch die verfügbaren Web-Server.

2.5 Die Verbindung zur Datenbank

Das Konzept, um Datenbanken an Servlets anzubinden, sieht folgendermaßen aus:

Bevor eine Verbindung zur Datenbank hergestellt werden kann, muß zunächst der entsprechende JDBC-Treiber in den Speicher geladen werden. Dieser registriert sich beim DriverManager und wird als gültiger Datenbanktreiber erkannt. Um den JDBC-Treiber zu laden, wird folgende Methode aufgerufen:

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

Danach kann die DriverManager-Klasse angefragt werden, ob eine Verbindung zu einer gegeben Datenbank hergestellt werden kann. Der DriverManager ist in java.sql.* implementiert.

```
Connection con = DriverManager.getConnection("jdbc:odbc:somedb", "user",
"passwd");
```

Ist ein Connection-Objekt erzeugt worden, können nun sogenannte Statement-Objekte erzeugt werden, mit denen SQL-Anweisungen an die Datenbank geschickt werden können.

```
      Statement stmt = con.createStatement();   // Erzeugen eines
                                                // Statementobjekts bzgl.
                                                // der Verbindung con,
                                                // mit Hilfe der Methode
                                                // createStatement die auf
                                                // das con Objekt angewendet
                                                // wird
```

Nun kann z.B. folgende SQL Anfrage gestellt werden:

```
ResultSet rs = stmt.executeQuery("SELECT * FROM anytable");
```

Jetzt kann das Ergebnis aus dem Result Objekt ausgelesen werden.

```
while(rs.next())
{
String event = rs.getSring("event");      // Aus der Spalte "event" werden
}                                         // alle Daten Reihe für Reihe
                                          // ausgelesen
```

Das *con* -Objekt ist eines der wichtigsten Konstrukte bei der Anbindung von Servlets an Datenbanken.

Durch die Verwendung des Driver Managers ist es sogar möglich, mehrere Datenbanken gleichzeitig anzumelden und zu benutzen. So können eigentlich unvereinbare Systeme dennoch gemeinsam genutzt werden. Als konkretes Beispiel lässt sich z.B. ein Skript für die Datenübernahme von einer Datenbank in eine andere leicht realisieren.

2.6 Die Datenbank "mySQL"

Da im Rahmen dieser Studienarbeit mit kostenloser Software gearbeitet werden soll, habe ich mich für die Verwendung von mySQL entschlossen. Es ist eine freie relationale Datenbank.

Sie ist schnell und nicht so schwergewichtig und komplex wie die „großen" Datenbanksysteme. Vor allen Dingen aber ist sie kostenlos, und es existieren JDBC Treiber (Typ 4), die eine einfache und effiziente Anbindung an Servlets erlauben.

In der Regel erhält man mySQL als tar.gz- oder rpm-Archiv. Sie ist entweder bei vielfältigen Unix-, vor allem aber Linux-Derivaten bereits enthalten, ansonsten kann man sie über http://www.mysql.com oder einen Mirror herunterladen.

Als Testbett wurde mySQL zunächst auf einer Linux-Plattform installiert. Das Programm lag als rpm-Archiv (Binärdistribution für Linux) vor. Die Installation verlief ohne Probleme.

Bei der Installation auf dem Zielrechner, einer SGI Maschine unter Irix 6.3, gab es jedoch Probleme.
Die Binärdistribution konnte nicht zum Laufen gebracht werden, und so musste das Programm selbst kompiliert werden. Dazu lädt man sich zunächst den als tar.gz gepackten Quellcode herunter, speichert ihn im Verzeichnis ~/temp und entpackt ihn mit dem Befehl `gunzip –dc ~/temp/mySQL-3.x.tar.gz | tar –xvf -` .Danach kann man das Konfigurationsskript aufrufen, welches die Daten des Rechners (Prozessortyp, Betriebssystem, Bibliotheken usw.) ermittelt und das Makefile erzeugt:

`./configure` (bei Bedarf können hier noch Parameter angegeben werden). Danach muß `make` und `make install` aufrufen werden. Letzterer Befehl und alle weiteren bis zum Abschluß der Installation müssen mit Root-Rechten ausgeführt werden, da Schreib- und Leserechte für die `/lib` und `/libexec` Verzeichnisse benötigt werden. Leider schlug auch diese Vorgehensweise fehl. Wir wiederholten den gesamten Vorgang auf einer SGI mit Irix 6.5, und dort lief die Datenbank dann ohne größere Schwierigkeiten.

Installiert wird sie über ein beiliegendes Skript mit Namen mysql_install_db, welches die Grant-Tabellen erzeugt, über die sich das System selbst konfiguriert. In diesen sind sämtliche Zugriffsrechte der einzelnen User und Datenbanken einzutragen. Das Rechtesystem von mySQL ist nicht standardisiert, aber sehr mächtig. Dieses Thema möchte ich hier jedoch nicht weiter ausführen.
Die Datenbank ist auch sehr gut dokumentiert, und es existieren User- und Newsgroups im Internet, bei denen man kostenlosen Support in Anspruch nehmen kann.

Außerdem existiert ein grafisches Frontend-System, basierend auf PHP, mit der sich die Datenbank komfortabel über den Web Browser administrieren läßt (http://www.htmlwizard.net).

Um eine mySQL-Datenbank in Java anzubinden, sind folgende Aufrufe nötig:

Zum einen muß der mySQL JDBC Treiber geladen werden (Typ 4 Treiber von *Terrence W. Zellers,* http://www.voicenet.com/~zellert/tjFM/twz1jdbcForMysql-doc.html).

```
Class.forName("twz1.jdbc.mysql.jdbcMysqlDriver");
```

Danach kann die Verbindung mit Hilfe des Driver-Managers hergestellt und das Connection-Objekt erzeugt werden (die Datenbank befindet sich auf minfo1.mi.fh-offenburg.de, der mysql-Dämon läuft auf Port 3306). Die Datenbank heißt *mi_tutorium.*

```
Connection con = DriverManager.getConnection("jdbc:z1MySQL://minfo1.mi.fh-
offenburg.de:3306/mi_tutorium?user=user&password=passwort");
```

An diesem Beispiel ist auch gut zu erkennen, daß lediglich diese zwei Zeilen abgeändert werden müssen, falls die Datenbank einmal gewechselt wird.

Bei diesen ganzen Aktionen ist es natürlich gut möglich, daß einmal ein Fehler auftritt. Entweder kann die Datenbankverbindung nicht hergestellt werden, oder der richtige JDBC –Treiber wird nicht gefunden, oder es findet eine anderer Ausnahmefall statt. Genau für diese Fehler- oder besser Ausnahmefälle gibt es in Java die Ausnahmebehandlung.

Diese besteht aus den drei Schlüsselwörtern `try`, `catch` und `throw` und wird folgendermaßen aufgebaut:

```
//Fehlerbehandlung mit try und catch

Connection con = NULL;

try   {
      // Laden und registrieren des JDBC-Treibers
      // Kann ein ClassNotFoundException Fehler verursachen

      Class.forName("mysql.jdbc.driver.mysqldriver");

      // Verbindung zur Datenbank herstellen
      // Kann ein SQLException Fehler verursachen

      con = DriverManager.getConnection("jdbc:mysql", "user", "passwd");

      //... Rest des Quellcodes
      }

catch (ClassNotFoundException e) {

      // Error Meldung: JDBC Treiber konnte nicht geladen werden !

}

catch (SQLException e) {

      // Error Meldung: Datenbankverbindung konnte nicht
      //                hergestellt werden !

}

finally {

      // Datenbankverbindung schließen

      try {

            if (con != NULL) con.Close();
      }

      catch (SQLException ignored) { }
  }
```

Das Schlüsselwort `throw` wird nur dann benötigt, wenn eine selbstgeschriebene Methoden eine Fehlermeldung erzeugen soll.

2.7 Wichtige SQL Anweisungen

Dieses Kapitel soll einen Überblick über die, meines Erachtens nach, wichtigsten SQL Anweisungen geben. Es stellt daher keinen Anspruch auf Vollständigkeit.

DDL Anweisungen

Um eine neue "Datenbank" zu erstellen, ist folgendes einzugeben:

```
CREATE DATABASE databasename;
```

Da relationale Datenbanken in Tabellen organisiert sind, muß zuerst diese Struktur eingerichtet werden:

```
CREATE TABLE      tabellenname
                  (     spaltenname-1 datentyp [NOT NULL] auto_increment,
                        spaltenname-2 datentyp [NOT NULL],
                        ...
                        spaltenname-n datentyp [NOT NULL],
                        PRIMARY KEY (spaltenname-1)
                  )
```

Hiermit wird eine Tabelle mit den Spalten spaltenname 1-n und ihren jeweiligen Datentypen angelegt. Wird NOT NULL angegeben, so muß dieses Feld auf jeden Fall einen Wert haben. Werden NOT NULL-Felder nicht initialisiert, so gibt das Datenbanksystem eine Fehlermeldung aus. Außerdem wird die Spalte *spaltenname-1* als Primärschlüssel deklariert. D.h. dieser Wert ist einmalig und über Ihn kann die Zeile später eindeutig referenziert werden.

DML Anweisungen

Einer der häufigsten SQL Anweisungen ist SELECT:

```
      SELECT * FROM anytable;
```

Mit diesem Befehl werden alle Spalten der Tabelle *anytable* ausgelesen.

```
SELECT name, pnumber, date  FROM anytable

WHERE Id_number IS 12;
```

Hiermit werden die Spalten *name, pnumber* und *date* der Tabelle *anytable* ausgelesen. Aber nicht alle Zeilen der Spalten, sondern nur diese, in denen Id_number (z.B. ein Primärschlüssel) den Wert 12 hat.

Eine weitere wichtige Anweisung ist INSERT. Damit lassen sich neue Daten in die Datenbank eintragen.

```
INSERT INTO anytable VALUES (wert1, wert2, wert 3, ...);
```

Dieser Befehl legt eine neue Zeile in der Tabelle anytable mit den Werten, die in der VALUE Klammer stehen, an (die Reihenfolge muß exakt der Spaltenreihenfolge in der Tabelle entsprechen).

Soll ein bereits vorhandener Eintrag geändert werden, muß die UPDATE-Anweisung verwendet werden.

```
UPDATE anytable SET

                Spalte 1 = Wert 1,
                Spalte 2 = Wert 2,
                ...
                Spalte n = Wert n;
WHERE bedingung
```

Zum Löschen von Daten existiert der DELETE Befehl:

```
DELETE FROM anytable
WHERE bedingung
```

Dieser löscht eine ganze Zeile aus der Tabelle *anytable*, an der die Bedingung *bedingung* zutrifft.

2.8 Wichtige Klassen, Objekte und Methoden

Die JDBC-Schnittstelle wird als Java-Paket (Package) java.sql bereitgestellt. Dieses Paket besteht im wesentlichen aus Schnittstellen (Interfaces), die von den jeweiligen JDBC-Treibern als Klassen implementiert werden müssen. Die wichtigsten Schnittstellen bzw. Klassen sind:

java.sql.Driver

Der JDBC-Treiber implementiert die Schnittstellen durch Datenbank-spezifische Klassen.

java.sql.Drivermanager

Der Treiber-Manager verwaltet und lädt die angemeldeten JDBC-Treiber und ist für den Verbindungsaufbau und -abbau zu einer Datenbank zuständig.

java.sql.Connection

Diese Klasse bzw. Schnittstelle repräsentiert eine Verbindung zu einer Datenbank.

java.sql.Statement

Dies ist eine Containerklasse, die eine SQL-Anweisung für eine gegebene Verbindung repräsentiert.

java.sql.ResultSet

Diese Klasse bzw. Schnittstelle repräsentiert die von einer SQL-Anweisung zurückgegebene Ergebnismenge.

3 Beispielanwendung: Tutorium – Datenbank

3.1 Beschreibung und Konzept

Als konkrete Anwendung soll ein System entwickelt und implementiert werden, in dem Angebot und Nachfrage für Tutorien des Studiengangs MI an der FH Offenburg über das WWW verwaltet werden. Die Anmeldebestätigungen werden über E-Mail verschickt.

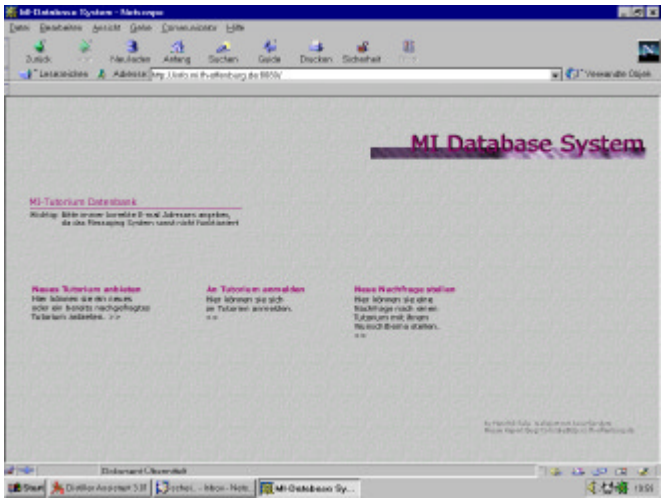
Studenten soll es möglich sein, selbst Tutorien anzubieten oder sich an bereits angebotenen anzumelden. Ferner soll die Möglichkeit bestehen, eine Anfrage für ein bestimmtes Tutorium zu veröffentlichen. Außerdem soll es einer bestimmten Person möglich sein administrative Aufgaben zu erledigen. Diese umfassen insbesondere das Einsehen und Löschen von Daten und das Versenden von E-Mail-Nachrichten an Personengruppen (z.B. an alle Tutoriumsteilnehmer).
Die Administrationsfunktionen sind nicht mehr Teil der Dokumentation.
Um die Datenbank vor dem Zugriff von außen zu schützen, werden mittels der .htaccess Datei des Webservers nur Anfragen die von innerhalb der FH-Offenburg kommen beantwortet.

Implementiert wird diese Anwendung mit Java-Servlets. Als Betriebsystem wird IRIX und als Web-Server Apache 1.3.6 verwendet. Als Datenbank kommt mySQL zum Einsatz.

3.2 Funktionen

Allgemeine Funktionen

F10 Anmelden eines neuen Tutoriums
F20 Anmelden an ein bestehendes Tutorium
F30 Nachfrage für ein Tutorium stellen
F40 Ein nachgefragtes Tutorium anbieten
F50 Angebotene Tutorien abrufen



Administrator Funktionen (Passwortgeschützt)

F100 Aufruf Überblickseite
F110 Löschen von Tutorien und deren Anmeldungen
F120 Versenden von E-Mail-Nachrichten an Tutoriumsteilnehmer

Auf die Servlets, die administrative Aufgaben übernehmen, wird nicht eingegangen.

3.3 Datenbankdesign

Zuerst wird eine neue Datenbank mit dem Namen *mi_tutorium* erstellt:

```
CREATE DATABASE mi_tutorium;
```

Danach werden drei Tabellen angelegt, die später die Daten enthalten:

Die Tabelle *tutorien* enthält die Beschreibung der einzelnen angebotenen Tutorien und die Daten der Person, die dieses anbietet.

```
CREATE TABLE tutorien

      (t_id       int(10)          DEFAULT '0'  NOT NULL  auto_increment,
       tutname    varchar(255)     DEFAULT ''   NOT NULL,
       name       varchar(255)     DEFAULT ''   NOT NULL,
       semester   varchar(255)     DEFAULT ''   NOT NULL,
       email      varchar(255)     DEFAULT ''   NOT NULL,
       tdatum     date             DEFAULT '0000-00-00' NOT NULL,
       PRIMARY KEY (t_id)
      )
```

In der Tabelle *anmeldungen* werden die Daten derer gespeichert, die sich bereits an ein Tutorium angemeldet haben:

```
CREATE TABLE anmeldungen

      (a_id       int(10)          DEFAULT '0'  NOT NULL  auto_increment,
       t_id       int(10)          DEFAULT '-1' NOT NULL,
       aname      varchar(255)     DEFAULT ''   NOT NULL,
       asemester  varchar(255)     DEFAULT ''   NOT NULL,
       aemail     varchar(255)     DEFAULT ''   NOT NULL,
       adatum     date             DEFAULT '0000-00-00' NOT NULL,
       PRIMARY KEY (a_id)
      )
```

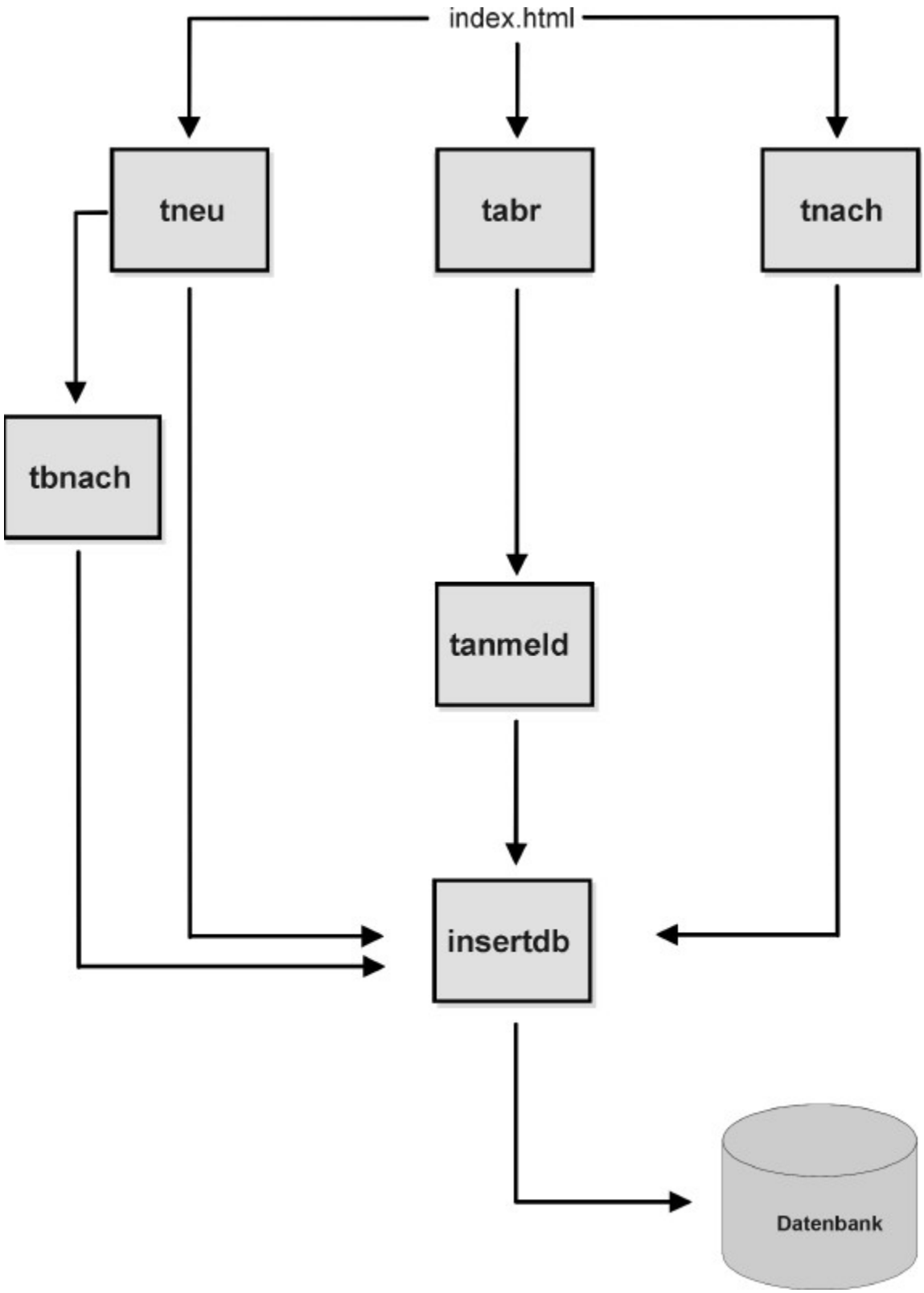
In der dritten Tabelle *nachfrage* werden die Daten derer Personen abgelegt, die ein Tutorium mit einem bestimmten Thema nachfragen:

```
CREATE TABLE nachfrage

      (n_id       int(10)          DEFAULT '0'  NOT NULL  auto_increment,
       ntutthema  varchar(255)     DEFAULT ''   NOT NULL,
       nname      varchar(255)     DEFAULT ''   NOT NULL,
       nsemester  varchar(255)     DEFAULT ''   NOT NULL,
       nemail     varchar(255)     DEFAULT ''   NOT NULL,
       ndatum     date             DEFAULT '0000-00-00' NOT NULL,
       PRIMARY KEY (n_id)
)
```

3.4 Die Servlets

Das gesamte System ist folgendermaßen aufgebaut:



tabr.java

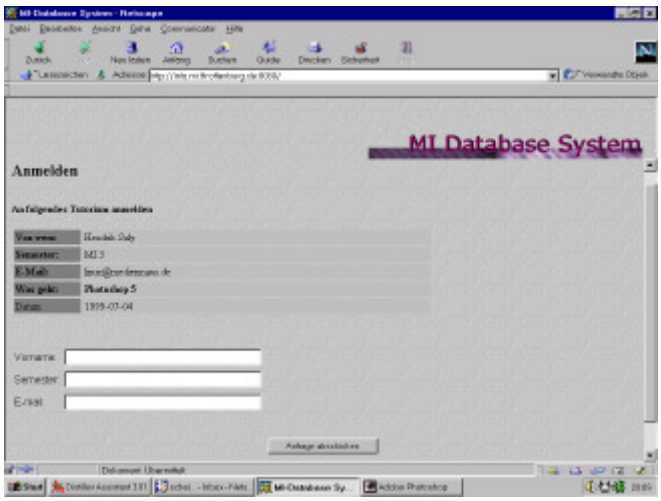
F50

Dieses Servlet listet alle verfügbaren Tutorien auf und bietet mittels Hyperlink die Möglichkeit, sich an eines anzumelden. Außerdem werden die nachgefragten Tutorien aufgelistet.
Mittels GET wird *t_id* (Primärschlüssel für Tutorien) an tanmeld.java übergeben.
Parametername ist *tut*.

tanmeld.java

F20

Hat man sich für eines der Tutorien entschieden, so müssen hier die persönlichen Daten eingegeben werden.



Erhält die Information, an welches Tutorium der Benutzer sich anmelden möchte über den GET Parameter *tut* von tabr.java.

Übergibt die POST Parameter *name, semester, email* an insertdb.java.

Außerdem werden als Hidden Fields folgende Parameter übergeben: *table, tut, tut_leiter, mail_tut_leiter, tut_plain.*

Mit Hilfe von *table*, kann das insertdb-Servlet feststellen in welche Datenbanktabelle die Daten eingetragen werden müssen. Die restlichen Parameter enthalten den Primärschlüssel für Tutorien (*tut*), den Namen des Tutoriumleiters (*tut_leiter*), die E-Mail-Adresse des Tutoriumleiters (*mail_tut_leiter*), und *tut_plain* enthält das Thema des Tutoriums. Diese Parameter sind vor allem für das Versenden der E-Mails notwendig.

tnach.java

F30

Hier kann eine Nachfrage abgesetzt werden. Sie wird dann mittels tabr.java aufgelistet. Ruft insertdb.java auf.

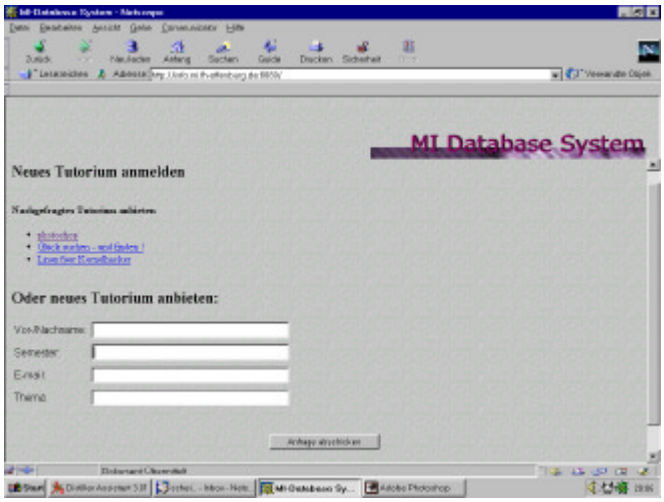
Übergibt die POST Parameter *name, semester, email, tut* an insertdb.java.

Außerdem werden als Hidden Fields folgende Parameter übergeben: *table.*

tneu.java

F10, F40

Das Servlet ermöglicht es ein neues Tutorium anzubieten. In diesem Falle wird dann direkt mit diesen Daten insertdb.java aufgerufen.



Es besteht aber auch die Möglichkeit, über dynamisch erzeugte Links ein Tutorium anzubieten, nachdem bereits nachgefragt wurde. In diesem Falle wird tbnach.java aufgerufen.

Übergibt die POST Parameter *name, semester, email, tut* an insertdb.java.
Außerdem werden als Hidden Fields folgende Parameter übergeben: *table.*

Wird ein nachgefragtes Tutorium aufgerufen, so wird mittels GET der Parameter *tut* (Primärschlüssel Tutorien) an tbnach.java übergeben.

tbnach.java

F40

Hier werden die persönlichen Daten eingegeben, und das Tutorium wird hinzugefügt.
Das Thema ist bereits bekannt und wird als Hidden Field (*tut*) übergeben.

Übergibt die POST Parameter *name, semester, email* an insertdb.java.

Außerdem werden als Hidden Fields folgende Parameter übergeben: *table, tut, deletenf.*

Der Parameter *deletenf* enthält den Primärschlüssel für das nachgefragte Tutorium, damit dies dann aus der Tabelle "nachfrage" gelöscht werden kann. Diese Aktion wird von insertdb.java ausgeführt.

insertdb.java

Dies ist das zentrale Servlet, das für jeden Schreibzugriff auf die Datenbank aufgerufen werden muß. Außerdem werden hier die Eingaben des Benutzers auf unerlaubte Zeichen überprüft und die E-Mails verschickt.

Folgende Aktionen werden in dieser Reihenfolge ausgeführt:

1) Parameter holen
2) Diese von Sonderzeichen befreien (über cleanString)
3) Falls der Parameter *tut_leiter* existiert (also insertdb von tanmeld aufgerufen wurde), werden E-Mails an den Tutoriumsleiter und an die Person, die sich angemeldet hat, verschickt
4) Je nach Inhalt des Paramters *table* wird das betreffende SQL Statement ausgeführt
5) Existiert der Parameter *deletenf* (also insertdb wurde von tbnach aufgerufen), wird die betreffende Nachfrage, die nun keine mehr ist, gelöscht

Die Entscheidung ein solch zentrales Servlet einzusetzen, habe ich aus folgenden Überlegungen heraus getroffen: Spätere Erweiterungen sind einfacher möglich, da maßgebliche Änderungen nur an einer Stelle durchzuführen sind. Außerdem wird die Zahl der benötigten Servlets für das gesamte System überschaubar gehalten, da bei dezentraler Entwicklung für jedes Servlet ein eigenes Folgeservlet mit den entsprechenden Datenbankoperationen existieren müßte.

Einige Besonderheiten des insertdb Servlets sollen hier erläutert werden:

Aktuelles Datum erzeugen

Zunächst muß ein neues Datumsobjekt erzeugt werden

```
GregorianCalendar datum = new GregorianCalendar();
```

Danach kann mit

`datum.get(Calendar.YEAR)` das Jahr ausgegeben werden.

`datum.get(Calendar.MONTH)+1)` der Monat ausgegeben werden.

`datum.get(Calendar.DATE)` der Tag ausgegeben werden.

Die Methode cleanString

```
//Methode cleanString zum bereinigen von Sonderzeichen
public String cleanString(String eingabe, String clearof){

            StringTokenizer st = new StringTokenizer(eingabe,clearof);

            eingabe = "";

            while (st.hasMoreTokens()){

                  eingabe = eingabe+st.nextToken();}

                  return eingabe;
            }
```

Der String *eingabe* wird von allen Zeichen befreit die im String *clearof* vorkommen.

So ist es z.B. sinnvoll die HTML-Klammern (<>) zu entfernen, damit der Benutzer keinen HTML-Code eingeben kann. Außerdem werden noch einige andere Zeichen entfernt, die einen Fehler der Datenbank verursachen (u.a. \'`´).

E-Mails mit Java-Servlets versenden

SMTP-Objekt erzeugen, mit Hilfe dessen auf die SMTP Funktionen des Betriebsystems zugegriffen werden kann:

```
import sun.net.smtp.SmtpClient;
SmtpClient smtp = new SmtpClient();
```

Danach können Absender und Empfänger der Nachricht festgelegt werden:

```
smtp.from("dbmaster@info.mi.fh-offenburg.de");
smtp.to("hsaly@cip.rz.fh-offenburg.de");
smtp.to(email);
```

Um die eigentliche Text-Nachricht versenden zu können, muß ein weiteres Objekt vom Typ PrintStream erzeugt werden.

```
PrintStream msg = smtp.startMessage();
```

Danach kann die Nachricht zeilenweise aufgebaut werden.

```
msg.println("Subject: MDS-Anmeldebestaetigung");
msg.println();
msg.println("Sie haben folgende Nachricht vom MI Database System:");
msg.println(...);
```

Verbindung schließen und E-Mail versenden:

```
smtp.closeServer();
```

3.5 Schlußbetrachtung

Java eignet sich aufgrund der besprochenen Eigenschaften sehr gut zur Anbindung von Datenbanken an das Internet. Es steht, im Gegensatz zu den Skriptsprachen, eine vollständige objektorientierte und netzwerkfähige Programmiersprache zur Verfügung. Diese Tatsache bietet hohe Flexibilität und erlaubt auch die Realisierung komplexer Anwendungen. Jedoch muß man auch erkennen, daß Java nicht auf dieses Einsatzgebiet optimiert ist, weshalb es mit anderen speziellen Produkten vielleicht einfacher ist, Datenbankanwendungen bzw. dynamische Webseiten zu erzeugen. Aber dafür stehen ja schon Mechanismen wie Java Server Pages zur Verfügung. Die Entwicklung der MI-Tutoriums-Datenbank verlief relativ problemlos, war jedoch stellenweise etwas umständlich. So stehen z.B. bei den für das WWW optimierten Sprachen einfachere Funktionen für Mail- und Datumsoperationen zur Verfügung. Auch die etwas umständliche Art und Weise HTML-Code mit der Print Funktion zurückgeben zu müssen, dürfte bei großen und komplizierten Projekten problematisch und unübersichtlich werden.

Die Beispielanwendung Tutorium-Datenbank ist unter http://info.mi.fh-offenburg.de zu finden.

3.6 Weiterführende Aspekte

In diesem Kapitel möchte ich einige weitere Modifikationen und neue Anwendungen vorschlagen, die an der FH Offenburg mit Hilfe der Java-Servlet Technologie (z.B. als Laborversuche oder als weitere Studien-/Projektarbeiten) entwickelt werden könnten:

Der Tutoriumsverwaltung könnte ein **Passwortsystem** hinzugefügt werden, bei dem jeder Anbieter ein Passwort erhält, mit Hilfe dessen er einsehen kann, wer an seinem Tutorium angemeldet ist, er den Titel des Tutoriums ändern kann und er auch das gesamte Tutorium löschen kann (jeweils mit Benachrichtigung der Teilnehmer).

Weiterhin wäre es sinnvoll, wenn bereits **laufende Tutorien auf einer speziellen Seite ausgeschrieben und übersichtlich dargestellt würden**, um spontane Tutoriumsbesuche zu ermöglichen. Außerdem würde diese Seite einen Überblick bieten.

Wird die Tutorium-Datenbank irgendwann einmal zu groß und zu unübersichtlich, könnte sich eine **Stichwortsuche** nach bestimmten Themen als nützlich erweisen.

Als Neuentwicklung wäre ein **Online-Ausleihsystem für die Geräte oder Schlüssel vom Studio-Ohlsbach** denkbar. Es könnte ein Gerät für ein gewissen Zeitraum angefragt werden, mit sofortiger Bestätigung ob das Gerät verfügbar ist. Außerdem behält der dortige Labor-Assistent so den Überblick.

4 Anhang

4.1 Abkürzungsverzeichnis

API	Application Programming Interface
APS	Application Server
ASP	Active Server Pages
CGI	Common Gateway Interface
DBI	Database Independend
DBMS	Database Management System
GIF	Graphic Interchange Format
GNU	GNU's not Unix
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IIS	Internet Information Server
IP	Internet Protocol
JSP	Java Server Pages
JVM	Java Virtual Machine
JDBC	Java Database Connectivity
JDK	Java Development Kit
MIME	Multipurpose Internet Mail Extensions
ODBC	Open Database Connectivity
PERL	Practical Extraction and Report Language
PHP	PHP Hypertext Präprozessor
SSI	Server-Sided Includes
SQL	Structured Query Language
TCP	Transmission Control Protocol
URL	Unified Resource Locator
WWW	World Wide Web

4.2 Literaturverzeichnis

Hobbs, Ashton (1997):
Datenbanken programmieren mit JDBC, 1. Auflage, Haar bei München: SAMS/Markt&Technik

Hunter, Jason (1998):
Java Servlet Programming, 1.Auflage, Sebastopol: O´Reilly & Associates

Kähler, Wolf-Michael (1990):
SQL – Bearbeitung relationaler Datenbanken, 1. Auflage, Braunschweig: Vieweg Verlag

Schmid, Egon (1999):
PHP – Dynamische Webauftritte professionell realisieren, 1. Auflage, München: Mart&Technik

Lemay, Laura ;Perkins, Charles L. (1997):
Java in 21 Tagen, 3. Auflage, Haar bei München: SAMS/Markt&Technik

Krüger, Guido (1997):
Java 1.1. lernen, 1. Auflage, Bonn: Addison-Wesley-Longman

4.3 Weitere Quellen

Kaffee&Kuchen:
die führende deutsche Java-Seite von Christoph Bergmann und Hannes Gamperl,
zu finden unter dem URL http://java.seite.net/jdbc/index.html

Pulverkaffee:
Dynamische HTML-Seiten mit PHP/FI, mSQL und gd von Tobias Häcker,
zu finden unter dem URL http://www.ix.de/ix/artikel/9608056/

mySQL:
Ofizielle mySQL Homepage,
zu finden unter dem URL http://www.mysql.com

Java:
Ofizielle Java Page von Sun,
zu finden unter dem URL http://java.sun.com/

JDBC for MySQL:
A type 4 JDBC driver for MySQL von Terrence W. Zellers,
zu finden unter dem URL http://www.voicenet.com/~zellert/tjFM/twz1jdbcForMysql-doc.html

4.4 Quelltextauszüge

```
/* insertdb.java
 Studienarbeit von Hendrik Saly, MI4
 Thema: Datenbankanbindung von Web-Servern in Java

 Dieses Servlet ist das zentrale Servlet
 fuer alle Datenbankschreibzugriffe

 16.8.1999 */


import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
import java.util.*;

//fuer E-Mail Funktion
import sun.net.smtp.SmtpClient;

public class insertdb extends HttpServlet {


    //Methode cleanString zum bereinigen von Sonderzeichen
    public String cleanString(String eingabe, String clearof) {

        StringTokenizer st = new StringTokenizer(eingabe,clearof);

        eingabe = "";

        while (st.hasMoreTokens()) {

            eingabe = eingabe+st.nextToken();

        }

        return eingabe;
    }



    public void doPost(HttpServletRequest req, HttpServletResponse res) //Method=POST
    throws ServletException, IOException {


        Connection con = null;
        Statement stmt = null;
        ResultSet rs = null;

        res.setContentType("text/html");

        ServletOutputStream out = res.getOutputStream();

        //Datumobjekt erzeugen
        GregorianCalendar datum = new GregorianCalendar();


        //Parameter aus dem Socket holen (via POST)
        String    table = req.getParameter("table");
        String    tut       = req.getParameter("tut");
        String    name      = req.getParameter("name");
        String    semester = req.getParameter("semester");
        String    email     = req.getParameter("email");


        //Hidden fields via Post holen
        String deletenf = req.getParameter("deletenf");          //Wenn von tbnach, alte Nachfrage loeschen
```

```
String     tut_leiter = req.getParameter("tut_leiter");                    //Name Tutoriumsleiter
String     tut_plain = req.getParameter("tut_plain");                      //Tutoriumname als Klartext
String     mail_tut_leiter = req.getParameter("mail_tut_leiter");          //Emailadresse des Tutoriumleiters

//Datum setzen
String     datum = datum.get(Calendar.YEAR)+"-"+(datum.get(Calendar.MONTH)+1)+"-
"+datum.get(Calendar.DATE);

//Sonderzeichen definieren
String sonderzeichen = "\"\\\'`´|<>{}";


//Strings von Sonderzeichen befreien
tut                = cleanString(tut, sonderzeichen);
name               = cleanString(name, sonderzeichen);
semester           = cleanString(semester, sonderzeichen);
email              = cleanString(email, sonderzeichen);


//Gebe HTML an den Browser

out.println("<HTML><HEAD><TITLE>Datenbankeintrag</TITLE></HEAD><BODY
background=../back.jpg>");
out.println("<BR><P>");
out.println("<H2>Daten erfolgreich eingetragen !</H2>");
out.println("<BR>");



// ---------------------------------------------------------------------
// Emails verschicken


if (tut_leiter != null) {  //Nur Mail verschicken, wenn tut_leiter existiert, d.h. bei Anmeldung

      out.println("Es wurden E-mail Nachrichten an folgende Personen verschickt:");
      out.println("<BR>");
      out.println("<BR>");
      out.println("<ul>");


      //Anmeldebestaetigung fuer Teilnehmer

      try {

            SmtpClient smtp = new SmtpClient();
            smtp.from("dbmaster@info.mi.fh-offenburg.de");
            smtp.to("hsaly@cip.rz.fh-offenburg.de");
            smtp.to(email);
            //smtp.to(mail_tut_leiter);
            out.println("<li>"+email);
            //out.println("<li>"+mail_tut_leiter);
            //out.println("</ul>");


            PrintStream msg = smtp.startMessage();

            msg.println("Subject: MDS-Anmeldebestaetigung");
            msg.println();
            msg.println("Sie haben folgende Nachricht vom MI Database System:");
            msg.println();
            msg.println("----------------------------------------------------------------");
            msg.println();
            msg.println("Sie haben sich an das Tutorium >>"+tut_plain+"<< von >>"+tut_leiter+ "<< angemeldet");
            msg.println();
            msg.println();
            msg.println("Die Anmeldung erfolgte am: "+datum);
            msg.println();
```

```
            msg.println("---------------------------------- MI Database System V0.5 ---- ");
            msg.println();
            msg.println("Report to hsaly@cip.rz.fh-offenburg.de");

            smtp.closeServer();

        } catch (IOException e) { out.println("<h1>Bitte geben sie eine KORREKTE E-mail Adresse an !!<h1>"); }


        //Mail an des Tutoriumsleiter, informiert ueber neue Anmeldung


        try {

            SmtpClient smtp = new SmtpClient();
            smtp.from("dbmaster@info.mi.fh-offenburg.de");
            smtp.to("hsaly@cip.rz.fh-offenburg.de");
            //smtp.to(email);
            smtp.to(mail_tut_leiter);
            //out.println("<li>"+email);
            out.println("<li>"+mail_tut_leiter);
            out.println("</ul>");


            PrintStream msg = smtp.startMessage();

            msg.println("Subject: MDS-Anmeldung fuer Tutorium");
            msg.println();
            msg.println("Sie haben folgende Nachricht vom MI Database System:");
            msg.println();
            msg.println("-----------------------------------------------------------------");
            msg.println();
            msg.println("Es hat sich jemand fuer Ihr Tutorium >>"+tut_plain+"<< angemeldet");
            msg.println();
            msg.println();
            msg.println("Die Anmeldung erfolgte am: "+datum);
            msg.println("von: "+name+" ("+semester+", "+email+")");
            msg.println();
            msg.println("---------------------------------- MI Database System V0.5 ---- ");
            msg.println();
            msg.println("Report to hsaly@cip.rz.fh-offenburg.de");

            smtp.closeServer();

        } catch (IOException e) { out.println("<h1>Fehler: Emailadresse Tutoriumausrichter falsch !<h1>"); }



    } //Ende Email


    try {

        //Mysql Treiber laden
        Class.forName("twz1.jdbc.mysql.jdbcMysqlDriver");


        //Verbindung zur Datenbank mi_tutorium aufbauen
        Connection connection = DriverManager.getConnection("jdbc:z1MySQL://minfo1.mi.fh-
        offenburg.de:3306/mi_tutorium?user=user&password=passwort");


        //Statement Objekt erzeugen
        Statement statement = connection.createStatement();


        if (table.equals("anmeldungen")) {
```

```
        statement.executeUpdate("INSERT INTO anmeldungen
        VALUES(","+(Integer.parseInt(tut))+","+name+","+semester+","+email+","+datum+")");

        }

        else {

              if (table.equals("tutorien")) {

            statement.executeUpdate("INSERT INTO tutorien
            VALUES(","+tut+","+name+","+semester+","+email+","+datum+")");

              }


              else {
                    if (table.equals("nachfrage")) {

        statement.executeUpdate("INSERT INTO nachfrage
        VALUES(","+tut+","+name+","+semester+","+email+","+datum+")");

                    }

                    else { out.println("Es ist ein Fehler aufgetreten. "+table+" konnte nicht gefunden werden<BR>Daten
                          nicht erfolgreich verarbeitet"); }

        } }

        out.println("<BR>");
        out.println("<BR><p><H2><a href=\"../index.htm\">Zurueck</H2>");
        out.println("</body></html>");


        //Nachfrageeintrag loeschen ?
        if (deletenf.equals(null)) {

        }

        else {
              //SQL Abfrage (hier: loeschen aus nachfrage)

              statement.executeUpdate("DELETE FROM nachfrage WHERE n_id ="+deletenf);


        }

} catch(java.lang.ClassNotFoundException e) {

        out.println("JDBC-ODBC-Treiber nicht gefunden" + e.getMessage());

}


catch(java.sql.SQLException e) {

        out.println("Fehler beim Abfragen der Datenbank  : " + e.getMessage());

}


finally {

        //Datenbankverbindung schließen
        try {
              if (con != null) con.close();

        }
             catch (SQLException ignored) { }       }}}
```

```
/* tabr.java
 Studienarbeit von Hendrik Saly, MI4
 Thema: Datenbankanbindung von Web-Servern in Java

 Dieses Servlet bietet ein Uebersicht
 ueber alle Tutorien mit der Möglichkeit zur
 Anmeldung

 16.8.1999 */


import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;


public class tabr extends HttpServlet {

      public void doGet(HttpServletRequest req, HttpServletResponse res)
      throws ServletException, IOException {


            Connection con = null;
            Statement stmt = null;
            ResultSet rs = null;

            res.setContentType("text/html");

            ServletOutputStream out = res.getOutputStream();


            try {

                  //Mysql Treiber laden
                  Class.forName("twz1.jdbc.mysql.jdbcMysqlDriver");


                  //Verbindung zur Datenbank mi_tutorium aufbauen
                  Connection connection = DriverManager.getConnection("jdbc:z1MySQL://minfo1.mi.fh-
                  offenburg.de:3306/mi_tutorium?user=user&password=passwort");


                  //Statement Objekt erzeugen
                  Statement statement = connection.createStatement();


                  //SQL Abfrage (hier: Abfragen der bestehenden Tutorien)
                  rs = statement.executeQuery("SELECT * FROM tutorien ORDER BY tdatum DESC");


                  //Gebe Ergebnis in HTML an den Browser zurück

                  out.println("<HTML><HEAD><TITLE>Verfuegbare Tutorien</TITLE></HEAD><BODY
                  background=../back.jpg>");
                  out.println("<BR><P>");
                  out.println("<H2>Verfügbare Tutorien</H2>");
                  out.println("<BR>");

                  out.println("<TABLE CELLSPACING=2 CELLPADDING=2 BGCOLOR=silver>");


                  // Ergebnis des SQL Statements zuzueckgeben
                  while(rs.next()) {

                 out.println("<TR><TD WIDTH=100 BGCOLOR=gray><B>Von wem:</B></TD><TD
                 WIDTH=540>"+ rs.getString("name")+"</TD><TD></TD></TR>");
```

```
                out.println("<TR><TD BGCOLOR=gray><B>Semester:</B></TD><TD>"+
                rs.getString("semester")+"</TD><TD></TD></TR>");

                out.println("<TR><TD BGCOLOR=gray><B>E-Mail:</B></TD><TD>"+
                 rs.getString("email")+"</TD><TD></TD></TR>");

                out.println("<TR><TD BGCOLOR=gray><B>Was geht:</B></TD><TD><B>"+
                rs.getString("tutname")+"</B></TD><TD BGCOLOR=e1e1e1>");

                out.println("<A HREF=\"/servlets/tanmeld?tut = "+rs.getInt("t_id")+"\"><B>Interessiert
                ?</B></A></TD></TR>");

                out.println("<TR><TD BGCOLOR=gray>Datum:</B></TD><TD>"+
                 rs.getString("tdatum")+"</TD><TD></TD></TR>");

                out.println("<TR><TD BGCOLOR=FFFFFF></TD></TR>");
                }



        out.println("</TABLE>");


        out.println("<BR><P>");
        out.println("<H2>Es besteht Nachfrage nach folgenden Tutorien</H2>");
        out.println("<BR>");

        out.println("<TABLE CELLSPACING=2 CELLPADDING=2 BGCOLOR=silver>");


        //Resultobjekt neu initalisieren
        rs = null;

        //SQL Abfrage (hier: Abfragen nach nachgefragten Tutorien)
        rs = statement.executeQuery("SELECT * FROM nachfrage ORDER BY ndatum DESC");


        // Ergebnis des SQL Statements zuzueckgeben
        while(rs.next()) {

                out.println("<TR><TD WIDTH=100 BGCOLOR=gray><B>Von wem:</B></TD><TD
                WIDTH=540>"+ rs.getString("nname")+"</TD><TD></TD></TR>");

                out.println("<TR><TD BGCOLOR=gray><B>Semester:</B></TD><TD>"+
                rs.getString("nsemester")+"</TD><TD></TD></TR>");

                out.println("<TR><TD BGCOLOR=gray><B>E-Mail:</B></TD><TD>"+
                rs.getString("nemail")+"</TD><TD></TD></TR>");

                out.println("<TR><TD BGCOLOR=gray><B>Wunschthema:</B></TD><TD><B>"+
                rs.getString("ntutthema")+"</B></TD><TD BGCOLOR=e1e1e1>");

                out.println("<TR><TD BGCOLOR=gray>Datum:</B></TD><TD>"+
                rs.getString("ndatum")+"</TD><TD></TD></TR>");

                out.println("<TR><TD BGCOLOR=FFFFFF></TD></TR>");
        }



        out.println("</TABLE>");


        out.println("</BODY></HTML>");

} catch(java.lang.ClassNotFoundException e) {
```

```
            out.println("JDBC-ODBC-Treiber nicht gefunden" + e.getMessage());

        }

        catch(java.sql.SQLException e) {

            out.println("Fehler beim Abfragen der Datenbank  : " + e.getMessage());

        }

        finally {

            //Datenbankverbindung schließen
            try {
                if (con != null) con.close();

            }

            catch (SQLException ignored) { }

} } }
```