Seminar Datenbanksysteme – Anwendungsentwicklung

Vortrag: Datenbankanbindung im World Wide Web

Von Christian Bayerlein

4. Februar 2000

AG DBIS
Universität Kaiserslautern

Inhalt

1. Einführung 3
2. Wozu Datenbankanbindung 3
2.1 Einbindung von unternehmensinternen Daten 4
2.2 Einbindung vorhandener Daten in das WWW 4
2.3 Entwicklung von stark interaktiven Anwendungen 5
3. Realisierungsansätze zur Datenbankanbindung 5
3.1 Serverseitige Applikationslogik 6
3.1.1 Einstufige Verfahren 6
3.1.1.1 Verarbeitung mit Hilfe von CGI-Bin's 7
3.1.1.2 Verwendung von Server-API's 8
3.1.2 Zweistufige Lösung 10
3.1.3 Funktionsabarbeitung im Datenbank-Server 11
3.2 Client-seitige Applikationslogik 11
3.2.1 Positive Aspekte beim Einsatz von Java 12
3.2.2 Negative Aspekte beim Einsatz von Java 13
3.2.3 Datenbankanbindung in Java mittels JDBC 15
3.3 Benutzung von SQL3 15
3.3.1 Entwicklungsmöglichkeiten durch neue Aspekte in SQL3 15
3.3.2 Einbindung der Neuerung zur Aktualisierung von Web-Seiten 15
4. Zusammenfassung 16
5. Literaturverzeichnis 18

1. Einführung

Das WWW gehört mittlerweile zu den größten Anwendungen überhaupt. Das Informationsangebot im Web wächst täglich – und ein Ende dieses Wachstums ist nicht in Sicht. Gleichzeitig wächst der Bedarf an interaktiven Anwendungen, z. B. Online-Einkaufszentren.

Zu Beginn bestand das WWW nur aus statischen Seiten, die der Benutzer durch Anwahl von Verknüpfungen – sog. Links – durchstöbern konnte. Mit Einführung der Formulartechnik wurde es erstmals möglich, dynamisch auf Benutzeranfragen zu reagieren, indem man die Werte in den ausgefüllten Formularen auf Seite des Servers mit speziellen Programmen – sog. CGI's – auswertete.

Durch das Wachstum der Datenbestände wurde es notwendig, die benötigten Daten für die dynamische Generierung der aufgrund der Benutzeranfragen erstellten Seiten strukturiert zu speichern und zu verwalten. Hierfür verwendet man im allgemeinen Datenbanksysteme (DBMS – Datenbankmanagementsysteme).

Dieser Beitrag soll nun die Möglichkeiten zur Anbindung von Datenbanksystemen an das WWW erörtern.

Im ersten Teil beschäftige ich mich mit den groben Zielen einer Datenbankanbindung: Wozu werden Web-basierte Datenbank-Anwendungen benötigt? Danach gehe ich über zu dem „wie". Welche grundsätzlichen Möglichkeiten bestehen, mit Datenbanksystemen verwaltete Daten in das WWW zu bringen.

2. Wozu Datenbankanbindung?

Grundsätzlich kann man die Gründe, Datenbanken mit dem Web zu verbinden in mindestens drei Kategorien aufteilen [2]:

- Einbindung von unternehmensinternen Daten mit Hilfe der Technologie des Internets – Aufbau eines sog. Intranets (Interne Nutzung im Unternehmen)

- Bereitstellung vorhandener Daten im WWW (für aus Unternehmenssicht externe Benutzer)

- Erstellung von stark interaktiven Anwendungen, z. B. Online-Einkaufszentren oder -Auskünften

Im folgenden will ich näher auf die drei Punkte eingehen, um sie genauer zu analysieren und kategorisieren. Natürlich sind noch weitere Gründe denkbar, auf welche ich aber im Rahmen dieser Arbeit nicht näher eingehen will.

2.1 Einbindung von unternehmensinternen Daten

Die Verwendung der Technologie des Internets für die Bereitstellung der Daten im Unternehmen bietet vielfältige Vorteile. Ich will hier nur einige nennen:

- Plattform-Unabhängigkeit – die Daten können mit Hilfe eines sog. Browser, der für fast alle Plattformen existiert, angezeigt werden. Dadurch ist man z. B. nicht an ein bestimmtes Betriebssystem gebunden.

- Einfache Administration – die Daten können zentral verwaltet werden und sind dennoch mit einfachen Hilfsmitteln bearbeitbar. In der Regel muß auf den einzelnen Arbeitsplatz-Rechnern nur ein sogenannter Browser installiert sein.

- Benutzerfreundlichkeit – das Web gilt durch seine einfache Strukturierung und Übersichtlichkeit als sehr benutzerfreundlich.

- Schnelle Anwendungsentwicklung – hierfür steht die schnell zu erlernende Sprache HTML und viele graphische Werkzeuge zur Verfügung

- Einfache Anbindung an Außenstellen – durch das dem Internet zugrundeliegende Protokoll TCP/IP können einfach und sicher Verbindungen nach außen realisiert werden. Die Sicherheit kann dabei durch sogenannte Firewalls und Verschlüsselung gewährleistet werden.

2.2 Einbindung vorhandener Daten in das WWW

In vielen Unternehmen besteht ein Bedarf, vorhandene Daten in das Web zu bringen, um Besucher der Unternehmens-Website eine umfassende Information zu ermöglichen.

Hierbei besteht die Schwierigkeit, daß die Daten für gewöhnlich nicht schon in HTML vorliegen, sondern erst konvertiert werden müssen. Der Benutzer kann mit den proprietären Dateiformaten in der Regel nichts anfangen.

Außerdem werden die Daten, die zur Verfügung gestellt werden sollen oft durch Datenbanksysteme verwaltet. Hier muß in irgendeiner Weise der Zugriff über das WWW geregelt werden.

Grundsätzlich gibt es die Möglichkeit, solche Datenbestände regelmäßig in statische Web-Seiten zu konvertieren. Hierfür gibt es unzählige Werkzeuge. Diese Methode hat allerdings den Nachteil, daß die erzeugten Seiten in der Regel nicht aktuell sind – d. h,. sie entsprechen nur zum Zeitpunkt der Konvertierung dem aktuellen Zustand der im Datenbanksystem modellierten Miniwelt.

Eine andere Möglichkeit besteht darin, zum Zeitpunkt der Benutzeranfrage dynamisch eine HTML-Seite zu erstellen. Hierzu dienen spezielle Gateways, die die Verbindung zum Datenbanksystem aufbauen und die Anfrage entsprechend aufbereitet weiterleiten. Die zurückgelieferten Daten werden daraufhin nach HTML konvertiert und formatiert und dann an den Browser des Benutzers geschickt. Nachteil: diese Methode ist wesentlich langsamer als die erste, da hier ein Programm ausgeführt werden und die Verbindung zum Datenbank-

Server aufgebaut werden muß und nicht einfach eine Datei aus dem Dateisystem gelesen und zum Benutzer geschickt werden kann.

Seid der Einführung von SQL3 besteht eine weitere Möglichkeit. Bei ihr werden die Daten bei einer Änderung neu in statische Seiten konvertiert. Diese Methode verbindet sozusagen die Vorteile der beiden anderen, nämlich ständige Aktualität der Daten und schnelle Verfügbarkeit. Auf diese Möglichkeit will ich noch mal näher im letzten Kapitel eingehen.

2.3 Entwicklung von stark interaktiven Anwendungen

Gerade im Bereich des e-Commerce – d. h. im elektronischen Handel – wächst die Zahl der stark interaktiven Anwendungen zur Zeit sehr schnell. Unter solchen Anwendungen will ich hier die meinen, die folgende Kriterien erfüllen:

- Interaktionsmöglichkeiten gehen weit über das Hindurchnavigieren mit Hilfe von Links hinaus
- Eigenhändige Eingaben mit Hilfe von Formularen möglich
- Die Sitzungen erstrecken sich über mehrere Schritte, bei denen immer wieder Eingaben vom Benutzer erforderlich sind.

Ein gutes Beispiel für eine stark interaktive Anwendung ist das sogenannte Online-Einkaufszentrum. Hier kann ein Benutzer durch das Sortiment von Waren stöbern und diese in einen virtuellen Einkaufskorb legen. Am Ende gibt er seine persönlichen Daten – z. B. die Anschrift und gewünschte Zahlungsweise ein und bestätigt noch einmal seine Bestellung. Bei einem solchen System muß ständig der Warenkorb des Benutzers bereit gehalten werden. Für solch komplexe Aufgaben kann in der Regel nur ein Datenbanksystem eingesetzt werden.

3. Realiesierungsansätze der Datenbankanbindung

In diesem Abschnitt möchte ich mich damit beschäftigen, wie eine Datenbankanbindung in das WWW grundsätzlich realisiert werden kann. Dazu unterscheide ich mehrere Architektur-Formen.
Prinzipiell gibt es drei Möglichkeiten, Datenbanksysteme anzubinden: erstens können periodisch die Daten der Datenbank auf statische HTML-Seiten abgebildet werden. Hierbei werden Skripte benutzt, die Daten aus der Datenbank abfragen und daraus HTML-Seiten konstruieren. Diese Skripte werden periodisch aufgerufen, um die Seiten zu bestimmten Zeiten – z. B. einmal in der Stunde – oder manuell durch den Web-Administrator zu aktualisieren. Dies wird oft für die Erzeugung von z. B. Indexseiten eingesetzt. Vorteile bei dieser Methode sind z. B., daß keine besonderen CGI-Programme oder Module für den Web Server gebraucht werden. Bei einer Benutzer-Anfrage wird lediglich eine Datei aus dem Datei-System des Web-Servers gelesen und an den Browser des Benutzers geschickt, was die Antwortzeiten wesentlich verringert.

Eine zweite Möglichkeit besteht daran, dynamisch bei Benutzeranfrage die Seiten zu erzeugen. Dazu werden z. B. nach dem ausfüllen eines Formulars durch den Benutzer die Daten an den Web-Server geschickt, der daraufhin ein Programm startet oder ein Modul aufruft, welches die Daten ausgewertet und anschließend eine HTML-Seite als Antwort erzeugt. Es könnte aber auch ein Java-Applet auf den Browser übertragen und dort gestartet

werden, welches die graphische Benutzeroberfläche aufbaut und die Verbindung mit dem Datenbank-Server direkt aufnimmt. Auf diese Methode, HTML-Seiten bei Benutzer-Anfrage dynamisch zu erzeugen, bzw. mit Java-Applets zu arbeiten will ich im folgenden Abschnitt näher eingehen und den größten Teil dieser Ausarbeitung widmen.

Zum Ende dieser Ausarbeitung möchte ich dann noch auf die dritte Möglichkeit eingehen. Hierbei werden neue Konzepte in SQL 3 benutzt, um z. B. durch Trigger die Web-Seiten bei einer Änderung in den Tabellen der Datenbank automatisch zu aktualisieren um sie mit dem aktuellen Zustand der Miniwelt konsistent zu halten.

Doch zunächst zum Thema dynamische Seiten-Erzeugung und Benutzung von Java-Applets:

Im Prinzip kann danach unterschieden werden, wo die Applikationslogik sitzt (auf Server- bzw. Client-Seite) und wo und wie die Daten dargestellt werden. Ferner kann bei serverseitiger Applikationslogik danach unterschieden werden, ob für die Anbindung an die Datenbank ein eigener Prozeß läuft, der die Verbindungen verwaltet und aufrecht erhält. In einem solchen Fall hat man es typischerweise mit einem sog. Applikationsserver zu tun.

3.1 Serverseitige Applikationslogik

Bei einer Realisierung mit serverseitiger Applikationslogik verhält es sich in der Regel so, das vom Browser des Benutzers eine Anfrage via http an den Server geschickt wird, der die Daten für die Anfrage entgegen nimmt und einen neuen Prozeß startet oder es wird mittel einer sogenannten Server-API ein Modul im Web-Server aufgerufen. Dieser bzw. dieses dient zur Verarbeitung der Daten., um daraus dynamisch eine HTML-Seite zu erzeugen [1,2].

Ein großer Vorteil dieser Methode ist, daß zum Browser des Benutzers keine Programm-Daten übertragen werden müssen, und dieser dadurch "schlank" bleiben kann. Dadurch ist mit erheblich kürzerer Ladezeit der Anwendung zu rechnen. Gerade bei kleineren Anwendungen lohnt es sich nicht, ein eigenes Java-Applet zu entwickeln.

Ein Problem bei diesem Verfahren ist allerdings die Zustandslosigkeit es HTTP-Protokolls [1]. Dies bedeutet, der Server kann sich bei mehrschrittigen Vorgängen nicht mehr daran erinnern, welche Abläufe bereits geschehen sind und welche Daten bereits vom Benutzer eingegeben wurden. Dies möchte man jedoch vor allem bei Anwendungen wie Online-Einkaufszentren irgendwie implementieren und muß dazu Tricks anwenden (z. B. versteckte Formularvariablen oder Cookies), damit dies möglich wird.

Außerdem ist mittels HTML die Gestaltung der Oberfläche nicht so flexibel wie bei einem Java-Applet.

Auf die verschiedenen Formen der Verarbeitung auf Seite des Servers will ich im folgenden näher eingehen. Dafür möchte ich zunächst diese verschiedenen Architekturen klassifizieren.

Unter System-Architektur versteht man dabei im allgemeinen den logischen Aufbau der Werkzeuge und ihre Verteilung auf entsprechende Prozesse [1]. Dabei möchte ich zwischen einstufigen und zweistufigen Verfahren sowie der Verarbeitung im Datenbank-Server unterscheiden.

3.1.1 Einstufige Verfahren

Bedingt durch den einfachen Aufbau dieser Architektur wird das einstufige Verfahren am häufigsten zur Realisierung einfacher Datenbank-Web-Anwendungen eingesetzt [1]. Dabei wird eine direkte (logische) Datenbank-Verbindung aufgebaut. Dies geschieht entweder durch ein CGI-Programm oder durch eine Server-Erweiterung. Für jede Benutzeranfrage muß dies erneut geschehen, was allerdings relativ viel Rechenzeit erfordert. Die Datenbank muß nämlich jedes Mal neu authentifizieren, und die Rechte des Benutzers, unter dessen Kennzeichen das CGI-Programm bzw. die Server-Erweiterung läuft, überprüfen. Bis zum eigentlichen Verarbeitungsbeginn werden also schon ziemlich viele Ressourcen benötigt. Daher ist diese Methode auch nur verwendbar, wenn nicht mit allzu hohen Besucher-Zahlen zu rechnen ist. Gute Beispiele hierfür sind Adressdatenbanken und Gästebücher, sowie Einschritt-Vorgänge wie das Online-Tracking. (Paketverfolgung z. B. bei UPS)

Für die Realisierung der Applikationslogik kommen im allgemeinen entweder CGI-Programme oder Server-Erweiterungen in Frage. Im folgenden möchte ich näher auf diese Begriffe eingehen, und deren Vorteile und Nachteile erörtern.

3.1.1.1 Verarbeitung mit Hilfe von CGI-Bin's

Bei der Verarbeitung mit Hilfe von sog. CGI-Bin's handelt es sich um die älteste Form der Interaktion zwischen Benutzer und Web-Server. CGI steht hierbei für „Common Gateway Interface" [4]. Es wird also eine allgemeine Schnittstelle zwischen Applikationslogik und Web-Server definiert. Dabei wird anhand des URL festgestellt, daß es sich bei der Anfrage nicht um eine statische Seite handelt, sondern das ein Programm (genau genommen ein Prozeß) aufgerufen werden soll, dem die vom Benutzer in ein Formular eingegebenen Daten übergeben werden. Dieses Programm verarbeitet die Daten und liefert daraufhin eine Seite zurück, die der Web-Server dann an den Browser des Benutzers schickt.

Prinzipiell kann hierfür jede beliebige Programmiersprache verwendet werden. Zur Anbindung an die Datenbank stehen je nach Programmiersprache hierfür Klassenbibliotheken, Module oder ähnliches zur Verfügung. Dies gilt natürlich nicht nur für CGI-Programme, sondern auch generell zur Datenbank-Client-Programmierung. Insofern handelt es sich also bei dieser Methode um spezielle Datenbank-Clients, die eine gewisse Spezifikation als Schnittstelle zum Web-Server implementieren [2].

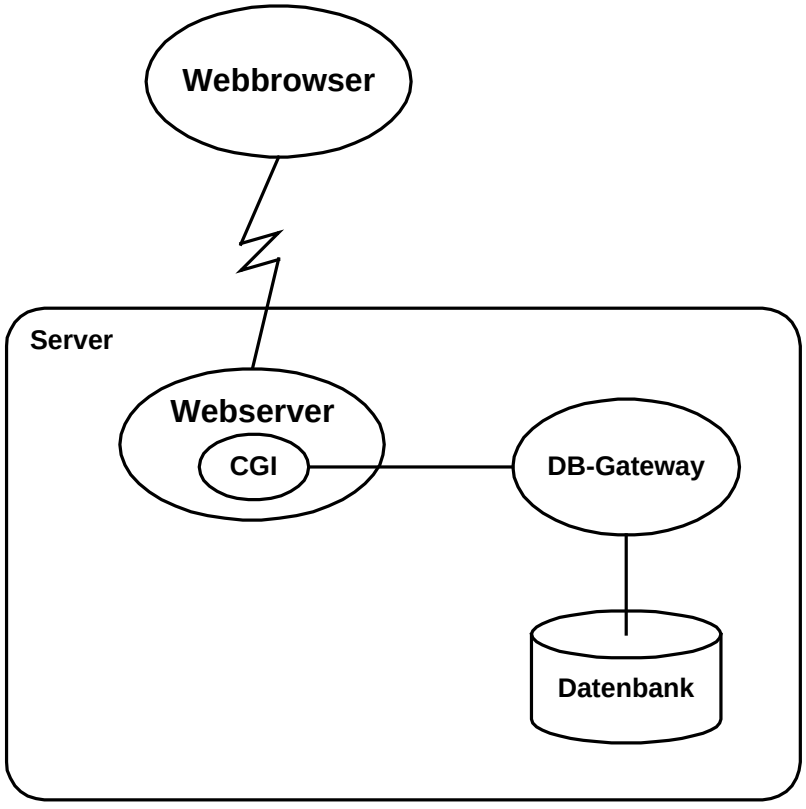
Diese Methode hat allerdings den Nachteil, daß für jede Anfrage ein neuer Prozeß gestartet werden muß, der eine Verbindung zur Datenbank aufgebaut [1, 2]. Ist die Anfrage schließlich bearbeitet, wird der Prozeß beendet, und die Verbindung steht nicht weiterhin zur Verfügung. Für jede neue Anfrage muß also eine neue Verbindung zum Datenbanksystem aufgebaut werden. Dies verlangsamt die Bearbeitung der Anfragen sehr, besonders bei sehr hohen Server-Belastungen.

Für Anwendungen, bei denen es allerdings nicht so sehr auf die Ausführungszeit ankommt bzw. bei denen man mit nicht sehr hohen Besucher-Zahlen rechnet, stellt dies allerdings eine einfache Möglichkeit dar, die Datenbankanbindung zu realisieren. Für CGI-Bin's werden in der Regel Skriptsprachen wie Perl [5] verwendet, die leicht zu erlernen sind, und mit denen man schnell einfache Programme erstellen kann.

Das folgende Beispiel illustriert die Verwendung von Perl als Skriptsprache zur Implementierung einer Datenbank-Abfrage:

```
#!/usr/bin/perl
# Msql-Package laden:
use Msql;
# Seitenkopf ausgeben:
print "Content-type: text/html\n\n"
# [...]
# Verbindung mit dem DB-Server herstellen
$testdb = Msql->connect;
# Datenbank auswählen
$testdb->selectdb("bookmarks");
# Datenbankabfrage erstellen
$sth = $testdb->query("select name, url from bookmarks order by name");
# Resultat ausgeben
print "<TABLE BORDER=1>\n";
print "<TR><TH>Name</TH><TH>URL</TH></TR>\n";
$rows = sth->numrows;
while ($rows>0)
{
   @sqlrow = sth->fetchrow;
   print "<TR><TD>".@sqlrow[0]."</TD><TD><A HREF=\".@sqlrow[1]."\">"
          .@sqlrow[1]."</A></TD></TR>\n";
   $rows--;
}
print "</TABLE>\n";
# Seitenende ausgeben
# [...]
```

Die folgende Grafik stellt die Verwendung von CGI-Programmen noch einmal bildhaft dar:



3.1.1.2 Verwendung von Server-API's

Der größte Nachteil bei der Verwendung von CGI-Programmen ist, daß für jede Benutzeranfrage ein neuer Prozeß gestartet werden muß, was bei größeren Prozeßen lange Ladezeiten bedeutet [1, 2]. Um diesen Nachteil zu beheben haben die Hersteller der Web-Server verschiedene, von einander unabhängige Schnittstellen zur Programmierung entwickelt. Bei dieser Methode können zum eigentlichen Server Module hinzu gebunden werden, um die Möglichkeiten des Servers zu erweitern.

Folgende Tabelle zeigt die populärsten API's [2]:

API	Hersteller	Web-Server
NSAPI	Netscape	Fast Track Server, Enterprise Server
ISAPI	Microsoft	Internet Information Server
WSAPI	O'Reilley	WebSite Pro
Apache API	Apache Group	Apache

Die Module, die zum Server hinzu gebunden wurden, können jede Aktion des Servers, vom Bearbeiten des URL bis hin zum Logging erweitern [1]. Außerdem teilen sie den gemeinsamen Adressraum und das starten eines neuen Prozesses entfällt. Hierdurch ergeben sich wesentlich kürzere Ladezeiten, einer der grundsätzlichen Vorteile von Server-Erweiterungen gegenüber CGI-Programmen. Außerdem kann die Verbindung zum Datenbank-Server aufrecht erhalten werden, und muß nicht bei jeder Benutzeranfrage neu aufgebaut werden.

Der große Nachteil von Server-API's besteht darin, daß bislang noch keine gemeinsame Schnittstelle definiert ist. Somit ist man bei Verwendung einer Server-API auf diesen speziellen Server-Hersteller festgelegt.

Eine der am weitesten verbreiteten Verwendungen von Server-Erweiterungen besteht darin, Module zum Server hinzu zu binden, die ihm die Fähigkeit verleihen, angereicherte HTML-Dokumente zu verarbeiten. Dies geschieht dann z. B. durch Präprozessoren. Diese Analysieren die Dokumentdateien und durchsuchen sie nach speziellen Tags, welche Programmcode enthalten. Dieser Code wird dann vom Präprozessor ausgeführt und so dynamisch eine Seite erzeugt. Ein gutes Beispiel hierfür ist PHP3 [6]. Der Server erkennt durch die Endung der Datei (.php3) ,daß es sich bei dem abgerufenen Dokument um eines mit speziellen PHP3-Tags angereichertes HTML-Format handelt, und leitet dieses durch den Preprozeßor.

Ein großer Vorteil von angereicherten HTML-Dokumenten besteht darin, daß unter anderem direkt erkennbar ist, wann im Dokument welcher Befehl abgearbeitet wird. Außerdem können solche Dokumente mit jedem beliebigen Standard-HTML-Editor erstellt und bearbeitet werden, da vom Browser diese speziellen Tags ignoriert werden, und die sonstige Formatierung nicht beeinflussen.

Folgendes in PHP3 formuliertes Beispiel illustriert die Anbindung an eine Datenbank durch ein angereichertes HTML-Dokument:

```
<!-- Seitenkopf... --><?PHP
// Verbindung zum Datenbank-Server aufnehmen
mysql_connect("localhost", "username", "password");
// Datenbank auswählen
mysql_select_db("bookmarks")  or  die("Error  opening  database");
// Datenbankabfrage erstellen
$result = mysql_query("select name, url from bookmarks order by name");
// Ergebnis ausgeben
?><TABLE BORDER=1><TR><TH>Name</TH><TH>URL</TH></TR>
<?PHP
while ($row = mysql_fetchrow($result)) { ?>
<TR><TD><?PHP echo $row[0]; ?></TD>
<TD><A HREF="<?PHP echo $row[1]."\">".$row[1]; ?></TD></TR>
</TABLE><!—Seiteiende ... --->
```

3.1.2 Zweistufige Lösung

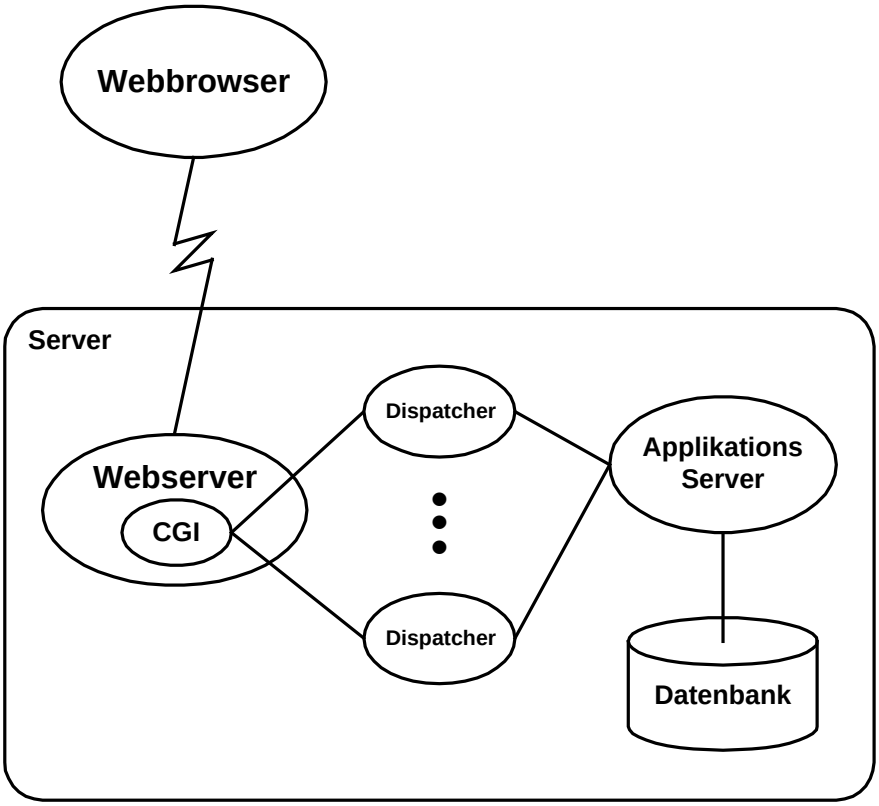
Wie oben besprochen, ist das hauptsächliche Problem bei der einstufigen Lösung, daß bedingt durch wiederholten Verbindungsaufbau und Größe des Programms mit langen Antwortzeiten zu rechnen ist. Dies wird bei der zweistündigen Lösung dadurch versucht zu vermeiden, das man die komplette Applikationslogik nach außen in einen gesonderten Prozeß verlegt. Dieser Verarbeitungsprozeß wird dann auch Applikationsserver genannt.

Das startende CGI-Programm bzw. die Server-Erweiterung dient hierbei nur noch zur Kommunikation mit dem eigentlichen Verarbeitungsprozeß und wird auch "Stub", "Dispatcher" oder "Connector" genannt [2]. Er parst die Benutzeranfrage und leitet diese an den Applikationsserver weiter. Dieser läuft als Dämon-Prozeß bzw. als System-Dienst ständig im Hintergrund und wartet auf solche Anfragen. Erhält er eine solche Anfrage, verarbeitet er diese und schickt das Ergebnis formatiert als HTML-Seite zurück an den Dispatcher. Dieser übergibt die Seite daraufhin an den Web-server, welcher sie an den Browser des Benutzers schickt.

Durch die vom Dispatcher in den Applikationsserver ausgelagerte Applikationslogik ergibt sich eine erheblich kleinere Programmgröße, was die Ladezeit verkürzt [1, 2]. Außerdem kann die Verbindung zwischen Verarbeitungsprozeß und Datenbank-Server aufrecht erhalten werden. Zudem können häufig gebrauchte HTML/Makro-Dateien gepuffert werden, was die Verarbeitung noch einmal beschleunigt [1]. Ein weiterer Vorteil ist eventuell die bessere Verteilung der Last. Dadurch, daß die Applikationslogik auf einen eigenen Prozeß ausgelagert wird, muß die Verarbeitung nicht auf dem gleichen Rechner wie der Web-server laufen. Hierdurch eignet sich die zweistufige Lösung besonders für große Anwendungen mit hoher Besucher-Zahl.

Die Nachteile der zweistufigen Lösung liegen unter anderem darin, daß bedingt durch den ständig im Hintergrund laufenden Verarbeitungsprozeß auch in Zeiten mit geringer Belastung System-Ressourcen belegt werden. Außerdem sind aufgrund der zweiten Schicht und damit durch die höhere Komplexität bedingt die Werkzeuge zur Erstellung solcher Lösungen wesentlich aufwendiger und auch teurer.

Folgende Grafik veranschaulicht die zweistufige Lösung mit Applikationsserver:



3.1.3 Funktionsabarbeitung im Datenbank-Server

Eine dritte Möglichkeit zur Gestaltung der Architektur besteht darin, die Applikationslogik im Datenbank-Server ausführen zu lassen [1]. Hierbei wird die Erweiterungsfähigkeit moderner objekt-relationaler Datenbanksysteme (ORDBS) ausgenutzt. Besonders die benutzerdefinierten Funktionen machen dies möglich. Diese werden direkt im Datenbank-Server ausgeführt.

Ein wesentlicher Vorteil dieser Methode liegt darin, daß die Kommunikation verringert wird. Es ist schließlich kein separater Prozeß für die Applikationslogik notwendig und das Ergebnis der benutzerdefinierten Funktion ist lediglich eine Zeichenkette, welche die HTML-Seite repräsentiert, die über den Web-server zum Browser des Benutzers geschickt wird.

Nachteilig ist, daß der Datenbank-Server hierdurch mehr belastet wird, und auch eine geringere Flexibilität, da die Applikationslogik nicht auf einen eigenen Rechner verlagert werden kann. Außerdem ist eventuell mit einer komplexen Applikations-Entwicklung zu rechnen.

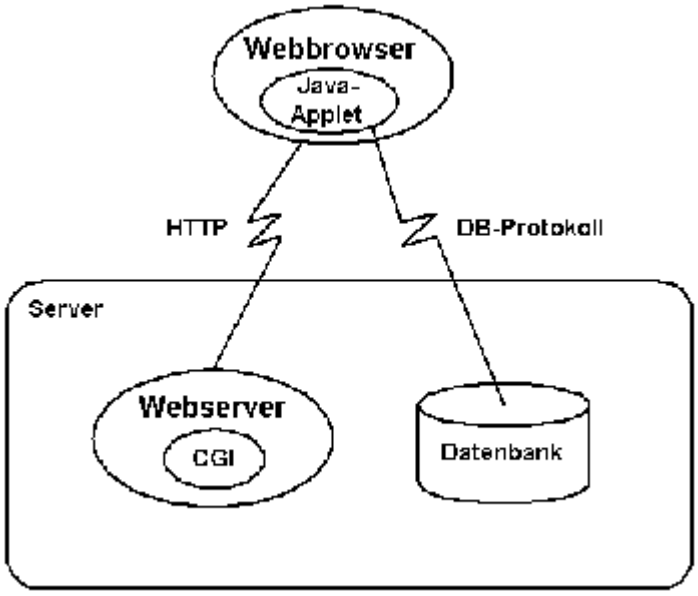
3.2 Client-seitige Applikationslogik

Im folgenden Teilabschnitt beschäftige ich mich mit Methoden, die den Verarbeitungsprozeß in den Rechner des Benutzers verlagern. Diese Verfahren beruhen im allgemeinen entweder auf Browser-Erweiterungen (sog. Plug-Ins) oder auf Java-Applets [2].

Bei Plug-Ins handelt es sich um Zusatz-Programme, die vom Browser aufgerufen werden, wenn er einen Dateityp vom Server erhält, die er selbst nicht darstellen kann. Dies kann verwendet werden, um ein lokal installiertes Programm aufzurufen, das mit dem Datenbank-Server direkt kommuniziert und die komplette Applikationslogik enthält. Das WWW dient hierbei lediglich dazu, den Benutzer zur Applikation zu führen und diese zu starten [2]. Auf diese Methode will ich hier allerdings nicht näher eingehen, da sie nur den eigentlichen klassischen Client/Server-Ansatz darstellt und in diesen Sinne für das Thema "Datenbankanbindung im WWW" nicht von besonderem Interesse ist.

Die andere Vorgehensweise ist, in eine HTML-Seite ein sogenanntes Java-Applet einzubinden, was von der im Browser integrierten virtuellen Maschine ausgeführt wird. Dieses kann potentiell direkt mit dem Datenbankserver (bzw. Applikationsserver) kommunizieren – der Umweg über den Web-Server ist nicht mehr notwendig.

Folgende Grafik soll die Architektur mit Java auf Clientseite noch einmal anschaulich darstellen:



Im folgenden will ich mich näher mit diesen Applet-basierten Techniken beschäftigen, deren allgemeinen Vor- und Nachteile, erörtern um anschließend noch kurz die Anbindung an Datenbanken in Java mittels JDBC zu besprechen.

3.2.1 Positive Aspekte beim Einsatz von Java

Ein wesentlicher Vorteil bei der Nutzung von Java besteht darin, daß die Applikationslogik im Browser ausgeführt werden kann, ohne hier vorher ein spezielles Programm installieren zu müssen. Der Administrationsaufwand bei Updates der Software wird hierdurch erheblich reduziert, da nur die Dateien der Java-Klassen(-Archive) gepflegt werden müssen, was zentral auf dem Server erfolgt [2].

Nach der Übertragung und dem Start des Applets auf dem Clientrechner lassen sich von dort aus Netzwerkverbindungen öffnen [1]. Dadurch kann der Umweg über den Web-server vermieden werden. Daraus resultiert eine wesentlich verbesserte Lastverteilung und eine erhebliche Reduzierung des Datenverkehrs.

Im Internet stellt dieser oft den Flaschenhals dar. Bei der Verwendung von Java müssen nur die eigentlichen Daten vom Datenbank-Server zum Applet übertragen werden. Dieses bereitet die übertragenen Daten dann auf und stellt sie grafisch dar. Information zur Formatierung muß nicht – wie bei HTML üblich – übertragen werden. Ein gutes Beispiel hierzu ist der Verlauf des Kurses einer Aktie. Muß bei HTTP-basierten Verfahren hierzu auf Seite des Servers eine Grafik generiert werden, die dann zum Browser übertragen wird, brauchen bei einem Java-basierten Ansatz nur die eigentlichen Verlauf-Daten übertragen zu werden.

Probleme wie die Zustandslosigkeit von HTTP lassen sich so vermeiden und es lassen sich auch komplexe Datenauswertefunktionen implementieren. Außerdem ist es durch diese Fähigkeit, Zustände zu speichern ohne größeren Aufwand möglich, mehrere voneinander abhängige Datenbank-Anweisungen in einer Transaktion auszuführen. Dies bedeutet, daß nun auch "lange" Transaktionen durchgeführt werden können.

Außerdem ist es durch die Grafikfähigkeit von Java auch möglich, neben den von HTML unterstützten alphanumerischen Geschäftsdaten in Tabellenform auch komplexe Zusammenhänge zu visualisieren [1]. Dazu gehören z. B. Meßwerte oder Geometrie-/CAD-Daten. Außerdem verbessern sich durch diese Grafikfähigkeit die Möglichkeit, die Gestaltung der Benutzeroberfläche (GUI) an besondere, individuelle Anforderungen anzupassen.

3.2.2 Negative Aspekte beim Einsatz von Java

Bei der Nutzung von Java auf Client-Seite gibt es leider aber auch Nachteile. Ein großer Nachteil ist z. B. die relativ lange initiale Ladezeit [1]. Dies folgt aus der Tatsache, daß zunächst das Applet, welches die Applikationslogik enthält, vom Web-server in den Browser geladen werden muß. Durch Einsatz neuerer Java-Versionen kann dies allerdings doch beschleunigt werden. So ist es z. B. möglich, mit signierten Java-Applets zu arbeiten. Diese können dauerhaft auf Client-Seite vorgehalten werden, und müssen nicht bei jedem Programmstart neu übertragen werden. Eine andere Lösung besteht darin, zunächst nur die graphische Benutzer-Schnittstelle zu laden und das Laden anderer Komponenten soweit zu verzögern, bis diese tatsächlich gebraucht werden. Dadurch kann mit einem vergleichsweise kleinen Applet und geringen Ladezeiten gestartet werden.

Desweiteren wirkt sich bei dieser Methode das Sicherheitsprinzip von Java-Applets hinderlich aus. Durch die sogenannte "Sandbox", in dem Java-Applets laufen, ist es im allgemeinen nur möglich, Netzwerkverbindungen zu dem Server aufzubauen, von dem auch das Applet zum Browser übertragen wurde [1]. Dies bedeutet wiederum eine potentiell schlechtere Lastverteilung, da entweder der Datenbank-Server auf dem gleichen Rechner wie der Web-server laufen muß oder auf diesen Rechner ein spezieller Verbindungsserver zwischengeschaltet werden muß. Eine Ausnahme bilden hier wiederum signierte Java-Applets. Nach einmaliger Bestätigung durch den Benutzer haben diese das Recht, Netzwerkverbindungen zu jedem beliebigen Server aufzubauen.

3.2.3 Datenbankanbindung in Java mittels JDBC

Nun möchte ich zum Ende dieses Abschnitts noch kurz besprechen, wie eine Anbindung von Datenbanken in Java mittels JDBC realisierbar ist.

JDBC steht für "Java DataBase Connectivity" [7]. Dieser Name wurde in Anlehnung an "ODBC" vergeben. ODBC steht für "Open DataBase Connectivitiy" und ist ein von Microsoft

und verschiedenen anderen Software-Herstellern entwickelter Standard für den API-Zugriff auf Datenbanksysteme.

Bei JDBC handelt es sich um spezielle Klassenbibliotheken in Java, die Schnittstellen (sog. Interfaces) definieren, welche zur Datenbankanbindung verwendet werden. Die JDBC-Bibliotheken sind seit der Java-Version (JDK) 1. 1 in der Standard-Java-API enthalten und werden ständig um weitere Funktionalität ergänzt. JDBC-Treiber, welche unter anderem von DBVS-Herstellern angeboten werden, implementieren diese Schnittstellen und können nun vom Java-Programm zur Laufzeit (bei der Instanziierung der JDBC-Klassen) geladen werden, um auf die jeweilige Datenbank zugreifen zu können.

JDBC-basierte Programme sind abgesehen von produktspezifischen Einschränkungen und der SQL-Syntax unabhängig von speziellen Datenbanksystemen (und dem dazu gehörigen JDBC-Treiber). Die sich dadurch ergebende leichte Portierbarkeit ist ein wesentlicher Vorteil von JDBC.

Ein weiterer Vorteil von JDBC liegt in der großen Mächtigkeit. Hierdurch lassen sich auch komplexe und dynamisch konfigurierbare Datenbankapplikationen entwickeln.

JDBC-Treiber werden je nach ihrer Architektur in vier verschiedene Typen aufgeteilt. Folgende Tabelle listet diese Typen mit ihren jeweiligen Eigenschaften auf.

JDBC-Treiber-Typ	***Beschreibung***
Typ 1 (JDBC-ODBC-Bridge)	• Dient zur Anbindung über vorhandene ODBC-Treiber • Benötigt nativen Binärcode
Typ 2 (Native-API-partly-Java)	• Wandelt JDBC-Aufrufe in solche für proprietäre Client-API's • Benötigt ebenfalls nativen Binärcode
Typ 3 (Net-Protocol-All-Java)	• Übersetzt JBDC-Aufrufe in DBVS-unabhängiges Netzwerkprotokoll für Middleware • Middleware übersetzt NW-Aufrufe und verbindet Client mit dem DBVS
Typ 4 (Native-Protocol-All-Java)	• Übersetzt JBDC-Aufrufe in DBVS-spezifisches Protokoll • Direkte Verbindung zwischen Client und DBVS

Wie man der Tabelle entnehmen kann, benötigen JDBC-Treiber vom Typ 1 und 2 nativen Binärcode, so daß diese für die Realisierung Web-basierter Datenbankanwendungen nicht in Frage kommen. Hierfür müsste nämlich noch neben einem Java-fähigen Browser zusätzliche Software auf dem Client-Rechner installiert sein, was dem Gedanken einer plattformunabhängigen Web-basierten Applikation widerspricht.

Bei Treibern vom Typ 3 wird ein Verbindungsserver (Middleware) zur Kommunikation benutzt. Dies ist wohl die flexibelste Lösung. Ein weiterer Vorteil hierbei ist die erhöhte Sicherheit, da hier keine direkte Kommunikation mit dem Datenbanksystem stattfindet. Dieses kann durch eine Firewall abgesichert werden, und ist so keinen direkten Angriffen ausgesetzt. Im Gegensatz dazu bringen Treiber vom Typ 4 jedoch Vorteile bezüglich der Antwortzeit, da hier ein direkter Kontakt zum Datenbank-Server aufgebaut wird. Da jedoch kein Verbindungsserver genutzt wird, ergeben sich bei der Verwendung unsignierter Applets wiederum, wie weiter oben schon erwähnt, Nachteile bezüglich der Lastverteilung, da der

Datenbankserver auf dem gleichen Rechner, wie der Web-Server laufen muß, und dieser so eventuell zum Flaschenhals werden kann.

3.3 Benutung von SQL3

Im vorangegangenen Teil dieser Ausarbeitung habe ich mich hauptsächlich damit beschäftigt, wie man dynamisch auf Benutzeranfrage hin HTML-Seiten erzeugen kann. Dies war bisher die einzige Möglichkeit, aktuelle Daten bereit zu stellen. Dabei ist allerdings mit großen Nachteilen, z. B. langen Antwortzeiten zu rechnen. Außerdem entstehen durch hohen Ressourcen-Verbrauch zusätzliche Probleme.

Deshalb will ich mich im folgenden damit beschäftigen, wie man aktuelle Daten zur Verfügung stellen kann, ohne mit dem Problem hohen Ressourcen-Verbrauchs leben zu müssen.

3.3.1 Entwicklungsmöglichkeiten durch neue Konzepte in SQL3

Eine gute Möglichkeit hierfür bietet SQL 3 mit dem neuen Konzept der Trigger und benutzerdefinierten Funktionen [3].

Diese benutzerdefinierten Funktionen bieten die Chance, das Datenbanksystem zu erweitern, um dadurch flexiblere Anwendungsmöglichkeiten desselben zu schaffen. Hierbei sei als Beispiel das Schreiben einer Datei in das Dateisystem genannt.

Ein Trigger wird dazu benötigt, zu definieren, ob bei einer Änderungsoperation an einer Datenbank-Tabelle eine benutzerdefinierte Funktion aufgerufen werden soll.

3.3.2 Einbindung der Neuerungen zur Aktualisierung von Web-Seiten

Mit einem Trigger kann definiert werden, welche Aktionen bei einer Aktualisierung einer Datenbank-Tabelle zu erfolgen haben. Ein Trigger kann zum Beispiel wie folgt definiert sein:

```
CREATE TRIGGER docGenerate AFTER INSERT ON news REFERRENCING NEW AS n
(EXECUTE PROCEDURE generateDoc(n.id, 'table=news'));
```

In diesem Beispiel wird ein Trigger definiert, der die Ausführung einer benutzerdefinierten Funktion "generateDoc" anstößt, sobald eine Aktualisierung der Tabelle "news" auftritt. Diese Funktion könnte nun die Daten aus der Tabelle lesen und daraus eine HTML-Seite erzeugen, die daraufhin in das Datei-System des Web-Servers geschrieben wird.

Diese Methode hat den großen Vorteil, daß bei Anfrage des Benutzers nur eine statische Seite aus dem Datei-System des Web-Servers gelesen werden muß, was die Antwortzeiten erheblich verkürzt. Trotzdem sind die Seiten ständig aktuell, da bei jeder Aktualisierung der Datenbank auch eine Aktualisierung der betreffenden HTML-Seiten vorgenommen wird.

Das oben genannte Szenario ist allerdings ein sehr einfaches, da hier nur die Aktualisierung einer einzelnen Tabelle zu beachten ist. Hängt der Inhalt einer HTML-Seite jedoch von mehreren Tabellen ab, so steigt die Komplexität erheblich. Es müssen mehrere Trigger (jeweils für die betreffenden Tabellen) erstellt und verwaltet werden.

Deshalb ist es sinnvoll, ein Werkzeug einzusetzen, welches die Verwaltung solcher Trigger, die Erzeugung der HTML-Seiten und der Dokumente im Datei-System vereinfacht. Ein solches System stellt "iWebDB" [8] dar, welches an der Universität Kaiserslautern entwickelt wird. Mit diesem Werkzeug ist es sehr einfach, ein solches System zu erstellen und zu verwalten. Mit Hilfe von Dokument-Vorlagen können die HTML-Seiten in Abhängigkeit der Daten aus der Datenbank spezifiziert werden. Mit einer graphischen Benutzeroberfläche können alle Einstellungen (Trigger, Dokumente, Vorlagen...) administriert werden.

3. Zusammenfassung

Abschließend möchte ich noch einmal zusammenfassen, welche Themen ich in diesem Beitrag erörtert habe.

Zunächst habe ich mich damit beschäftigt, zu welchen Zwecken eine Datenbankanbindung benutzt wird. Dabei bin ich auf 3 Gründe näher eingegangen: Einbindung von unternehmensinternen Daten mit Hilfe der Technologie des Internets – Aufbau eines sog. Intranets, Bereitstellung vorhandener Daten im WWW und Erstellung von stark interaktiven Anwendungen, z. B. Online-Einkaufszentren oder –Auskünften. Dabei bin ich näher auf die Aspekte dieser drei typischen Anwendungsgebiete eingegangen und habe Besonderheiten erörtert.

Danach habe ich erörtert, welche Möglichkeiten zur Realisierung einer Datenbankanbindung bestehen. Hierbei habe ich verschiedene System-Architekturen unterschieden und diese im einzelnen besprochen. Als erstes habe ich erklärt, wie man eine Datenbankanbindung mit Applikationslogik auf Seite des Servers realisiert. Ein Vorteil dieser Methode ist unter anderem, das hierbei keine Programm-Daten zum Browser des Benutzers übertragen werden müssen. Allerdings sind hierbei Probleme wie die Zustandslosigkeit von http zu beachten.

Bei dieser Architektur habe ich zwischen Formen mit einstufiger und zweistufiger Logik und Funktionsabarbeitung im Datenbank-Server unterschieden. Einstufige Logik kann man mit Hilfe von CGI-bin’s oder sogenannte Server-API implementieren. Der Aufbau dieser Architektur ist sehr einfach, wodurch schnell kleine Anwendungen realisierbar werden. Da bei dieser Methode allerdings für jede Anfrage ein Prozeß gestartet werden muß und dieser sich neu am Datenbanktabellen anmelden muß, ist allerdings mit langen Wartezeiten zu rechnen.

Dieser Nachteil wird durch die zweistufige Applikationslogik behoben. Hierbei läuft ständig im Hintergrund ein Prozeß, der die Verbindung zum Datenbanksystem aufrecht erhält und Daten zur Verarbeitung über sogenannte Dispatcher entgegen nimmt. Diese Dispatcher sind relativ klein und deshalb schnell zu laden, was die Antwortzeit erheblich verkürzt.

Eine dritte Möglichkeit zur Realisierung der Datenbankanbindung auf Seite des Servers besteht darin, die Funktionsabarbeitung im Datenbank-Server zu verwirklichen. Hierbei wird die Erweiterungsfähigkeit moderner Datenbanksysteme benutzt. Ein großer Vorteil liegt in der verringerten Kommunikation, da das Ergebnis der Anfrage direkt in Datenbanksystem ermittelt werden kann und so lediglich eine Zeichenkette übertragen werden muß. Der Datenbank-Server wird hierbei allerdings mehr belastet, was einer Last-Verteilungs-Strategie eventuell widerspricht.

Die andere Möglichkeit, die Datenbankanbindung zu realisieren besteht darin, die Applikationslogik auf Seite des Clients auszuführen. Dabei wird mit Java-Applets gearbeitet. Dadurch, daß dieses Applet direkt eine Verbindung zum Datenbanksystem aufbauen kann, wird der Umweg über einen Web-Server vermieden. Außerdem müssen bei dieser Verbindung nur wirklich relevante Daten übertragen werden, was die Kommunikation verringert. Ein weiterer Vorteil ist die Möglichkeit, gegenüber http relativ einfach Zustände speichern zu können und „lange“ Transaktionen ausführen zu können. Außerdem ist es wesentlich einfacher, eine flexible Benutzer-Schnittstelle zu programmieren. Nachteile ergeben sich bei der Verwendung von Java allerdings durch eine initial lange Ladezeit, da zunächst das Programm in den Browser geladen werden muß. Außerdem sind dem Applet durch das Prinzip der „Sandbox“ einige Restriktionen auferlegt.

Eine neue Methode besteht darin, die Möglichkeiten von SQL 3 zu benutzen, um Web-Seiten aktuell zu halten. Bei Änderungen in Datenbanktabellen werden mit Hilfe von sogenannten Triggern und benutzerdefinierten Funktionen die zugehörigen HTML-Seiten aktualisiert. Hierdurch können die Antwortzeiten erheblich verkürzt werden, da bei einer Anfrage lediglich eine Seite aus dem Dateisystem gelesen und zum Browser des Benutzers übertragen werden muß. Allerdings kann diese diese Methode nicht zum verarbeiten von dynamischen Anfragen verwendet werden.

Abschließend kann man sagen, daß die Auswahl der Methode zur Realisierung der Datenbankanbindung sehr davon abhängt, welche Anwendung verwirklicht werden soll. Für kleinere Anwendungen, wie z. B. ein Gästebuch kommt wahrscheinlich am ehesten eine einstufige Realisierung mit CGI in Frage, wohingegen bei komplexen Anwendungen, z. B. bei Onlinebanking eher mit Java gearbeitet würde.

5. Literatur

[1] Loeser, H.: Techniken für Web-basierte Datenbankanwendungen - Anforderungen, Ansätze, Architekturen, in: Informatik - Forschung und Entwicklung 13(4), Springer, 1998.

[2] Hoffmann, W. : Datenbankankanbindung für das WWW – Untersuchung von Gateways und Entwicklung einer prototypischen Web-Datenbank-Anwendung – Diplomarbeit Fachhochschule Gummersbach, 1997.

[3] Loeser, H. : Keeping Web-pages Up-to-Date with SQL3, interner Bericht, Universität Kaiserslautern, 1999.

[4] The CGI Specification: http://hoohoo.ncsa.uiuc.edu/cgi/

[5] http://www.perl.com

[6] http://www.php.net

[7] JDBC 2.0 API : http://www.javasoft.com/products/jdbc/

[8] Loeser, H., Ritter, N. : iWebDB – Integrated Web Content Management baesed on Object-Relational Database Technology, in: Proc. Int. Database Engineering and Application Symposium (IDEAS'99), Montreal, Canada, 1999.