Datenbanken im WWW — Von CGI bis JDBC und XML

Holger Meyer Meike Klettke
Andreas Heuer

Lehrstuhl Datenbank- und Informationssysteme
Fachbereich Informatik, Universität Rostock
18051 Rostock
Telefon: ++49(381) 498–3401, Fax: –3443
Email: {hme,meike,heuer}@informatik.uni-rostock.de

20. Mai 2000

Zusammenfassung

Datenbanken im Internet sind ein aktuelles Entwicklungsgebiet, das sowohl in der Forschung als auch in kommerziellen Entwicklungen aktiv verfolgt wird. Von besonderem Interesse ist hier das *World Wide Web*, das als globales Informationssystem verstanden werden kann.

Der Erfolg des WWW bringt Probleme mit sich, die mit der Entwicklung und dem Einsatz von Datenbanksystemen in Unternehmen überwunden geglaubt waren: Heterogenität, Verteiltheit und Unstrukturiertheit der Information. Jeder kann Informationen im WWW bereitstellen und dabei Inhalt und Form im Rahmen der technischen Möglichkeiten frei bestimmen. Dieser „anarchistische“ Zustand ist bei einem globalen, dezentralen System wie dem WWW sowohl zum Teil gewollt als auch unvermeidlich.

Als Resultat stellt sich folgerichtig oft die Frage: Wie kann man aus heterogenen, verteilten Informationen im WWW homogene, recherchierbare Informationen eines Informationssystems gewinnen? Wir werden in diesem Beitrag kurz den möglichen vorteilhaften Einsatz der Datenbanktechnologie zur Beantwortung dieser Frage diskutieren.

Keywords: Datenbanken, Internet, XML, Java, CGI, JDBC, SQLJ

1 Vorteile der Nutzung von Datenbanken im WWW

Der klassische Ansatz des WWW basiert auf Dokumenten in Form von HTML-Dateien, die über eine URL adressiert sind und durch Links miteinander verbunden werden. Die derart dargestellten Informationen sind oft redundant und ab einer gewissen Größe unübersichtlich.

Die Kombination von Hypermedia-Systemen im Internet (WWW) und Datenbankinformationen kann Suchen in Dokumenten und Anfragen in Datenbanken kombinieren. Die Integration von DBMS als Informationsquellen kann die Anfragen und Transaktionen von Datenbanksystemen möglichst weitgehend nutzen. Im Gegensatz zu reinen HTML-Strukturen ermöglicht eine derartige Kombination den Einsatz von Optimierern, Integritätskontrolle und Zugriffskontrolle der Datenbanksysteme.

In den meisten Bereichen werden Datenbanken in lokalen Informationssystemen für operative Daten bereits eingesetzt, um große Datenmengen in kompakter Form zu verwalten. Datenbanken sind leichter aktualisierbar als Textdokumente und vermeiden fehlerträchtige Redundanz durch Integration.

In vielen Anwendungen bringt auch der Einsatz von Datenbankanfragesprachen Vorteile, da in diesen die Wünsche des Anwenders genauer spezifiziert werden können, exakte Treffer geliefert werden können und verschiedene Sichten auf Datenbanken mit Anfragen formulierbar sind.

Die Kombination von Datenbanken und Anfragen im WWW führt zu dem Begriff der *dynamischen HTML-Dokumente*. Dynamische HTML-Dokumente werden aufgrund von Anfrageauswertungen generiert, entsprechen also Sichten auf den Datenbestand in Form von Dokumenten.

Für die Einbindung und Ausführung von Programmen im Rahmen des WWW gibt es zwei unterschiedliche Ansätze. Die *client-seitige* Ausführung wird durch Einbetten von Programmcode (z.B. Java-Applets, JavaScript) in ein Dokument und die Ausführung im Browser realisiert. Das WWW dient demzufolge nur noch als Verteilungsmedium.

Bei der *server-seitigen* Ausführung werden Dienstprogramme als Erweiterungen des WWW-Servers aufgerufen um so z. B. einen Datenbankzugriff zu realisieren. Diese Programme können entweder externe Programme sein, die über CGI aufgerufen werden, oder als server-seitige Skripte in die Dokumente eingebettet und vor der Auslieferung durch den Server ausgeführt werden.

2 Suchen und Anfragen im WWW

Bei der Frage, welche der weiter unten einzuführenden Architekturen sich für eine bestimmte Anwendung eignen, muß zunächst die Frage geklärt werden, welche Art des Zugriffs auf die Informationen im WWW gewünscht wird:

- Möchte man strukturierte Datenbankdaten im WWW darstellen, so wird der Hauptzugriffsweg auf diese Daten über eine Datenbankanfragesprache wie SQL oder ein Anfrageformular laufen. Mit WWW-Techniken wird hier nur die externe Darstellung der Daten

realisiert. Die Anbindung des Datenbanksystems erfolgt in diesen Fällen meist über die unten beschriebenen Standard-Techniken.

- Möchte man (in HTML oder zukünftig in XML beschriebene) Internet-Dokumente in Dokumentverwaltungssystemen speichern und auf diese zugreifen, so spricht man vom Content Management. Üblicherweise werden Suchmaschinen eingesetzt, um nach Merkmalen des Inhalts der Daten — also inhaltsbasiert — (bei Texten: Stichworten, bei Bildern: etwa Farben) suchen zu können.

Wir werden hier zunächst die wesentlichen Merkmale und Unterschiede von Such- oder Retrieval-Verfahren zu Datenbank-Anfragemechanismen darstellen.

Die Prozesse *Suche* oder *Retrieval* (Information Retrieval) kann man durch folgende Eigenschaften charakterisieren:

- Die Suche wird auf Texte, Multimedia-Daten oder andere unstrukturierte Daten angewendet.
- Oft wird das Boolesche Retrieval verwendet: Elemente aus dem Inhalt des Dokuments (content-based retrieval) werden mit booleschen Operatoren (und, oder, nicht) verknüpft.
- Die Vorteile der Suchverfahren sind die Unterstützung vager Anfragen, Ranking-Funktionalitäten zur Gewichtung der Ergebnisdokumente und ein Relevance Feedback zum iterativen Verfeinern und Verbessern von Suchergebnissen.

Anfragen an Datenbanken sind dagegen durch folgende Merkmale gekennzeichnet:

- Anfragen werden auf strukturierte Daten angewendet.
- Übliche Operationen sind eine attributierte Selektion von Daten (Attribut Nachname = ‘Meyer‘), Verknüpfung von Daten und Dokumenten verschiedener Typen und eine Restrukturierung von Dokumentmengen.
- Die Vorteile von Datenbankanfragen sind attributierte Anfragen, eine Ausnutzung des Typs der Daten (typspezifische Operatoren wie Vergleiche), Verknüpfungen und Restrukturierungen.

Wie kann man diese beiden Zugriffsmöglichkeiten nun bei einer Mischung von strukturierten und unstrukturierten Anteilen, allgemeiner also auf sogenannten *semistrukturierten Daten*, kombinieren?

Hat man Datenbankdaten über eine der Standard-Techniken in eine WWW-Präsentation eingebunden, muß ein Datenbankanfrageformular an eine Suchmaschine angebunden werden. Werden dagegen HTML- oder XML-Dokumente in Datenbanken gespeichert, so werden die Suchfunktionen aus einer Datenbankanfrage heraus aufgerufen, wie im neuen SQL-Standard SQL-99/MM

Text, oder die Anfragesprache (z. B. XQL und XML-QL) gleich auf XML-Dokumenten definiert, wie es der Ansatz bei den XML-Servern ist.

Bevor wir nun auf die einzelnen Techniken zur Verwaltung von Daten und Dokumenten im WWW eingehen, werden wir die verschiedenen Ansätze noch auf einer detaillierteren Ebene als in den bisherigen Abschnitten klassifizieren.

3 Klassifizierung der Lösungsansätze

Die einzelnen Architekturen unterscheiden sich insbesondere in der Dynamik der angezeigten Dokumente. Klassische reine HTML-Dokumente entsprechen *statischen Seiten*, deren Inhalt als Text im Dokument festgeschrieben ist. *Dynamische Seiten* beinhalten Daten, die erst beim Abruf einer Seite durch spezielle Filterprogramme oder -funktionen des WWW-Servers aus einer Datenbank geholt und eingefügt werden. *Dynamisch erzeugte Seiten* können komplett aus der Datenbank generiert werden. Diese Varianten geben mögliche Koppelungen von Datenbanksystemen mit WWW-Diensten vor, die ein Informationssystem im WWW realisieren können.

WWW-Dienst als Informationssystem

Die erste Variante realisiert das Informationssystem als Sammlung von statischen Seiten. Eine mögliche Koppelung mit einem DBMS erfolgt dadurch, daß die statischen Seiten aus dem Datenbankinhalt *generiert* werden (etwa durch Einsatz von Report-Generatoren oder Skriptsprachen). Die Verwaltung der Information erfolgt durch die Bereitstellung von Dateien, die über URL erreichbar sind, in der Dateiverwaltung des Betriebssystems.

Die Vorteile dieser Lösung sind insbesondere, daß statische Seiten billig zu erstellen sind, ein schneller Zugriff auf die Dokumente möglich ist und keine Kosten für ein explizites Verwaltungssystem anfallen.

Diesen Vorteilen stehen einige Nachteile gegenüber:

- Eine aktuelle Darstellung im WWW kann nur durch regelmäßiges Anpassen oder erneutes Generieren der Dokumente erfolgen. Bei großen Informationssystemen ist dies mit beträchtlichem Aufwand verbunden.
- Es fehlen die Möglichkeiten der Datenbanktechnik, so gibt es keine generischen Anfrage- und Änderungsoperationen, keine Optimierung, keine Konsistenzkontrolle, keine Zugriffskontrolle und keine Transaktionen.
- Da die HTML-Seiten mehrere Benutzungsprofile unterstützen sollten, werden Daten oft redundant und damit fehlerträchtig dargestellt.

Leider ist diese Variante oft noch der Regelfall bei Web-Präsentationen, auch wenn die Daten in einer Datenbank enthalten sind.

Web-Server mit Schnittstelle zum Datenbanksystem

Bei dieser Realisierungsvariante besteht das Informationssystem aus statischen und dynamischen Seiten, wobei letztere (zum Beispiel über CGI-Skripte) beim Aufruf mit Daten aus einer Datenbank angereichert oder neu generiert werden.

Abbildung 1 WWW-Seiten mit CGI-Schnittstellen

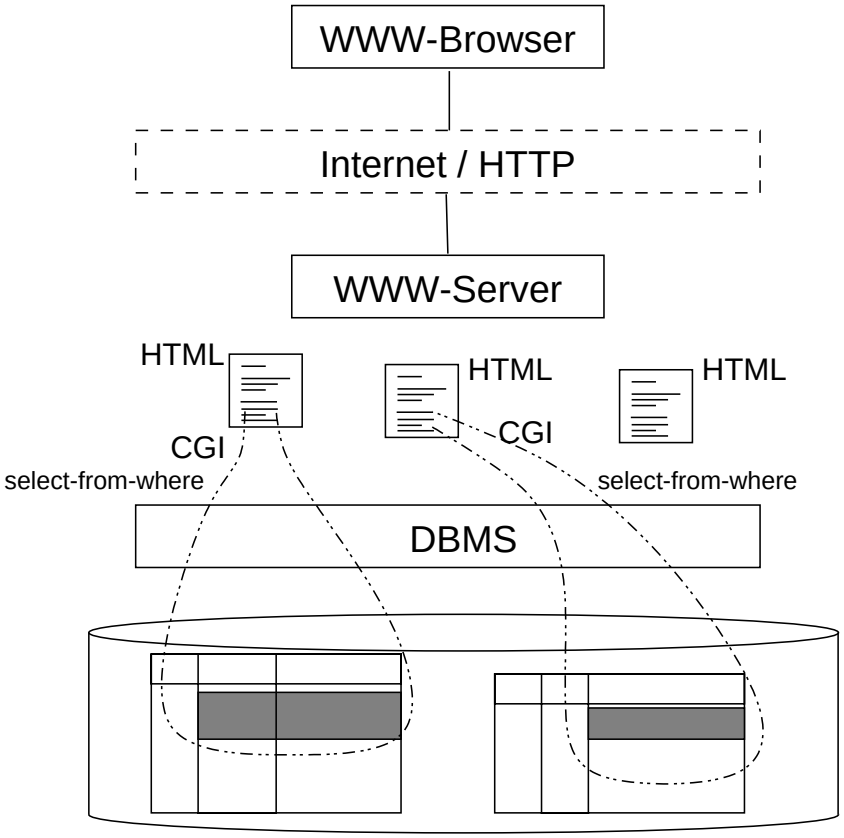


Abbildung 1 skizziert den prinzipiellen Aufbau dieser Lösung. Die Schnittstelle zwischen WWW-Server und den Anfrageprogrammen wird hierbei über das *Common Gateway Interface (CGI)* realisiert. Ein CGI-Programm wird durch eine URL identifiziert und über eine normale Anfrage beim WWW-Server aufgerufen. Zusätzlich können noch Parameter übergeben werden, die z.B. die Anfrageparameter repräsentieren. Als Ergebnis der Bearbeitung liefert ein CGI-Programm eine HTML-Seite mit den Anfrageresultat. Ein Problem ist hierbei, daß die Programme bei jedem Aufruf die Verbindung zur Datenbank herstellen müssen. Erweiterte Ansätze wie FastCGI oder proprietäre Server-Erweiterungen bieten hier jedoch Abhilfe. Konzeptionell ähnlich zum CGI sind *Servlets*, wobei hier Java-Klassen als Server-Erweiterung eingesetzt werden. Der Zugriff auf die Datenbank kann damit z.B. über JDBC erfolgen.

Ein anderer Ansatz ist die Verwendung von server-seitigen Skripten. Dabei werden Anweisun-

gen einer Skriptsprache in die HTML-Seiten eingebunden und bei einer Anforderung vom Server ausgeführt. Die Ergebnisse der Abarbeitung werden in die HTML-Seite übernommen. So können Datenbankanfragen derart eingebunden werden, daß die Anfrageergebnisse Teil des ausgelieferten Dokumentes sind. Für den Client ist diese Einbettung vollständig transparent, es wird nur das HTML-Ergebnis übertragen. Zur Implementierung der Skripte sind verschiedene Sprachen geeignet, z. B. JavaScript, Visual Basic oder spezielle Datenbankprogrammiersprachen Sprachen wie W3SQL.

Die Vorteile der CGI-Lösung sind insbesondere, daß sie mit jedem WWW-Server zusammenspielt, aber die strukturierten Daten in der Datenbank gespeichert werden — mit all den Vorteilen, die eine Datenbanklösung mit sich bringt. Die Einbettungslösung erfordert spezielle Erweiterungen des WWW-Servers. Bei beiden Verfahren ist jedoch insbesondere die Aktualität der präsentierten Daten gewährleistet und die Redundanz wird kontrolliert. Für die Datenhaltung kann ein klassisches, etwa relationales, Datenbanksystem genutzt werden.

Für die unstrukturierten (statischen) Anteile der HTML-Seiten gelten die gleiche Nachteile wie im vorherigen Szenario. Durch die Einbettungslösung ergibt sich eventuell ein langsamer Zugriff auf die Datenbank (indirekter Zugriff).

Ausführliche Beschreibungen der möglichen Varianten finden sich zum Beispiel in den Artikeln von Loeser [Loe97, Loe98] oder in [AGW98].

HTML-Seiten im Datenbanksystem

Eine engere Kopplung der WWW-Informationen mit einem DBMS erhält man, indem die Seiten im DBMS gespeichert werden. Hierzu können CLOBs, BLOBs oder ein eigener Datentyp für HTML-Dokumente, etwa in Form eines Datenbankerweiterungsmoduls, genutzt werden. CLOBs und BLOBs können leider nur uninterpretierte Informationen verwalten.

Die Variante der Speicherung von WWW-Seiten in einem DBMS ist in Abbildung 2 skizziert. Vorteilhaft ist bei dieser Lösungsvariante, daß strukturierte und unstrukturierte Daten gemeinsam in der Datenbank gespeichert werden. Die bekannten Datenbankvorteile greifen daher für alle Arten von Informationen.

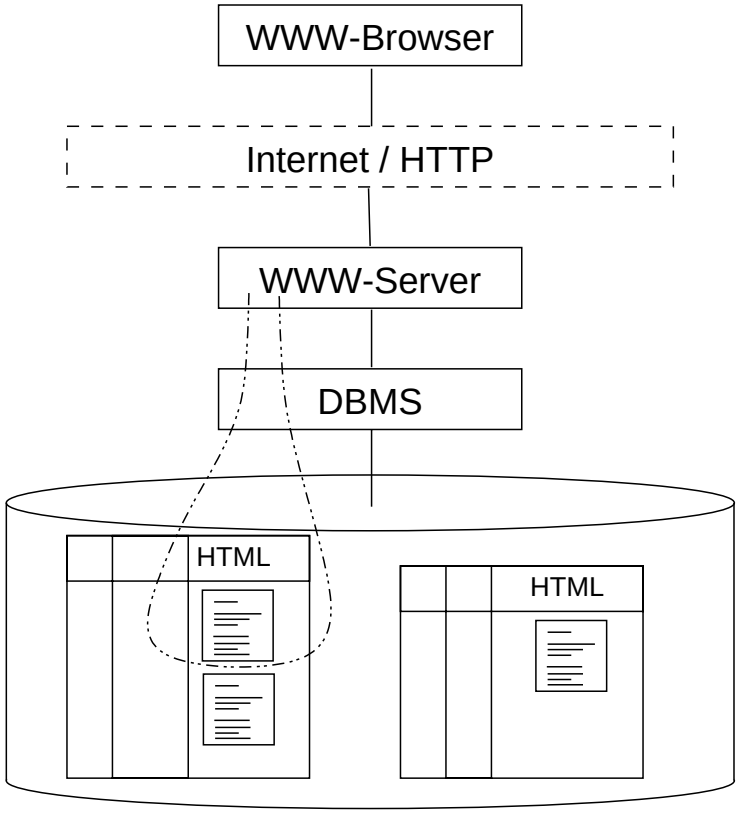
Als Nachteil muß angemerkt werden, daß diese Lösung nicht immer mit jedem WWW-Server zusammenspielt, und daß die Link-Struktur dem Datenbanksystem nicht bekannt ist.

Diese Lösung wird von RDBMS-Herstellern wie Informix und Oracle angeboten. Hierbei sind die HTML-Seiten mit speziellen syntaktische Erweiterungen angereichert, die die Einbettung von SQL-Anweisungen ermöglichen.

Hyperlink-Strukturen im Datenbanksystem

Werden nur HTML-Seiten im DBMS verwaltet, kann die Konsistenz der Vernetzung durch Links nicht durch das DBMS kontrolliert werden. Um diesen Nachteil zu vermeiden, können komplette Hyperlink-Strukturen direkt in Datenbankstrukturen abgebildet werden. Ein Beispiel (O_2Web)

Abbildung 2 WWW-Seiten im Datenbanksystem



wird weiter unten im Rahmen der Vorstellung konkreter Systeme noch eingeführt.

Web-Server als Verteilungsplattform

Ein Problem des Zugriffs auf Datenbanken über das WWW ist die zustandslose Protokoll HTTP, das ein Sitzungs-Management erschwert. Auch sind die Möglichkeiten von HTML hinsichtlich der Gestaltung von interaktiven Benutzerschnittstellen für viele Anwendungen zu eingeschränkt. In diesen Fällen kann das WWW als Verteilungsplattform für Anwendungsfunktionen genutzt werden. Diese Anwendungsfunktionen können u.a. Java-Applets sein, die im Browser ablaufen und über spezielle Protokolle und Schnittstellen (z.B. JDBC, RMI, CORBA) mit dem DBMS oder einen entsprechenden Anwendungsfunktions-Server kommunizieren.

Die Vorteile dieser Lösung sind die Nutzung geeigneter Kommunikationsprotokolle, eine freie Gestaltung der Benutzerschnittstelle und der im Vergleich zur konventionellen (d.h. nicht WWW-basierten) Verteilung geringere Installations- und Verwaltungsaufwand. Allerdings erfordert dieser Ansatz entsprechende Browser-Unterstützung. Außerdem sind geeignete Sicherheitsmaßnahmen zum Schutz der Client-Systeme vor „böswilligen“ Applets vorzusehen, wie etwa die Signie-

rung des Applet-Codes oder die Bereitstellung einer sicheren Ausführungsumgebung (Sandbox).

XML-Dokumente im Datenbanksystem

Die Zukunft der WWW-Unterstützung durch Datenbanksysteme wird voraussichtlich zum Teil in der Verwaltung semistrukturierter Daten durch DBMS liegen. Eine HTML- oder XML-Seite hat feste und variable Bestandteile. Diese Struktur kann bei der Speicherung semistrukturierter Daten direkt in die Datenbank übernommen werden. Die Speicherung von solchen Dokumenten in Datenbanksystemen wird auch als *Content Management* bezeichnet, da hier nicht nur die beschreibenden Daten, sondern auch die semistrukturierten Inhalte direkt mit Datenbanktechniken verwaltet werden.

Die hiermit zusammenhängenden Herausforderungen an die Datenbanktechnologie werden ausführlicher in Abschnitt 6 diskutiert, wo auch die zukünftige Eignung von XML als Datenaustauschformat näher betrachtet wird.

Zunächst werden aber Grundtechniken zur Kopplung von Datenbanksystemen und Web-Anwendungen dargestellt. Neben der CGI-Programmierung mit HTML und Formularen sind dies Java-basierte Ansätze, die genutzt werden können. Auf spezielle, häufig auftretende Probleme, z. B. dem Fehlen einer Sitzungsschicht im HTTP-Protokoll, wird eingegangen. Im darauf folgenden Abschnitt werden drei Datenbanksystem, Oracle8*i*, Informix Dynamic Server und O_2, und ihre Web-Anbindungen vorgestellt.

4 Standard-Techniken

Die einfachsten Formen der Anbindung von Datenbanksysteme an das WWW, sind die Nutzung der Formular-Programmierung in HTML, von HTTP als Standard-Protokoll und der Server-seitigen Anbindung von Anwendungsprogrammen über die CGI-Schnittstelle.

4.1 HTML und Formulare

Der aktuelle Standard-Vorschlag für das WWW ist HTML 4.0.1 [Hyp, Tol97]. Mittlerweile ist die Markup-Sprache zur Gestaltung von Web-Seiten auch in XML [Ext] formuliert. Ein begleitender Vorschlag definiert *Cascading Style Sheets* zur flexible Definition des Layouts. Wesentliche Bestandteile von HTML sind neben der normalen Auszeichnung von Textsorten mächtige Möglichkeiten zum Mathematiksatz (HTML 3.0, MathML) und Formulare, die seit der HTML Version 3.0 existieren und auch separat in XML als XForms definiert wurden.

HTML stammt von SGML ab und übernimmt deren syntaktischen Geflogenheiten. Es kennt drei Arten von Elementen: Entities spezifiziert Zeichen-ähnliche Grundbausteine, einfache Tags wirken an der Position ihres Auftretens, und Umgebungen werden mit Anfangs- und Ende-Tag begrenzt. Im Gegensatz zu XML können Tags beliebig geschachtelt oder überlappend benutzt

werden. Attribute für Tags enthalten weitere Daten und beeinflußen in der Regel deren Wirkung, z. B. steuern beim `<IMG>`-Tag die Größe von Bildern. Die verschiedenen HTML-Tags können für Textelemente, komplexe Text-Strukturen (z. B. Tabellen) und zur logischen und physischen Textauszeichnung verwandt werden.

Formulare können in HTML in das `<FORM>`-Tag eingebettet werden. Das Attribut `ACTION` ist zwingend, und das Argument stellt eine Ziel-URL dar. Hinter dieser URL verbirgt sich üblicherweise ein Programm, das die Formulardaten übermittelt bekommt und als Ergebnis wieder ein HTML-Dokument erzeugt. Das Attribut `METHOD` legt die Methode der Datenübermittlung fest.

Beispiel 1 HTML-Formular

```
<FORM ACTION="/cgi-bin/sample.cgi" METHOD=get>
Vorname: <INPUT NAME="name" TYPE=text SIZE=20 MAXLENGTH=30>

Geschlecht: <SELECT NAME="gender" SIZE=1>
<OPTION VALUE="maennlich"> m\&auml;nnlich
<OPTION VALUE="weiblich" SELECTED> weiblich
</SELECT>
<INPUT TYPE=submit>
</FORM>
```

Eingabe-Elemente innerhalb der `<FORM>`-Umgebung bilden den Datensatz. Mehrere Formularumgebungen auf einer WWW-Seite sind möglich, dürfen aber nicht geschachtelt werden. Neben einfachen Eingabefeldern oder mehrzeiligen Eingabebereichen, können Radio-Buttons, Auswahllisten, oder das Einlesen von externen Dateien eingesetzt werden. Attribute steuern die Darstellung, legen Standardwerte fest oder beeinflußen das Verhalten des WWW-Browsers bei der Arbeit mit HTML-Formularen. Spezielle Formularelemente erlauben das Rücksetzen auf Standardwerte und das Auslösen der Datenübertragung. Eingabeelemente vom Typ *hidden* sind im Browser nicht manipulierbar und können zur Repräsentation von Zustandsvariablen herangezogen werden.

Das Beispiel 1 zeigt ein HTML-Formular mit einem einfachen Eingabefeld, einer Auswahlliste, die als Pop-up dargestellt wird (`SIZE=1`) und einem Submit-Knopf, der dafür sorgt, daß die Formulardaten an den Web-Server mit der angegebenen Methode (`get`) geschickt werden. Der Web-Server startet das angegeben CGI-Programm (code/cgi-bin/sample.cgi), übergibt die Formulardaten (die eingegebenen Werte von `name` und `gender`) und sendet die vom CGI-Programm als Ergebnis erzeugte Web-Seite an den Browser zurück.

Als Übertragungsmethoden können zwei Varianten eingesetzt werden. `METHOD=get` hängt die Formulardaten an die URL an, sie werden über die Umgebungsvariable `QUERY_STRING` an das CGI-Programm vom Web-Server übermittelt. `METHOD=post` überträgt die Daten per HTTP, sie werden vom Web-Server über die Standard-Eingabe an das CGI-Programm vermittelt.

4.2 CGI-Programme und `DB::perl`

Eine der ersten und sehr verbreiteten Techniken zur Erstellung von CGI-Programmen [Mau96] ist die Skriptsprache Perl. Mit `DB::perl`[YRK99] ist man in der Lage, Datenbankanwendungsprogramme in Perl zu realisieren. Diese Datenbankschnittstelle besteht aus zwei Anteilen:

- **DBI** als einheitliche Programmierschnittstelle, die in Anlehnung an SQL/CLI bzw. ODBC entworfen wurde, und
- **DBD**-Treibern für eine Vielzahl von Datenbank- und andere Systeme.

`DB::perl`-Treiber existieren z. B. für Oracle, Informix, Ingres und DB2, aber auch für Volltextdatenbanksysteme wie Fulcrum und für Dateiverwaltungssysteme.

Charakteristisch für die Schnittstelle ist ein Cursor-basierter Zugriff auf die Anfrageergebnisse. Da Perl weitgehend untypisiert ist, gibt es keine Probleme mit beliebigem SQL — alle SQL-Datentypen werden als Zeichenketten in Perl repräsentiert. Über die API kann ein Zugriff auf Schema-Information erfolgen, damit ist eine flexible Programmierung möglich.

Die Grundelemente der API sind in der Tabelle 1 übersichtsartig dargestellt.

Tabelle 1 `DB::perl` API Grundelemente

Anweisung	**Bedeutung**
`$dbh = DBI->connect(`*`Parameter`*`)`	Datenbankverbindung herstellen
`$dbh->disconnect`	Verbindung trennen
`$sth = $dbh->prepare(`*`SQL-String`*`)`	Vorbereiten von SQL-Anweisung
`$rv = $sth->execute`	Ausführen von SQL-Anweisung
`$sth->fetchrow`	Ergebniss satzweise lesen
`$sth->finish`	SQL-Anweisungen abschließen
`$sth->NAMES`	Zugriff auf Ergebnisschema
`$DBI::errstr`	Fehlercode testen, auch mit `$rv`

Ein Beispiel, das das Einfügen in eine Datenbank realisiert, soll die wesentlichen Konzepte von `DB::perl` verdeutlichen. Ein Formular (siehe 1) ist ein normales HTML-Dokument mit FORM-Elementen. Mit `FORM ACTION="`*`Script`*`"` wird ein Skript referenziert. Wenn der Benutzer den `submit`-Knopf im Browser drückt, wird die Ausführung des Skriptes auf dem Web-Server veranlaßt. Das Skript (siehe 3) zur Auswertung des Formulars (siehe 1) bekommt die Inhalte der Formularvariablen als Parameter vom Web-Server vermittelt. Es führt die Datenbankanweisungen mittels `DB::perl` aus und erzeugt als Ergebnis eine HTML-Seite, die vom Web-Server an den Browser als Ergebnis zurückgesandt wird.

Der Einsatz von Perl erlaubt eine flexible Programmierung — speziell für die Arbeit mit CGI, HTML und Formularen sind viele vorgefertigte Module frei verfügbar. Transaktionen sind zwar nachbildbar aber schwierig umzusetzen, wenn sie über mehrere CGI-Programmaufrufe verteilt sind, was generell für die CGI-Programmierung gilt. Das fehlende Typsystem in Perl macht

Beispiel 2 `DB::perl`-Formular

```
<HTML>
<HEAD><TITLE>perl DBI Beispiel Formular</TITLE></HEAD>
<BODY><H1>Eingabe-Formular</H1>
<FORM ACTION="/~schlegel/wwwdb/updatebsp.cgi" METHOD=get>
  Name <INPUT NAME=name TYPE=text SIZE=20><BR>
  Anzahl <INPUT NAME=quant TYPE=text SIZE=2><BR>
  <CENTER>
    <INPUT TYPE=submit VALUE="Einf&uuml;gen">
  </CENTER>
</FORM>
</BODY>
</HTML>
```

Anwendungen, die mehr als nur Operationen über Zeichenketten nutzen wollen, aufwendig. Perl als Sprache ist sehr komplex, da sie Konzepte aus einer Vielzahl von Skriptsprachen und UNIX-Werkzeugen vereinigt (Shell, sed, AWK).

4.3 Server API's und FastCGI

Das Common Gateway Interface (CGI) ist eine einfache, robuste Schnittstelle für die Abarbeitung von Anwendungsprogrammen auf dem Web-Server. CGI ist sprachunabhängig, Server-unabhängig und ein offener Standard. Die Nachteile bestehen darin, daß für jeden einzelnen HTTP-Request ein neuer Prozeß erzeugt wird, und es nur einen Antwortmodus gibt.

Um diese Nachteile von CGI zu überwinden, haben verschiedene Hersteller eigene Server-APIs implementiert, wie NSAPI von Netscape und ISAPI von Microsoft. Auch der aus der Open source stammende und meisteingesetzte Web-Server Apache hat einen eigenen Erweiterungsmechanismus.

Leider haben diese Server-APIs selbst wiederum eine Reihe von Nachteilen. Sie sind komplex und mit einer entsprechenden Lernkurve verbunden. Meist ist eine Sprachbindung gegeben, oftmals an C, C++ oder an eine spezielle Sprache (Pike), wie im Fall von Roxen. Die Server-Architektur wirkt sich auf die API bezüglich des Prozeßmodells aus, *Multi threading* erlaubt zwar eine schnelle Prozeßerzeugung und effizienten Datenaustausch zwischen Server und Server-Programm (*Servlet*) über gemeinsamen Speicher, ist aber nicht sehr robust und eine Prozeßisolation fehlt. Generell sind diese Server-APIs an die Herstellervorgaben gebunden.

Eine Kombination der Unabhängigkeit von CGI mit den Vorteilen von Server-API's stellt FastCGI [Fas] dar. FastCGI-Prozesse können mehrere Requests behandeln, der Aufwand der Prozeßerzeugung pro HTTP-Anforderung entfällt. Die *FastCGI application library* erlaubt eine einfache Migration von CGI-Applikationen (C, Java, Perl, Python) und ist abwärtskompatibel. Genau-

Beispiel 3 `DB::perl`-Skript

```
#!/opt/bin/perl5
use DBI; $ENV{'II_SYSTEM'}='/users/db10';
use CGI param;

print 'Content-type: text/html\n\n';
print '<HTML>';
print '<HEAD><TITLE>Updates mit DBI perl</TITLE>';
print '</HEAD>';
print '<BODY><H2>Updates mit <TT>DBI perl</TT></H2>';
# QUERY_STRING analysieren
$qs = new CGI;
$name = $qs->param('name');
$quant = $qs->param('gender');
# Verbindung zur Datenbank aufbauen
$dbh = DBI->connect( 'testdb', '', '', 'Ingres' );
# SQL-Anweisung ausfuehren
$rv = $dbh->do("insert into test values ('$name', $gender)");
# explizites commit oder rollback, Verbindung trennen
$dbh->commit;
$dbh->disconnect;
print '<B>fertig!</B></BODY></HTML>'; exit 0;
```

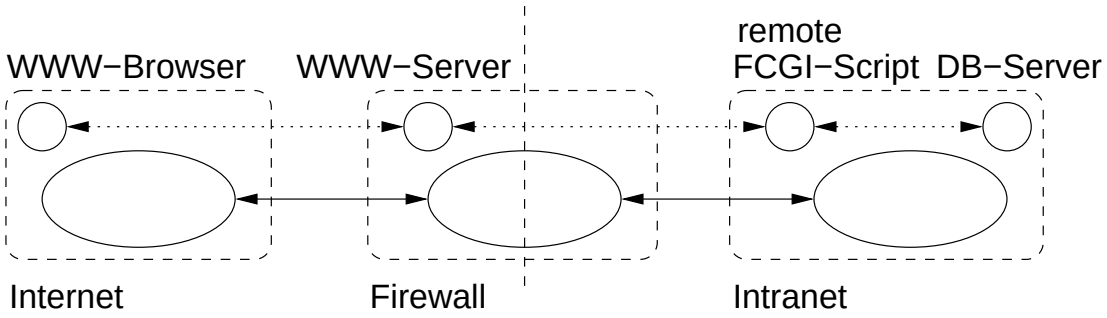
so wie bei CGI ist Programmiersprachunabhängigkeit, Prozeßisolation und eine gewisse Herstellerunabhängigkeit[1]gegeben. FastCGI ist unabhängig von der Server-Architektur (non-threaded, threaded) und unterstützt verteile Anwendungen — FastCGI Applikationen können entfernt vom Web-Server ausgeführt werden.

Remote FastCGI. Der Server benutzt lokal eine Stream-Pipe (Duplexverbindung auf Basis Stream socket) und entfernt eine TCP-Verbindung für die Kommunikation mit dem FastCGI-Prozeß. Dies erlaubt zum einen eine sehr gute Lastverteilung, FastCGI-Prozesse können auf unbelastete Rechner verteilt werden, und eine hervorragende Integration in Firewall-Umgebungen (siehe Abbildung 3).

FastCGI Anwendungsmodi. Gegenüber CGI können FastCGI-Anwendungen in drei Modi arbeiten, nicht wie CGI-Anwendungen nur im Request/Response-Mode:

[1]FastCGI wird von Open Market's server, Apache, NCSA, Microsoft, Netscape, Roxen und weiteren Web-Servern unterstützt

Abbildung 3 FastCGI und Firewall



- *Responder*, arbeitet wie in CGI,
- *Filter*, die Anwendung filtert die angeforderte Web-Seite, bevor sie an den Client geht. Dies erlaubt eine Formatumwandlung On-the-fly, Realisierung von dynamischen Dokumenten, z. B. Embedded SQL, Einblendung von Werbung und eine Template-Verarbeitung.
- *Authorizer*, die Anwendung entscheidet, ob ein Request ausgeführt wird oder nicht. Damit wird eine Nutzer- und Paßwort-Auswertung über spezielle Datenbasen, die Umsetzung von komplexen Zugriffsstrategien, z. B. nach Tageszeit, Lastsituation, und ein dynamisches Umleiten von Anforderungen (*Redirect*) möglich.

Ein rudimentäres FastCGI-Programm in C bzw. wird in den Beispielen 4 und 5 gezeigt: in einer `while`-Schleife wird auf eingehende HTTP-Requests mit einem sehr einfachen HTML-Dokument geantwortet.

Die generelle Unterschiede gegenüber normaler CGI-Bearbeitung bestehen bei FastCGI darin, daß Prozesse nicht nach Bearbeitung eines Requests enden, sondern auf neue Requests warten. Der Datenaustausch mit dem Server erfolgt nicht über UNIX-Environment und Pipes, sondern über eine Duplex-Verbindung.

4.4 Zustände speichern mit Cookies

Das HTTP-Protokoll hat das Problem, zustandslos zu sein. Anfragen liefern einzelne, unabhängige Dokumente. Damit können per HTTP keine Transaktionen über mehrere Stufen (HTML-Dokumente, CGI-Aufrufe) realisiert werden. Dies macht es notwendig, eine der folgenden Techniken zu benutzen:

1. Es werden zusätzliche Daten zur Nachbildung einer Sitzungsschicht gesammelt. Diese Daten werden per HTML-Variable als Zustand an spätere Seiten weitergeben. Durch ein abschließendes HTML-Formular wird die Gesamttransaktion ausgelöst.

Beispiel 4 FastCGI-Programm in C

```
#include <fcgi_stdio.h>

void
main()
{
   int count = 0;

   while (FCGI_Accept() >= 0) {
      printf("Content-type: text/plain\n");
      printf("\n");
      printf("Hello World!\n");
      printf("Request # %d!\n", ++count);
   }
   exit(0);
}
```

2. Es wird ein dedizierter Transaktionsdämon eingesetzt. Eine zentrale Instanz (Dämon) vergibt Transaktionsnummern und verwaltet den damit verbundenen Zustand. Durch spezielle Kodierung über versteckte Variablen in HTML-Dokumenten oder in der URL bzw. mittels Cookies wird diese Transaktionsnummer mitgeführt. Probleme bei dieser Lösung bereiten Time-outs und der Abbruch der Verbindung.

3. Die Anwendung setzt spezielle Server-Erweiterungen wie FastCGI ein.

Neben der Technik, Zustandsinformationen in der URL bzw. durch versteckte (*hidden*) Formularvariablen zu kodieren, sind *Cookies* eine weit verbreitete Möglichkeit, dieses unabhängig von HTML-Seiten zu realisieren. Dabei speichert ein Zustandsobjekt, der *Cookie*, mittels des Web-Browser auf dem Client Informationen.

Das Setzen eines Cookies erfolgt durch einen speziellen HTTP-Header, der typischerweise von einem CGI-Skript oder durch den Web-Server selbst erzeugt wird. Dieser Server-Header hat folgende Syntax:

```
Set-Cookie: name=value; expires=date;
     path=path; domain=domain-name; secure
```

Mit `name=`*`value`*`;` erfolgt das Setzen der Variablen (Cookie), Sonderzeichen werden wie in URLs bei der HTTP-GET-Methode kodiert. `expires=`*`date`*`;` gibt ein Gültigkeitsdatum an, d.h., wann der Cookie auf Client-Seite gelöscht werden kann. Eine Einschränkung der Gültigkeit kann durch `path=`*`path`*`;` erfolgen: stimmt die Pfad-Komponente der URL mit *path* überein,

Beispiel 5 FastCGI-Programm in Python

```
import fcgiapp

while 1:
   try:
      input, output, error, environ = fcgiapp.Accept()
      data = input.read()
      data = process(data, environ)
      output.write(data)
   except:
      pass
```

wird der Cookie vom Client zurückgesandt, wobei die ersten `strlen(`*`path`*`)` Bytes verglichen werden. Mittels `domain=`*`domain-name`*`;` dürfen nur Server aus der Domain *domain-name* den Cookie setzen. Es wird der letzte Teil des FQDN (Full qualified domain name) verglichen. Durch Angabe von `secure` wird spezifiziert, daß der Cookie nur via HTTPS gesandt werden darf.

Immer wenn die Bedingungen (Gültigkeitsdatum, Pfad- und Domain-Komponente) erfüllt sind, sendet der Client (Web-Browser) bei den HTTP-Request folgenden zusätzlichen Client-Header mit, der den Cookie-Wert enthält.

```
Cookie: name=value; name=value
```

In CGI-Programmen kann man dann die gesetzte Cookies in den entsprechenden Umgebungsvariablen `COOKIE_`*`name`* und als <Name, Wert>-Tupel in der Variablen `HTTP_COOKIES` abfragen. Zusätzlich enthält die Variable `COOKIES` die Liste aller Cookie-Namen. Das Löschen eines Cookies kann durch Angabe eines `expire`-Datum, das in der Vergangenheit liegt, erfolgen.

Das Beispiel 6 zeigt das Setzen und Abfragen von Cookies durch ein kleines AWK-Programm.

Vorteilhaft bei der Nutzung von Cookies sind die Entkopplung von den eigentlichen HTML-Dokumenten und die Einflußnahme auf die Gültigkeit bezüglich Domain-Namen, Pfad-Komponenten und Lebenszeit. Leider gibt es die berechtigte Sorge vieler Nutzer, daß Cookies mißbräuchlich[2]eingesetzt werden, so daß die Cookie-Nutzung im Browser oftmals abgeschaltet wird.

4.5 JDBC: Java Database Connectivity

Java als Sprache für verteilte System und das Web entkoppelt vom Ausführungsort und der Ausführungsplattform, JDBC [Deh98, SS00] soll vom Datenbanksystem entkoppeln. Intension war

[2]Man kann Cookies auch dazu benutzen, um Informationen über das Nutzerverhalten zu erlangen.

Beispiel 6 Cookie-Programm in AWK

```
#!/usr/bin/nawk -f
BEGIN {
     setcookie("muppets_cookie_munster", "custard pie",
               "/show/muppets/");
     exit(0);
}
function printcookies() {
     for (e in ENVIRON) {
          if (e ~ /COOKIE/) {
               printf("<PRE>%s=\"%s\"</PRE>\n", e, ENVIRON[e]);
          }
     }
}
function setcookie(name, val, path) {
     printf("Content-Type: text/html\n");
     printf("Set-Cookie: %s=%s; path=%s\n\n", name, val, path);
     printf("<HTML>\n");
     printf("<TITLE>%s</TITLE>\n", name);
     printf("<BODY>\n");
     printf("<H1>A cookie (%s) for the munster!</H1>\n", val);
     printf("<P>Cookie\n...\n");
     printcookies();
     printf("</BODY>\n</HTML>\n");
}
```

es, eine „einfache“, leicht zu benutzende Schnittstelle zu schaffen, die als „Low-level“ API zur Realisierung weiterer „High-level“ APIs geeignet ist. JDBC steht für Java DataBase Connectivity und ist eine Datenbankschnittstelle für Java von JavaSoft, die als Vorbild SQL/CLI, ODBC und X/Open/CLI hat. JDBC ist als Schnittstelle zu Relationalen DBMS gedacht, für den Zugriff auf Objekt-orientierte oder Objekt-relationale DBMS gibt es ODMG-Java-Bindings bzw. SQLJ in SQL-99 [ME00].

JDBC erlaubt die Unterstützung von vorübersetzten Anfragen und von Datenbankprozeduren (*Stored procedures*). Dynamische Datenbankzugriffe werden genauso unterstützt wie der Zugriff auf Metadaten. Die Entkopplung von Anwendung und dem konkret benötigtem Laufzeitsystem wird wie bei ODBC durch dynamische Treiberauswahl und dynamisches Laden des Treibers ermöglicht.

In Abbildung 4 werden die möglichen Realisierungsformen von JDBC-Treibern gegenübergestellt. Nach Anzahl der beteiligten Komponeten unterscheidet man

- *One-tier* (Typ 4), der JDBC-Treiber ist vollkommen in Java realisiert,
- *Two-tier* (Typ 1 und 2), wenn der JDBC-Treiber lokale Laufzeit-Bibliotheken benutzt, und
- *Three-tier* (Typ 3), wenn JDBC-Treiber indirekt über einen speziellen Server (*Gateway*) auf das DBMS zugreift.

Ein Beispiel für einen *two-tier*-Treiber wäre die JDBC-ODBC-Brücke aus dem JDK in Verbindung mit einem ODBC-Treiber. Eine *three-tier* Lösung wäre eine Middleware-Anwendung in Java in Verbindung mit einem Datenbanksystem-unabhängigem Protokoll wie RMI, die JDBC-Aufrufe auf dem Server in ein Datenbanksystem-spezifisches Protokoll umsetzt.

Ausschlaggebend für die SQL-Konformität ist neben der Abbildung von SQL-Datentypen auf entsprechende Java-Datentypen, die Bereitstellung von Eigenschaften des angebundenen Datenbanksystems über Metadaten und die Einhaltung des ISO-Standards SQL-92 Entry Level. Teilweise werden Java-Implementierungen für spezielle SQL-Typen (DATE, TIME) durch die API selbst bereitgestellt.

Die JDBC API ist als Menge von abstrakten Java Schnittstellen für das Erzeugen von Verbindungen zu einer bestimmten Datenbank, die Ausführung von SQL-Anweisungen und die Verarbeitung der Ergebnisdaten realisiert. Das Beispiel in Abbildung 7 zeigt die wesentlichen Elemente eines JDBC-Programmes, das Laden des Treibers und den Verbindungsaufbau mit `getConnection`, dem Erzeugen eines Containers für die Ausführung von SQL-Anweisungen `createStatement`, dem Ausführen von Anweisungen und dem satzweisen Verarbeiten von Ergebnistupeln mit `executeQuery`, `next` und `getXXX`.

Die Adressierung von Treibern erfolgt analog zum bekannten URL-Mechanismus. Mit folgender Adresse kann die Datenbank `testdb` des Systems `mSQL` auf dem Server `baltic.de` angesprochen werden:

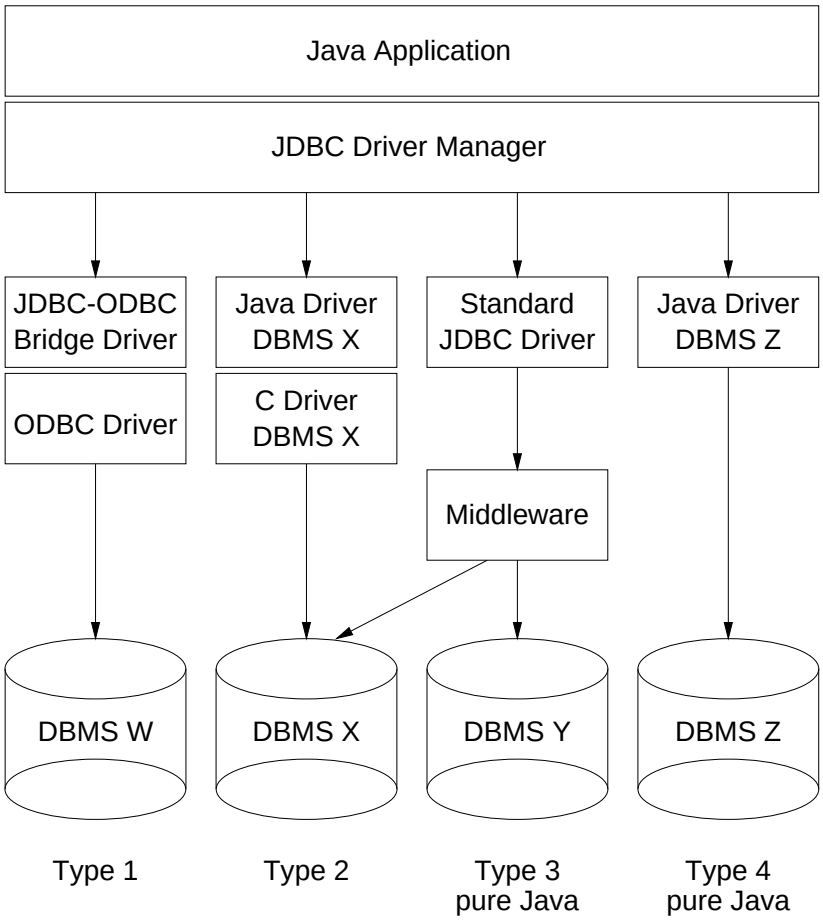
```
jdbc:mSQL://baltic.de/testdb
```

Mit JDBC sind genauso wie mit Java allgemein folgende Anwendungsszenarios denkbar:

- *Trusted code*: eine lokale Java-Anwendung hat uneingschränkten Zugriff auf alle Ressourcen.
- *Trusted applets*: aufgrund einer Signatur mit kryptographischen Schlüssel oder der Nutzerentscheidung anhand der Quelle erlaubt Zugriffe auf das lokales Dateisystem.
- *Untrusted code*: Java applets aus dem Internet gestatten nur eine Verbindung zu dem Rechner, von dem das Applet geladen wurde.

Wesentlich für die schnelle Verbreitung von JDBC ist die Kompatibilität zu ODBC-Anwendungen durch eine frei verfügbare JDBC-ODBC-Brücke: Applikationen können mit der JDBC-API erstellt werden, die API-Rufe werden durch JDBC-ODBC-Brücke gefiltert, und es erfolgt eine Konvertierung in ODBC-Funktionsaufrufe, die an DBMS geschickt werden.

Abbildung 4 JDBC — Treiberarchitektur



Die JDBC API liegt mittlerweile in der Version 2.0 vor [WFC+99, Deh98] und ist Bestandteil des JDK 1.2. Sie verfügt über ein verbessertes Cursor-Konzept und bessere Unterstützung von Updates, eine SQL-99 Unterstützung, Konzepte für verteilte Transaktionen und skalierbare Anwendungen (z. B. *Connection pooling*). Nachteilig bei JDBC bleibt die rein relationale Anbindung und die Koexistenz von SQL- und Java-Datentypen.

4.6 SQLJ: Embedded SQL für Java

SQLJ [ME00, SS00] ist ein Vorschlag verschiedener Hersteller (Oracle, IBM, Sybase, Tandem, JavaSoft, Microsoft, Informix), der bei der ANSI/ISO eingereicht ist, und in den SQL-Standard einfließen soll. Analog zur Einbettung von SQL in C, C++, Cobol und Ada bettet SQLJ SQL-Anweisungen mit einem speziellen Präfixsymbol (`#sql{}`) in Java ein. Dieser Vorschlag besteht aber nicht nur aus der Einbettungsprogrammierung in Java, sondern umfaßt folgende drei

Beispiel 7 Einfaches JDBC-Programm

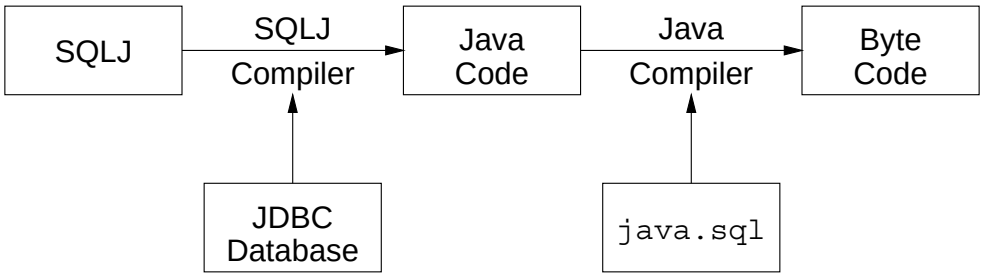
```
Connection con =
      DriverManager.getConnection("jdbc:mSQL://baltic.de/testdb");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT NAME, QUANTITY FROM PERSON");

while (rs.next()) {
      String name = rs.getString("NAME");
      float price = rs.getFloat("QUANTITY");
      ...
}
```

Anteile:

- *SQLJ Embedded SQL*: Einbettung von SQL in Java, vorübersetzte SQL-Anweisungen werden auf JDBC-Rufe abgebildet.
- *SQLJ Stored Procedures und UDFs*: statische Java-Methoden können zur Programmierung von SQL Stored procedures und *User defined functions* (UDFs) benutzt werden.
- *SQLJ Data Types*: reine Java-Klassen können in SQL als abstrakte Datentypen benutzt werden. Sie stellen damit eine Alternative zu den SQL-99 *Abstract data types* (ADTs) dar.

Abbildung 5 SQLJ Übersetzung



Der Vorteil der Einbettungsprogrammierung (siehe Abbildung 5) liegt in der Aufdeckung von Fehlern zur Übersetzungszeit und führt zu robusteren Anwendungsprogrammen im Vergleich zu JDBC. Die eigentlichen Vorteile von SQLJ sind erstens, die wesentlich bessere Integration von Java und Objekt-relationalen Datenbanken, zweitens, Java-Klassen als Datenbanktypen benutzen und drittens, SQL-99 ADTs in Java implementieren zu können. Das Beispiel 9 zeigt die Einbettung von SQL in Java.

Weiterhin erfolgt die Abbildung von SQL-Datentypen auf entsprechende Java-Klassen pro Verbindung automatisch (siehe die Benutzung von `getObject` im Beispiel 8).

Beispiel 8 SQLJ Benutzung von `getObject`

```
Statement stmt;
ResultSet rs = stmt.executeQuery("SELECT CUSTOMER FROM ACCOUNTS");
rs.next();
Customer cust = (Customer)rs.getObject(1);
```

SQL Embedded in Java. SQLJ stellt einen einfachen Mechanismus zur Einbettung von SQL-Anweisungen in Java dar. Ziel ist es, verschiedene Komponenten durch unterschiedliche Systeme (SQLJ-Übersetzer) oder Hersteller (JDBC-Treiber) erzeugt, binärkompatibel zu halten und damit ein gutes Integrationswerkzeug zu schaffen und Portabilität über verschiedene Datenbanksystem zu erhalten. Der Vorteil der Vorübersetzung (siehe Abbildung 5) besteht in:

- der Syntax- und Typüberprüfung zur Übersetzungszeit,
- streng typisierten Iteratoren (Cursor-Variablen) und
- der Offline-Optimierung mit entsprechendem Performance-Gewinn.

SQLJ-Klauseln werden mit `#sql` eingeleitet und enden mit einem Semikolon `';'`, Hostvariablen werden durch einen Doppelpunkt `':'` ausgezeichnet, und der eigentliche SQL-Anweisungsteil wird in geschweifte Klammern { } eingeschlossen. SQLJ-Anweisungen können mit einem Verbindungskontext, der die zu verwendenden Tabellen, Sichent und Rechte festlegt, assoziiert werden (im Beispiel 9 ist der Kontext `dept`).

Beispiel 9 SQLJ Einbettung mit Kontext

```
#sql context Department;
Department dept = newDepartment("jdbc:msql://baltic.de/:testdb");
int n;
String name;
Address addr;
...
#sql [dept] { insert into person values (:num, :name, :addr)};
```

Erweiterbarkeit wird in SQLJ durch Plug-ins für SQL-Syntax- und Semantik-Checker, die herstellerspezifisch über das *SQLChecker framework* bereitgestellt werden können, und der Standardanbindung von SQLJ an beliebige JDBC-Treiber garantiert.

Java-Methoden als SQL-Prozeduren. Statische Java-Methoden können zur Implementierung von SQL Stored procedures und nuzterdefinierten Funktionen (UDFs, *User defined functions*)

eingesetzt werden. Auf Datenbankebene erlangt man damit Zugriff auf existierende, umfangreiche Java-Bibliotheken. Mit Java steht dem Datenbanksystem eine prozedurale, skriptorientierte Sprache zur Verfügung, die portabel über DBMS-Grenzen ist und gut im Middleware-Bereich zur Systemintegration herangezogen werden kann. Die Java-Methoden selbst können wiederum JDBC-Rufe und/oder SQLJ für den Datenbankzugriff verwenden.

Java-Klassen als SQL-Datentypen. Durch SQLJ wird die Verwendung von Java-Klassen zur Implementierung von Datentypen möglich, die für die Definition von Attributen in SQL-Tabellen und Sichten verwandt werden können. Diese neuen Datentypen können genauso wie normale SQL-Datentypen benutzt werden, z. B. auch für Parameter von SQL-Prozeduren.

Der Vorteil für SQL besteht in einem Typerweiterungsmechanismus und in der Alternative bzw. Ergänzung der abtrakten Datentypen (ADTs bzw. UDTs, *User defined types*) in SQL-99. Aus Java-Sicht bekommt man eine direkte Unterstützung von Java-Objekten in SQL-Datenbanken. Damit ist es nicht mehr notwendig, Java-Objekte umständlich auf skalare SQL-Datentypen oder BLOBs (*Binary large objects*) in der Anwendung abbilden zu müssen.

Beispiel 10 SQLJ-Klasse `Address`

```
public class Address
        implements java.io.Serializable {
    public String street;
    public String zip;
    public static int recommended_width = 25;
    // A default constructor
    public Address() {
        street = "Unknown";
        zip = "None";
    }
    // A constructor with parameters
    public Address(String S, String Z) {
        street = S;
        zip = Z;
    }
    // A method to return a string representation
    // of the full address
    public String toString() {
        return "Street=" + street + " ZIP=" + zip;
    }
};
```

5 Datenbanksysteme und Standard-Techniken

Beispielhaft sollen die Web-Lösungen der Datenbanksysteme Oracle, Informix und O_2 beschrieben werden. In allen Fällen übernimmt das Datenbanksystem die volle Kontrolle über die HTML-Seiten. Speziell integriert Oracle den Web-Server, stellt Informix ein Autorenwerkzeug bereit, und O_2Web bildet Objektstrukturen elegant auf Hyperlink-Strukturen ab.

5.1 Oracle8*i*

Als Produkt setzt Oracle8*i* auf den Einsatz von Java und PL/SQL für die Realisierung von Web-Anwendungen [PM99] und entsprechender Middleware-Techniken. Speziell steht mit Oracle WebDB eine Web-basierte Nutzerschnittstelle zur Erstellung und Verwaltung von kompletten Web-Anwendungen zur Verfügung. Im weiteren werden die Komponenten von Oracle8*i* beispielhaft aufgezeigt.

- Der *integrierte Web-Server* von Oracle erlaubt die Abarbeitung von Java Servlets direkt im Datenbanksystem.
- Für die Realisierung von verteilten Transaktionen kann auf einen *Enterprise JavaBeans Server* zugegriffen werden.
- Alternativ hat man die Möglichkeit, einen *Object request broker* von VisiBroker einzubinden, wenn mit CORBA-gestützten Anwendungen kommuniziert werden sollen.
- *SQLJ* und *Java Stored procedures* erlauben die Integration von SQL, PL/SQL und Java-Code.
- Das *Internet File System* (*i*FS) erlaubt den Daten- und Dokumentenzugriff über eine hierarchische Dateisystemsicht mittels Standard-Protokollen (SMB, HTTP, FTP, SMTP, POP, IMAP).
- *Oracle ConText* stellt einen abstrakten Datentyp zur Speicherung von HTML-, XML- und Volltextdokumenten im Datenbanksystem mit zugehörigen Information Retrieval-Operationen bereit.

5.2 Informix Dynamic Server

Bei Informix Dynamic Server (IDS) [Inf] handelt es sich um ein erweiterbares objekt-relationales Datenbanksystem. Als objekt-relationale Konzepte werden ADTs, Vererbung, Typkonstruktoren und die *DataBlade*-Technik bereitgestellt.

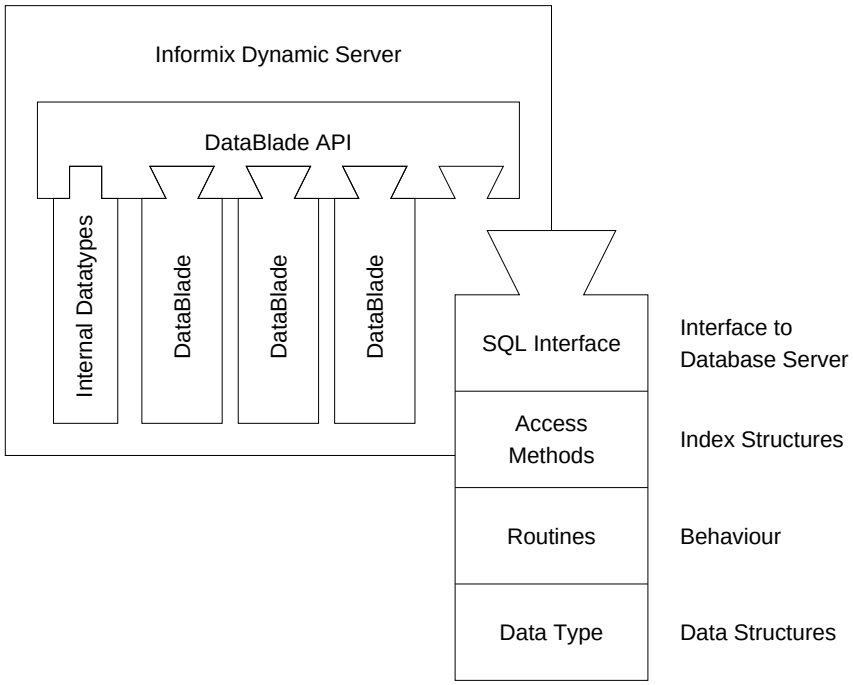
Die DataBlade-Technologie (siehe Abbildung 6) erlaubt die Integration beliebiger Datentypen mit zugehörigen Zugriffpfaden und Optimierungstechniken in das Datenbanksystem. Prinzipiell

können über diese Technik beliebige Fremdsystem, z. B. ein Volltextdatenbanksystem, eingebunden werden. Es stehen umfangreiche Objektbibliotheken und ein Developer's Kit für die DataBlade-Entwicklung zur Verfügung.

Gegenüber anderen objekt-relationalen Systemen (z. B. IBM DB2) besitzt Informix eine Vielzahl von nutzbaren DataBlades, die teilweise Fremdprodukte sind. Hier eine Auswahl: Es gibt ein 2D/3D DataBlade, das 18 Datentypen wie Punkte, Vektor, Flächen durch Polygone und ca. 1000 Funktionen wie Länge, Durchschnitt, Enthalten, Überlappen, Mittelpunkt bereitstellt. Mit R-Bäumen existieren Zugriffsstrukturen, die sehr gut für raumbezogene Anfragen geeignet sind. Neben verschiedenen Volltext-DataBlades gibt es ein Image-DataBlade, das 50 Formate mit Anfrage-Methoden (`GetColor`, `GetHeight`, `Scale`), und Update-Methoden für Skalierung und *Cut and Paste* implementiert.

Mit *Visual Information Retrieval* von Virage sind bildinhaltsorientierte Anfragen nach z. B. Farbe und Form möglich, die Resultate werden gewichtet. Das Informix Web-DataBlade gestattet es, HTML im Datenbanksystem zu verwalten. Z.Z. ist leider keine Anbindung über CORBA möglich, wodurch eine Transaktionskontrolle bei Fremdprodukten erschwert wird.

Abbildung 6 Informix DataBlade Architektur



Web DataBlade. Als Web-Server kann bei Informix ein beliebiger Server eingesetzt werden. Die Informix Anbindungstechnik besteht aus folgenden drei Komponenten:

1. Der *WebDriver* bildet mit der CGI- oder NSAPI-Schnittstelle zum WebServer die Web-Zugriffe auf die Datenbank ab.

2. Ein *WebDaemon* fordert dabei HTML-Seiten aus der Datenbank an.

3. *WebExplore* ist als Methode im Web-DataBlade der Datenbank realisiert, diese setzt dynamisch HTML-Seiten zusammen aus statischen HTML-Anteilen und Informix-Tag-Erweiterungen zusehen in Abbildung 11. Diese Erweiterungen können SQL-Anfragen und Anfrageergebnisse spezifizieren.

Die Informix-HTML-Erweiterungen sind SGML-kompatibel und können damit auch von Fremdsoftware bearbeitet werden. Neben der Verwaltung von HTML-Dokumenten können spezielle Tags als direkte Datenbankattribute abgelegt und beliebige Multimediatypen normalisiert in einer erweiterbaren Tabellenhierarchie gespeichert werden. Außerdem stellt Informix mit dem *Application Page Builder* (APB) ein eigenes Autorenwerkzeug auf der Basis von HTML 4.0 bereit.

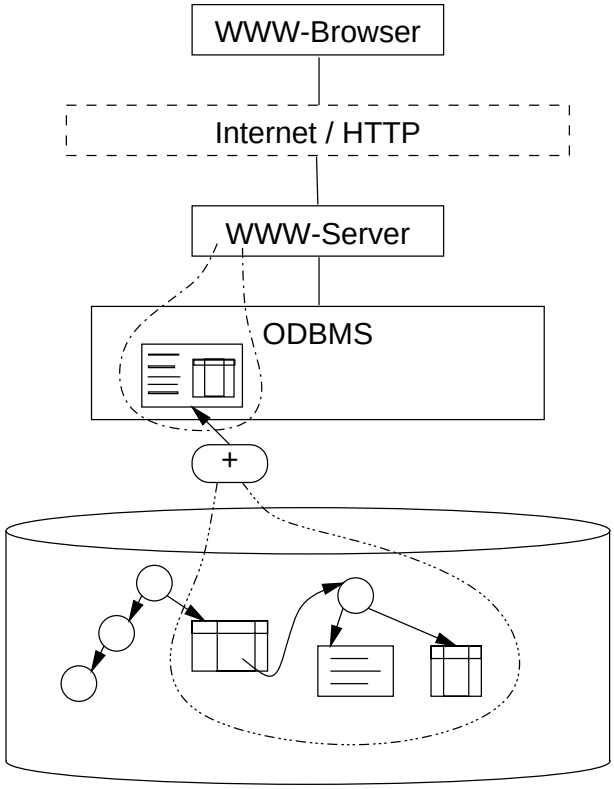
Beispiel 11 Web-DataBlade Application Page Builde-Fragment

```
<HTML>
<HEAD><TITLE></TITLE></HEAD>
<BODY>
<H1></H1>
<!-- Parametervariable definieren -->
<?MIVAR NAME="pname" DEWFAULT="J&uuml;rgen">$pname<?/MIVAR>
<TABLE>
  <TR><TH></TH><TH></TH></TR>
  <!-- Anfrage und tupelweise Verarbeitung -->
  <?MISQL QUERY="select name, quant
                 from test
                 where name LIKE '$pname';">
    <!-- Zeilen fuer Tabelle erzeugen -->
    <TR><TD>$1</TD><TD>$2</TD></TR>
  <?/MISQL>
</TABLE>
</BODY>
</HTML>
```

5.3 O_2Web: Links in Datenbanken

Abbildung 7 skizziert die Speicherung kompletter WWW-Strukturen in einem Objektdatenbanksystem. [3] Die Objektstrukturen des ODBMS werden mit geeigneten Operationen (im Bild vereinfacht als + notiert) durch das ODBMS zu vernetzten WWW-Dokumenten zusammengesetzt, wobei auch die Links in Form von URLs generiert werden.

Abbildung 7 O_2Web — WWW-Strukturen im Datenbanksystem



Die Datenbankvorteile gelten nun auch für Link-Strukturen, so sind entsprechende Konsistenzprüfungen möglich. Die gesamte darzustellende Information wird nun voll datenbankgestützt verwaltet. Bei heutigen Systemen ergibt sich der Nachteil, daß HTML-Informationen normalisiert gespeichert werden müssen, also Textanteile getrennt von Links gespeichert werden. Auch müssen Änderungen am HTML-Standard im DBMS berücksichtigt werden, so daß die Datenunabhängigkeit in der Regel nicht mehr gegeben ist.

Da die Anwendungsdaten nun verzeigerte Datensätze verschiedenster Formate sind, kann diese Lösung besonders gut von Objektdatenbanken unterstützt werden. Ein Beispiel für diesen Ansatz ist das Produkt O_2Web von O_2.

[3] Nach der Übernahme der Produktlinie durch Informix ist z. Z. unklar, was aus O_2 und darauf aufbauenden Produkten wird.

6 XML-basierte Techniken

Das Akronym XML steht für Extensible Markup Language. Dahinter verbirgt sich ein durch das W3C entwickeltes Dokumenten-Austauschformat.

Die immer größere Vernetzung von Rechnersystemen und die immer stärker werdende Nutzung des WWW für den Austausch von Informationen erfordern Austauschformate, die leicht anwendbar, verständlich und durch Applikationen problemlos lesbar sind.

Aus diesem Grund ist das Interesse an XML sehr groß, die Anzahl der Anwendungen, die XML einsetzen, steigt sprunghaft an.

Zur Einführung soll anschließend in kurzer Form der Aufbau von XML-Dokumenten erläutert werden. Ein XML-Dokument besteht aus einer Anzahl von Elementen, diese haben jeweils ein Beginn- und Ende-Tag, dazwischen steht der Wert des Elementes. Die Tags liefern Informationen über die Bedeutung der konkreten Werte, stellen also in der Datenbankterminologie Strukturinformationen dar. Elemente können hierarchisch geschachtelt werden. Zu den einzelnen Elementen können Attribute ergänzt sein, die weitere Eigenschaften eines Elementes darstellen. Im folgenden kurzen XML-Beispiel 12 treten die Elemente `buch`, `titel`, und `autor` auf. Dem Element `buch` ist das Attribut `isbn` zugeordnet.

Beispiel 12 XML-Dokument

```
<buch isbn='0-201-342855'>
    <titel>The XML companion</titel>
    <autor>
         <name>Bradley</name>
         <vorname>Neil</vorname>
    </autor>
</buch>
```

Für eine Menge von Dokumenten kann eine DTD (Document Type Definition) spezifiziert sein, diese gibt an, welche Elemente und Attribute in den Dokumenten auftreten können und wie diese verschachtelt werden. Idealerweise wird die DTD vor Erstellen der Dokumente aufgestellt. Beim Parsen von Dokumenten wird deren Gültigkeit anhand der DTD überprüft. Eine zu Beispiel 12 gehörende DTD ist im Beispiel 13 zu finden.

In XML-Dokumenten treten Strukturinformationen und Daten in einem Dokument auf. Man nennt diese Eigenschaft „selbstbeschreibend“, für viele Anwendungen wird XML aufgrund dieser Eigenschaft gewählt. Die gegenwärtigen Hauptanwendungen von XML lassen sich wie folgt angeben:

- EDI (Electronic Data Interchange)
- Verbindung verschiedener Softwarekomponenten

Beispiel 13 Buch-Document Type Definition

```
<!ELEMENT buch (titel, autor+)>
<!ATTLIST isbn #REQUIRED>
<!ELEMENT titel (#PCDATA)>
<!ELEMENT autor (name,vorname)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT vorname (#PCDATA)>
```

- Anbindung von Datenbanken an Applikationen
- Import/Export von Datenbank-Inhalten

Darüber hinaus ist XML zur Darstellung semistrukturierter Daten geeignet. Semistrukturierte Daten können Dokumente mit irregulärer oder nur partiell definierter Struktur sein, der Typ der Datenelemente kann wechseln, die Grenze zwischen Schema und Daten ist fließend ([Abi97], [Bun97]). In DTDs lassen sich Alternativen spezifizieren, über diese werden semistrukturierte Inhalte also wechselnde Strukturen der zugehörigen Dokumente dargestellt.

6.1 Abbildung auf Datenbanken

XML wird häufig als Austauschformat angewendet, zur Speicherung der Informationen sollen oft andere Mechanismen eingesetzt werden. Die Speicherung von XML-Dokumenten in Datenbanken bietet sich dabei an, weil auf diese Weise bewährte Technologien zur Speicherung und Anfrage von Daten eingesetzt werden können.

Die existierenden Vorschläge zur Speicherung von XML-Dokumenten in Datenbanken lassen sich dabei in zwei Klassen unterteilen.

Die erste Klasse verwendet relationale Datenbanken und speichert sowohl die Informationen über die Struktur der Daten als auch die Daten selbst auf Instanzebene der Datenbanken. Vorschläge dazu existieren in [FK99], [Bra98], und [SYU99]. Für das obere Beispiel würde eine solche Speicherung (etwas vereinfacht) wie folgt aussehen:

Element	**ID**	**Parent**	**Child#**	**String**
buch	100	-		
titel	101	100	1	The XML ...
autor	102	100	2	
name	103	102	1	Bradley
vorname	104	102	2	Neil

Attribute, in diesem Fall das Attribut `isbn`, werden in einer ähnlichen Relation gespeichert, hier ist die Zuordnung zur Element-ID erforderlich.

Damit hat man eine Variante zur Speicherung, die nur relativ wenige Relationen mit fester Struktur benötigt. Die Abspeicherung von Informationen ist einfach, es lassen sich auch Dokumente mit irregulärer Struktur leicht speichern. Für die Speicherung ist keine DTD erforderlich.

Die Probleme liegen hier jedoch bei der Realisierung von Anfragen. Sobald mehr als ein Element oder Attribut in der Anfrage auftritt, sind aufwendige Joins über diesen Relationen erforderlich.

Die zweite Klasse von Speicherungsvarianten ist die „natürlichere Speicherung“ von XML-Dokumenten. Dabei werden die Tags der Dokumente auf Strukturinformationen von Datenbanken abgebildet. Aufgrund der hierachischen Schachtelung bieten sich objekt-orientierte Datenbanksysteme wie POET und ObjectStore und objekt-relationale Datenbanksysteme wie Informix und DB2 für diese Art der Darstellung an. Bei dieser Variante ist eine DTD erforderlich, damit der Datenbank-Entwurf erfolgen kann.

Für das obere Beispiel sieht die zugehörige Datenbank wie folgt aus:

Buch			
ISBN	**Titel**	**Autor**	
		Name	**Vorname**
0-201-34285	The XML ...	Bradley	Neil

Problematisch kann für einige Anwendungen sein, daß semistrukturierte Informationen in strukturierte Datenbanken abgebildet werden. Dabei können Daten mit sehr vielen Nullwerten, also schwach besetzte Datenbanken entstehen.

Der große Vorteil dieser Speicherung ist, daß man SQL-Anfragefunktionalität von Datenbanken einsetzen kann — es lassen sich z. B. Joins, Aggregatfunktionen usw. ausführen, auch die Anfrageoptimierung übernimmt das Datenbanksystem.

Diese Art der Speicherung wird in dem System POET eingesetzt, das im Abschnitt 7 beschrieben wird. Auf objekt-relationale Datenstrukturen und unter Verwendung von DTDs und Statistiken über XML-Dokumentkollektionen bildet der Ansatz in [KM00] ab.

Eine Methode zur Abbildung der XML-Strukturen auf die Strukturen einer relationalen Datenbank wurde in [STH+99] vorgestellt, wobei hier viel Aufwand für die Übersetzung der Hierarchien in flache Relationen erforderlich ist.

6.2 XML-Anfragesprachen

Im vorherigen Abschnitt wurde dargestellt, wie Informationen, die für den Austausch in XML dargestellt werden, in Datenbanken gespeichert werden. Dafür werden die Informationen meist über XML-Parser aus den Dokumenten ausgelesen.

Sollen XML-Dokumente gespeichert werden, so müssen Anfragen auf diesen Dokumenten realisierbar sein. Es existiert jedoch noch keine Empfehlung des W3C für eine Anfragesprache, eine eigene Arbeitsgruppe des W3C (XML Query Group) beschäftigt sich mit diesem Problem. Festgeschrieben sind bislang nur die Mindestanforderungen für Anfragesprachen [FMR]. Zu diesen

gehören die Selektion von Dokumenten aufgrund von Inhalt oder Struktur, die Extraktion von ausgewählten Subelementen und die Restrukturierung von Dokumenten (Erzeugung einer neuen Elementmenge für die Anfrageergebnisse).

Weiterhin existieren mehrere eingereichte Vorschläge für XML-Anfragesprachen, die bekanntesten davon werden im folgenden kurz dargestellt.

Tabelle 2 Vergleich von XML-Anfragesprachen

Kriterium	**XQL**	**XML-QL**	**XML Lorel**	**QUILT**
Restructuring	$\surd$	$\surd$	$\surd$	$\surd$
n Documents	–	$\surd$	$\surd$	$\surd$
Ordering	$\surd$	–	$\surd$	$\surd$
Links	–	$\surd$	ID/IDREF	$\surd$
Join	–	$\surd$	$\surd$	inner, left outer join
Types	–	–	integer,real,string,id,idref	?

XQL. XQL ist ein Sprachvorschlag von Robie (Texcel Inc, gegenwärtig, Software AG), Lapp (webMethods Inc) und Schach (Microsoft), in dem XSL-Ansätze aufgegriffen werden. Die typische Anfragen folgen der im Beispiel 14 angegebenen Syntax.

Beispiel 14 XQL-Anfrage

```
<xsl:for-each select = buch[title='The XML Companion']>
  <result>
    <buch>
      <titel>
        <xsl:value-of select=titel />
      </titel>
      <xsl:for-each select=buch/autor>
        <autor><xsl:value-of /></autor>
      </xsl:for-each>
    </buch>
  </result>
</xsl:for-each>
```

Das Grundmuster von XQL-Anfragen besteht aus dem Selektieren zu verarbeitender Elemente mit Pfadausdrücken. Innerhalb eines Iterators kann dann auf die entsprechenden Werte zugegriffen werden, diese umgeformt und andere, zusätzliche Elemente als Ergebnis ausgegeben werden.

Mit XQL hat man eine Möglichkeit, mit einer recht kompakten Syntax Anfragen an einzelne Dokumente zu stellen. Aufgrund dessen wird XQL gegenwärtig in den meisten Systemen als Anfragesprache eingesetzt. Anfragen, deren Ergebnisse aus mehreren Dokumenten zusammengesetzt werden müssen, lassen sich jedoch nicht realisieren.

XML-QL. Ein anderer Sprachvorschlag ist XML-QL, ein Projekt der AT&T Labs, entwickelt von Deutsch, Fernandez, Florescu, Levy und Suciu. Bei dieser Sprache wird ein Mechanismus eingesetzt, der dem Query-by-Example ähnlich ist. Das Grundmuster einer Anfrage sind WHERE-CONSTRUCT-Klauseln, wobei die WHERE-Klausel spezifiziert, wonach gesucht wird, die CONSTRUCT-Klausel angibt, wie das Ergebnis gebildet wird. Es ist möglich, Variablen einzusetzen. Über diese können Anfragen an mehrere Dokumente gestellt werden. Ein einfaches Anfragebeispiel findet sich in Beispiel 15.

Beispiel 15 XML-QL-Anfrage

```
WHERE
  <buch>\$b</>
    IN www.amazon.com,
  <titel>\$t</></> IN \$b,
  <autor>\$a</></> IN \$b
CONSTRUCT
  <result>
    <buch><titel>\$t</>
      WHERE <autor>\$a</> IN \$b
      CONSTRUCT <autor>\$a</>
    </>
  </>
```

Ein wesentliches Problem, das bei XML-QL-Anfragen auftreten kann, ist, daß die Ordnung nicht erhalten bleibt. Am Beispiel Literatur kann man sich dieses anhand der Kapitel eines Buches deutlich machen, warum die Reihenfolge von Bedeutung ist. Werden die Kapitel eines Buches vertauscht, weil deren Ordnung nicht gewährleistet werden kann, so geht damit Information verloren und die resultierenden Ergebnisse sind falsch.

Lorel. Ein universitärer Prototyp für eine XML-Anfragesprache ist *Lorel* aus dem Projekt *Lore* (Lightweight Object REpository) der Stanford University. Diese Sprache wurde ursprünglich für semistrukturierte Daten konzipiert und später an die Besonderheiten von XML angepaßt.

QUILT. QUILT ist ein neuer Sprachvorschlag von Chamberlin (IBM), Robie (Software AG) und Florescu (INRIA), der Teile aus XQL, XML-QL, SQL und Lorel verwendet und eine Syntax für eine XML-Anfragesprache auf dieser Basis vorschlägt [CRF00]. Damit sind Anfragen über mehreren Dokumenten möglich, die Ordnung der Dokumente wird berücksichtigt, Aggregatfunktionen aus SQL und Views werden einbezogen. QUILT-Anfragen folgen dem Grundmuster FOR-LET-WHERE-RETURN. Eine einfache QUILT-Anfrage ist in Beispiel 16 dargestellt.

Bislang ist diese Spache jedoch nur syntaktisch und anhand von Beispielen definiert.

Beispiel 16 QUILT-Anfrage

```
FOR $b IN document("books.xml")//buecher
WHERE $b/keyword="XML"
  AND $b/year="2000"
RETURN
<new-xml-book>
  $b/titel
  $b/autor
</new-xml-book>
```

Vergleich der Anfragesprachen. Ein Vergleich (Tabelle 2) dieser vier Anfragesprachen zeigt die Vor- und Nachteile der einzelnen Sprachentwürfe. XQL ist nur geeignet, um die Informationen eines Dokumentes anzufragen. Anfragen über mehreren Dokumenten lassen sich nicht stellen. XQL hat eine große Bedeutung gewonnen, weil diese Sprache in vielen Systemen angewendet wird. XML-QL ist für Anfragen über Dokumentkollektionen geeignet, für einige Anwendungen kann jedoch die fehlende Ordnung der Ergebnisse problematisch sein.

Der Prototyp Lorel und der Sprachvorschlag QUILT bieten die Anfrage-Funktionalität, die man für alle Arten von Anfragen benötigt. Bei Lorel handelt es sich jedoch um einen univeristären Prototypen. Der Sprachvorschlag QUILT ist bisher nur eine Syntaxdefinition.

Das Problemfeld XML-Anfragesprache ist also noch sehr offen, die Verabschiedung einer Empfehlung durch das W3C wird hier starke Auswirkungen auf die Systeme, die XML-Dokumenten verwalten, haben.

7 Systeme mit XML-basierten Techniken

XML entwickelt sich zu einem der bedeutendsten Dokumenten-Austauschformate. So ist es nicht verwunderlich, daß viele Systeme sich der Speicherung von XML-Dokumenten annehmen. Einige davon sollen in diesem Abschnitt kurz dargestellt werden.

7.1 ORDBMS mit XML-Erweiterung

IBM DB2 UDB. Die Firma IBM bietet für das objekt-relationale Datenbanksystem DB2 UDB einen XML-Extender an, mit dem die Speicherung von XML-Dokumenten im Datenbank-System DB2 unterstützt wird. Dabei wird über eine XML-Beschreibung DAD (Data Access Definition) angegeben, wie die Zuordnung von Elementen und Attributen des Dokumentes auf die Attribute der Datenbank erfolgen soll. Ein Ausschnitt aus einer DAD-Datei ist im Beispiel 17 zu sehen. Der XML-Extender speichert entsprechend dieser Spezifikation die Dokumente in der Datenbank ab.

Beispiel 17 DB2 DAD-Fragment

```
<table name="buecher">
  <column name="isbn"
          type="varchar"
          path="/Books/Info/"
          multi_occurrence="NO">
  </column>
</table>
```

Es ist auch möglich, aus DB2-Datenbanken XML-Dokumente zu generieren. Auch hier kann der Benutzer das Aussehen der Dokumenten beeinflussen, indem er die Zuordnung zwischen den Datenbank-Attributen und Elementen und Attributen eines XML-Dokumentes festlegt. Entsprechend dieser Beschreibung werden aus dem Datenbestand Dokumente generiert. Das Vorgehen ist hochflexibel, da der Benutzer die Art der Abbildung jeweils selbst festlegen kann. Der Preis dafür ist die Erstellung einer DAD-Datei, die dafür notwendig ist.

Ergänzend gibt es für die Ableitung von XML-Dokumente aus DB2-Datenbanken ein Tool XPERANTO, das die Dokumente mittels XML-QL-Anfragen automatisch aus den objekt-relationalen Daten unter Zuhilfenahme von Informationen aus dem Data Dictionary erzeugt.

Oracle8*i*. Zum Datenbanksystem Oracle8*i* ist ebenfalls ein Tool zur XML-Speicherung verfügbar, das Oracle XML Developer's Kit (XDK). Dieses bietet XML-Parser an, die den Benutzer unterstützen, XML-Dokumente zu parsen und in Oracle-Datenbanken abzubilden. Für die Speicherung von XML-Dokumenten werden drei Varianten angeboten. Für strukturierte Anteile eine Speicherung in Relationen und Attributen dieser Relationen, für unstrukturierte Anteile als CLOB oder BLOB und als drittes eine Kombination aus diesen.

Eine Anwendung von XML-Erweiterungen für Digitale Bibliotheken wird in [HMPT00] vorgestellt.

7.2 XML-Server

Tamino. Tamino ist der von der Software AG entwickelte XML-Server. Er dient der Speicherung, Verwaltung und Verarbeitung von Text-, Bild- und Audiodaten. Tamino speichert XML-Dokumente direkt im XML-Format, wobei Dokumente mit und ohne DTD gespeichert werden können. Anfragen werden in einem XQL-Dialekt realisiert. Als zusätzliche Funktionalität wird der Zugriff auf externe relationale Datenquellen unterstützt.

eXcelon. eXcelon Explorer ist ebenfalls ein System zu Speicherung und Anfrage von XML-Daten. eXcelon stellt gespeicherte XML-Dokumente intern als XML-Datentypen dar, dieser Da-

tentyp baut auf ObjectStore auf. Als Anfragesprache dient XQL, zusätzlich besteht eine graphische Unterstützung bei der Anfrageformulierung. Durch Verwendung von XSL-Stylesheets können die Dokumente oder Teile der Dokumente sichtbar gemacht werden.

POET Content Management Suite. POET hat mit der Content Management Suite (CMS) ein System bereitgestellt, das auf der Plattform POET Object Server 5.0 geeignet ist, komplexe Medientypen zu speichern und zu verwalten. Dazu gehören neben XML und SGML auch Graphik-, Audio- und Videoformate. Gespeicherte XML-Dokumente werden, ähnlich wie in Abschnitt 6.1 beschrieben, auf objekt-orientierte Datenbanken abgebildet. Hervorzuheben ist hier, daß eine Volltext-Suchfunktionalität bereitsteht, die für viele Dokumenttypen, die größere Textanteile enthalten, erforderlich ist.

Tabelle 3 Vergleich Anbindungstechniken

Kriterium	**CGI**	**FastCGI**	**Server-API**	**DBMS**[4]	**Applets**
Sitzungsebene	–	√	√	√	√
Transaktionen	SST	MST	MST	MST	MST
verteilte TA	–	–	–	–	√
Kommunikation	C/S–C/S	C/S–C/S	C/S–C/S	C/S	C/S,C/S–C/S
Protokoll	HTTP	HTTP	HTTP	HTTP	Anwendung
Authentifizierung	indirekt	indirekt	indirekt	√	√
Sicherheit	HTTPS/SSL	HTTPS/SSL	HTTPS/SSL	HTTPS/SSL	Anwendung
Prozeßisolation	√	√	–	(√)	–
Sprachbindung	–	–	√	√	√
Offener Standard	√	(√)	–	–	√
Portabilität	√	√	–	–	√

8 Zusammenfassung und Ausblick

Es gibt eine Fülle an Anbindungsmöglichkeiten von speziellen Realisierungen bis hin zu allgemeinen, portablen Lösungen pro Datenbanksystem. Die Tabelle 3 stellt die vorgestellten Techniken vergleichend gegenüber. Dabei wurde neben den bereits diskutierten Aspekten (Portabilität, Kommunikationsprotokoll, Sitzungsschicht, Transaktionen) zusätzlich auf Datenschutzmechanismen eingegangen.

Die Authentifizierung kann direkt vom Applet aus oder mit einem besitzenden Server, wie z. B. bei Oracle8*i*, erfolgen. Hier sind Datenbanknutzer gleichzeitig auch die Web-Nutzer. Wenn die Authentifizierung indirekt erfolgt, muß eine Abbildung von Web- auf Datenbanknutzern in der Anwendung vorgenommen werden. Sicherheit bedeutet, daß Daten unter Nutzung von Verschlüsselungstechniken mit HTTPS und *Secure socket layer* (SSL) übertragen werden. Andernfalls muß sie durch die Anwendung selbst realisiert werden, wie z. B. bei Java-Applets.

Probleme bei der Implementierung einer Sitzungsschicht und von Transaktionen über mehrere Anweisungen (*Multiple statement transactions* MST) gibt es bei CGI. Lösungen, z. B. mit Cookies, wurden angedeutet. Für verteilte Transaktionen gibt es sinnvolle Konzepte im Java-Umfeld.

Bei der Nutzung von Applets (one-, two-tier) oder beim Einsatz eines Datenbanksystems mit direkter HTTP-Schnittstelle kann die Kommunikation direkt (C–S) zwischen Web-Browser und Datenbanksystem erfolgen, alle anderen Techniken kommunizieren indirekt (C–S/C–S) über Datenbankprogramme auf dem Web-Server. Das Kommunikationprotokoll ist bei fast allen Anbindungsverfahren der HTTP-Standard. Nur Java-Applets benutzen datenbankspezifische Protokolle, Java-RMI oder CORBA-Techniken. Portabilität bietet Java und durch die Programmiersprachenunabhängigkeit auch CGI und FastCGI.

Neben der weitverbreiteten CGI-Programmierung in Skriptsprachen werden zunehmend in großen, verteilten Anwendungen Middleware-Lösungen eingesetzt. Sie basieren auf Java-Servlets und -Applets und nutzen Techniken und Entwicklungsumgebungen wie JavaBeans [Jav] und San Francisco [San].

XML wird als Austausch- und Repräsentationsformat immer häufiger verwendet. Die Standardisierung ist in diesem Bereich sehr dynamisch und kaum zu überschauen. Trotzdem existieren bereits viele Systeme, die XML einsetzen. HTML bzw. die XML-konforme Definition als XHTML wird aber weiterhin als Darstellungsformat im Web-Browser seine Bedeutung behalten.

Generell ist heute das Web die Integrationsplattform für verteilte Informationssysteme. Techniken aus dem Bereich Föderierter Datenbanken, vor Jahren schon ein aktuelles Thema in der Datenbankforschung, finden hier ihren Einsatz speziell bei der Einbindung von Altsystemen (*Legacy systems*). Ein auch in der Forschung hochaktueller Aspekt ist die Integration von strukturierten und semi-strukturierten Anfrage- und Verarbeitungsmechanismen, die Verfahren aus dem Bereich Datenbanken und dem Information Retrieval vereinigen.

Literatur

[Abi97] Serge Abiteboul. Querying Semi-Structured Data. In *Database Theory - ICDT '97, 6th International Conference*, volume 1186 of *Lecture Notes in Computer Science*, pages 1–18, Delphi, Greece, January 1997. Springer Verlag.

[AGW98] Rolf Assfalg, Udo Goebels, and Heinrich Welter. *Internet Datenbanken — Konzepte, Methoden, Werkzeuge*. Addison Wesley Longman, 1998.

[Bra98] Neil Bradley. *The XML companion*. Addison Wesley, 1998.

[Bun97] Peter Buneman. Semistructured Data. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 117–121, Tucson, Arizona, USA, May 1997. ACM Press.

[CRF00] Don Chamberlin, Jonathan Robie, and Daniela Florescu. QUILT: An XML Query Language for heterogeneous Data Sources. In *WebDB*, 2000.

[Deh98] Wolfgang Dehnhardt. *Anwendungsprogrammierung mit JDBC*. Carl Hanser Verlag, 1998.

[Ext] Extensible Markup Language (XML) 1.0. `http://www.w3.org/TR/REC-xml`.

[Fas] FastCGI. `http://www.fastcgi.com/`.

[FK99] Daniela Florescu and Donald Kossmann. Storing and Querying XML Data Using an RDBMS. *Bulletin of the Technical Committee on Data Engineering*, 22(3):27–34, September 1999.

[FMR] Peter Fankhauser, Massimo Marchiori, and Jonathan Robie. Xml query requirements. `http://www.w3.org/TR/xmlquery-req`.

[HMPT00] Andreas Heuer, Holger Meyer, Beate Porst, and Patrick Titzler. BlueView: Virtual Document Servers for Digital Libraries. In *Proceedings of the IEEE International Conference on Advances in Digital Libraries*, May 2000.

[Hyp] Hypertext Markup Language (HTML) 4.0.1. `http://www.w3.org/TR/REC-html`.

[Inf] Informix Dynamic Server. `http://www.informix.com/IDS/`.

[Jav] JavaBeans Component API. `http://www.javasoft.com/beans/`.

[KM00] Meike Klettke and Holger Meyer. XML and Object-Relational Database Systems — Enhancing Structural Mappings Based on Statistics. In *Proceedings of the Third International Workshop on the Web and Databases*, pages 257–266, May 2000.

[Loe97] Henrik Loeser. Datenbankanbindung an das WWW - Techniken, Tools und Trends. In *Datenbanksysteme in Büro, Technik und Wissenschaft, BTW'97, GI-Fachtagung, Ulm, März 1997*, Informatik aktuell, pages 83–99, Berlin, 1997. Springer-Verlag.

[Loe98] Henrik Loeser. Techniken für Web-basierte Datenbankanwendungen: Anforderungen, Ansätze, Architekturen. *Informatik — Forschung und Entwicklung*, 13(4):196–216, 1998.

[Mau96] Rainer Maurer. *HTML und CGI Programmierung*. dpunkt Verlag, 1996.

[ME00] Jim Melton and Andrew Eisenberg. *Understanding SQL and Java Together: A Guide to SQLJ, JDBC, and Related Technologies*. Morgan Kaufmann Publishers, 2000.

[PM99] Steven Ponndorf and Wolf-Gert Matthäus. *Oracle 8i und Java*. Addison Wesley Longman, 1999.

[San] The San Francisco Project. `http://www.ibm.com/Java/SanFrancisco/`.

[SS00] Gunter Saake and Kai-Uwe Sattler. *Datenbanken & Java — JDBC, SQLJ und ODMG*. dpunkt Verlag, 2000.

[STH+99] Jayaval Shanmugasundaram, Kristin Tufte, Gang He, Chun Zhang, David DeWitt, and Jeffrey Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of the 25th VLDB Conference, Edinburgh, Scotland*, pages 302–314, 1999.

[SYU99] Takeyuki Shimura, Masatoshi Yoshikawa, and Shunsuke Uemura. Storage and Retrieval of XML Documents Using Object-Relational Databases. In *DEXA*, pages 206–217, 1999.

[Tol97] Robert Tolksdorf. *Die Sprache des Web: HTML 4*. dpunkt Verlag, 1997.

[WFC+99] Seth White, Maydene Fisher, Rick Cattell, Graham Hamilton, and Mark Hapner. *JDBC API Tutorial and Reference, Second Edition*. Addison Wesley Longman, 1999.

[YRK99] Randy Jay Yarger, George Reese, and Tim King. *MySQL & mSQL*. O'Reilly & Associates, 1999.