Entwicklungen für die Oracle8*i*-Datenbank

Ab Oracle8*i* werden zwei wichtige Programmiersprachen in der Datenbank unterstützt: PL/SQL und Java. Bei zwei Datenbankprogrammiersprachen stellt sich natürlich die Frage, wie werden mit PL/SQL und Java am besten Anwendungen für Oracle8*i* erstellt?

Kurz zusammengefasst bietet PL/SQL enorme Vorteile für Oracle-Datenbankentwickler, insbesondere hinsichtlich der Leistungsfähigkeit, der einfachen Bedienbarkeit, der Robustheit und der nahtlosen Integration in SQL. Heutzutage ist PL/SQL eine ausgefeilte prozedurale Sprache zur Entwicklung von Datenbankanwendungen und ideal für die Errichtung von datenintensiven SQL-Anwendungen geeignet. Mit Oracle8*i* hat Oracle Java auf dem Datenbank-Server eingeführt, um eine robuste, skalierbare Plattform für diese enorm populäre, universal einsetzbare Sprache zu schaffen. Sie können mit Java mehrstufige, komponentenorientierte Anwendungen mit Enterprise JavaBeans und dem Common Object Request Broker API (CORBA), wie auch herkömmliche Stored Procedures für die Datenbank erstellen. Oracle8*i* bietet mehrere Einrichtungen, die die Erstellung von Anwendungen mit PL/SQL und Java vereinfachen und auch die Kombination von Anwendungen, die in beiden Sprachen geschrieben wurden, erleichtern.

Die umfassende Java-Lösung von Oracle zeichnet sich durch Einfachheit, Flexibilität und freie Wahlmöglichkeiten aus, ohne die für unternehmensweite Anwendungen erforderliche Leistung zu schmälern. Die Java-Lösung von Oracle kombiniert das Beste aus der traditionellen Programmierwelt mit den neuen Internetstandards, um führende Java-Anwendungen zu ermöglichen.

Oracle bietet verschiedene Komponenten, Dienstprogramme und Schnittstellen an, mit denen die XML-Technologie in Datenbankanwendungen genutzt werden kann (vgl. Kapitel 2). Welche Produkte Sie verwenden, hängt von Ihren Anwendungsanfor-

derungen, Programmiervorlieben und der Entwicklungs- und Produktionsumgebung ab. Oracle8*i* beinhaltet die native Unterstützung für Internetstandards einschließlich Java und XML. Sie können Oracle XML-Komponenten und damit erstellte Anwendungen in der Oracle-Datenbank JServer einrichten – der integrierten Java Virtual Machine von Oracle8*i*. Für Geräte und Anwendungen, die kleinere Datenbanken erfordern, können Sie mit Oracle8*i* Lite XML-Daten speichern und abrufen.

Dieses Kapitel beschäftigt sich zunächst mit den Grundlagen der Oracle JServer-Architektur, und erklärt, wie Sie die Java-XML-Komponenten von Oracle im Oracle JServer verwenden können. Wir erörtern auch verschiedene Ansätze zur Speicherung und für den Abruf von XML-Dokumenten mit XML-Komponenten. Am Ende dieses Kapitels erfahren Sie, wie man mit diesen XML-Komponenten eine einfache Online-Bookshop-Website entwickeln kann.

3.1 Die XML-fähige Datenbank Oracle8*i*

Datenbanken und XML bieten einander ergänzende Funktionalitäten bei der Speicherung von Daten. Die Datenbank speichert Daten zum effizienten Abruf, XML unterstützt den einfachen Austausch von Informationen, wodurch die Interoperabilität zwischen Anwendungen gewährleistet wird. Oracle8*i* ermöglicht die Speicherung von XML und die Errichtung von XML-fähigen Anwendungen. Wenn Sie XML-Dokumente in der Datenbank speichern, können Sie hierzu auch die Datenbankverwaltungswerkzeuge und Prozeduren wie das Backup nutzen. Sie können diese zur Durchsetzung von Regeln für Daten und Sicherheit verwenden, und um Operationen zu blockieren, die die Datenintegrität betreffen, indem Sie Regeln und Logik in einer Datei einbetten. Außerdem können Sie über die Konvertierung von Datenbanktabellen in XML-Dokumente die XML-Funktionen nutzen. Sie können XML-Dokumente mit XSLT-Stylesheets als HTML-Seiten formatieren, sie mit XML-basierten Abfragespachen durchsuchen oder sie als Datenaustauschformat verwenden.

Die objektrelationalen Funktionen von Oracle8*i* erlauben die Abbildung der komplexen Strukturen von XML-Daten. Sie können XML-Daten auf der gewünschten Granularitätsstufe einsetzen und verwalten, und damit auf effiziente Weise dynamische XML-Dokumente aus den resultierenden Teildokumenten erstellen. Sie können ebenso XML-Dokumente als einzelnes Dokument mit seinen Tags in einem *Character Large Object* (*CLOB*) speichern oder als Daten, indem diese ohne Tags über objektrelationale Tabellen verteilt werden. Das Internet File System (vgl. Kapitel 5) von Oracle8*i* kann auf XML-Dokumente zugreifen, die als externe Dateien oder im Web gespeichert sind. Sie können mit interMedia Text (vgl. Kapitel 6) von Oracle8*i* Suchen in XML-Dokumenten durchführen, die in Oracle8*i* gespeichert werden. Sie können XML als reinen Text indizieren oder als Dokumentenabschnitte für Detailsuchen, z.B. um „Oracle WITHIN title“ zu finden, wobei „title“ ein Bereich in einem Do-

kument ist. interMedia in Oracle8*i* unterstützt auch die Volltextindizierung der Dokumente und Funktionen zur Ausführung von SQL-Abfragen über Dokumente.

3.2 JServer und Java-XML-Komponenten von Oracle

Oracle bietet die derzeit umfassendste und leistungsstärkste Java-Lösung, eine Vielzahl an Servern zur Ausführung von Java-Anwendungen, Standardprogrammierschnittstellen (APIs) zur Erstellung von unternehmensweiten Anwendungen und ein produktives Set von Werkzeugen für den Einsatz von Java-Anwendungen. Oracle verfügt über zwei Ausführungsumgebungen für Java – Oracle8*i* und Oracle Application Server. Beide Server nutzen ein durchgehendes Entwicklungsmodell mit gemeinsamen APIs und gemeinsamen Verwendungs- und Verwaltungsstrukturen, mit denen Sie Java-Anwendungen erstellen können, die leicht über mehrstufige Architekturen partitioniert werden können. In der Regel können Sie datenintensive Geschäftslogik auf dem Datenbankserver, und Computing-intensive Logik auf dem Application Server ablegen. In Kapitel 4 wird die Entwicklung für den Oracle Application Server behandelt.

Oracle8*i* JServer ist der industrieweit robusteste Java-Server. Er wurde von Grund auf als erste echte unternehmensweit skalierbare Java-Programmierplattform entwickelt. Oracle8*i* JServer wird den technischen Herausforderungen gerecht, die bislang die breitgestreute Verwendung von Java bei unternehmensweiten Anwendungen eingeschränkt haben. Der JServer von Oracle ist vollständig kompatibel zu den Java-Standards im Java Development Kit und hat alle Tests bestanden, die erforderlich sind, um als „Standard-Java“ zertifiziert zu werden. Oracle8*i* wird derzeit mit Java-Unterstützung auf vielen Betriebssystemen und Hardware-Plattformen einschließlich Solaris, NT, HP-UX, DEC, AIX, Sequent und anderen Systemen ausgeliefert.

3.2.1 Grundkonzepte des JServers

JServer ist eng in die Oracle8*i*-Datenbank integriert und besitzt eine Reihe von Komponenten: einen Byte-Code Compiler, einen Garbage Collector, einen integrierten Java-Klassen Loader und einen Java-through-C Compiler, die alle für die optimale Leistung und Skalierbarkeit in der Datenbankumgebung ausgelegt sind (siehe Abbildung 3-1). Die Anwendung läuft im gleichen Prozess- und Adressraum wie der Datenbankkernel, so dass verschiedene Speicherstapel für die optimale Leistung gemeinsam genutzt werden. JServer-Sitzungen sind analog zu normalen Oracle-Sitzungen. Wenn Sie sich bei Oracle8*i* anmelden, starten Sie eine Datenbanksitzung. Wenn Sie in dieser Sitzung das erste Mal eine Java-Methode aufrufen, wird für diese Sitzung eine private

Java Virtual Machine erstellt. Eine Sitzung umfasst die Lebensdauer aller Objekte, auf die statische Java-Variablen verweisen, alle Objekte, die auf diese Objekte verweisen und so weiter (deren transitives Ende). Unter dem Gesichtspunkt der Client/Server-Interaktion betrachtet, pflegt jede JServer-Sitzung ihren eigenen Java-Status und wird so als spezielle JCM interpretiert. In Wirklichkeit nutzt die Oracle-Implementierung fast den ganzen Code, die Infrastruktur und die Metadaten der aktiven JVM zwischen den Benutzern.

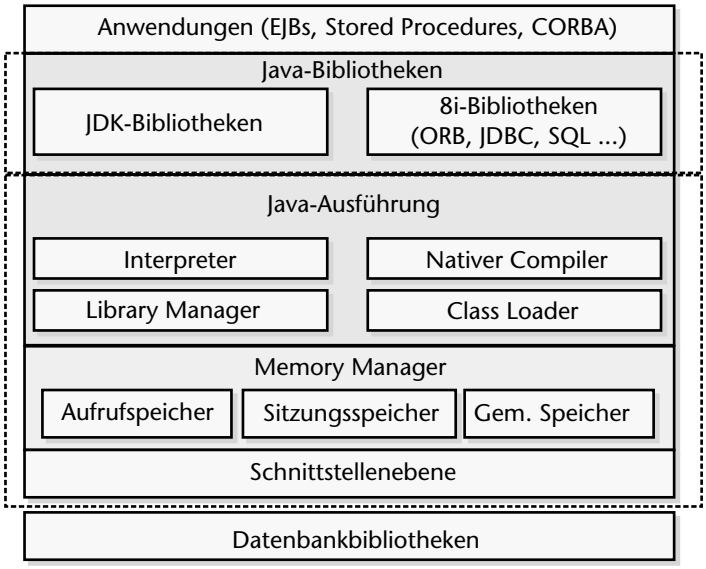

Abbildung 3-1: Die Architektur des Oracle JServers

Im Unterschied zur herkömmlichen JVM, wo die Java-Dateien kompiliert und geladen werden, kompiliert und lädt das JServer-System Schemaobjekte. Die drei Arten von Java-Schemaobjekten, die in Abbildung 3-2 dargestellt werden, sind:

- **Schemaobjekte für Java-Klassen –** Entsprechen den Java-Klassendateien
- **Schemaobjekte für Java-Quellcode –** Entsprechen den Java-Quelldateien
- **Schemaobjekte für Java-Ressourcen** - Entsprechen den Java-Ressource-Dateien

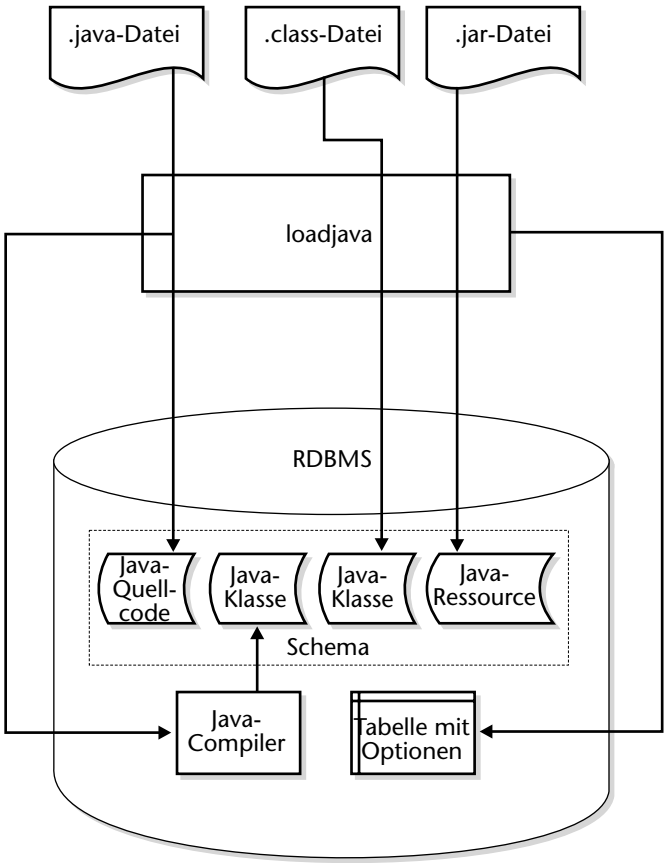

Abbildung 3-2: Java-Schemaobjekte

Die Klassen, die eine Java-Anwendung definieren, werden in der Oracle8*i*-Datenbank unter einem SQL-Schema des Besitzers gespeichert. Da diese Klassen-/Schemaobjekte in der Regel lange in der Datenbank existieren, kompiliert JServer sie alle in gut optimierten nativen Code mit der *Way ahead of time (WAT)*-Kompilierungsmethode. WAT übersetzt Java-Standardklassenbinärcodes in spezielle C-Programme, die dann in native dynamische Bibliotheken kompiliert werden (mit einem integrierten, plattformabhängigen C-Compiler).

Dadurch, dass der Interpreter-Overhead ganz eliminiert wird und Optimierungen in Zeilen und Objekten unterstützt werden, beschleunigt WAT die Ausführungsleistung von Java-Anwendungen erheblich.

3.2.2 Java-XML-Komponenten

Oracle bietet verschiedene Komponenten, Dienstprogramme und Schnittstellen, mit denen Sie die XML-Technologie in Ihren Datenbankanwendungen nutzen können. Hierzu gehören der XML-Parser, der XSL-Processor, der XML Class Generator, XML TransViewer-Beans und XSQL-Servlets. Der XML-Parser ist eine stand-alone XML-Komponente, die ein XML-Dokument parst, das dann von einer Anwendung verarbeitet werden kann. Der Parser unterstützt die Schnittstellen *Document Object Model (DOM)* und *Simple API for XML (SAX)* sowie XML-Namespaces. Er kann XML-Dokumente in auswertendem und nicht-auswertendem Modus parsen. Im nicht-auswertenden Modus stellt der Parser sicher, dass das XML-Dokument korrekt aufgebaut ist. Im auswertenden Modus wird das Dokument gegen die DTD ausgewertet und auf die Gültigkeitsbeschränkungen geprüft. Die XSL-Umwandlungsengine ist im XML-Parser integriert und führt Umwandlungen von XML-Dokumenten unter Verwendung von XSL-Stylesheets durch. Der XML Class Generator erzeugt eine Reihe von Java-Quellcode-Dateien aus einer XML-DTD. Sie können mit den erzeugten Quelldateien ein XML-Dokument, das zur Eingabe-XML-DTD kompatibel ist, erstellen, optional auswerten und drucken.

Das XSQL-Servlet ist ein Werkzeug, das SQL-Abfragen verarbeitet und die Ergebnisse als XML ausgibt. Dieser Prozessor ist als Java-Servlet implementiert und verwendet eine XML-Datei, die integrierte SQL-Abfragen als Eingabe verwendet. Es werden alle erwähnten Komponenten zur Durchführung der Operationen verwendet. Die XML TransViewer-Beans sind ein Set von XML-Komponenten für Java-Anwendungen oder Applets. Sie können mit diesen Beans XML-Dateien asynchron parsen, oder die XML/XSL-Dateien mit der farblichen Hervorhebung der XML/XSL-Syntax anzeigen. Sie können auch XSL-Stylesheets anwenden, um ein XML-Dokument in nahezu jedes textbasierte Format zu übertragen, einschließlich XML, HTML und DDL, und die Ergebnisse der Umwandlungen sofort einsehen. Sie können diese visuellen und nicht-visuellen Komponenten in Oracle JDeveloper integrieren, um Entwicklern das schnelle Erstellen und Einsetzen von XML-basierten Datenbankanwendungen zu ermöglichen.

Java-XML-Komponenten können flexibel als einzelstehende Java-Anwendungen mit einem JDBC-Treiber laufen, um mit der Datenbank zu interagieren, oder effizient im Oracle JServer laufen mit dem fein-getunten serverseitigen JDBC-Treiber. Die einzelstehende Java-Anwendung kann sich entweder auf der Client-Seite oder auf dem Oracle Application Server in der mittleren Ebene befinden. In der Regel verwenden Sie in Java-Einzelanwendungen JDBC, um eine Verbindung zur Datenbank mit der Klasse DriverManager einzurichten, die die Gruppe der JDBC-Treiber verwaltet. Sind die JDBC-Treiber einmal geladen, können Sie die Methode getConnection aufrufen, die ein Connection-Objekt liefert, das eine Datenbanksitzung darstellt. Der serverseitige interne JDBC-Treiber läuft in einer Standardsitzung und einem Standardtransaktionskontext. So sind Sie bereits mit der Datenbank „verbunden“ und alle Ihre SQL-

Operationen sind Teil des Standardtransaktionskontexts. Sie können mit der folgenden Anweisung ein Connection-Objekt erhalten:

```
Connection c = DriverManager.getConnection("jdbc:default:connection");
```

Wenn Sie Anwendungen mit Java Stored Procedures entwickeln, achten Sie darauf, dass der serverseitige interne JDBC-Treiber nicht zur Anbindung an eine Ferndatenbank verwendet werden kann. Sie können sich nur bei einem Server anmelden, der Ihre Java Stored Procedure ausführt. Für Server-zu-Server-Verbindungen verwenden Sie einen serverseitigen oder clientseitigen JDBC Thin-Treiber.

Wie zu Beginn dieses Kapitels erwähnt, können Sie seit Oracle8*i* mit Java Datenbankanwendungen entwickeln und diese in der Datenbank als Stored Procedures einsetzen. In Java entwickelte Stored Procedures laufen im gleichen Adressraum wie SQL und PL/SQL, so dass sie nahtlos mit anderen PL/SQL-Anwendungen zusammenarbeiten können. Da diese Java-Programme im gleichen Adressraum laufen wie der Datenbankserver, und die eingebetteten serverseitigen internen JDBC-Treiber nutzen können, müssen sie keine Netzwerkverbindungen auf sich nehmen, um auf SQL-Daten zuzugreifen, und sind daher hervorragend für abfrageintensive Aufgaben und die Reduzierung des Netzwerkverkehrs optimiert. Um gespeicherte Java-Programme in einer Oracle8*i*-Datenbank einzusetzen, müssen Sie Ihre Java-Programme in die Datenbank laden und die Java-Methoden für SQL veröffentlichen. Die Datenbank unterstützt eine Vielzahl verschiedener Möglichkeiten zum Laden eines Java-Programms. Sie können Java-Quelltext, Standard-Java-Klassendateien oder Java-Archive (jar) laden. Eine in die Datenbank geladene Java-Quelle wird automatisch vom Java Byte-Code-Compiler kompiliert, der sich in der Datenbank befindet. Sie können Java-Objekte in Oracle8*i* auf verschiedene Arten laden. Als Erstes sollten Sie einen neuen DDL-Befehl in der Form „CREATE JAVA ...“ über SQL*Plus absetzen, um den Java-Quellcode, Binär- oder Ressourcendateien in die Datenbank zu laden:

```
Create a directory object on the server's file system
SQL> CREATE DIRECTORY bfile_dir as '/home/user/oracle/xml/parser/v2';
# Then load the Java class using the "CREATE JAVA CLASS ..." statement
SQL> CREATE JAVA CLASS USING BFILE (bfile_dir, 'XMLParser.class');
```

Um den Ladeprozess zu vereinfachen, enthält Oracle8*i* ein in Java geschriebenes Dienstprogramm namens LOADJAVA, das den Ladeprozess von Java-Code in die Datenbank automatisiert. Da das Dienstprogramm LOADJAVA die JDBC-Treiber von Oracle zur Kommunikation mit der Datenbank verwendet, können Sie Java-Programmeinheiten über das Netzwerk in die Datenbank laden.

```
loadjava -user scott/tiger@myhost:1521:orcl xmlparserv2.jar
```

3.2.3 Veröffentlichung und Aufruf von Java-XML-Komponenten

Es ist wichtig anzumerken, dass es mit dem Aufkommen der Java Stored Procedures nicht obsolet wurde, Stored Procedures in PL/SQL zu schreiben. Sie können die bestehende Bibliothek an PL/SQL Stored Procedures nutzen, während Sie neue serverseitige Stored Procedures in Java implementieren. Dieser Ansatz ist möglich, weil die Stored Procedures in PL/SQL und in Java die gleiche Aufrufspezifikation haben – Letztere ist für SQL veröffentlicht. Auf der Anwendungsebene ist es daher transparent, welche Technologie eingesetzt wird.

Sie müssen die Java-Methoden im Oracle8*i* Data Dictionary veröffentlichen, bevor Sie diese über SQL aufrufen. Wenn Sie eine Java-Klasse in die Datenbank laden, werden die Methoden nicht automatisch veröffentlicht, weil Oracle nicht weiß, was sichere Einstiegspunkte für SQL-Aufrufe sind. Um die Methoden zu veröffentlichen, müssen Sie Aufrufspezifikationen erstellen, die Java-Methodennamen, Parametertypen und Rückgabetypen für deren SQL-Gegenstücke festlegen. Für eine bestimmte Java-Methode deklarieren Sie eine Funktion oder Prozedur für eine Aufrufspezifikation mit den Anweisungen SQL CREATE FUNCTION oder CREATE PROCEDURE. Sie veröffentlichen Java-Methoden, die Werte zurückgeben, als Funktionen, und leere Java-Methoden als Prozeduren. Anwendungen rufen die Java-Methode über deren Aufrufspezifikation auf, indem der Name der Aufrufspezifikation angegeben wird. Das Laufzeitsystem sucht die Definition der Aufrufspezifikation im Oracle8*i* Data Dictionary und führt die entsprechende Java-Methode aus, wie in Abbildung 3-3 dargestellt.

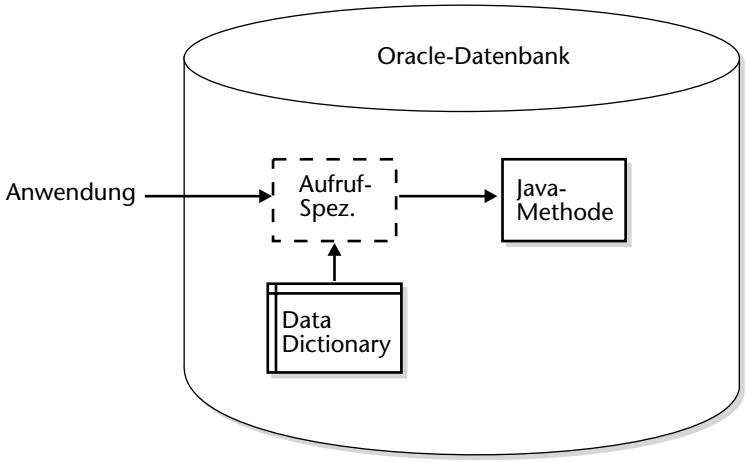

Abbildung 3-3: Aufruf einer Java-Methode

Eine Aufrufspezifikation und die Java-Methode müssen sich im gleichen Schema befinden (außer die Java-Methode besitzt das Synonym PUBLIC). Sie können eine Aufrufspezifikation deklarieren als

- Einzelstehende (Top-Level) PL/SQL-Funktion oder Prozedur
- Gepackte PL/SQL-Funktion oder Prozedur
- Elementmethode eines SQL-Objekttyps

Eine Aufrufspezifikation gibt Oracle den obersten Einstiegspunkt einer Java-Methode bekannt. Aus diesem Grund können Sie nur statische Methoden veröffentlichen – mit einer Ausnahme: Sie können Instanzmethoden als Elementmethode eines SQL-Objekttyps veröffentlichen. Die entsprechenden SQL- und Java-Parameter (und Funktionsergebnisse) müssen kompatible Datentypen in der Aufrufspezifikation besitzen. Oracle8*i* führt automatisch Konvertierungen zwischen SQL-Typen und Java-Klassen durch.

Sie können die folgende Java-Quelle oder Klassendatei in die Oracle8*i*-Datenbank laden und deren Methoden als PL/SQL-Funktionen veröffentlichen. Um diese Methoden zu veröffentlichen, können Sie die folgende Aufrufspezifikation erstellen:

```
public class CaseConvert
{
 public static String toUpper(String str)
 {
    return str.toUpperCase();
 }
 public static String toLower(String str)
 {
    return str.toLowerCase();
 }
}
```

```
CREATE OR REPLACE FUNCTION toUpper (str VARCHAR2) return VARCHAR2AS LAN-
GUAGE JAVA NAME 'CaseConvert.toUpper(java.lang.String) returns
                                    java.lang.String';
CREATE OR REPLACE FUNCTION toLower (str VARCHAR2) return
                                       VARCHAR2AS LANGUAGE JAVA
NAME 'CaseConvert.toLower(java.lang.String) returns java.lang.String';
```

Ähnlich können Sie den Java-XML-Parser laden und den folgenden Java-Code schreiben, um ein XML-Dokument, das in einer Tabelle als CLOB gespeichert ist, zu parsen und auszuwerten:

```
public class Validate {
public static String validateCLOB(String xmldoc) {
 SAXParser parser = new SAXParser();
 try {
 .    parser.parse(new StringInputStream(xmldoc));
      return "Valid XML Document";
 } catch (SAXException e) {
      return "Invalid XML Document" + e.getMessage();
 }
}

CREATE OR REPLACE FUNCTION validateCLOB (xmldoc CLOB) return
VARCHAR2AS LANGUAGE JAVA NAME
'Validate.validateCLOB (oracle.sql.CLOB) returns java.lang.String';
```

Wenn Sie eine Java Stored Procedure veröffentlicht haben, sieht diese wie eine PL/SQL Stored Procedure aus. Sie können sie mit dem SQL-Namen in der Aufrufspezifikation in verschiedenen Kontexten aufrufen:

- In einer Zeile mit einer SQL-Anweisung:

```
SELECT validateCLOB(XMLDOC) FROM XMLSTORE;
```

- Von der obersten Ebene mit CALL:

```
CALL validateCLOB(:xmldoc) INTO :x
```

- Aus einer PL/SQL-Prozedur, einem Paket oder anonymen Blöcken mit genau derselben Syntax, mit der auch andere PL/SQL-Prozeduren aufgerufen werden.

```
DECLARE
result VARCHAR;
xmldoc CLOB;
BEGIN
...
result := validateCLOB(xmldoc);
dbms_output(result);
...
END;
```

Nun wissen wir einiges darüber, wie Java Stored Procedures in der Oracle8*i*-Datenbank entwickelt und eingesetzt werden. Mit einigen einfachen Beispielen haben wir die Schritte gezeigt, die zum Laden, Auflösen, Veröffentlichen und Aufrufen von Java-Programmen in der Datenbank erforderlich sind. Eine der zentralen Anwendungen der Java Stored Procedures ist, die Oracle8*i*-Datenbank als Speicher für XML-Dokumente zu verwenden. Im nächsten Abschnitt betrachten wir verschiedene Ansätze zur Abbildung der Datenbankschemata auf XML-Dokumente und umgekehrt.

3.3 Datenbankschema und XML-Dokumente

XML-Dokumente sind Texte, die einer hierarchischen Struktur oder Baumstruktur entsprechen, die in einer DTD oder einem XML-Schema angegeben sind. Sie können diese hierarchischen Daten einfach in einer optimalen internen Form unter Verwendung von objektrelationalen Tabellen speichern. Alle bestehenden und zukünftigen internen Anwendungen können mit dieser Information in der effizientest möglichen Weise arbeiten. Wenn Sie Informationen abrufen, um diese mit anderen Parteien oder Anwendungen gemeinsam zu nutzen, können Sie die entsprechende Ansicht der Daten und des Dokumentkontexts speziell für die anstehenden Aufgaben als integriertes XML darstellen. Die Objektansichten von Oracle8*i* erlauben Ihnen die Präsentation der Daten in beliebigen „logischen" Kombinatonen, wobei Details des unterliegenden physikalischen Speichers verborgen werden. Sie können die Struktur einer oder mehrerer unterliegender Tabellen in eine für die Anforderungen einer bestimmten Anwendung nützlichere oder passendere Struktur übertragen. Wenn Sie Informationsansichten mit anderen Views von in Beziehung stehenden Informationen verbinden, bilden diese ganz natürlich „Bäume" oder „Graphen" der verknüpften Daten. Wenn Sie Datenbankinformationen als XML darstellen, stellen die zuvor verknüpften Views die Grundlage dar für viele verschiedene baumartig strukturierte XML-Dokumente.

Dies ist ein einfaches Beispiel der Zuordnung einer Datenbanktabelle an eine XML-DTD der folgenden Form:

```
<!ELEMENT table (rows)*>
<!ELEMENT rows (column1, column2, ...)>
<!ELEMENT column1 (#PCDATA)>
<!ELEMENT column2 (#PCDATA)>
...
```

Eine Datenbank bietet aber mehr Möglichkeiten als eine DTD zum Ausdruck von Regeln. Mit XML und DTDs können Sie keine Typinformation definieren. Das Datenbankschema definiert Typinformation und Beschränkungen, wie die zulässigen Wertebereiche. Ein Datenbankschema erlaubt die Definition von Beziehungen und Abhängigkeiten. Beispielsweise kann Ihr e-Business Aufträge als XML-Dokumente erhalten. Durch die Verwendung einer Datenbank können Sie die Kunden- und Bestellinformation verbinden und Regeln definieren, um Bestellungen für geschlossene Konten nicht zu verarbeiten. Trotz der Beschränkungen in DTDs wird bei der Zuordnung eines Datenbankschemas an eine DTD für die Werkzeuge, die XML-Dokumente als Eingabe benötigen, die Datenbank als virtuelles XML-Dokument dargestellt.

Das Format für DTDs ist ein etablierter internationaler Standard, der weiterbestehen und in den nächsten Jahren weiter verbessert werden wird. Wegen der den DTDs innewohnenden Beschränkungen und der zunehmend datenorientierten Rolle, die XML

aufgrund der Entwicklungen im e-Business und e-Commerce übernehmen soll, arbeitet das W3C-Gremium an der Förderung eines neuen Standards namens XML-Schema, statt die aktuellen DTD-Standards weiter voranzutreiben. Mit der Verbreitung des XML-Schemas können die Einschränkungen der DTDs überwunden werden. XML-Schemas erlauben die Angabe der Typinformation und der Beschränkungen. Mit dem XML-Schema können Sie eine einfache Datenbanktabelle an ein XML-Schema in der folgenden Form zuweisen:

```
<schema targetNamespace="someNSURI">
 .<element name="table">
     <element name="rows" minOccurs="0" maxOccurs="*">
        <element name="column1">
           <datatype source="string">
               <length value="1000"/>
           </datatype>
        </element>
      <element name="column2" type="decimal">
 ...
     </element>
  </element>
</schema>
```

Um Daten zwischen einem XML-Dokument und einer Datenbank zu übertragen, müssen Sie eine Dokumentenstruktur an das Datenbankschema zuweisen und umgekehrt. Eine XML-Dokumentenstruktur ist mit einer DTD oder einem XML-Schema verbunden. Eine DTD wird verwendet, um die Elemente und Attribute zu beschreiben. Wie DTDs beschreiben Schemas Daten, bieten aber den zusätzlichen Vorteil, dass auch die Datentypen der Daten angegeben werden können, von einfachen Typen wie Zeichenketten, Datumsangaben und Integern bis hin zu komplexen Strukturen wie Mailadressen oder Punkten in einem Graph.

3.3.1 Zuweisung von XML-Dokumenten an ein Datenbankschema

Für die Zuweisung eines mit einer XML-DTD oder einem XML-Schema assoziierten XML-Dokuments an ein Datenbankschema, um XML in Oracle8*i* zu speichern, gibt es drei Grundstrategien:

- Zuweisung des vollständigen XML-Dokuments an ein einzelnes intaktes Objekt, wie CLOBs
- Zuweisung von XML-Elementen an objektrelationale Tabellen und Spalten im Datenbankschema
- Zuweisung von Fragmenten von XML-Dokumenten als CLOBs, und des übrigen Teils als objektrelationale Tabellen

Sie können je nach Struktur des XML-Dokuments und den von der Anwendung ausgeführten Operationen einen der obigen Ansätze auswählen. Sie können auch das XML-DTD oder Schema in der Datenbank speichern, um die XML-Dokumente auszuwerten.

XML-Dokumente in CLOBs

Die Speicherung eines intakten XML-Dokuments in einem CLOB oder Binary Large Object (BLOB) ist eine gute Strategie, wenn das XML-Dokument statische Inhalte enthält, die nur aktualisiert werden, wenn das gesamte Dokument ersetzt wird. Die Beispiele hierfür schließen geschriebenen Text wie Artikel, Anzeigen, Bücher, Gesetze und Verträge ein. Dokumente dieser Art werden als dokumentenzentriert bezeichnet und in der Datenbank als Ganzes gehalten. Die Speicherung dieser Dokumentenart als Ganzes in Oracle8*i* verbindet die Vorteile einer bewährten Datenbank mit einer höheren Zuverlässigkeit gegenüber der Speicherung auf Dateisystemebene. Wenn Sie ein XML-Dokument außerhalb der Datenbank speichern möchten, können Sie immer noch Oracle8*i*-Funktionen zur Indizierung, Abfrage und für den effizienten Abruf des Dokuments über BFILES, URLs und textbasierte Indizierung verwenden.

XML-Dokumente als objektrelationale Daten

Wenn das XML-Dokument eine gut definierte Struktur besitzt und Daten enthält, die aktualisierbar sind oder anderweitig verwendet werden können, so ist das Dokument datenzentriert. Typischerweise enthält das XML-Dokument Elemente und Attribute, die komplexe Strukturen besitzen. Beispiele dieser Dokumentenart sind Bestellungen, Rechnungen oder Flugpläne. Oracle8*i* hat mit den objektrelationalen Erweiterungen die Möglichkeit, die Struktur der Daten in der Datenbank über Objekttypen, Objektreferenzen und Sammlungen abzubilden. Es gibt zwei Optionen, um die Struktur von XML-Daten in objektrelationaler Form zu speichern und beizubehalten:

- Speichern der Elementattribute in einer relationalen Tabelle und Definition der Objektansichten zur Abbildung der Struktur der XML-Elemente
- Speichern der strukturierten XML-Elemente in einer Objekttabelle

Einmal in objektrelationaler Form gespeichert, können die Daten mit SQL einfach aktualisiert, abgefragt, neu arrangiert und neu formatiert werden.

Das XSQL-Servlet erlaubt die Speicherung eines XML-Dokuments durch die Zuordnung an unterliegenden objektrelationalen Speicher, aber auch umgekehrt, objektrelationale Daten als XML-Dokument abzurufen. Ist ein XML-Dokument strukturiert, aber die Struktur des XML-Dokuments ist nicht kompatibel zur Struktur des unterliegenden Datenbankschemas, müssen Sie die Daten in das richtige Format übertragen, bevor Sie sie in die Datenbank schreiben. Sie können dies mit XSL-Stylesheets oder anderen Programmieransätzen erreichen, oder das datenzentrierte XML-Dokument als intaktes Einzelobjekt speichern. Oder Sie können Objekt-Views definieren, die den

verschiedenen XML-Dokumentstrukturen entsprechen, und definieren Instead-of-Trigger, um die entsprechenden Umwandlung vorzunehmen und die Basisdaten zu aktualisieren.

XML-Dokumente als Fragmente und objektrelationale Daten

Außerdem können Sie mit Oracle8*i*-Views eine Kombination aus strukturierten und nicht-strukturierten XML-Daten als Ganzes anzeigen und damit arbeiten. Views erlauben die Konstruktion von Objekten *on the fly*, indem verschiedenartig gespeicherte XML-Daten kombiniert werden. Daher können Sie strukturierte Daten (von Angestellten, Kunden etc.) an einem Ort in objektrelationalen Tabellen speichern und gleichzeitig in Beziehung stehende unstrukturierte Daten (wie Beschreibungen und Kommentare) in einem CLOB speichern. Wenn Sie die Daten als Ganzes wieder abrufen möchten, stellen Sie einfach die Struktur aus den verschiedenen Datenstücken mit Typkonstruktoren in der SELECT-Anweisung der View wieder zusammen.

Über das XSQL-Servlet können dann die zusammengestellten Daten aus der View in einem einzigen XML-Dokument abgerufen werden.

3.3.2 Zuweisung eines Datenbankschemas an virtuelle XML-Dokumente

Für die Zuweisung eines Datenbankschemas an virtuelle XML-Dokumente gibt es zwei Grundstrategien:

- Abbildung der vollständigen Datenbank in einem virtuellen XML-Dokument
- Abbildung der Ergebnismenge einer Abfrage in einem virtuellen XML-Dokument

Ja nach Anwendung können Sie einen dieser Ansätze verwenden. Gehen wir beispielsweise von einer Anwendung aus, die eine Datenbankkopie von einer Datenbank auf eine andere mit einem anderen Schema durchführt. Diese Anwendung kann eine XML-Umwandlung auf dem virtuellen Dokument verwenden, das von der ersten Datenbank dargestellt wird, und das Ergebnis der Transformation in die zweite Datenbank einfügen.

Das vollständige Schema

Eine Datenbank besteht aus einem Schema, das mit jedem Datenbankbenutzer verknüpft ist. Jedes Schema, das mit einem Benutzer verknüpft ist, ist eine Sammlung von Schemaobjekten, auf die der Benutzer zugreifen kann. Bei der Zuordnung eines Datenbankschemas an eine DTD wird jeder Benutzer als ein Kindelement des obersten Elements zugewiesen, das über die SID der Datenbankinstanz bestimmt wird. Ein Element, das ein Benutzerschema und dessen Kindelemente darstellt, verwendet einen eindeutigen Namespace, um Konflikte mit Schemaobjekten zu vermeiden, die in anderen Benutzerschemas definiert sind.

Die folgende (vereinfachte) Prozedur erzeugt eine DTD aus einem relationalen Schema:

1. Erstellen Sie für jede Tabelle eine Element.
2. Erstellen Sie für jede Spalte in einer Tabelle ein reines PCDATA-Kindelement.
3. Für jedes Objekt oder jede verschachtelte Tabellenspalte erstellen Sie ein ELEMENT-Inhalt-Kindelement mit Attributen oder verschachtelten Spalten als Kindelemente.

Beispielsweise entspricht die folgende DTD einer einfachen Datenbank:

```
<!ELEMENT dbschema (sys, scott, ...)>
<!ATTLIST dbschema
          xmlns CDATA #FIXED http://www.oracle.com/xml/dbschema
          sid CDATA #REQUIRED>
<!ELEMENT scott (BookList, ...)>
<!ATTLIST scott
         xmlns CDATA #FIXED "http://www.oracle.com/xml/dbschema/scott">
<!ELEMENT BookList (Book)*>
<!ATTLIST Book row_num CDATA #IMPLIED>
<!ELEMENT Book (Title, ISBN, Author, Publisher, (Review)*)>
...
```

Leider gibt es verschiedene Einschränkungen bei der Zuordnung eines Datenbankschemas an eine DTD. Beispielsweise gibt es keine Möglichkeit, die Datentypen oder Spaltenlängen definitiv aus der DTD vorherzusagen. Die Lösung für dieses Problem ist die Verwendung von Datentypen in XML-Dokumenten, die ein XML-Schema verwenden. Sie können auch die Primärschlüssel/Fremdschlüssel-Einschränkungen im XML-Dokument durch die Verwendung des XML-Schemas bewahren.

Abfragen

Die Ergebnismenge einer Abfrage kann einem virtuellen XML-Dokument zugewiesen werden, genau wie beim Prozess der Zuordnung der Datenbank an ein XML-Dokument. Das Ergebnis der Abfrage enthält eine Zeilenmenge, die dem XML-Dokument mit dem Root-Element ROWSET zugeordnet wird, und jede Zeile wird in einem Element ROW gekapselt. Sie können andere Element-Tags für die Element-Tags ROWSET und ROW angeben. Das folgende Beispiel illustriert die Zuordnung einer einfachen Abfrage an ein XML-Dokument:

```
SQL> select * from scott.BookList;
<!ELEMENT BookList (Book)*>
<!ATTLIST BookList
        xmlns CDATA #FIXED "http://www.oracle.com/xml/dbschema/scott">
<!ELEMENT Book (Title, ISBN, Author, Publisher, (Review)*)>
<!ATTLIST Book row_num CDATA #IMPLIED>
...
```

Sie können ein XML-Schema auch verwenden, um die XML-Dokumentenstruktur und die Datentypen anzugeben. Das folgende XML-Schema entspricht der zuvor ausgewählten Abfrage:

```
<schema targetNamespace="someNSURI">
  <type name="Person">
    <datatype name="Lastname" source="string"/>
    <datatype name="Firstname" source='string"/>
  </type>
  ...
  <element name="BookList">
    <element name="rows" minOccurs="0" maxOccurs="*">
       <element name="Title" type="string"/>
       <element name="ISBN" type="ISBN">
       <element name="Author" type="Person">
        ...
    </element>
  </element>
</schema>
```

Zusätzlich zu einfachen relationalen Abfragen können Sie Abfragen mit verschachtelten SELECT-Anweisungen oder Objektnavigationen einem XML-Dokument zuordnen, um dem Dokument Tiefe zu geben. Das folgende Beispiel ruft das Attribut **Lastname** vom Objekt **Author** ab:

```
SQL> select Title, Author.Lastname, ISBN from scott.BookList;
<!ELEMENT BookList (Book)*>
<!ATTLIST BookList
        xmlns CDATA #FIXED "http://www.oracle.com/xml/dbschema/scott">
<!ELEMENT Book (Title, Author_Lastname, ISBN)>
<!ATTLIST Book row_num CDATA #IMPLIED>
...
```

Die Instanz eines XML-Dokuments, die der obigen Abfrage entspricht, lautet:

```
<BookList>
   <Book row_num="1">
     <Title>Introducing XML</Title>
     <Author_Lastname>Smith</Author_Lastname>
     <ISBN>11-0342000123</ISBN>
   </Book>
   <Book row_num="2">
     <Title>XML for web sites</Title>
     <Author_Lastname>Jackson</Author_Lastname>
     <ISBN>15-7812000423</ISBN>
    </Book>
<BookList>
```

3.3.3 Speichern und Abrufen von XML-Daten

XML-Daten können dokumentenzentriert oder datenzentriert sein. Beide Dokumenttypen können aus einer Datenbank oder einem XML-Dokument stammen. Wenn die Daten aus der Datenbank stammen, möchten Sie sie vielleicht als XML darstellen, und umgekehrt, wenn die Daten aus einem XML-Dokument stammen, möchten Sie diese womöglich in einer relationalen Datenbank speichern. Ein Beispiel für Ersteres sind die vielen Altdaten, die in relationalen Datenbanken gespeichert werden. Ein Beispiel für Letzeres sind Daten, die im Web als XML angezeigt werden, die in der Datenbank gespeichert werden sollen. Je nach Anforderung benötigen Sie verschiedene Techniken, um Daten aus einem XML-Dokument in der Datenbank zu speichern, aus der Datenbank in einem XML-Dokument oder beides. Im vorherigen Abschnitt haben wir verschiedene Ansätze für die Zuordnung des Datenbankschemas an die DTD und umgekehrt besprochen. Nun stellen wir Implementierungstechniken vor, um die obigen Anforderungen zu verwirklichen.

Datenbank-DOM

Die meisten XML-Werkzeuge arbeiten entweder mit dem SAX- oder dem DOM-API. Sie können mit diesen Werkzeugen eine Datenbank als XML-Dokument anzeigen lassen, wenn auf die Datenbank mit einem DOM-API zugegriffen wird. Anders ausgedrückt kann mit einem DOM-API für Datenbanken die Datenbank wie ein virtuelles XML-Dokument dargestellt werden, das mit einer DTD verknüpft ist, die aus dem Datenbankschema abgeleitet ist. Ein Element im DOM-Modell kann ganz allgemein als einzelne Ergebnismenge betrachtet werden (bei der Übertragung von Daten aus der Datenbank in XML), oder als einzelne Tabelle oder aktualisierbare View (bei der Übertragung von Daten aus XML in die Datenbank). Sie können für die Datenbanktabelle einfach einen DOM-Baum erstellen, indem Sie die Zeilen und Spalten iterativ durchgehen und Knoten erstellen, wenn Sie auf einen Eintrag treffen. Diese Prozedur kann auf die ganze Datenbank angewendet werden oder auf eine Ergebnismenge, die von einer Abfrage geliefert wird. DOM-APIs ermöglichen es Applikationen, im Arbeitsspeicher auf XML-Dokumente zuzugreifen, so dass schon einfache Implementierungen sehr speicherintensiv sein können. Das Speichereffizienzproblem kann behoben werden, indem eine Lazy Node-Konstruktion verwendet wird: ein Knoten im DOM-Baum wird erst dann erstellt, wenn er angefordert wird.

Obwohl DOM-APIs den vollständigen navigatorischen Zugriff auf XML-Dokumente erlauben, besitzt DOM keine APIs zur Abfrage eines Dokuments. Ein Ansatz ist hier die Verwendung von Oracle8*i* interMedia Text (siehe Kapitel 6), um XML-Dokumente zu durchsuchen, die in der Datenbank gespeichert sind. Im nächsten Abschnitt erörtern wir einen neuen Ansatz zur Abfrage der Datenbank, um dynamische XML-Seiten mit SQL-Abfragen aufzubauen.

3.4 XSQL: XSLT/SQL-Serverseiten

Viele Anwendungsentwickler erstellen ihre Geschäftsanwendungen für den Einsatz im Web, da das Internet die starke Nachfrage nach flexiblem Informationsaustausch vorantreibt. Die Entwickler benötigen für dieses Problem standardbasierte Lösungen. SQL, XML und XSLT sind Standards, die diese Aufgabe in der Praxis lösen können.

SQL ist der Standard, den Sie bereits aus Ihren Produktionssystemen kennen. XML ist ein plattformneutrales Industriestandardformat zur Darstellung der Ergebnisse von SQL-Abfragen als Datagramme für den Austausch. XSLT definiert die standardbasierte Umwandlung von XML-Datagrammen in das benötigte Zielformat XML, HTML oder ein anderes Textformat.

Sie können mit dem XSQL-Servlet einfach dynamische XML-Datenseiten aus den Ergebnissen von einer oder mehreren SQL-Abfragen errichten. Die Ergebnisse können dann über das Web als XML-Datagramme oder HTML-Seiten mit serverseitiger XSLT-Transformation bereitgestellt werden. Sie können auch XML-Code erhalten, der auf dem Web-Server vorliegt, und in eine Datenbank einfügen. Das XSQL-Servlet nutzt hierzu den Oracle XML-Parser, die Oracle XSL Transformation Engine und die Oracle XML SQL Utilities.

Durch die leistungsstarke Kombination von SQL, XML und XSLT im Server mit einem überall verfügbaren HTTP-Protokoll für den Transport können Sie

- webbasierte Informationsanforderungen von jedem Clientgerät im Web empfangen.
- die entsprechende logische Sicht auf Geschäftsdaten, die die Anforderung benötigt, abfragen.
- die Datagramme in XML über das Web an die anfordernde Partei zurückliefern.
- die Informationen flexibel in jedes beliebige XML-, HTML- oder Textformat übertragen.

Natürlich enthalten Oracle8*i* und das Oracle XML Developer's Kit die gesamte Kerntechnologie, die Entwickler zur Implementierung dieser Lösung benötigen. Sie können aber auch Oracle XSQL Pages verwenden, was die Verwendung der unterliegenden XML-Technologiekomponenten automatisiert, um die meisten gängigen Fälle ohne Programmierung zu lösen.

Oracle XSQL Pages sind Schablonen, mit denen jeder, der mit SQL vertraut ist, deklarativ

- dynamische XML-Datenseiten zusammenstellen kann, die auf einer oder mehreren parametrisierten SQL-Abfragen basieren.

- die Datenseiten umwandeln kann, um ein Endergebnis in jedem beliebigen XML-, HTML- oder Textformat mit einer dazugehörenden XSLT Transformation zu erstellen.

Beispielsweise ruft die folgende URL-Anforderung eine Liste der verfügbaren Flüge für einen beliebigen Zielort ab, wie in Abbildung 3-4 dargestellt.

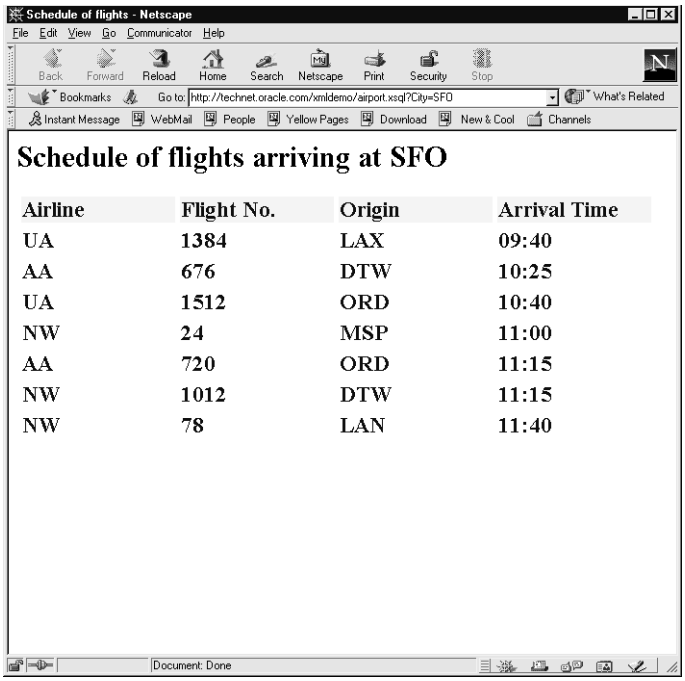

Abbildung 3-4: Screenshot von airpot.sql

```
http://yourcompany.com/AvailableFlightsToday.xsql?City=SFO
```

Sie könnten als Antwort auf die obige URL-Anforderung mit der folgenden XSQL-Seite die Liste aus der unternehmensweiten Datenbank abrufen.

```
<?xml version="1.0"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
    SELECT Carrier, FlightNumber, Origin,
           TO_CHAR(ExpectedTime,'HH:MI') Due
      FROM FlightSchedule
    WHERE TRUNC(ArrivalTime) = TRUNC(SYSDATE)
       AND Destination = '{@City}'
  ORDER BY ExpectedTime
</xsql:query>
```

3.4.1 Architektur von XSQL-Seiten

Das XSQL-Servlet ist ein Werkzeug, das SQL-Abfragen verarbeitet und die Ergebnismenge als XML ausgibt. Dieser Prozessor ist als Java-Servlet implementiert und verwendet eine XML-Datei, die eingebettete SQL-Abfragen als Eingabe verwendet. Es verwendet das XML Developer's Kit zur Durchführung vieler Operationen.

Sie können das Servlet auf jedem Web-Server einsetzen, der Java-Servlets unterstützt. Abbildung 3-5 zeigt, wie die Daten von einem Client zum Servlet und wieder zum Client zurück fließen. Die Abfolge der Ereignisse ist wie folgt:

- Der Benutzer gibt eine URL über einen Browser ein, der das XSQL-Servlet über den Java Web-Server interpetiert und weitergibt. Die URL enthält den Namen der XSQL-Zieldatei (**.xsql**) und optional Parameter wie Werte und einen XSL-Stylesheetnamen. Alternativ kann der Benutzer das XSQL-Servlet aus der Befehlszeile abrufen, und so den Browser und den Java Web-Server umgehen.
- Das Servlet übergibt die XSQL-Datei an den XML Parser for Java, der das XML parst und eine API für den Zugriff auf die XML-Inhalte erstellt.
- Die Seitenprozessorkomponente des Servlets verwendet das API zum Abruf der XML-Parameter und SQL-Anweisungen (die zwischen den Tags <**xsql:query**> und <**/xsql:query**> stehen). Der Seitenprozessor übergibt auch alle XSL-Verarbeitungsanweisungen an den XSLT-Prozessor.
- Der Seitenprozessor erstellt dann eine Datenbank-DOM, indem die SQL-Abfragen an die unterliegende Oracle8*i*-Datenbank gesendet wird, die die Abfrageergebnisse liefert. Die Ergebnisse werden in die XML-Datei an der gleichen Stelle wie die ursprünglichen <**xsql:query**>-Tags eingebettet.
- Wenn gewünscht, werden die Abfrageergebnisse und andere XML-Daten vom XSLT-Prozessor mit einem angegebenen XSL-Stylesheet umgewandelt. Die Daten können dann in HTML oder ein anderes Format umgewandelt werden, das über das Stylesheet definiert wird. Der XSLT-Prozessor kann selektiv verschiedene Stylesheets auf der Basis des Clienttyps anwenden, der die ursprüngliche URL-Anforderung abgesetzt hat. Diese HTTP_USER_AGENT-Information erhält der Client über eine HTTP-Anforderung.
- Der XSLT-Prozessor übergibt das fertiggestellte Dokument zurück an den Client-Browser, zur Anzeige für den Benutzer.

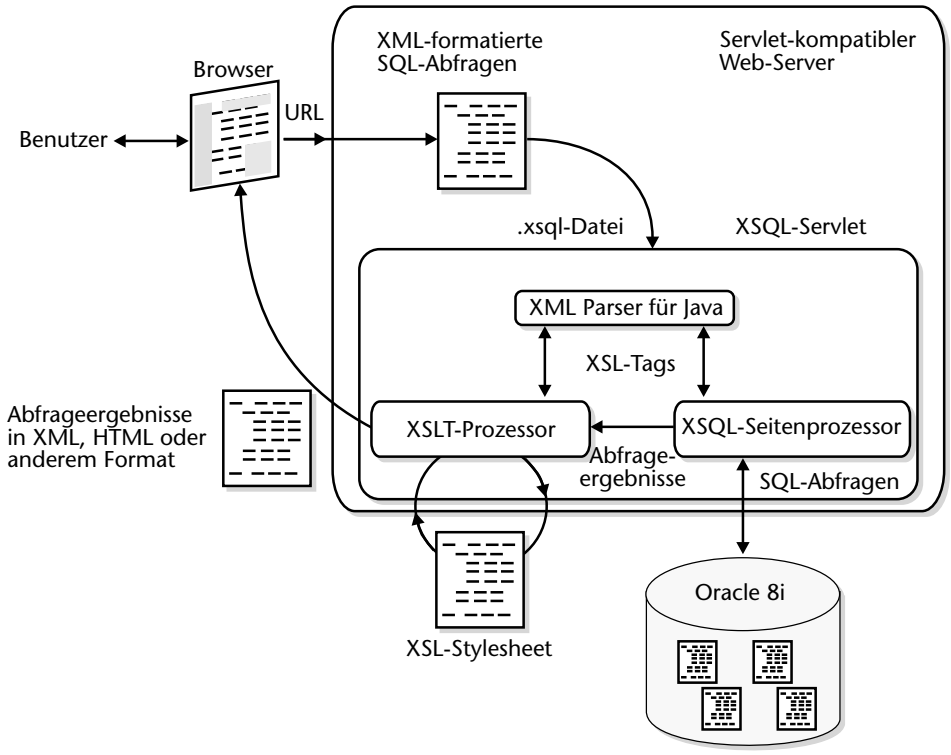

Abbildung 3-5: Architektur von XSQL-Seiten

3.4.2 Installation des XSQL-Servlets

Sie können das XSQL-Servlet auf vielen verschiedenen Web-Servern installieren und konfigurieren, wie Oracle8*i* Lite Web-to-Go Server, Apache 1.3.9 mit JServ 1.0 und dem Sun JavaServer Web Development Kit (JSWDK) 1.0.1 Web Server.

In diesem Abschnitt beschreiben wir kurz die Installationsschritte für den Apache 1.3.9 Web Server. Detaillierte Anleitungen zur Installation des XSQL-Servlets auf vielen verschiedenen Web-Servern finden Sie in Kapitel 8. Zur Installation des XSQL-Servlets führen Sie folgende Schritte durch:

1. Stellen Sie sicher, dass alle JAR-Dateien, die auf dem XSQL-Servlet laufen sollen, im CLASSPATH für die Apache JServ Laufzeitengine stehen. Sie sollten die folgenden Zeilen in die Datei jserv.properties aufnehmen:

```
# Oracle XSQL Servlet
wrapper.classpath=C:\xsql\lib\oraclexsql.jar
# Oracle JDBC (8.1.5)
```

```
wrapper.classpath=C:\xsql\lib\classes111.zip
# Oracle XML Parser V2 (with XSLT Engine)
wrapper.classpath=C:\xsql\lib\xmlparserv2.jar
# XSQLConfig.xml File location
wrapper.classpath=C:\xsql\lib
```

2. Registrieren Sie die Dateinamenserweiterung **.xsql** für die Zuordnung der Java-Servlet-Klasse namens **oracle.xml.xsql.XSQLServlet**. Sie sollten die folgenden Zeilen in die Konfigurationsdatei mod_jserv.conf einfügen:

```
# Executes a servlet passing filename with proper extension
# property of servlet request.
# Syntax: ApJServAction [extension] [servlet-uri]
# Defaults: NONE
# Notes: This is used for external tools.
#ApJServAction .jsp /servlets/nl.nmg.jsp.JSPServlet
ApJServAction .xsql /servlets/oracle.xml.xsql.XSQLServlet
```

Nachdem Sie die Dateierweiterung **.xsql** registriert haben, starten Sie den Web-Server neu und blättern eine XSQL-Datei durch, um die XML-Ausgabe oder die umgewandelte HTML-Ausgabe anzuzeigen.

3.4.3 Dynamische XML-Dokumente aus SQL-Abfragen

Oracle XSQL-Seiten sind XML-Datenseiten mit integrierten SQL-Abfragen zum Abrufen oder Einfügen von Daten. Sie können eine XSQL-Seite erstellen, indem Sie einen <**xsql:query**>-Tag in Ihre XML-Datei an dem Platz einsetzen, an dem der SQL-Code ausgeführt werden soll. Das <**xsql:query**>-Element wird durch das XML-Ergebnis Ihrer Abfrage ersetzt.

Das XSQL-Servlet verwendet eine Konfigurationsdatei XSQLConfig.xml für den Zugriff und die Berechtigungsprüfung der Datenbankverbindung. Ein Beispiel für eine Konfigurationsdatei ist das Folgende:

```
<?xml version="1.0" ?>
<XSQLConfig>
   <connectiondefs dumpallowed="no">
      <connection name="demo">
      <username>scott</username>
      <password>tiger</password>
      <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
      <driver>oracle.jdbc.driver.OracleDriver</driver>
    </connection>
  </connectiondefs>
   :
 </XSQLConfig>
```

Sie können weitere Verbindungselemente definieren, indem Sie verschiedene Benutzer bestimmen oder verschiedene JDBC-Treiber verwenden. Das XSQL-Servlet erwartet ein Attribut namens „connection“ im Root-Element Ihres XML-Dokuments, dessen Wert der Name einer Verbindung sein muss, die in Ihrer Konfigurationsdatei definiert ist. Die einfachste Verwendung des Tags <**xsql:query**> ist:

```
<?xml version="1.0"?>
 <xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo">
      SELECT 'Hello World' AS "GREETING" FROM DUAL
 </xsql:query>
```

Die obige XSQL-Seite erzeugt das dynamsich erstellte standardmäßige XML-Dokument:

```
<?xml version = '1.0'?>
 <ROWSET>
    <ROW id="1">
        <GREETING>Hello World</GREETING>
    </ROW>
 </ROWSET>
```

Sie können das kanonische XML-Dokument in eine andere XML-Form oder in HTML umwandeln. Sie können ein XSL-Stylesheet in Ihrer XSQL-Seite über eine Verarbeitungsanweisung verknüpfen:

```
<?xml-stylesheet type="text/html" href="transform.xsl"?>
```

Sie können objektrelationale Abfragen in XSQL-Seiten verwenden. Sie können auch Parameter an diese Seiten mit der URL senden. Beispielsweise können Sie die mit den objektrelationalen Einrichtungen von Oracle8*i* einen benutzerdefinierten Objekttyp namens POINT erstellen. Sie können den neuen POINT-Typ als Datentyp der Spalte ORIGIN in Ihrer Tabelle LOCATION mit den folgenden DDL-Anweisungen einsetzen:

```
CREATE TYPE POINT AS OBJECT (X NUMBER, Y NUMBER);

CREATE TABLE LOCATION (
 NAME VARCHAR2(80),
 ORIGIN POINT
);
```

Sie können eine Zeile in diese LOCATION-Tabelle durch eine INSERT-Anweisung mit dem POINT()-Konstruktor einfügen:

```
SQL> INSERT INTO LOCATION VALUES ( 'Someplace', POINT(11,17) );
SQL> COMMIT;
```

Dann können Sie eine XSQL-Seite wie die folgende **point.xsql** verwenden, um die Tabelle LOCATION mit einem Parameter x-coord abzufragen.

```
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  SELECT name, origin
    FROM location loc
  WHERE loc.origin.x = {@x-coord}
</xsql:query>
```

Sie können mit der folgenden URL alle Positionen mit dem Wert x-coord = 11 abfragen.

```
http://yourmachine.com/xsql/demo/point.xsql?x-coord=11
```

```
<ROWSET>
  <ROW num="1">
     <NAME>Someplace</NAME>
     <ORIGIN>
         <X>11</X>
         <Y>17</Y>
     </ORIGIN>
  </ROW>
</ROWSET>
```

Dies zeigt, wie die verschachtelten *X*- und *Y*-Attribute in der POINT-Datentypstruktur der Spalte ORIGIN automatisch als verschachtelte <**X**>- und <**Y**>-Elemente in der XML-Ausgabe angezeigt werden.

3.4.4 Unterstützung von XSLT-Tags in XSQL

Oracle XSQL-Seiten sind Schablonen, mit denen Sie dynamische XML-Datenseiten auf der Basis einer oder mehrerer parametrisierter SQL-Abfragen zusammenstellen können. Der Prozessor verwendet dann XSLT zur Übertragung der Datenseiten, um das Endergebnis in jedem gewünschten XML-, HTML- oder Textformat zu liefern. Sie können aber auch mit XSLT XML-Datenseiten zusammenstellen. Beispielsweise können Sie mit XSLT SQL-Abfragen ausführen, die entweder <**xsl:choose**> oder <**xsl:if**> verwenden. Die folgende XSQL-Seite zeigt die Verwendung von **xsl:if** beim Abruf einer Zusammenfassung oder von Detailinformationen zur Bücherliste:

```
<?xml version="1.0"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:if test="$detail = 'yes'">
    SELECT * FROM scott.BookList
 </xsl:if>
 <xsl:if test="$detail = 'no'">
    SELECT Title, Authorname FROM scott.BookList</xsql:query>
 </xsl:if>
</xsql:query>
```

3.5 Das Beispiel Buchladen

In diesem Abschnitt wird die Erstellung einer Website mit XSQL-Seiten behandelt, wobei Oracle8*i* als Daten-Repository zum Einsatz kommt. In diesem Abschnitt werden die in diesem Kapitel erörterten Konzepte eingesetzt, um ein „Praxisbeispiel" zu illustrieren, das auf dem einfachen Geschäftsvorgang der Verwaltung eines Online-Buchkatalogs basiert. Die Schritte vom Entwurf bis zur Implementierung zeigen Ihnen, wie Sie XML-fähige Websites erstellen können.

3.5.1 Entwurf des Datenbankschemas

Das Ziel ist die Entwicklung einer einfachen Website für die Verwaltung eines Buchkatalogs. Als Erstes müssen Sie das Datenmodell festlegen, um den Katalog in der Datenbank zu speichern. Das Datenmodell, das in Abbildung 3-6 gezeigt wird, illustriert die Grundobjekte – Bücher, Autoren, Kritiken – die in diesem Beispiel verwendet werden.

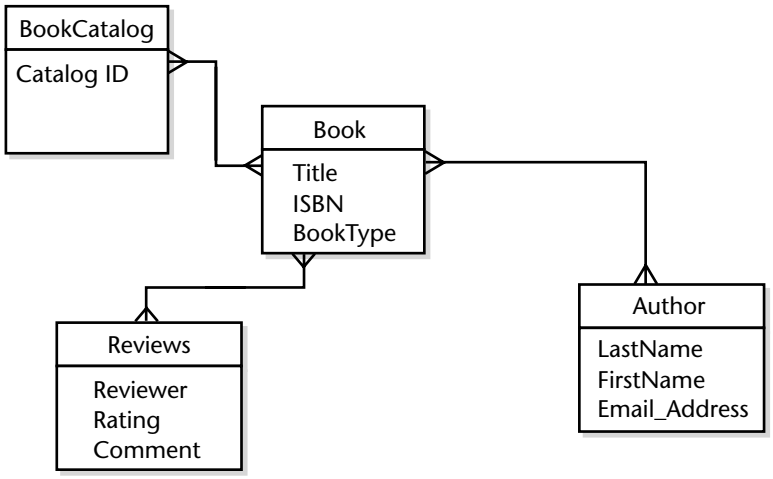

Abbildung 3-6: Datenmodell für den Buchkatalog

Eine Buch besitzt eine N:N-Beziehung mit dem Autor, da ein Buch viele Autoren haben kann (mindestens einen), und jeder Autor viele Bücher verfasst haben kann. Ein Buch besitzt außerdem eine 1:N-Beziehung zu Kritiken. Diese Beziehung ist optional, da ein Buch auch keine Kritiken bekommen kann. Dieses Datenmodell kann in folgende DTD übersetzt werden:

```
<!-- DTD for Book Data -->
<!ELEMENT BOOKCATALOG (BOOK)*>
<!ELEMENT BOOK (TITLE, AUTHOR+, PUBLISHER?, PUBLISHYEAR?, PRICE?,
                REVIEWS*)>
<!ATTLIST BOOK ISBN CDATA #REQUIRED>
<!ATTLIST BOOK BOOKTYPE (Fiction|SciFi|Fantasy) #IMPLIED>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT AUTHOR (LASTNAME, FIRSTNAME, EMAIL_ADDRESS)>
<!ELEMENT LASTNAME (#PCDATA)>
<!ELEMENT FIRSTNAME (#PCDATA)>
<!ELEMENT EMAIL_ADDRESS (#PCDATA)>
<!ELEMENT PUBLISHER (#PCDATA)>
<!ELEMENT PUBLISHYEAR (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
<!ELEMENT REVIEWS (REVIEWER, RATING, COMMENTS)>
<!ELEMENT REVIEWER (#PCDATA)>
<!ELEMENT RATING (#PCDATA)>
<!ELEMENT COMMENTS (#PCDATA)>
```

Das XML-Dokument (der Buchkatalog), der zur obigen DTD kompatibel ist, kann in der Datenbank auf verschiedene Arten gespeichert werden. Das Dokument kann gespeichert werden als CLOB, und interMedia kann zur Suche und für den Abruf von Informationen aus dem XML-Dokument verwendet werden. Das CLOB-Speichermodell hat eine Einschränkung. Bei Aktualisierungen verwendet ein CLOB immer das vollständige Dokument. Ein anderer Ansatz, um ein XML-Dokument in der Datenbank zu speichern, ist, die DTD einem Datenbankschema zuzuordnen. Ein einfaches Schema für die obige DTD würde drei Objekte enthalten: eine BOOK-Tabelle, ein AUTHOR-Objekt und ein REVIEW-Objekt. Die BOOK-Tabelle enthält die Listenbücher und zwei verschachtelte Tabellen: eine für die Liste der Autoren und die andere für die Liste der Kritiken.

Nachdem Sie das XML-Dokument in die Datenbank geladen haben, müssen Sie das XSQL-Servlet und die XDK-Komponenten auf dem mittleren Web-Server installieren, und über JDBC mit der Datenbank kommunizieren. Schließlich müssen Sie Ihre Website mit XSQL-Dateien erstellen, um XML-Dokumente abzurufen und diese in HTML-Ausgaben umzuwandeln.

3.5.2 Entwurf der Website mit XSQL

Sie können eine Website mit JavaScript, Java Server Pages und Dynamic HTML auf viele verschiedene Arten entwerfen und erstellen. In diesem Abschnitt zeigen wir einen einfachen Entwurf mit reinem HTML. Diese Schritte illustrieren die Verwendung der XSQL-Seiten zum Abruf der Daten aus der Datenbank. Diese Seiten können als Bausteine in einem fortgeschritteneren Website-Design dienen. Wir zeigen zwei Webseiten aus den Daten in der Tabelle BOOKCATALOG:

- Eine Seite, die eine Liste aller Bücher im Katalog enthält.
- Eine Seite, die eine detaillierte Ansicht eines Buchs enthält, das über die ISBN-Nummer bestimmt wurde.

Zwei Schritte sind beim Einrichten einer Webseite wichtig. Als Erstes müssen Sie entscheiden, welche Daten für die Webseite unter Verwendung der XSQL-Datenseite erforderlich sind. Dann müssen Sie entscheiden, welche HTML-Formatierung Sie für eine XSLT-Umwandlung nutzen. Im Fall eines fortgeschritteneren Webseiten-Entwurfs können Sie XSLT benutzen, um die Datenseite in Java-Script zu übertragen. Sie können auch die Datenseite in Java Server Pages aufnehmen und DOM-APIs benutzen, um die Daten abzurufen.

Die Katalogansicht

Für die Anzeigeseite des Katalogs müssen Sie nicht die vollständige Information zu jedem Buch haben. Sie können nur diejenigen Attribute des Buchs auswählen, die Sie in der Katalogansicht darstellen möchten. Sie können die ISBN-Nummer, den Titel, den Nachnamen des Autors, den Verlag und den Preis mit den folgenden Datenseiten auswählen:

```
?xml version="1.0"?>
<?xml-stylesheet type="text/html" href="catalog.xsl"?>
<xsql:query xmlns:xsql="urn:oracle-xsql"
            connection="demo"
            rowset-element="BOOKCATALOG"
            row-element="BOOK">

  SELECT ISBN, Title, Author.Lastname, Publisher, Price
    FROM scott.BookCatalog

</xsql:query>
```

Die obige XSQL-Seite liefert ein XML-Standarddokument, das die Liste der Bücher enthält. Der XSQL-Prozessor wandelt das XML-Dokument mit dem XSLT-Stylesheet „catalog.xsl“ um, das mit der Verarbeitungsanweisung des XML-Stylesheets verknüpft ist.

```
<?xml version="1.0"?>
<BOOKCATALOG>
  <BOOK>
    <ISBN>1234-123456-1234</ISBN>
    <TITLE>C Programming Language</TITLE>
    <AUTHOR_LASTNAME>Kernighan</AUTHOR_LASTNAME>
    <PUBLISHER>EEE Editions</PUBLISHER>
    <PRICE>7.99</PRICE>
  </BOOK>
```

```
  <BOOK>
    <ISBN>3456-34567890-3456</ISBN>
    <TITLE>C++ Primer</TITLE>
    <AUTHOR_LASTNAME>Lippmann</AUTHOR_LASTNAME>
    <PUBLISHER>McGraw Hill</PUBLISHER>
    <PRICE>4.99</PRICE>
  </BOOK>
  <BOOK>
    <ISBN>2137-598354-65978</ISBN>
    <TITLE>Twelve Red Herrings</TITLE>
    <AUTHOR_LASTNAME>Archer</AUTHOR_LASTNAME>
    <PUBLISHER>Harper Collins</PUBLISHER>
    <PRICE>12.95</PRICE>
  </BOOK>
  <BOOK>
    <ISBN>237864-4787834-3459</ISBN>
    <TITLE>The Eleventh Commandment</TITLE>
    <AUTHOR_LASTNAME>Archer</AUTHOR_LASTNAME>
    <PUBLISHER>Harper Collins</PUBLISHER>
    <PRICE>3.99</PRICE>
  </BOOK>
  <BOOK>
    <ISBN>1230-23498-2349879</ISBN>
    <TITLE>Emperor's New Mind</TITLE>
    <AUTHOR_LASTNAME>Penrose</AUTHOR_LASTNAME>
    <PUBLISHER>Oxford Publishing Company</PUBLISHER>
    <PRICE>15.99</PRICE>
  </BOOK>
</BOOKCATALOG>
```

Sie können ein einfaches Stylesheet wie das Folgende verwenden, um das erzeugte XML-Dokument in eine HTML-Tabelle umzuwandeln. Das Ergebnis der Konvertierung zeigt Abbildung 3-7.

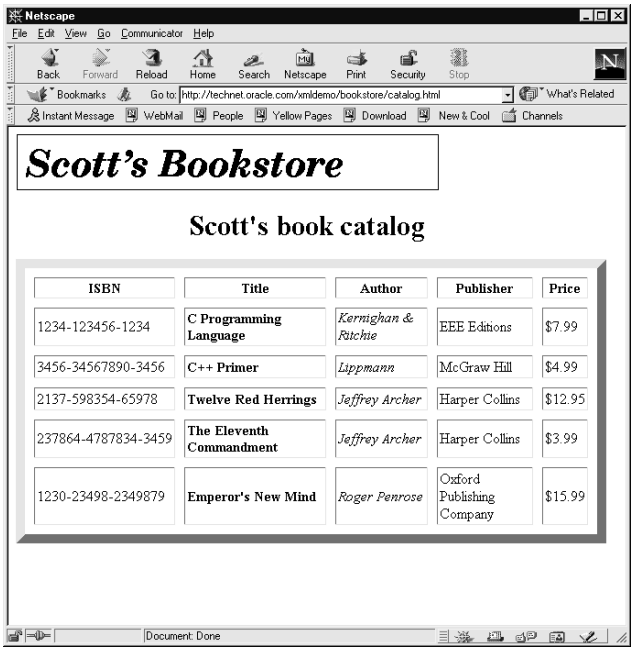

Abbildung 3-7: Screenshot des Buchkatalogs

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
 <HTML>
   <body bgcolor="#FFFFFF" text="#000000">
   <img align="center" src="mylogo.gif" border="0"/>
   <h1 align="center"> Scott's book catalog </h1>
   <xsl:apply-templates/>
   </body>
 </HTML>
</xsl:template>

<xsl:template match="BOOKCATALOG">
  <table align="center" border="10" cellspacing="10" cellpadding="2">
    <th>ISBN</th>
    <th>Title</th>
    <th>Author</th>
    <th>Publisher</th>
    <th>Price</th>
    <xsl:apply-templates select="BOOK"/>
  </table>
</xsl:template>
```

```
<xsl:template match="BOOK">
 <tr><xsl:apply-templates/></tr>
</xsl:template>

<xsl:template match="TITLE">
 <td><b><xsl:apply-templates/></b></td>
</xsl:template>

<xsl:template match="AUTHOR_LASTNAME">
 <td><i><xsl:apply-templates/></i></td>
</xsl:template>

<xsl:template match="PUBLISHER">
 <td><xsl:apply-templates/></td>
</xsl:template>

<xsl:template match="ISBN">
 <td><xsl:apply-templates/></td>
</xsl:template>

<xsl:template match="PRICE">
 <td>$<xsl:apply-templates/></td>
</xsl:template>

</xsl:stylesheet>
```

Detailansicht

Sie müssen weitere Informationen abrufen, um eine Detaildarstellung des Buchs anzuzeigen. Sie können auch alle Informationen abrufen und mit der XSLT-Transformation die erforderlichen Buchattribute formatieren. Die folgende XSQL-Datenseite ruft alle Attribute des Buchs mit dem Parameter **isbn-no** ab:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/html" href="catalog.xsl"?>
<xsql:query xmlns:xsql="urn:oracle-xsql"
            connection="demo"
            rowset-element="BOOKCATALOG"
            row-element="BOOK">

    SELECT *
      FROM scott.BookCatalog
     WHERE ISBN = {@isbn-no}

</xsql:query>
```

Das folgende XML-Dokument wird von der XSQL-Datenseite abgerufen und kann dann als HTML-Seite wie in Abbildung 3-8 formatiert werden.

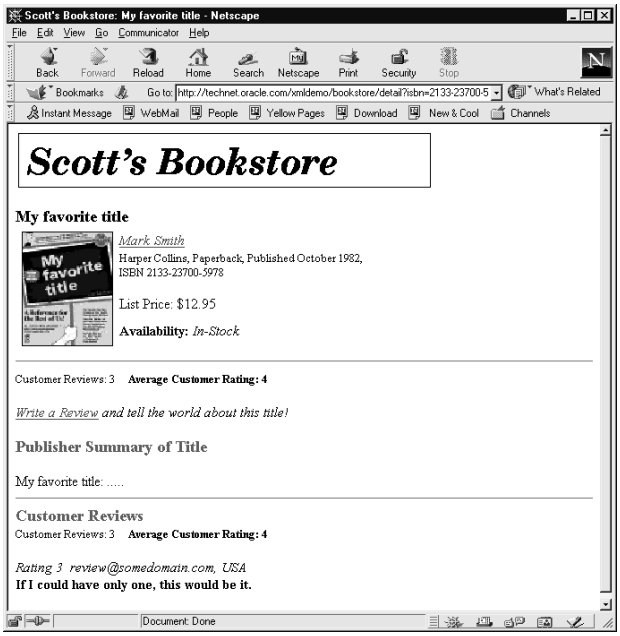

Abbildung 3-8: Screenshot der Buchdetails

```
<?xml version="1.0"?>
<BOOKCATALOG>
 <BOOK>
   <ISBN>2133-23700-5978</ISBN>
   <TITLE>My favorite title</TITLE>
   <AUTHOR>
       <LASTNAME>Smith</LASTNAME>
       <FIRSTNAME>Mark</FIRSTNAME>
       <EMAIL>msmith@mydomain.com</EMAIL>
    </AUTHOR>
    <BOOKTYPE>Fiction</BOOKTYPE>
    <PUBLISHER>Harper Collins</PUBLISHER>
    <PUBLISHYEAR>1982</PUBLISHYEAR>
    <PRICE>12.95</PRICE>
    <REVIEW_LIST>
  <REVIEW>
    <REVIEWER>review@somedomain.com</REVIEWER>
    <RATING>3</RATING>
    <COMMENTS>
    ...
    </COMMENTS>
  </REVIEW>
    ...
    <REVIEW_LIST>
 </BOOK>
</BOOKCATALOG>
```