WISH XML Query Composer

Wei Hua Liang

A Thesis

in

The Department of Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

August 2003

Your file Votre référence
ISBN: 0-612-83911-7
Our file Notre référence
ISBN: 0-612-83911-7

ABSTRACT

WISH XML Query Composer

Wei Hua Liang, M.Sc.
Concordia University, 2003

WISH XML Query Composer is a tool for composing form-based queries and their associated reports for relational databases. This tool incorporates two industry standards to generate user-friendly customizable queries and reports. The *Structured Query Language* (SQL) has been a universal standard for accessing data in relational databases since 1989. The Extensible Markup Language (XML) is a relatively new standard documented by World Wide Web Consortium (W3C) that supports data exchange on the World Wide Web. Today, many different Relational Database Management Systems (RDBMS) utilize SQL as the primary means for accessing and manipulating data. The design of the WISH XML Query Composer is an attempt to use the very simple but flexible XML semantics to represent database schemas, SQL queries and result datasets, regardless of in what RDBMS the data is stored.

This thesis describes the design and implementation of the WISH XML Query Composer. The tool is written in Java programming language with Swing components, and connects to the database through Java Database Connectivity (JDBC). Java Architecture for XML Binding (JAXB) is used to automate the mapping between XML documents and Java objects.

ACKNOWLEDGEMENTS

I would like to take this opportunity to give grateful thanks to my supervisor, Professor Greg Butler, for his supports and guidance throughout the this research project. This work would not have been completed without his concise instructions and flexible supervision.

Thanks to the Graduate Program Director and all the instructors who have taught or helped me during my studies at Concordia University, especially Professor Joey Paquet in Software Design Methodology, Professor Ahmad Seffah in Human Computer Interface Design, Professor Lixin Tao in Internet and Java Programming and Professor Peter Grogono for his generous help and encouragement. Special thanks to Halina for her smiles and help that I have received during the years.

I would also like to express appreciations to my husband Fei Liao for his endless support and patience; my parents for setting high expectations and being consistently serious about my education; my friend Sylvain Berdard, for his inspiration and confidence in me so that I made the decision of pursuing a Master's Degree at Concordia University; my schoolmates Ms. Yuanyao Xiong, Mr. Xuede Chen, Ms. Qixia Deng and Ms. Liqian Zou, for their help in my studies; to my partners in Market America for their understanding and moral support; and to my employers, CGI and Harlequin, for their financial support.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1 The Problem

Databases today are essential to every business or scientific investigation. They are used to maintain records, to present data to the public on the World-Wide-Web, and to support many other processes. The power of databases comes from a body of knowledge and technology that has developed over serveral decades and is embodied in specialized software called a Database Management System (DBMS). One of the most common operations a DBMS supports is database query, an operation to extract specified data from databases. [DSI01] For this purpose, database query languages are developed and integrated into the user interfaces of DBMS's. Of which the most popular are Structured Query Language(SQL) for relational databases and Object Query Language (OQL) for object-oriented databases.

Generally, there are three ways to access and query a relational database, as shown in Table 1.1 below. Each access method has its intended audience:

User	*Database Access*	*Example*
Database Administrator (DBA)	Writes SQL statement through DBMS built-in interface	Query Analyzer of Microsoft SQL Server
Application Developer	Writes SQL statement through programming interface	Java Database Connectivity (JDBC)
End User	Runs pre-programmed fixed SQL statement through application interface	Database driven web sites

Table 1.1: General Ways to Access Database

Today, most database applications embed pre-programmed query statements, hence, the end users do not have the flexibility to specify their own search criteria. The use of query languages are somehow restricted to Database Administrators (DBA) and application developers, because, in order to query a database, one needs the database model, data dictionary or schema, and the following details:

- Method used to connect to the database
- Name of the database and the tables
- Column names and the meanings of the columns
- Names of the key columns and information on how to use keys to join the data from several tables
- Syntax of the query language

For a user with no programming background or database knowledge, it is a tedious task to query a database with specific criteria. Let alone there are so many different DBMS vendors in the market, each have their own query interface and extensions to the standard SQL syntax.

This is the main motivation of WISH XML Query Composer, hereafter refered to as "WISH". Aiming to help novice users perform search as they wish, the name WISH stands for "With Intuitive Search Help".

Another major goal of this project is to use the eXtensible Markup Language (XML) as the data interface of databases. A versatile markup language, XML is capable of labeling the information content of diverse data sources including structured and semi-structured documents, relational databases, and object

repositories. XML acts like the ASCII of the Internet, allowing interaction between hosts regardless of their operating systems, database managers, or data formats. Hence, a query tool that uses the structure of XML intelligently can express queries across all kinds of data.

XML allows the adding of metadata, in forms of tags, to the content of documents. Consequently sophisticated searching is allowed in XML documents, specifying not only the search string but also the type of content it represents. For these kind of searches, the XML Query (XQuery) is being developed by the W3C as the industry standard. At the time of this writing, the XML Query Requirements has just been released (June 27, 2003). [XQL03] Through the translation between XML and existing database objects, however, existing query utilities such as SQL can be used to achieve the goal.

1.2 Project Goal

The objective of this project is to develop a tool for composing form-based queries and their associated reports for relational databases. The tool is written in Java, and use XML to communicate with the database(s). The design of the tool should adhere to existing industry standards, such as XML standards for schemas, queries, and results/tables. Whether the tool can really communicate to a database like MySQL depends on the facilities of the database. The initial version does not have to actually connect to databases to demonstrate the functionality and effectiveness of the tools.

1.3 Contribution

To come up with a solid design, intensive researches have been carried out on industry standards, including the ANSI/ISO SQL-92 Standard and Transact-SQL, the W3C XML 1.0, XQuery 1.0, DTD, XML Schema and Extensible Stylesheet Language (XSL) 1.0. [TSP99, XML00, XQL03, XSP01, XSL99] Furthermore, many existing commercial and scientific database query tools and XML-database transformation tools are examined and experimented, as outlined in Table 1.2 below:

Tool	*Origin*	*Reference*
XML Light-weighted Extractor	IBM alphaWork	[XLE01]
XML to Object Oriented Database Translation System	University of Newcastle	[XTS00]
Visual Object-Oriented Database for ODMG OQL	University of Texas	[VOO99]
Semi-structured Object Database	University of New South Wales	[SOD00]
Aqua Data Studio	AquaFold	[ADS03]
DB-XML Vision	DataMirror	[DXV02]
Dynamo Control Center	ATG	[DCC00]
SQL Server Query Analyzer	Microsoft	[SQA00]

Table 1.2: Some Database Query and Transformation Tools

One of the great strengths of XML is its flexibility in representing many different kinds of information from diverse sources. RDBMS has been chosen as the database model of this project because of its popularity and the well-adopted

SQL standard for querying data. The object-oriented database model is also studied but, due to time constraints, is not implemented in this version of WISH.

WISH is designed as a result of the research mentioned above. Besides browsing and reporting database metadata and data, it provides an easy-to-use form-based query tool for complex data retrieval and publishing, in XML and HTML formats designed by the advanced user. The original objectives have been met and the implemented solution even actually connects to databases through JDBC drivers / JDBC-ODBC bridge.

WISH implements several key Java Technologies: JDBC for database connectivity, Swing for Graphic User Interface (GUI) design, and of most importance, Java Architecture for XML Binding (JAXB) for mapping between XML documents and Java classes/objects.

1.4 Organization of the Thesis

This thesis presents the design and verification of WISH in its different phases. Chapter 2 describes the major related topics, namely relational databases, SQL queries, form-based queries, XML basics, Java and JAXB. Chapter 3 contains the details of XML definitions, classes and GUI design for the three components of the WISH: Schema Tree, Form Composer and Query Executer; furthermore, test cases are presented for validation and testing purpose. Chapter 4 concludes by discussing the usefulness of WISH and the future work.

2. Background

2.1 Relational Databases

These days, Relational Database Management Systems (RDBMS) are the primary engines of information systems everywhere. The primary reason why relational database systems win out over hierarchical and network database systems is their simplicity. Most RDBMS vendors utilize the Structured Query Language (SQL) for accessing data in a database. This relational high-level language interface is much simpler to learn and more intuitive than non-relational databases.

A relational database is a system whose users view data as a collection of tables related to one another through common data values. In a relational database, data is modeled by *tables (also called relations)*, which are composed of *rows* and *columns*. Data items are rows of the tables, and rows have a fixed number of components, each of a fixed type determined by the table's schema. Column headers, called *attributes*, represent the meaning of each component of the rows. The table name and its attribute names and types are the *schema* for the table. Tables of independent data can be linked, or *related*, to one another if all have columns of data that represent the same data value, called *keys*. [DSI00, TSP99]

A *database schema* is a collection of table schemas. A relational database schema is often illustrated in a database Entity-Relational (E-R) diagram. In an E-R diagram, each table schema (also called entity) is shown in a box, with its

name as heading and the attributes as remaining fields; the entities are linked with lines to show their relationships.

In this thesis, relational databases are the main object of data queries. The example data model is demonstrated by an ER-diagram.

2.2 SQL Queries

The Structured Query Language, universally referred to by its acronym, SQL, is a language for interacting with relational databases. SQL traces its origins to IBM in the late 1970s, but the main version in use today is SQL-92, which is published in 1992 by both American National Standards Institute (ANSI) and International Organization for Standardization (ISO). The standards covered lots of important details concerning the querying and manipulation of data, and helped formalize many of the behaviors and syntax structures of SQL.

SQL has a large number of capabilities, including statements that query the database. Queries are generally expressed with a "select-from-where" statement, which has the general form shown in Figure 2.1. Only the first two lines, the clauses introduced by the keywords SELECT and FROM, are required. The clauses in a SELECT statement must be specified in the proper order.

SELECT *attribute_list*
FROM *table_source*
[WHERE *search_condition*]
[GROUP BY *attribute_list*]
[HAVING *search_condition*]
[ORDER BY *attribute_list* [ASC | DESC]]

Figure 2.1: General Form of a SQL Query

The full syntax of the SELECT statement is complex, but the main clauses can be summarized as:

1. SELECT *attribute_list*: Describes the columns of the result set. It is a comma-separated list of expressions. Each expression defines both the format (data type and size) and the source of the data for the result set column. Each select list expression is usually a reference to a column in the source table or view the data is coming from, but can be any other expression, such as a constant or a Transact-SQL function. Using the * expression in a select list specifies that all columns in the source table are returned.

2. FROM *table_list:* Contains a list of the tables from which the result set data is retrieved. These sources can be base tables or views. The FROM clause can also contain join specifications, which define the specific path in navigating from one table to another.
3. WHERE *search_conditions:* A filter that defines the conditions each row in the source tables must meet to qualify for the SELECT. Only rows that meet the conditions contribute data to the result set. Data from rows that do not meet the conditions are not used.
4. GROUP BY *attribute_list:* Partitions the result set into groups based on the values in the columns of the *group_by_list*.
5. HAVING *search_conditions:* An additional filter that is applied to the result set. HAVING clauses are most commonly used with a GROUP BY clause, although a GROUP BY clause is not required before a HAVING clause.

6. ORDER BY *order_list* [ASC | DESC]: Defines the order in which the rows in the result set are sorted. *order_list* specifies the result set columns that make up the sort list. The ASC and DESC keywords are used to specify if the rows are sorted in an ascending or descending sequence. ORDER BY is important because relational theory specifies that the rows in a result set cannot be assumed to have any sequence unless ORDER BY is specified. ORDER BY must be used in any SELECT statement for which the order of the result set rows is important.

Each reference to a database object must be unambiguous. The tables and views specified in the FROM clause may have duplicate column names. It is especially likely that foreign keys will have the same column name as their related primary key. To resolve the ambiguity between duplicate names, the column name must be qualified with the table or view name.

One of the powerful features that SQL provides is the ability to use sub-queries within a WHERE clause. A sub-query is a complete select-from-where statement whose selected value is tested in the WHERE clause.

In this thesis, the queries eventually submitted to the relational databases must support the SQL features stated above.

2.3 Form-based Queries

For many people it is much easier to express a query visually than to write a set of statements in a language such as SQL. This is the motivation for the

development of a visual analog of SQL. Visual interfaces were introduced to help casual/novice users when querying databases. Visual query interfaces may be classified into form-based and graph-based. [WQL00]

Form-based interfaces permit to build queries from forms, which represent the queried database schema. The advantage of form-based queries is that many users are comfortable with forms and tables. Because the schemas of the tables are visible on the screen, the users do not have to remember column names. In a form-based interface, the queries are restricted to forms or documents that can be filled in various ways (e.g. the Query By Example method of database querying [QBE77]). These forms visualize tables for relational databases, nested tables for extended relational databases and classes for object-oriented databases. In some form-based query interfaces, users have to express selections, projections and joins. But in most of the form-based and graph-based query languages, query formulation only consists in condition expression because joins are already expressed. [WQL00]

Graph-based query interfaces provide a graph representation of both data and queries. Queries are graph patterns. Edges in queries represent edges or paths in the database. Regular expressions are used to qualify these paths.

This thesis concentrates on the form-based query interface. More in-dept discussions on graph-based query languages and tools can be found in other publications of the Know-It-All Framework [KIA02, DQL01], which is undertaken by Dr. Butler and his graduate students.

2.4 XML Basics

This section starts with an overview of XML and the reasoning behind generating XML from relational data, then continue on to discuss its definitions and representations. This is not intended to be complete, but it does cover all of the XML features used in the following chapters.

2.4.1 XML

The eXtensible Markup Language (XML) is a simple, very flexible text format derived from Standard Generalized Markup Language (SGML). The term XML is also used to refer to data that's marked up in a format defined using XML. [XDT00]

XML is based on the ISO standard language SGML. This means that XML, like SQL, is not vendor-specific, and is therefore likely to remain in use for a long time. XML is defined by the World Wide Web Consortium (W3C) as a subset of SGML, intended to be easier for programmers to implement.

As a meta-language, XML is used to describe the structure of data. No matter where the data is input or output, stored, or transmitted from one place to another, it can benefit from the application of XML. XML is a way of defining simple text-based representations of arbitrarily complex structured information. With the emergence of XML, it is now feasible to construct intelligent Internet agents that can interact reliably with multiple, diverse hosts. Therefore, XML is especially

valuable in web development to supplement traditional databases, and in the transfer of data between businesses or organizations.

An XML document contains one or more elements. An element consists of a start tag and an end tag delimiting the boundaries of the content, or just an empty tag. Each element has a type, identified by its tag name, and may have a set of attributes. The attributes, in name-value pairs, are specified in the start tag or the empty tag. Here is a quick example (Author.xml):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Author id = "1">
   <FirstName>Diane</FirstName>
   <LastName>Palmer</LastName>
   <Birthday year="1948" month="September" day="1" / >
   <WebSite>www.DianePalmer.com</WebSite>
   <Introduction>She is good</Introduction>
</Author>
```

Figure 2.2: XML Example

In the example above, there are three kinds of information:

- The beginning XML declaration identifies the document as an XML document. It contains information about the version of XML used, and possibly the encoding and the existence of an underlying DTD within the document.
- The author's names and web site are examples of textual information.
- The blanketed <Introduction> and </Introduction> are examples of start and end tags that identify an element and the meaning of the textual information.

This example shows six (6) different elements. The element with the type Author has an attribute id and the other five elements as its content. In contrast, the

empty element with the type Birthday has no content but three attributes named year, month and day with their respective values.

There are two important terms to describe the quality of an XML document: “well-formed” and “valid”. An XML documents is "well-formed" if there is exactly one root element, and every sub-element (and recursive sub-elements) have delimiting start- and end-tags, and the elements are properly nested within another. On the other hand, a *valid* document is "well-formed" and conforms to a specified set of production rules, which can be provided in either a DTD or XML Schema.

2.4.2 DTD and XML Schema

The tags used in an XML document can be declared in a Document Type Definition (DTD) or XML Schema.

Document Type Declaration (DTD) is used to define the legal building blocks of any SGML-based document. It defines the document structure with a list of legal elements. It calls for elements to consist of one of three things:

- A text string
- A text string with other child elements mixed together
- A set of child elements

DTD has been used for SGML and HTML since the 1970's, and was the primary validation method for most early XML implementations. However, DTD has

certain limitations when defining XML. A DTD does not have XML syntax and offers only limited support for types or namespaces. The new XML Schema system, a W3C recommendation since 2001, aims to provide a rich grammatical structure for XML documents that overcomes the limitations of DTD. The XML Schema Language is much richer than DTD. For example, schemas written in the XML Schema Language can describe structural relationships and data types that can't be expressed (or can't easily be expressed) in DTD.

An XML Schema is an XML document that defines the valid format of an XML dataset. This definition includes:

- what elements are (and are not) allowed at any point;
- what the attributes for any element may be;
- the number of occurrences of elements;
- the data type or possible values of elements and attributes, etc.

Figure 2.3 below shows the XML Schema (Author.xsd) for the XML example in Figure 2.2.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:annotation>
    <xsd:documentation xml:lang="en">
          XML Schema of Author example
    </xsd:documentation>
   </xsd:annotation>

   <xsd:element name="Author" type="AuthorType"/>

   <xsd:complexType name="AuthorType">
    <xsd:sequence>
     <xsd:element name="FirstName" type="xsd:string"/>
     <xsd:element name="LastName" type="xsd:string"/>
     <xsd:element name="Birthday" type="BirthdayType"/>
     <xsd:element name="Website" type="xsd:string" minOccurs="0"
```

```
    maxOccurs="unbounded"/>
      <xsd:element name="Introduction" type="xsd:string" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:string" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="BirthdayType">
    <xsd:attribute name="year" type="xsd:gYear"/>
    <xsd:attribute name="month" type="MonthType" use="required"/>
    <xsd:attribute name="day" type="xsd:gDay"/>
  </xsd:complexType>

  <xsd:simpleType name="MonthType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="January"/>
      <xsd:enumeration value="February"/>
      <xsd:enumeration value="March"/>
      <xsd:enumeration value="April"/>
      <xsd:enumeration value="May"/>
      <xsd:enumeration value="June"/>
      <xsd:enumeration value="July"/>
      <xsd:enumeration value="August"/>
      <xsd:enumeration value="September"/>
      <xsd:enumeration value="October"/>
      <xsd:enumeration value="November"/>
      <xsd:enumeration value="December"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:schema>
```

Figure 2.3. XML Schema Example

At the beginning of the XML schema is the schema element. It holds the definition of the target namespace. This schema defines <Author> as the root element that has a complex type named AuthorType. This means that it has child elements (such as <FirstName> and <Birthday>) and/or attributes (in this case, id). Each <Birthday> element also has a complex type named BirthdayType, which in turn has three attributes (year, month and day). The month attribute is of a restricted string type, defined as simple type name MonthType.

2.4.3 XSL

The Extensible Stylesheet Language (XSL) is a family of W3C recommendations for defining XML document transformation and presentation. It consists of three parts:

- XSL Transformations (XSLT): a language for transforming XML
- XML Path Language (XPath): an expression language used by XSLT to access or refer to parts of an XML document.
- XSL Formatting Objects (XSL-FO): an XML vocabulary for specifying formatting semantics

Simply put, XSL is a language for expressing style sheets. An XSL style sheet is, like Cascading Style Sheet (CSS), a file that describes how to display an XML document of a given type. A style sheet can be used to transform any instance of the DTD or XML Schema it is designed for.

Styling requires a set of source XML documents, containing the information that the style sheet will display and the style sheet itself. In our running example of Author.xml, the XML file doesn't contain any presentation information, which is contained in the style sheet. Separating the document's content and the document's styling information allows displaying the same document on different media (like screen, paper, cell phone), and enables users to view the document according to their preferences and abilities, just by modifying the style sheet.

Figure 2.4 below shows an XSL style sheet (Author.xsl) for the Author example:

```
<?xml version="1.0"?>
```

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="Author">
  <HTML>
   <BODY>
    <xsl:apply-templates select="FirstName" /> <xsl:apply-templates
select="LastName" /><BR/>
    <TABLE border="0">
     <TR><TH>Birthday: </TH>
      <TD><xsl:apply-templates select="Birthday" /></TD></TR>
     <TR><TH>Website: </TH>
      <TD><xsl:apply-templates select="Website" /></TD></TR>
    </TABLE>
    <xsl:apply-templates select="Introduction" />
   </BODY>
  </HTML>
 </xsl:template>
 <xsl:template match="FirstName">
    <B><xsl:value-of select="."/></B>
 </xsl:template>
 <xsl:template match="LastName">
    <B><xsl:value-of select="."/></B>
 </xsl:template>
 <xsl:template match="Birthday">
    <I><xsl:value-of select="@month"/> <xsl:value-of select="@day"/></I>
 </xsl:template>
 <xsl:template match="Website">
    <A><xsl:attribute name="HREF"><xsl:value-of select="."/></xsl:attribute>
    <xsl:value-of select="."/></A>
 </xsl:template>
 <xsl:template match="Introduction">
    <P><xsl:value-of select="."/></P>
 </xsl:template>
</xsl:stylesheet>
```

Figure 2.4: XSL Style Sheet Example

This XSL file builds a HTML page to display the author information in certain formats and positions. The names are in bold, birth month and day are in italic, and website is hyper-linked. <xsl:apply-templates select= "..."/> is a call to the template rules for a selected sub-elements of the current element.

To render an XML file with a style sheet, the style sheet processing command

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

should be added before the first element of the XML file, so that both the XML file and the XSL file are sent to the user's computer for transformation.

Figure 2.5 below shows the rendering of the XML file (Author.xml) and the style sheet (Author.xsl).

Diane Palmer

She is good

Birthday: *September 1*
Website: www.DianePalmer.com

Figure 2.5: XSL Rendering Example

2.4.4 Usefulness of XML Interface for Database Queries

There are at least four benefits to use XML as the format of database query and result dataset [XDT00].

1. Mobility of data: The XML representations of the database schema, query and report provide a single central hub format between DBMS's from different vendors, thus simplify the data exchange and portability of databases. Figure 2.6 below illustrates DBMS-to-DBMS translations, and Figure 2.7 below pictures a central hub format with XML.
2. Browser Views: As more browsers (e.g. Internet Explorer and Mozilla) support XML directly, the XML query or result can be displayed on the web with XSL transformation.

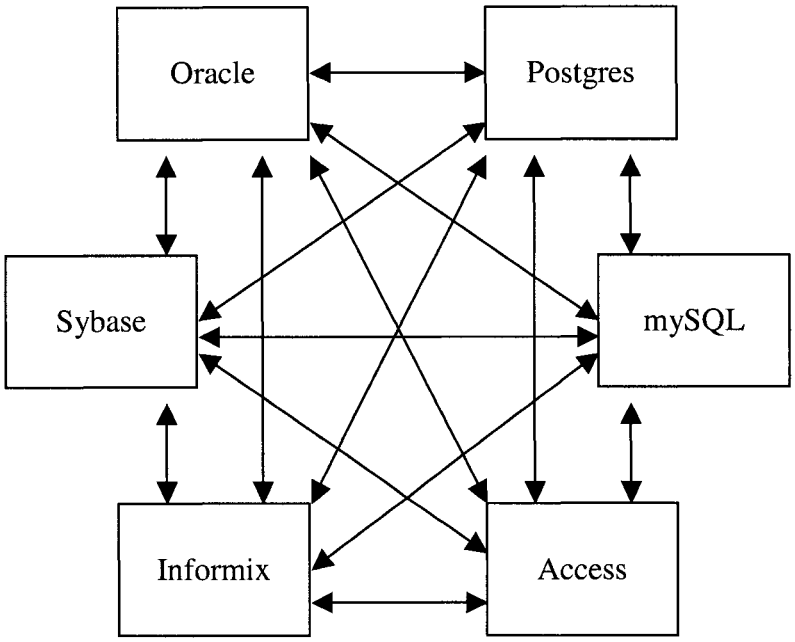

Figure 2.6 Six DBMS's Need 30 Filters

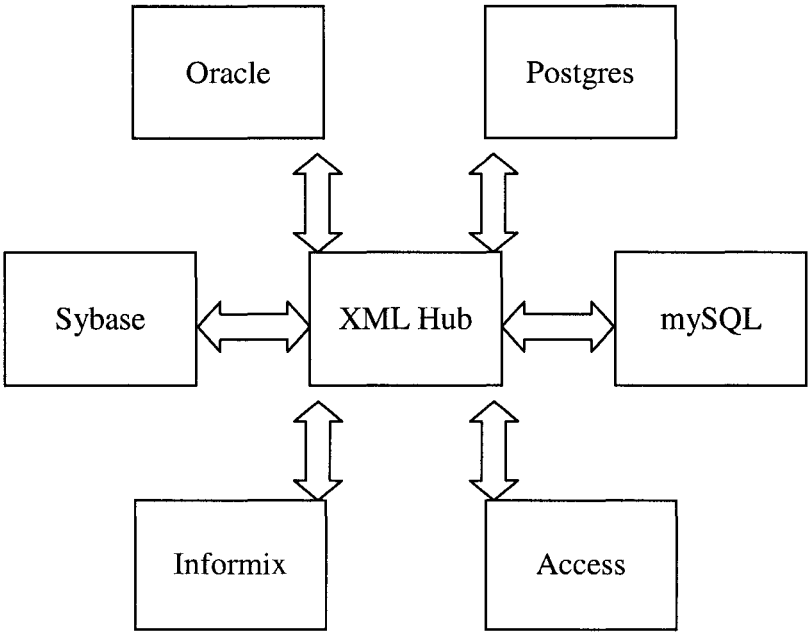

Figure 2.7 Six DBMS's and a XML Hub Need 12 Filters

3. Databases into Documents: For interoperability with a document-based system, the XML documents from a relational database can be loaded into a document repository or an object-oriented database.
4. XML Tools: Expressing database metadata, queries and results as XML allows the processing of this information in standalone XML-based tools, such as formatting or statistics packages.

2.5 Java and JAXB

WISH is coded and tested on the Java 2 Platform. Developed by Sun Microsystems, Java is one of today's most popular software development languages. We choose Java as the programming language because it is fully object-oriented, platform independent, free for download and offers a rich set of predefined classes that can be reused. Java's many predefined classes are grouped into categories of related classes called packages. The packages are referred to collectively as the Java class library or the Java Applications Programming Interface (Java API). In the following sections, we will introduce three Java API's that are important to the development of the WISH XML Query Composer:

- Java Database Connectivity (JDBC)
- Java Swing for Graphic User Interface (GUI)
- Java Architecture for XML Binding (JAXB)

2.5.1 JDBC

Java DataBase Connectivity (JDBC) is a high-level and a low-level API: the developer will use the JDBC API's classes and interfaces to make the necessary transactions with the RDBMS, while the JDBC driver vendor uses the JDBC to model its driver accordingly. JDBC is designed to free developers from low-level concerns about particular database types. It should permit, for example, to query an Access database with exactly the same source code as with an Oracle database.

JDBC is similar to Microsoft's ODBC (Open Database Connectivity), except that it is written in Java and therefore brings the added advantage of portability. Either JDBC or ODBC, however, needs a database driver to communicate with. The driver is specific to the particular type of database used. Most of the JDBC drivers are expensive. On the other hand, drivers for ODBC are already available with Windows installation. Since JDBC and ODBC are very similar, a set of routines were written to translate JDBC commands to ODBC, thus allowing JDBC to use ODBC drivers. These routines are called the JDBC-ODBC bridge. This bridge is free, and comes with versions of the JDK past 1.1.

Using Java in conjunction with JDBC provides a truly portable solution to writing database applications. WISH uses JDBC to access and query relational databases. For the sample data queries, it connects to Microsoft SQL Server 2000 through JDBC-ODBC bridge.

2.5.2 Java Swing

Java Swing is a Graphical User Interface (GUI) component kit, part of the Java Foundation Classes (JFC) integrated into Java 2 platform. Swing simplifies and streamlines the development of applications by providing a complete set of user-interface elements written entirely in the Java programming language [14].

One of the most important capabilities of the Swing toolkit is its *pluggable look and feel* -- a feature that enables developers to choose the appearance and behavior (or the look and feel) of the window components they use in their programs. Swing's look and feel standards promote flexibility and ease of use in cross-platform applications.

Swing components are lightweight. Swing components do not use any platform-specific implementation. Instead, Swing creates its components using pluggable look-and-feel modules that are written from scratch and do not use any platform-specific code at all. Consequently, Swing components use fewer system resources and produce smaller and more efficient applications than their heavyweight AWT counterparts. Several key Swing components used by WISH are listed in Table 2.1 below.

Component	Description
`JDesktopPane`	A container which supports creating multiple document interfaces.
`JFrame`	A window with a title bar, a border, a content pane, and an optional menu bar.

JInternalFrame	A special-purpose container which looks like a frame and has much the same API, but must appear within another window.
JBorderLayout	The default layout manager for JFrame, which arranges the components into five areas: North, South, East, West and Center.
JScrollPane	A general-purpose container which provides scroll bars around a large or growable component.
JOptionPane	A container which displays a dialog box containing information and buttons for user interaction.
JEditorPane	A text component to edit various kinds of content.
JMenuBar	A class for managing a menu bar.
JMenu	A class for managing menus.
JCheckBoxMenuItem	A class for managing menu items that can be toggled on or off
JMenuItem	A class for managing menu items.
JButton	An area that triggers an event when clicked.
JTree	A class that provides a tree view of hierarchical data.
JTextField	An area in which the user inputs data from the keyboard. The area can also display information.
JTable	A user-interface component that presents data in a two-dimensional table format.

Table 2.1: Swing Components Used in WISH

2.5.3 JAXB

A new Java API called Java Architecture for XML Binding (JAXB) makes it easier to access XML documents from applications written in the Java programming language. This API is available in the Java Web Services Developer Pack (JWSDP) pass 1.1.

JAXB is a Java technology that generates Java classes from XML schemas. As part of this process, the JAXB technology also provides methods for unmarshalling an XML instance document into a content tree of Java objects, and marshalling the content tree into an XML document. JAXB provides a fast and convenient way to bind an XML schema to a representation in Java code, making it easy for Java developers to incorporate XML data and processing functions in Java applications.

The JAXB data binding process involves the following steps, as shown in Figure 2.8 below: [JAX03]

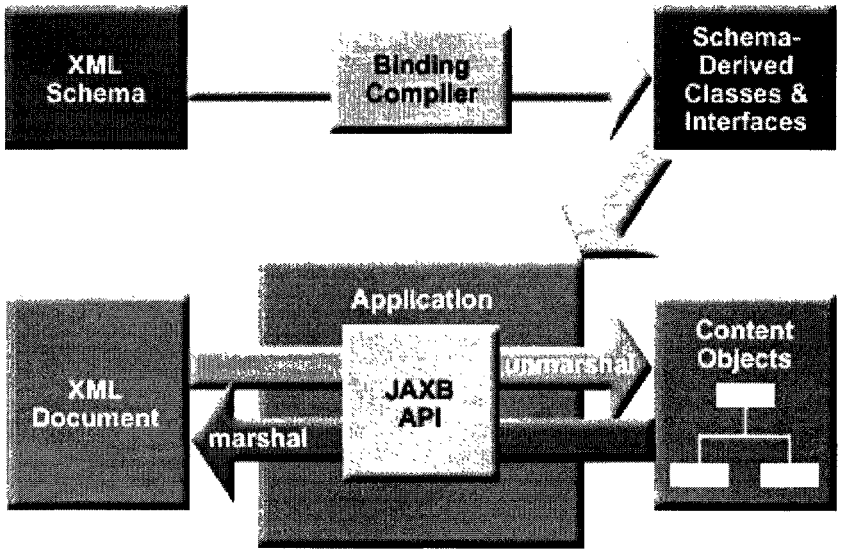

Figure 2.8: JAXB Data Binding Process

1) Generate classes from a source XML schema, and then compile the generated classes.
2) Unmarshal XML documents conforming to the schema. Unmarshalling generates a content tree of data objects instantiated from the schema-

derived JAXB classes; this content tree represents the structure and content of the source XML documents.

3) Unmarshalling optionally involves validation of the source XML documents before generating the content tree. If your application modifies the content tree, you can also use the validate operation to validate the changes before marshalling the content back to an XML document.
4) The client application can modify the XML data represented by a content tree by means of interfaces generated by the binding compiler.
5) The processed content tree is marshalled out to one or more XML output documents.

JAXB is an aid in developing Java-XML applications like WISH. The reasons and advantages of using JAXB in WISH are:

- JAXB simplifies access to an XML document from a Java program:
    - JAXB allows access to XML data without having to do XML processing. Unlike SAX-based processing, there is no need to create a SAX parser or write callback methods.
    - JAXB allows access to data in non-sequential order, but unlike DOM-based processing, it does not force to navigate through a tree to access the data.
    - By unmarshalling XML data through JAXB, Java content objects that represent the content and organization of the data are directly available to the program.

- JAXB uses memory efficiently: The tree of content objects produced through JAXB tends to be more efficient in terms of memory use than DOM-based trees.
- JAXB's binding behavior can be customized in a variety of ways.
- JAXB is flexible:
    - XML data can be unmarshalled from a variety of input sources, including a file, an InputStream object, a URL, a DOM node, or a transformed source object.
    - A content tree can be marshalled to a variety of output targets, including an XML file, an OutputStream object, a DOM node, or a transformed data object
    - JAXB allows access to XML data without having to unmarshal it. Once a schema is bound, the ObjectFactory methods can create the objects and then the set methods in the generated objects can create content.
    - Source data can be validated against an associated schema as part of the unmarshalling operation, and validation can be turned off.
    - A content tree can be validated by the Validator class, separately from marshalling. That is, the validating and the marshalling can be done at different points in time.

Another API of interest is Java API for XML Processing (JAXP), which may also be used in generating query and result XML. It is also available in JWSDP 1.1 and up.

3. Design and Implementation

3.1 Overview of WISH XML Query Composer

The design of WISH is an object-oriented approach that follows the Unified Modeling Language (UML) methodology.

3.1.1 Use Case View

Use cases are the patterns of behavior the system exhibits. The use case view models the functionality of the system as perceived by outside users, called actors [UML99]. The Use Case Diagram in Figure 3.1.1 below shows the relationships between actors and use cases of WISH.

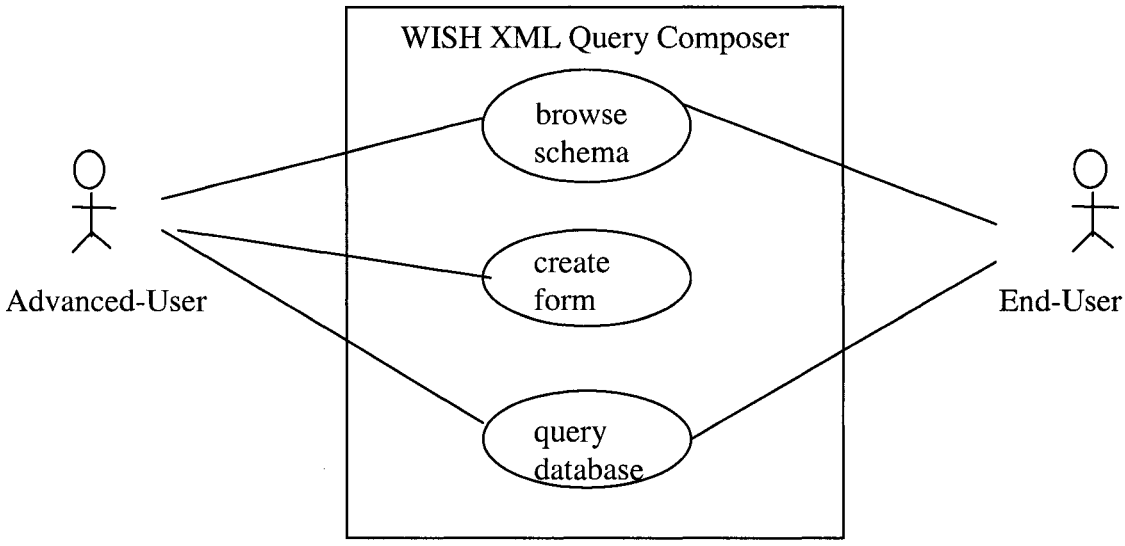

Figure 3.1.1 Use Case Diagram

There are two types of actors:

1) End-users: They wish to pose certain canned-queries using forms, and then to view the results as part of a canned-report. This is the primary actor.
2) Advanced-users: They know SQL and XML/HTML. Based on the database schema, they create the query forms and reports to be used by the end-users.

3.1.2 Model Management View

Any large system must be divided into smaller units so that developers can work with a limited amount of information at one time. Model management consists of packages and dependency relationships among packages. [UML99]

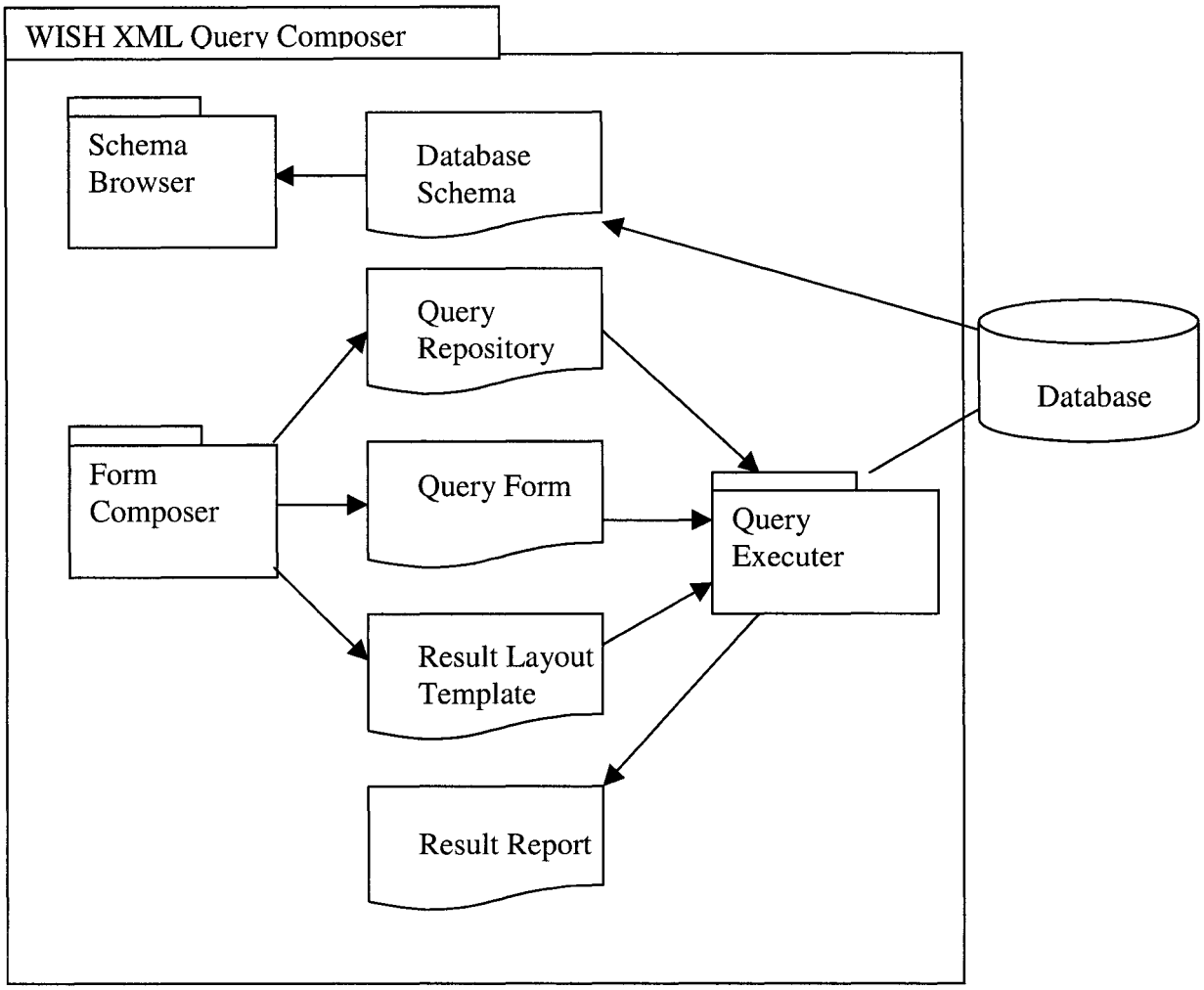

Figure 3.1.2 System Architecture Diagram

To handle the three main use cases indicated in Figure 3.1.1, WISH is divided into three subsystems (as shown in Figure 3.1.2 above):

1) a Schema Browser that allows the users easy access to information about tables, attributes, keys, and the types of data;
2) a Form Composer that lets the advanced-user design an XML/HTML form for query, an XML report for the database results, and a style sheet for presentation. The advanced-user needs to decide on the SQL query and its parameters, and then create the "form with slots" that is used by the end-user.
3) a Query Executer that lets the end-users enter values or parameters into the slots of the form, then select "submit", and the corresponding SQL query will be sent to the database. The query results can then be saved into an XML/HTML document in the report format designed by the advanced-user.

In a nutshell, with the visual help of Schema Browser, the advanced user uses the Form Composer to design the Query Forms in XML and Result Layout Templates in XSL, and puts their file names into the Query Repository as predefined query templates. The end-user then use the Query Executer to customize the query and save Result Reports as XML/HTML in the format predefined by the Result Layout Template.

Initially, these three subsystems were independent tools. At the end they were integrated into one system, as the advanced-user might do all three tasks at once. Very simple versions of form / report composer can be text editor of XML/HTML files.

Besides the three subsystems, there are five document type components in the System Architecture:

- Database Schema,
- Query Repository,
- Query Form,
- Result Layout Template,
- Result Report.

These are actually XML mechanisms that support the features of the three subsystems, and their individual XML Schema definitions are mapped to Java classes and packages using JAXB technology.

In the following sections, we will discuss the design rationale of each of the subsystems, and the XML derived packages they associate with.

3.2 Schema Browser

The first building block of WISH is the Schema Browser. The function of Schema Browser is to show the database metadata information on screen for the advanced-user to design form-based queries.

3.2.1 Database Schema in XML

The Schema Browser gets input from an XML file representing the database schema, i.e. the set of database tables. An XML database schema definition file is used to list all the tables and their respective data fields/columns. Each XML

database schema definition file defines one database schema. This includes one or more tables and the associated data columns and column properties. The XML file is processed by Schema Browser to generate a tree of database-table-column (with type and size) for the advanced users to browse.

To start with, we require that the database to be processed have an associated XML document for schema definition. Later on we can enhance the functionality by fetching the metadata from the actual database. After a connection has been established to a database, schema metadata can be retrieved using the JDBC DatabaseMetaData class. This JDBC interface is implemented by JDBC-complaint driver vendors to let users know the capabilities of a DBMS in combination with the driver based on JDBC technology. Many DBMS-independent database tools use this API because most JDBC drivers of major DBMS support it. Another related API is the ResultSetMetaData class.

Figure 3.2.1 shows the Document Type Definition (DTD) defining a core database schema (DBSchema.dtd).

```
<!ELEMENT DBSchema(SchemaNum, SchemaName, TablesDef)>
<!ELEMENT TablesDef(Table*)>
<!ELEMENT Table(TableNum, TableName, ColumnsDef)>
<!ELEMENT ColumnsDef(Column+)>
<!ELEMENT Column(ColumnNum, ColumnName, Caption?, DataType, Length)>
<!ELEMENT SchemaNum(#PCDATA)>
<!ELEMENT SchemaName(#PCDATA)>
<!ELEMENT TableNum(#PCDATA)>
<!ELEMENT TableName(#PCDATA)>
<!ELEMENT ColumnNum(#PCDATA)>
<!ELEMENT ColumnName(#PCDATA)>
<!ELEMENT Caption(#PCDATA)>
<!ELEMENT DataType(#PCDATA)>
```

<!ELEMENT Length(#PCDATA)>

Figure 3.2.1 XML DTD for Database Schema

One of the disadvantages of DTD is that it cannot specify data type for each #PCDATA element. As a better alternative, XML Schema serves data type mapping between XML definition and Java classes.

Figure 3.2.2 below is the XML Schema (DBSchema.xsd) corresponding to the same database schema definition as described in the DTD (DBSchema.dtd):

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:annotation>
   <xsd:documentation xml:lang="en">
     XML Schema for Database Schema in WISH XML Query Composer.
   </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="DBSchemaDef" type="DBSchema"/>

 <xsd:complexType name="DBSchema">
  <xsd:sequence>
    <xsd:element name="SchemaNum" type="xsd:positiveInteger"/>
    <xsd:element name="SchemaName" type="xsd:string"/>
    <xsd:element name="Description" type="xsd:string" minOccurs="0" />
    <xsd:element name="TablesDef" type="Tables"/>
  </xsd:sequence>
 </xsd:complexType>

 <xsd:complexType name="Tables">
  <xsd:sequence>
    <xsd:element name="Table" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="TableNum" type="xsd:positiveInteger"/>
            <xsd:element name="TableName" type="xsd:string"/>
            <xsd:element name="Description" type="xsd:string"
minOccurs="0" />
            <xsd:element name="ColumnsDef" type="Columns"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Columns">
```

```
  <xsd:sequence>
    <xsd:element name="Column" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="ColumnNum" type="xsd:positiveInteger"/>
          <xsd:element name="ColumnName" type="xsd:string"/>
          <xsd:element name="Description" minOccurs="0"
type="xsd:string"/>
          <xsd:element name="Caption" minOccurs="0" type="xsd:string"/>
          <xsd:element name="DataType" type="xsd:string"/>
          <xsd:element name="Length" type="xsd:positiveInteger"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
 </xsd:complexType>

</xsd:schema>
```

Figure 3.2.2 XML Schema for Database Schema

3.2.2 Static View

The static view captures object structure. An object-oriented system unifies data structure and behavioral features into a single object structure – class. [UML99]

From the XML Schema definition of database schema (DBSchema.xsd), the binding compiler (xjc) of JAXB can generate a Java package with a set of interface classes and corresponding implementation classes. For instructions on how to run xjc command, see JAXB documentations [JWS03].

Table 3.1 below summarizes the classes in the package Wish.DBSchema:

Class	**Description**
Columns	Java content interface class for Columns complex type.
Columns.ColumnType	Java content interface class for anonymous complex type for Column element.
DBSchema	Java content interface class for DBSchema complex type.
DBSchemaDef	Java content interface class for DBSchemaDef element

	declaration.
Tables	Java content interface class for Tables complex type.
Tables.TableType	Java content interface class for anonymous complex type for Table element.
ObjectFactory	This object contains factory methods for each Java content interface and Java element interface generated in the Wish.DBSchema package.
ColumnsImpl	Implementation of Columns complex type.
ColumnsImpl .ColumnType Impl	Implementation of anonymous complex type for Column element.
DBSchemaImpl	Implementation of DBSchema complex type.
DBSchemaDefImpl	Implementation of DBSchemaDef element declaration.
TablesImpl	Implementation of Tables complex type.
TablesImpl .TableTypeImpl	Implementation of anonymous complex type for Table element.

Table 3.1 Class Summary of JAXB Generated Package Wish.DBSchema

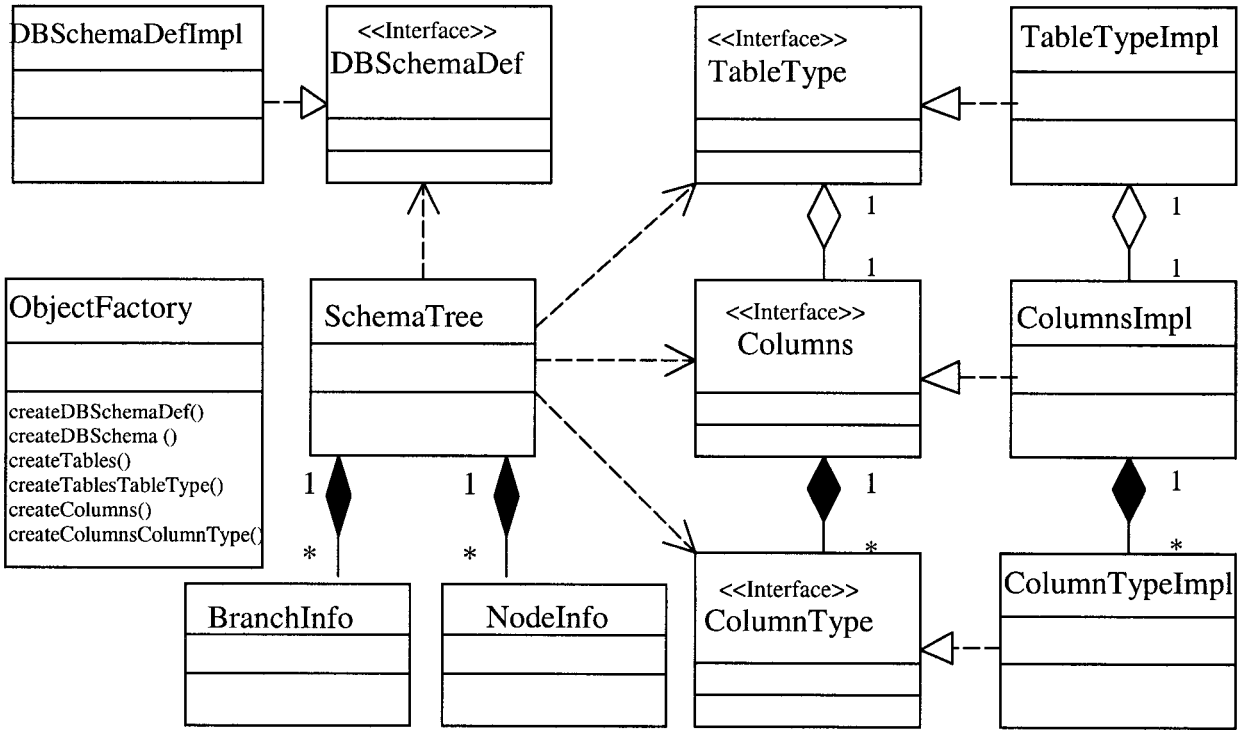

Figure 3.2.3 Class Diagram of Schema Browser Subsystem

Figure 3.2.3 above shows the class diagram for the Schema Browser Subsystem. A class SchemaTree is designed to evoke the JAXB generated classes and to populate the database schema into a schema tree on the left panel of the GUI. Two private classes BranchInfo and NodeInfo are designed to handle context-sensitive help documents and tips.

3.2.3 Example Database

Bookstore database is the example data model in this thesis. This database schema has four interrelated entities: Titles, Publishers, Authors, and Categories. Figure 3.2.4 below illustrates the data model, generated by Microsoft Access.

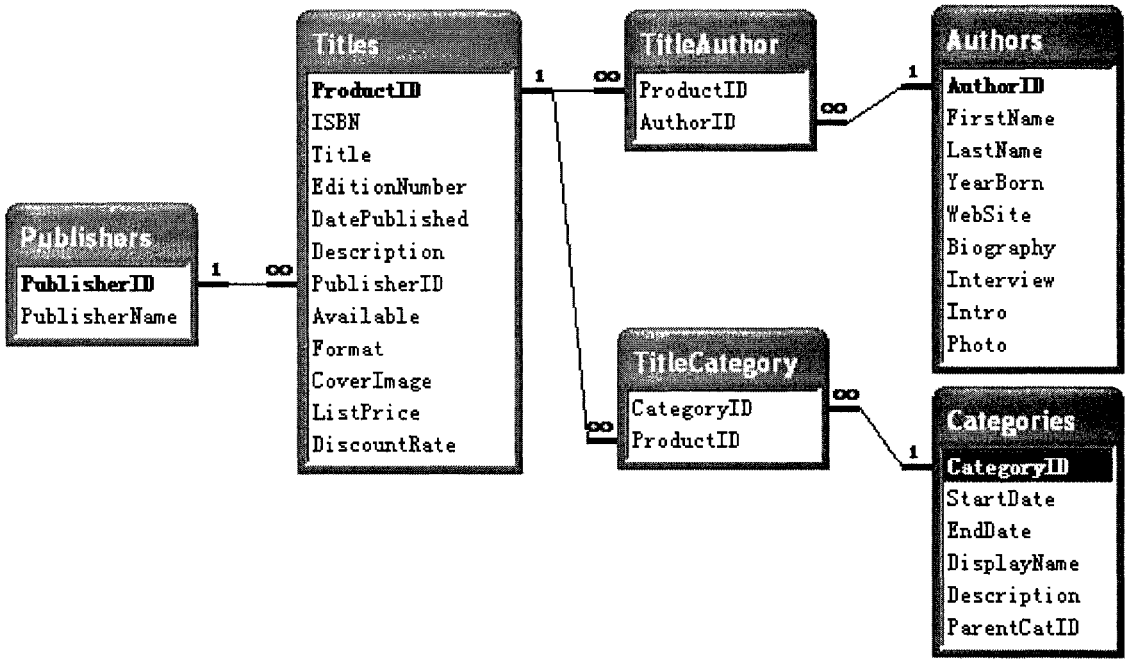

Figure 3.2.4 Bookstore Data Model

The XML file Bookstore.xml defines the Bookstore database schema in the XML format specified in DBSchema.xsd. The entire Bookstore.xml file is in Appendix

A. The portion in Figure 3.2.5 below should give enough information to understand the rules and components of the XML definition.

```
<DBSchemaDef>
  <SchemaNum>1</SchemaNum>
  <SchemaName>Bookstore</SchemaName>
  <Description>The Bookstore Data Model is the sample database schema
of WISH XML Query Composer.</Description>
  <TablesDef>
    <Table>
      <TableNum>1</TableNum>
      <TableName>Titles</TableName>
      <Description>Stores book related information.</Description>
      <ColumnsDef>
        <Column>
          <ColumnNum>1</ColumnNum>
          <ColumnName>ProductID</ColumnName>
          <Description>The unique identifier of the book.</Description>
          <Caption>BookID</Caption>
          <DataType>int</DataType>
          <Length>4</Length>
        </Column>
        ...
        </ColumnsDef>
      </Table>
      ...
  </TablesDef>
</DBSchema>
```

Figure 3.2.5 Bookstore Database Schema in XML

3.2.4 Graphical User Interface

The interface design for Schema Browser has gone through several iterations.

- **Command Console Versions**

Two tests are developed to use the JAXB generated classes and unmarshelling and to retrieve database schema from the Bookstore.xml file. The schema information is then printed on System Output. Figure 3.2.6 below shows the result of Test 1, which only retrieve table attributes:

```
E:\Thesis\Java\JAXB>java Test1
Schema Name: Bookstore
No of Tables 6
```

```
Table Num: 1
Table Name: Titles
No of Columns 12

Table Num: 2
Table Name: Authors
No of Columns 9

Table Num: 3
Table Name: Categories
No of Columns 6

Table Num: 4
Table Name: Publishers
No of Columns 2

Table Num: 5
Table Name: TitleCategory
No of Columns 2

Table Num: 6
Table Name: TitleAuthor
No of Columns 2
```

Figure 3.2.6 Schema Browser Test1: Using JAXB Generated Classes

Figure 3.2.7 below shows the output result of Test 2, with a tree-structured layout.

```
E:\Thesis\Java\JAXB>java Test2
*Database Schema
|
+-* Bookstore
  |
  +-* Tables (6)
    |
    +-* {1} Titles
    | +-* Columns (12)
    |   +-* {1} ProductID [int(4)]
    |   +-* {2} ISBN [char(13)]
    |   +-* {3} Title [varchar(255)]
    |   +-* {4} EditionNumber [char(20)]
    |   +-* {5} DatePublished [datetime(8)]
    |   +-* {6} Description [varchar(2000)]
    |   +-* {7} PublisherID [int(4)]
    |   +-* {8} Available [bit(1)]
    |   +-* {9} Formats [varchar(255)]
    |   +-* {10} CoverImage [varchar(100)]
    |   +-* {11} ListPrice [money(8)]
    |   +-* {12} DiscountRate [float(8)]
    |
    +-* {2} Authors
    | +-* Columns (9)
    |   +-* {1} AuthorID [int(4)]
    |   +-* {2} FirstName [varchar(30)]
    |   +-* {3} LastName [varchar(30)]
    |   +-* {4} YearBorn [char(4)]
    |   +-* {5} WebSite [varchar(100)]
    |   +-* {6} Biography [varchar(100)]
    |   +-* {7} Interview [varchar(100)]
```

```
|    +-* {8} Introduction [varchar(5000)]
|    +-* {9} Photo [varchar(100)]
|
+-* {3} Categories
| +-* Columns (5)
|    +-* {1} CategoryID [int(4)]
|    +-* {2} StartDate [datetime(8)]
|    +-* {3} EndDate [datetime(8)]
|    +-* {4} DisplayName [varchar(100)]
|    +-* {5} Description [varchar(2000)]
|
+-* {4} Publishers
| +-* Columns (2)
|    +-* {1} PublisherID [int(4)]
|    +-* {2} PublisherName [varchar(100)]
|
+-* {5} TitleCategory
| +-* Columns (2)
|    +-* {1} ProductID [int(4)]
|    +-* {2} CategoryID [int(4)]
|
+-* {6} TitleAuthor
| +-* Columns (2)
|    +-* {1} ProductID [int(4)]
|    +-* {2} AuthorID [int(4)]
```

Figure 3.2.7 Schema Browser Test2: Simulating the Tree

Simplified Graphical Version

Assuming the DBSchema.xsd is a complete database definition, a database schema only contains tables and a table only contains columns. Figure 3.2.8 below is a screenshot of the Schema Tree, inherited from JTree of Java Swing. Underneath the schema tree, there is a help area and a tip bar to show different context information based on the selected tree node or branch. The user could click on a database name, table name, or column name, and get relevant documentation in the help area and tip bar. The right side area is reserved for the form-based query composer and query results. When designing or executing queries, the advanced user can easily check properties of the database tables and columns from the Schema Tree on the left.

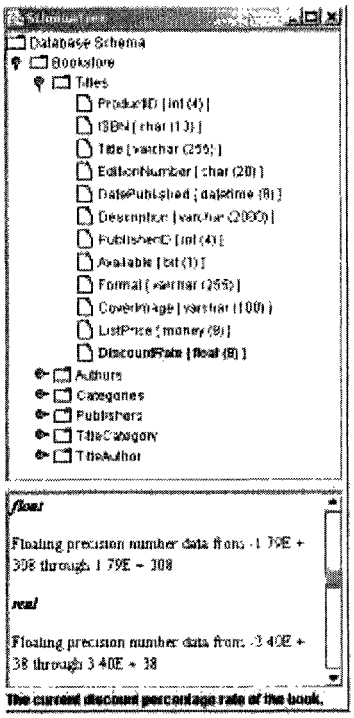

Figure 3.2.8 Schema Tree - Simplified Version

Extensible Graphical Version

Actually the DBSchema.xsd only defines a simplified version of a real-life database schema, which, besides tables, can contain other database objects such as views, stored procedures, users, roles, rules, defaults, user defined data types and functions. Similarly, tables can contain objects besides columns, such as indexes, constraints (primary key and foreign key, etc.) dependencies, triggers. Although these constraints and objects are not in the scope of this project, the Schema Tree can be designed in a format that can easily be extended to include them in the future. Figure 3.2.9 below is a screenshot of the extensible version of Schema Tree. Two layers are added to the hierarchy: Tables and Columns.

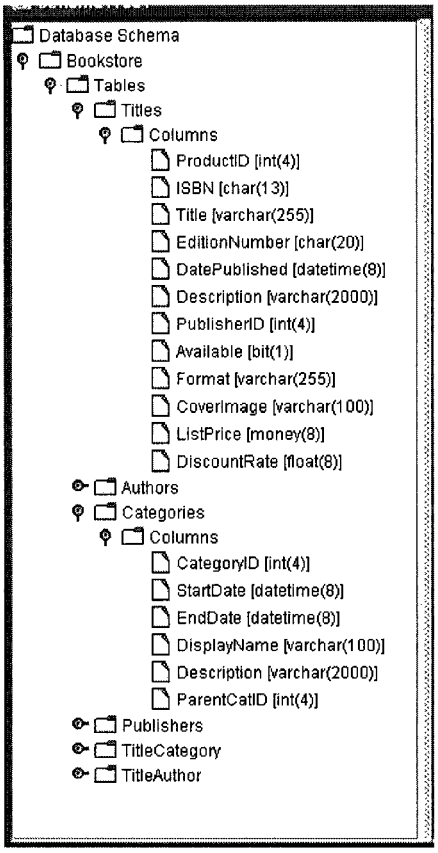

Figure 3.2.9 Schema Tree - Extensible Version

3.3 Form Composer

The second subsystem of WISH is the Form Composer. The function of Form Composer is to assist the advanced user in the creation and verification of form-based query and report templates.

3.3.1 Query Form in XML

The first step of building the Form Composer is to design the Query Form in XML format, which can be parsed and translated to standard SQL query string. Each XML Query Form file defines one query template. This includes one or more

data columns from one or more tables of the database, and possibly search conditions, sort orders and aggregations. Multiple joins, sub-queries and nested conditions should be allowed. Ambiguity should be eliminated.

These query forms act as bridges of the entire WISH system. They are the output of the Form Composer subsystem and the input of the Query Executer subsystem. In the Form Composer, the advanced user can edit, test and save the XML files, which can be used in the Query Executer later.

As for the DBSchema.xsd, JAXB binding compiler (xjc) is used to map the Query Form XML Schema (PreQuery.xsd) to a set of Java classes in a package (Wish.PreQuery). Each of the interface classes has a corresponding implementation class in package Wish.PreQuery.Impl. The interface classes and their specific bindings to the source XML Schema are described in Table 3.2 below.

XML Schema	Derived Class
<xsd:element name="QUERY" type="WishQuery"/>	QUERY
<xsd:complexType name="WishQuery"> <xsd:sequence> <xsd:element name="OUTPUT" type="WishOutput" maxOccurs="unbounded"/> <xsd:element name="SOURCE" type="WishSource" maxOccurs="unbounded"/> <xsd:element name="CONDITION" type="WishCondition" minOccurs="0" maxOccurs="unbounded"/> <xsd:element name="GROUP" type="WishOutput" minOccurs="0" maxOccurs="unbounded"/> <xsd:element name="HAVING" type="WishCondition" minOccurs="0" maxOccurs="unbounded"/> <xsd:element name="SORT" type="WishSort" minOccurs="0" maxOccurs="unbounded"/> </xsd:sequence> <xsd:attribute name="ID" type="xsd:string" use="required"/>	WishQuery

<xsd:attribute name="DESCRIPTION" type="xsd:string"/> <xsd:attribute name="SUBQUERY" type="xsd:boolean"/> </xsd:complexType>	
<xsd:complexType name="WishOutput"> <xsd:attribute name="NAME" type="xsd:string" use="required"/> <xsd:attribute name="COLUMN" type="xsd:string" use="required"/> <xsd:attribute name="TABLE" type="xsd:string"/> <xsd:attribute name="DISPLAY" type="xsd:string"/> </xsd:complexType>	WishOutput
<xsd:complexType name="WishSource"> <xsd:sequence> <xsd:element name="KEY" type="WishKey" minOccurs="0" maxOccurs="unbounded"/> </xsd:sequence> <xsd:attribute name="NAME" type="xsd:string" use="required"/> <xsd:attribute name="TABLE" type="xsd:string" use="required"/> <xsd:attribute name="DISPLAY" type="xsd:string"/> <xsd:attribute name="JOIN" type="WishJoin"/> <xsd:attribute name="PARENT" type="xsd:string"/> </xsd:complexType>	WishSource
<xsd:complexType name="WishCondition"> <xsd:sequence> <xsd:element name="EXPRESSION" type="WishExpression"/> <xsd:element name="OPERATOR" type="WishOperator"/> <xsd:element name="INPUT" type="WishInput" minOccurs="0"/> <xsd:element name="QUERY" type="WishQuery" minOccurs="0"/> </xsd:sequence> <xsd:attribute name="MORE" type="WishMore"/> </xsd:complexType>	WishCondition
<xsd:complexType name="WishSort"> <xsd:attribute name="DISPLAY" type="xsd:string" use="required"/> <xsd:attribute name="TABLE" type="xsd:string"/> <xsd:attribute name="COLUMN" type="xsd:string" use="required"/> <xsd:attribute name="ORDER" type="WishOrder"/> </xsd:complexType>	WishSort
<xsd:complexType name="WishKey"> <xsd:attribute name="COLUMN" type="xsd:string" use="required"/> <xsd:attribute name="PARENT-TABLE" type="xsd:string" use="required"/> <xsd:attribute name="PARENT-COLUMN" type="xsd:string" use="required"/> </xsd:complexType>	WishKey
<xsd:complexType name="WishOperator"> <xsd:sequence>	WishOperator

<xsd:element name="OPTION" type="WishOption" maxOccurs="15" /> </xsd:sequence> <xsd:attribute name="NAME" type="xsd:string" use="required"/> <xsd:attribute name="SELECTED" type="xsd:positiveInteger" use="required"/> </xsd:complexType>	
<xsd:complexType name="WishOption"> <xsd:attribute name="SEQNUM" type="xsd:positiveInteger" use="required"/> <xsd:attribute name="DISPLAY" type="xsd:string" use="required"/> <xsd:attribute name="SQLOP" type="WishSqlOp" use="required"/> </xsd:complexType>	WishOption
<xsd:complexType name="WishExpression"> <xsd:attribute name="DISPLAY" type="xsd:string"/> <xsd:attribute name="TABLE" type="xsd:string"/> <xsd:attribute name="COLUMN" type="xsd:string" use="required"/> </xsd:complexType>	WishExpression
<xsd:complexType name="WishInput"> <xsd:attribute name="NAME" type="xsd:string" use="required"/> <xsd:attribute name="SIZE" type="xsd:positiveInteger"/> <xsd:attribute name="VALUE" type="xsd:string"/> <xsd:attribute name="FIXED" type="xsd:boolean"/> <xsd:attribute name="TYPE" type="WishType"/> </xsd:complexType>	WishInput

Table 3.2: XML Schema Class Binding for Query Form

Besides the above portions for class binding, the rest of the Query Form XML Schema defines single type (such as WishSqlOp) restrictions that are used for validation in the implementations of the interface classes. The complete XML Schema (PreQuery.xsd) of the Query Form is in Appendix A.

3.3.2 Static View

Figure 3.3.1 below is the class diagram for Form Composer Subsystem. The classes in the Wish.PreQuery package are generated using the JAXB utility from the XML Schema for Query Form, as described in Table 3.2.

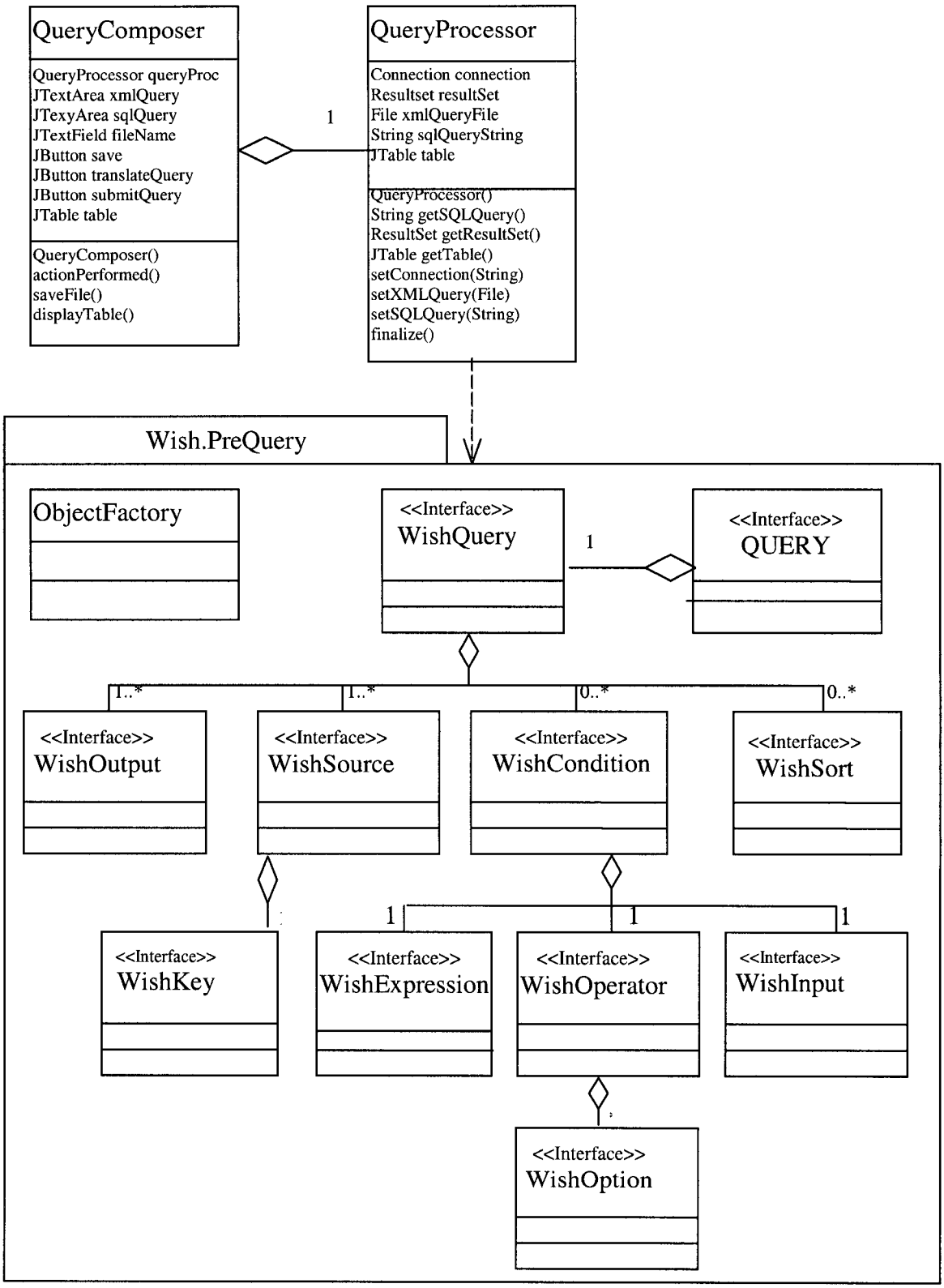

Figure 3.3.1 Class Diagram of Form Composer Subsystem

The ObjectFactory class can be used to create objects of all the interface classes in the package. Each interface class has a corresponding implementation class.

A class QueryProcessor is designed to unmarshell the XML file into objects of the Wish.PreQuery package to generate SQL statement, which can then be submitted to the underlying DBMS through JDBC. Result set returned from the DBMS will be returned or put in a table for display. The QueryProcessor class encapsulates the properties and methods of XML to SQL query translation and SQL query execution, thus hides the implementation details (such as parsing SQL query string using WishQuery object) from callers.

Another class QueryComposer is designed to handle the user interface features of the Form Composer. This class extends from JFrame and uses Java Swing components such as JPanel, JTextArea and JButton. It calls QueryProcessor class to translate queries and return results in a table.

3.3.3 Example Queries

In order to verify the effectiveness of the XML to SQL query translation, 13 example queries of the Boostore database are designed to cover different variations of SQL queries, including:

- single and multiple conditions (Q1),
- single and multiple ascending or descending sort orders (Q2, Q6),
- unambiguous, composite and aggregated output fields (Q3, Q1),
- single and multiple table sources with inner or outer join (Q4, Q5, Q9, Q10),

- nested queries or sub-queries (Q7, Q8, Q11, Q12),
- group by and having clauses (Q13).

The complete descriptions of these queries and their respective SQL statements are listed in Appendix B.

Based on the XML schema definition of Query Form (PreQuery.xsd), individual query XML files are written and tested for all example queries. They serve as test cases for the advanced user in the Form Composer, and later for end-users in the Query Executer. The XML files for all example queries are attached in Appendix C. Figure 3.3.2 below shows one XML query form file (Query13.xml) for example Q13.

```
<?xml-stylesheet type="text/xsl" href="WishQuery.xsl"?>
<QUERY ID="13" DESCRIPTION="Return average price of books in each
category sorted by average price.">
  <OUTPUT NAME="CategoryName" COLUMN="DisplayName"/>
  <OUTPUT NAME="AvgPrice" COLUMN="AVG(ListPrice)" />
  <SOURCE NAME="categories" TABLE="Categories" />
  <SOURCE NAME="ca_t" TABLE="TitleCategory" JOIN="inner"
PARENT="categories">
    <KEY COLUMN="CategoryID" PARENT-TABLE="Categories" PARENT-
COLUMN="CategoryID" />
  </SOURCE>
  <SOURCE NAME="ca_ti" TABLE="Titles" JOIN="inner" PARENT="ca_t" >
    <KEY COLUMN="ProductID" PARENT-TABLE="TitleCategory" PARENT-
COLUMN="ProductID" />
  </SOURCE>
  <GROUP NAME="CategoryName" COLUMN="DisplayName"/>
  <HAVING>
    <EXPRESSION DISPLAY="average price" COLUMN="Avg(ListPrice)" />
    <OPERATOR NAME="Op1" SELECTED="2">
      <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
      <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"/>
      <OPTION SEQNUM="3" DISPLAY="is undefined" SQLOP="is null" />
      <OPTION DISPLAY="is lower than" SQLOP="&lt;" SEQNUM="4" />
      <OPTION DISPLAY="is no lower than" SQLOP="&gt;=" SEQNUM="5"/>
      <OPTION SEQNUM="6" DISPLAY="is equal to" SQLOP="=" />
      <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;" SEQNUM="7" />
      <OPTION DISPLAY="is higher than" SQLOP="&gt;" SEQNUM="8"/>
      <OPTION DISPLAY="is no higher than" SQLOP="&lt;=" SEQNUM="9" />
```

```
    </OPERATOR>
    <INPUT NAME="Input1" TYPE="float" VALUE="0"/>
  </HAVING>
  <SORT DISPLAY="average price" COLUMN="AVG(ListPrice)"/>
</QUERY>
```

Figure 3.3.2 Bookstore Query Example in XML

The example XML query is quite intuitive to read. It defines the output fields, source tables and their join relations, aggregation and sort references, as well as search conditions, which contain expressions with a list of comparison operators to select from and the input fields to compare against.

3.3.4 Graphical User Interface

The Graphical User Interface of the Form Composer consists of a text field to input file name, a group of buttons to submit commands and three scrollable areas: XML editor, SQL editor and query result table.

Figure 3.3.3 below is a screenshot of the Form Composer GUI running example Q13.

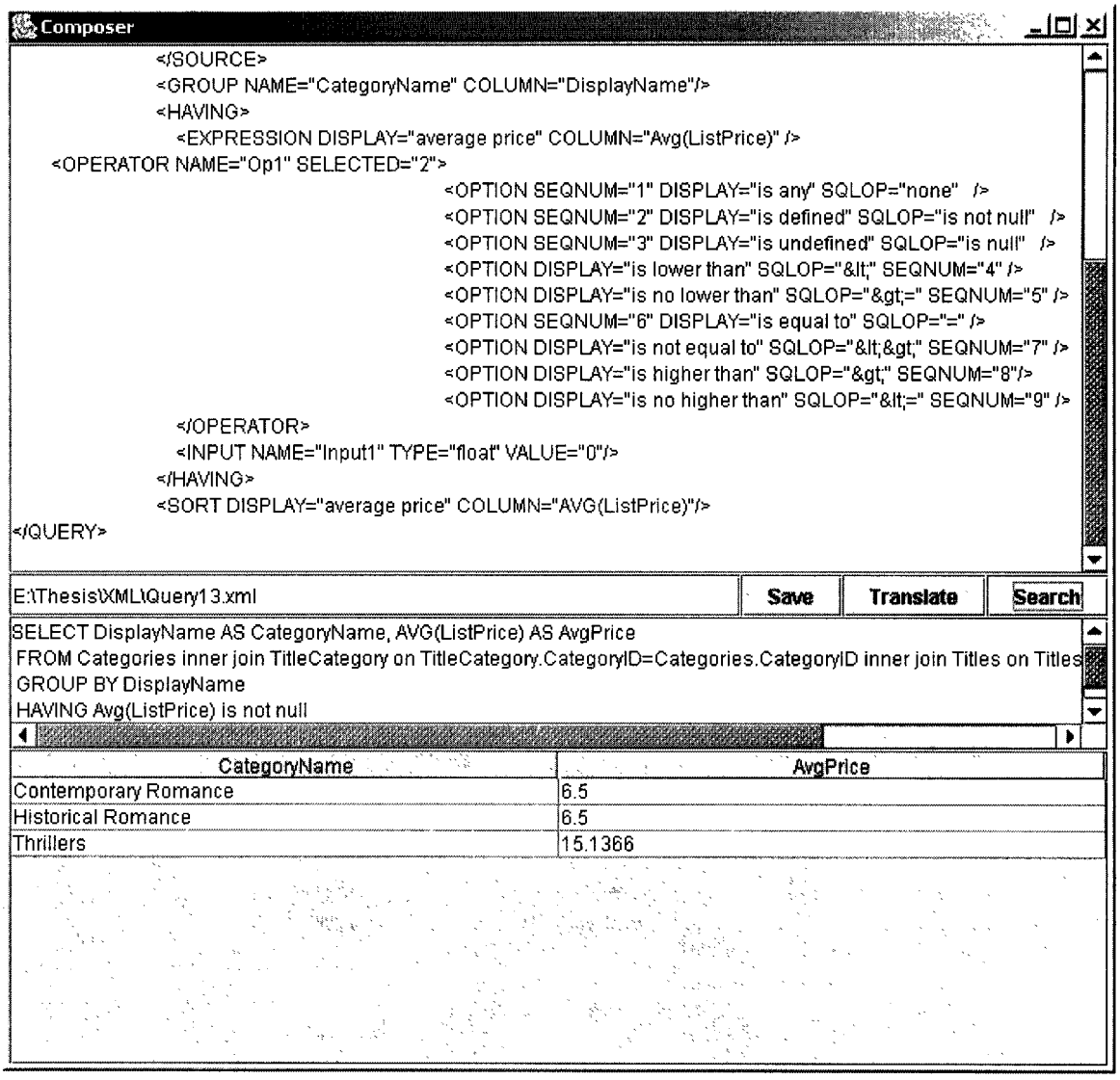

Figure 3.3.3 Form Composer User Interface

3.3.5 Advanced User Scenario

The advanced user can perform the following tasks with the Form Composer:

1. Compose an Query XML in the large XML editor at the top;
2. Enter the file name in the text field underneath and click the "Save" button to save the XML file. A "Save File" dialog box, as shown in Figure 3.3.4 below, will pop up for the user to choose the file path and name.

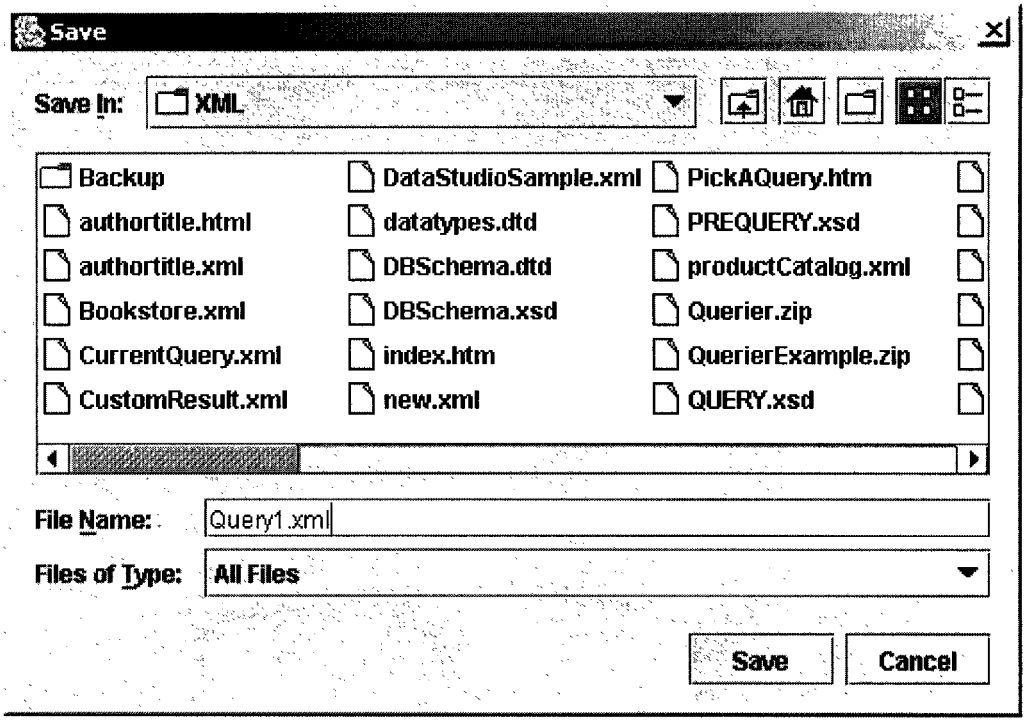

Figure 3.3.4 Save File Dialog Box

3. Type in the file name and hit "Enter" on the keyboard to open an existing XML file in the XML editor. After editing, perform Step 2 above to save the changes.
4. Click the "Translate" button to translate the XML into SQL query, which will be displayed in the stripped SQL editor in the middle. Validation of XML syntax is done in this step using the Wish.PreQuery classes.
5. Possibly edit the query string in SQL editor for testing purpose.
6. Click the "Search" button to submit the SQL query to database and preview the query results in the table at the bottom.

When composing queries, the advanced user can refer to the Schema Browser (described in Section 3.2) for database metadata information.

3.4 Query Executer

The most sophisticated subsystem of WISH XML Query Composer is the Query Executer. It is important because the end-users will use it most often to fulfill their various query requests. Based on the formed-based query and report templates designed by the advanced user, the end-users can customize any queries and generate the desired result reports in either XML or HTML format.

3.4.1 Result Report in XML

Now that the Query Forms are in XML, the Result Reports should have their XML representation as well. Designing a simple but very flexible XML schema for the results allows us to marshal the result set returned from the database into XML reports, which can then be transformed to different XML formats or HTML. Figure 3.4.1 below is the XML Schema for Result Report (Result.xsd).

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:annotation>
   <xsd:documentation xml:lang="en">
     XML Schema for Query Result in WISH XML Query Composer.
   </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="RESULTSET" type="WishResultset" />

 <xsd:complexType name="WishResultset">
     <xsd:sequence>
        <xsd:element name="ROW" maxOccurs="unbounded" type="WishRow" />
     </xsd:sequence>
     <xsd:attribute name="QUERYID" type="xsd:string"/>
     <xsd:attribute name="DESCRIPTION" type="xsd:string"/>
     <xsd:attribute name="CRITERIA" type="xsd:string"/>
     <xsd:attribute name="SQL" type="xsd:string"/>
 </xsd:complexType>

 <xsd:complexType name="WishRow">
   <xsd:sequence>
    <xsd:element name="FIELD" maxOccurs="unbounded" type="WishField" />
```

```
    </xsd:sequence>
    <xsd:attribute name="INDEX" type="xsd:int" use="required"/>
  </xsd:complexType>

  <xsd:complexType name="WishField">
    <xsd:simpleContent>
     <xsd:extension base="xsd:string">
       <xsd:attribute name="NAME" type="xsd:string" use="required"/>
       <xsd:attribute name="TYPE" type="xsd:int" />
     </xsd:extension>
    </xsd:simpleContent>
  </xsd:complexType>

</xsd:schema>
```

Figure 3.4.1 XML Schema for Result Report

The above XML Schema is generic enough to accommodate results of all kinds of queries, no matter how many and what data fields are specified in the report. However, the advanced user may want to apply different report layouts to different queries. This can be accomplished by designing style sheets for each query or a group of queries.

3.4.2 Result Layout Template in XSL

In this thesis, XSL style sheets are used to specify the structure of the report layout, that is, the placement of each XML element and attribute. The cosmetic styles, such as font and color, are taken care of by cascading style sheet (WishResult.css), which is referenced from the XSL template. Although the cosmetic styles can also be defined in the same XSL style sheet, the separation of structural styles from cosmetic styles provides more flexibility in designing templates. Figure 3.4.2 below is an example of query result layout template in XSL style sheet (Result.xsl).

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="RESULTSET">
    <HTML><HEAD>
      <TITLE>Query
       <xsl:value-of select="@QUERYID"/> -
       <xsl:value-of select="@DESCRIPTION"/></TITLE>
      <LINK REL="STYLESHEET" TYPE="text/css" HREF="WishResult.css"/>
    </HEAD>
    <BODY>
      <H1>QUERY RESULT PAGE</H1>
      <TABLE>
        <TR>
          <TH>Query ID: </TH>
          <TD><xsl:value-of select="@QUERYID"/></TD>
        </TR>
        <TR>
          <TH>Description: </TH>
          <TD><xsl:value-of select="@DESCRIPTION"/></TD>
        </TR>
        <TR>
          <TH>Criteria: </TH>
          <TD><xsl:value-of select="@CRITERIA"/></TD>
        </TR>
      </TABLE><BR/>
      <TABLE>
        <xsl:apply-templates select="ROW" />
      </TABLE>
    </BODY>
    </HTML>
  </xsl:template>

  <xsl:template match="ROW">
      <TR>
        <TD>
            <TABLE>
                  <TR><TD>#</TD></TR>
                  <TR><TD><xsl:value-of select="@INDEX"/></TD></TR>
            </TABLE>
        </TD>
        <xsl:apply-templates select="FIELD" />
      </TR>
   </xsl:template>

  <xsl:template match="FIELD">
      <TD>
            <TABLE>
                  <TR><TD><xsl:value-of select="@NAME"/></TD></TR>
                  <TR><TD><xsl:value-of select="."/></TD></TR>
            </TABLE>
      </TD>
  </xsl:template>
</xsl:stylesheet>
```

Figure 3.4.2 XSL Style Sheet Example for Result Report

The XSL and CSS stylesheets can be created and/or edited in any text editor, or through the XML editor of the Form Composer User Interface.

3.4.3 Query Repository in XML

A query repository is needed to store the list of different queries designed by the advanced user and to be used as query templates by the end-users. Each element represents a query template with the following attributes:

- ID – Unique identifier of the query template;
- XmlFile – File name of the XML Query Form;
- XslFile - File name of the XSL Result Layout Template;
- Description – Short introduction to the general purpose of the query template.

Once again, XML Schema is designed and then mapped to Java classes to serve as repository of queries. Figure 3.4.3 below is the XML Schema for Query Repository (QueryRepository.xsd):

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

 <xsd:annotation>
   <xsd:documentation xml:lang="en">
     XML Schema for Query Repository in WISH XML Query Composer.
   </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="QueryRepository">
   <xsd:complexType>
     <xsd:sequence>
       <xsd:element name="Query" maxOccurs="unbounded">
         <xsd:complexType>
           <xsd:attribute name="ID" type="xsd:string" use="required"/>
           <xsd:attribute name="xmlFile" type="xsd:string"
use="required"/>
           <xsd:attribute name="xslFile" type="xsd:string"/>
           <xsd:attribute name="Description" type="xsd:string"/>
             </xsd:complexType>
        </xsd:element>
```

```
      </xsd:sequence>
  </xsd:complexType>
 </xsd:element>

</xsd:schema>
```

Figure 3.4.3 XML Schema for Query Repository

Figure 3.4.4 below shows an example XML query repository (QueryRepository.xml) containing the 13 test cases used in the Bookstore model:

```
<QueryRepository>
      <Query ID="1" xmlFile="Query1.xml" xslFile="Result.xsl"
Description="Author named ..." />
      <Query ID="2" xmlFile="Query2.xml" xslFile="Result.xsl"
Description="Sorted Authors born in ..." />
      <Query ID="3" xmlFile="Query3.xml" xslFile="Result.xsl"
Description="Sorted Author names." />
      <Query ID="4" xmlFile="Query4.xml" xslFile="Resultset.xsl"
Description="All information related to title 'A Perfect Evil'." />
      <Query ID="5" xmlFile="Query5.xml" xslFile="Resultset.xsl"
Description="Titles published by ... between ... and ..." />
      <Query ID="6" xmlFile="Query6.xml" xslFile="Resultset.xsl"
Description="Sorted Titles priced between ... and ... after discount."
/>
      <Query ID="7" xmlFile="Query7.xml" xslFile="Resultset.xsl"
Description="Available titles in the same category as ..." />
      <Query ID="8" xmlFile="Query8.xml" xslFile="Resultset.xsl"
Description="Titles written by author named ..." />
      <Query ID="9" xmlFile="Query9.xml" xslFile="Resultset.xsl"
Description="Sorted Authors and titles written by them." />
      <Query ID="10" xmlFile="Query10.xml" xslFile="Resultset.xsl"
Description="Sorted Authors and available titles written by them." />
      <Query ID="11" xmlFile="Query11.xml" xslFile="Result.xsl"
Description="Authors writing titles in category ..." />
      <Query ID="12" xmlFile="Query12.xml" xslFile="Result.xsl"
Description="Authors only writing titles in category ..." />
      <Query ID="13" xmlFile="Query13.xml" xslFile="Resultset.xsl"
Description="Sorted Average price of books in each category." />
</QueryRepository>
```

Figure 3.4.4 Bookstore Example Query Repository in XML

As with the style sheets, the file QueryRepository.xml can also be created and/or edited in the XML Editor of the Form Composer user interface, as described in Section 3.3.3.

3.4.4 Static View

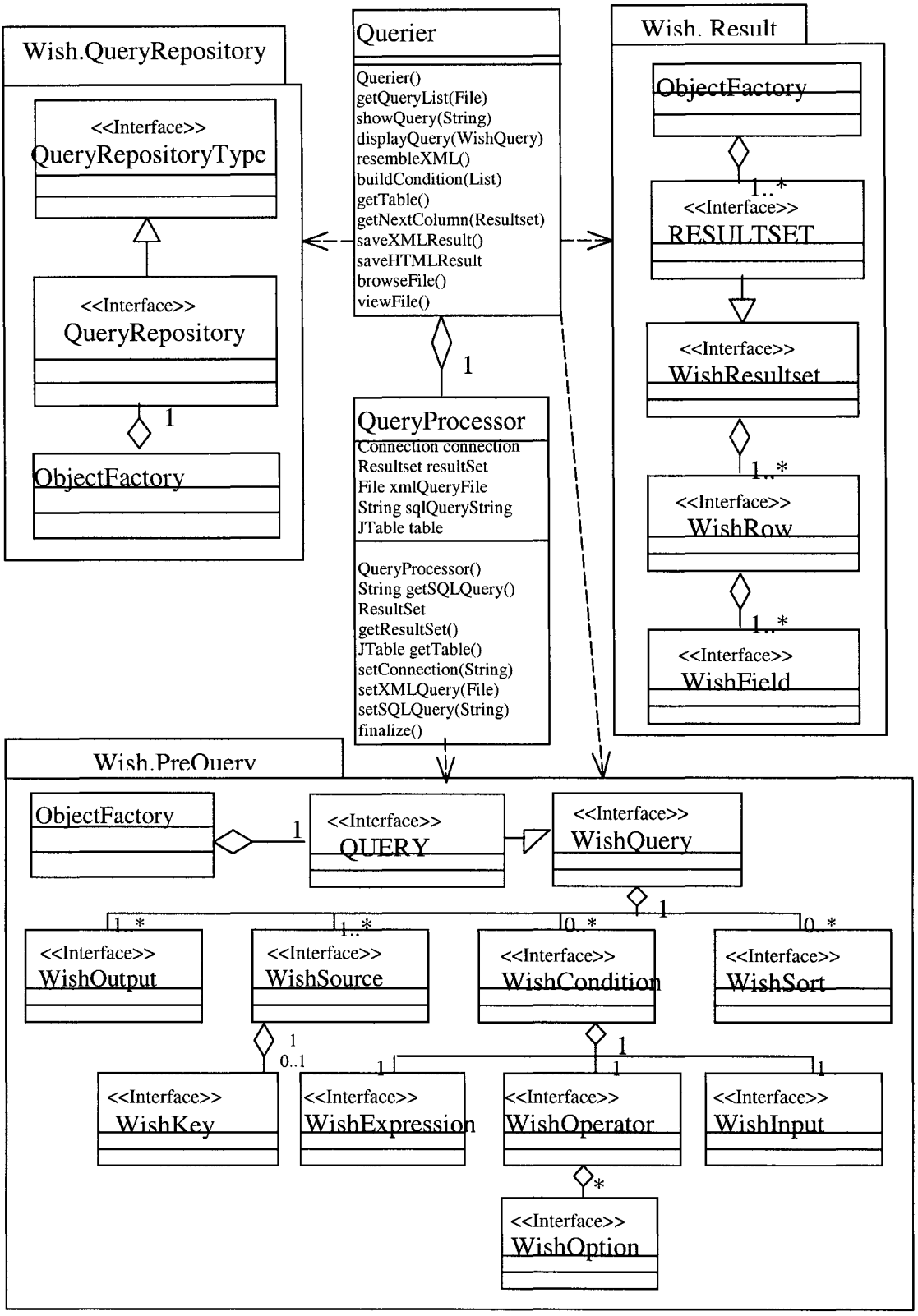

Figure 3.4.5 Class Diagram of Query Executer Subsystem

Figure 3.4.5 above is the class diagram of the Query Executer subsystem.

3.4.5 Graphical User Interface

Figure 3.4.6 below illustrates a screenshot of the Query Executer running example query 8:

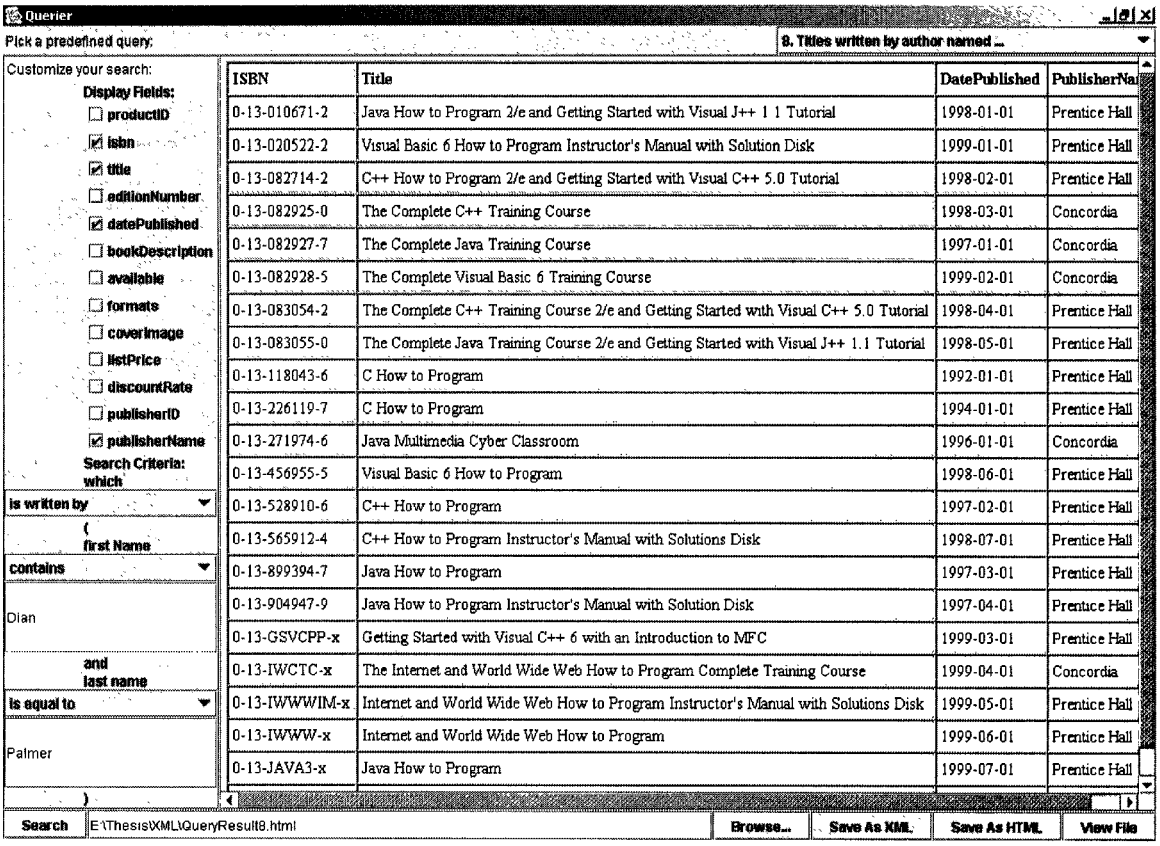

Figure 3.4.6 Query Executer User Interface

The majority of the Query Executer window is used as display area for result set table, XML or HTML display. On the top is a dropdown list of all available query templates in the repository. On the bottom are a text field for file name and a series of command buttons for search and file operation. On the left is the query form containing:

- Display fields (OUTPUT) in checkboxes;
- Search criteria comparison options (OPERATOR) in dropdown lists;
- Comparison values (INPUT) in text fields;
- Sort orders in radio buttons;
- All other fixed parts of the query in labels.

3.4.6 End-User Scenario and Example Results

The following steps outline a typical scenario when an end-user uses the Query Executer:

1. The ComboBox (drop-down list) at the top pulls queries from the QueryRepository.xml file, defined by QueryRepository.xsd. The first query is selected by default.
2. The user first selects a query from the drop-down list. The selected query will then be rendered on the left pane below, where the user can customize the display fields, comparison operator and value, grouping fields, sort order, etc.
3. When the user clicks "Search", the Query Executer will create a CurrentQuery object based on the query customization, generate SQL query string and retrieve the result set from the database.
4. The result set is by default displayed in table format in the display area.
5. The user can then specify a file name to save the result set in XML format that is defined in Result.xsd. In this step, the JAXB ObjectFactory class and Marshaller class are used to generate the XML file from the database

result set. Figure 3.4.7 below is an example Result Report in XML (QueryResult1.xml) generated by the Query Executer.

```
<?xml-stylesheet type='text/xsl' href='Result.xsl'?>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<RESULTSET SQL="SELECT AuthorID, FirstName, LastName, YearBorn,
WebSite, Biography, Interview, Introduction, Photo
 FROM Authors
 WHERE FirstName = 'Diane' and LastName = 'Palmer'" CRITERIA=" (first
name is equal to Diane) and (last name is equal to Palmer) "
DESCRIPTION="Author named ..." QUERYID="1">
<ROW INDEX="1">
<FIELD TYPE="4" NAME="AuthorID">2</FIELD>
<FIELD TYPE="12" NAME="FirstName">Diane</FIELD>
<FIELD TYPE="12" NAME="LastName">Palmer</FIELD>
<FIELD TYPE="1" NAME="YearBorn">1948</FIELD>
<FIELD TYPE="12" NAME="WebSite">www.dianePalmer.com</FIELD>
<FIELD TYPE="12" NAME="Biography">bio_dianepalmer.htm</FIELD>
<FIELD TYPE="12" NAME="Interview">inw_dianepalmer.htm</FIELD>
<FIELD TYPE="12" NAME="Introduction">She is good</FIELD>
<FIELD TYPE="12" NAME="Photo">dianepalmer.jpg</FIELD>
</ROW>
<ROW INDEX="2">
<FIELD TYPE="4" NAME="AuthorID">6</FIELD>
<FIELD TYPE="12" NAME="FirstName">Diane</FIELD>
<FIELD TYPE="12" NAME="LastName">Palmer</FIELD>
<FIELD TYPE="1" NAME="YearBorn">1968</FIELD>
<FIELD TYPE="12" NAME="WebSite">www.dp.com</FIELD>
<FIELD TYPE="12" NAME="Biography">bio_dianepalmer2.htm</FIELD>
<FIELD TYPE="12" NAME="Interview">inw_dianepalmer2.htm</FIELD>
<FIELD TYPE="12" NAME="Introduction">This is a new author with the same
name!</FIELD>
<FIELD TYPE="12" NAME="Photo">dianepalmer2.jpg</FIELD>
</ROW>
</RESULTSET>
```

Figure 3.4.7 Example Result Report in XML Generated by Query Executer

6. The user can view the result XML in the display area in place of the table.
7. For presentation of the Result1.xml, the advanced user can pre-define a Result.xsl which can be used to render into HTML, which can then be viewed in the display area. Figure 3.4.6 is showing the result HTML file (QueryResult8.html).

8. The generated XML and HTML files can also be used outside of the Query Executer, such as a standard browser. Figure 3.4.8 below demonstrates a screenshot of Result Report in XML (QueryResult7.xml) rendered in Internet Explorer.

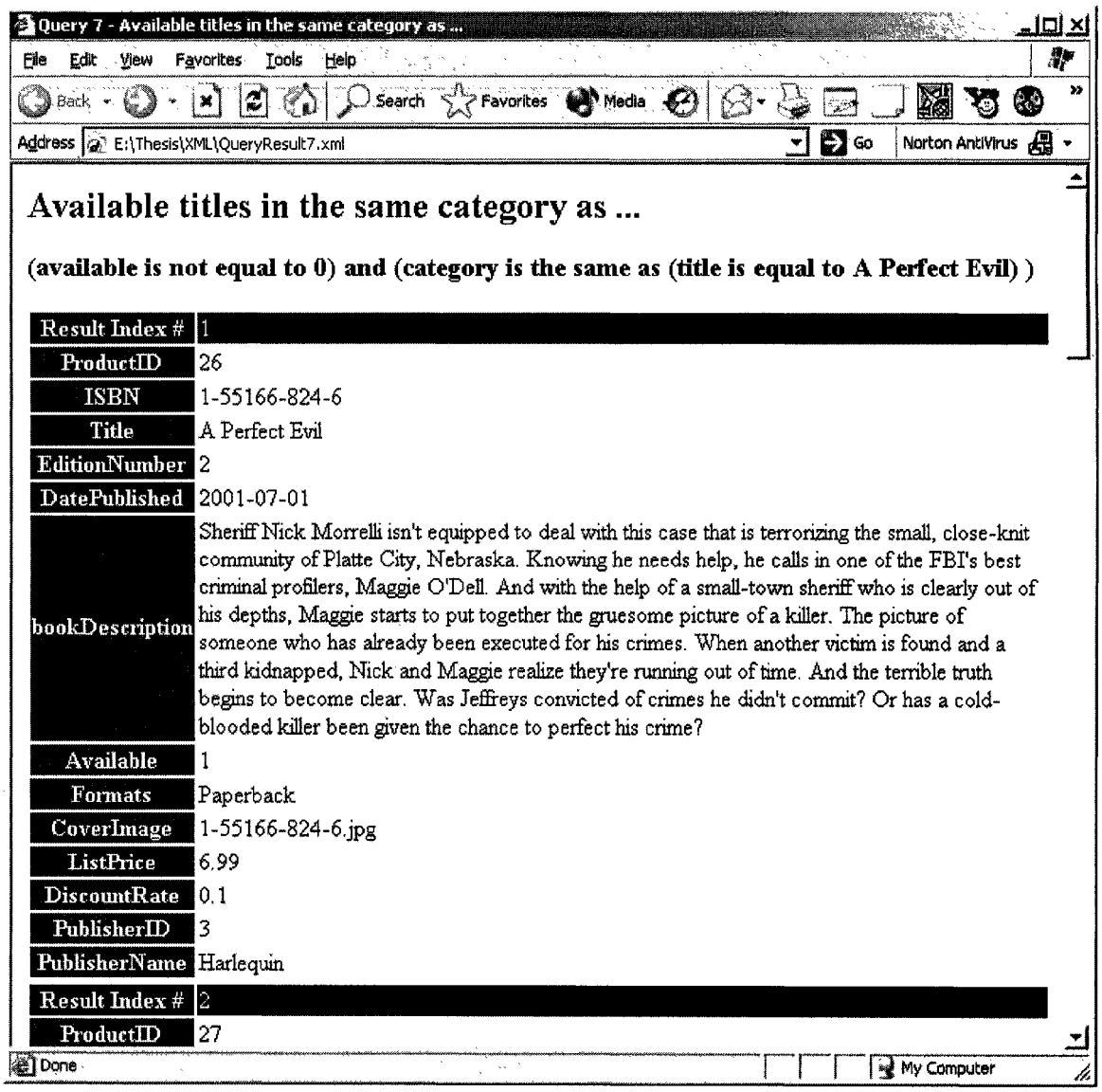

Figure 3.4.8 Example Result Report in XML with XSL Style Sheet

When composing queries, the end user can refer to the Schema Browser (described in Section 3.2) for tips and helps on metadata.

3.5 The Final Wrap

After all the subsystems are developed and tested, a final warp is implemented to bring all three individual user interfaces together. The wrap is the control console of WISH. It contains a menu bar and shows or hides the three subsystems as needed. Figure 3.5.1 below shows a screenshot of the entire WISH XML Query Composer system with the final wrap.

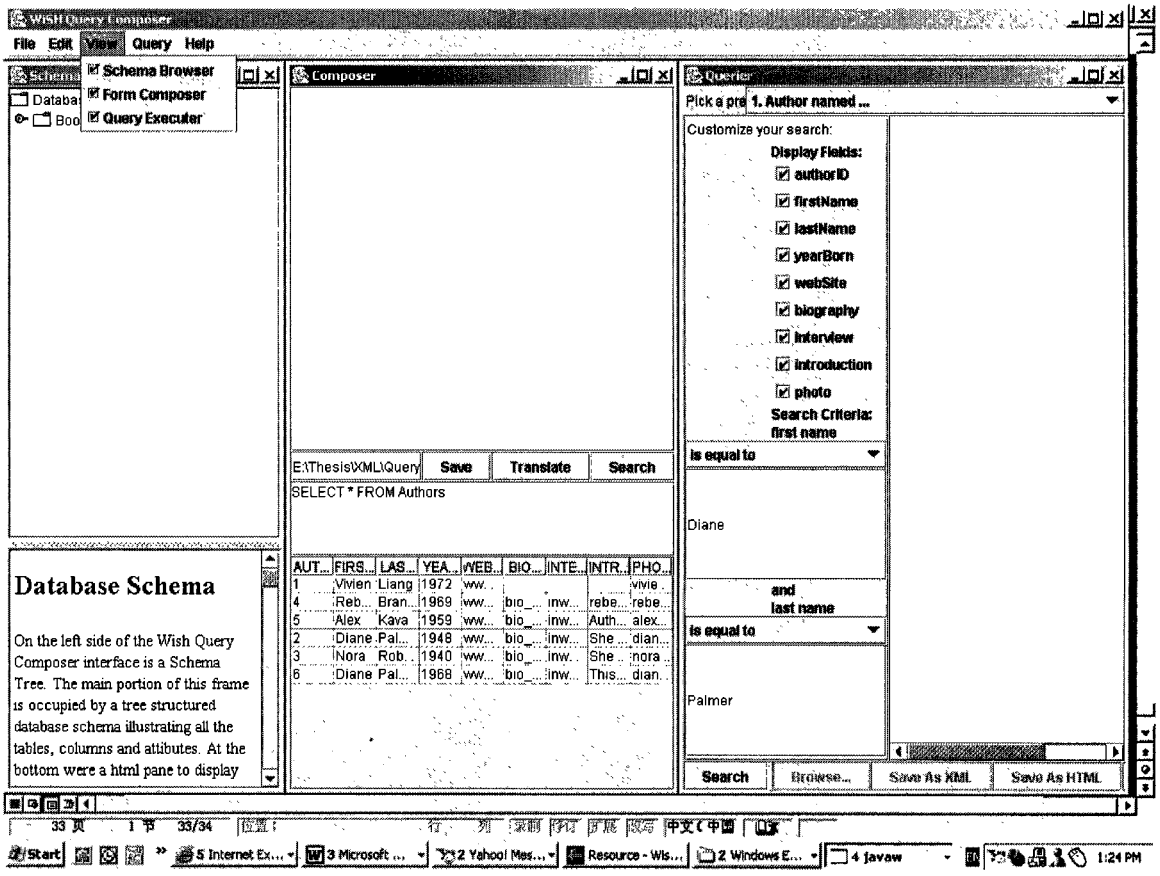

Figure 3.5.1 Screenshot of the WISH XML Query Composer

3.6 Implementation and Test Environment

3.6.1 J2SE

WISH was developed on Java 2 Platform Standard Edition (J2SE). The latest version (J2SDK 1.4.1) was downloaded from Sun Microsystem's website (http://java.sun.com). J2SE Swing components are used heavily for advanced and complex user interface design.

3.6.2 JAXB in JWSDP

The mapping between XML and Java classes was implemented using JAXB, one of the Java XML APIs in Java Web Services Developer Pack (Java WSDP). The latest version of Java WSDP (1.1) is an all-in-one download containing key technologies to simplify building of Web services using the Java 2 Platform.

3.6.3 JDBC or ODBC Drivers

A suitable database driver needs to be installed in order to access databases. Wish XML Query Composer supports JDBC 1.x or later APIs. JDBC driver classes are referenced by the CLASSPATH environmental variable. For more information about adding the driver classes to the CLASSPATH, refer to the documentation supplied by the driver vendor.

For database servers that do not have a JDBC driver, existing ODBC connection (in other words, an ODBC data source) can be used to connect to a database. The default JDBC-ODBC bridge driver contained in the JDK can communicate with any ODBC compliant database driver.

3.6.4 Test Database

For testing, the Bookstore database was created in Microsoft SQL Server 2000. Test data were imported from Books.mdb of "Java 2 How to Program" [JHP99] and www.mirabooks.com web site. JDBC-ODBC bridge driver is used to access the test database.

4. Conclusion

Revolutions often take place when new technologies are applied to traditional areas. For more than 20 years relational databases and SQL queries have been the dominating methods in data archiving and manipulation. In the recent decade, XML has emerged as the standard for exchanging data across disparate systems, and Java technology provides a platform for building portable applications. As natural partners in exchanging data and programs across different platforms, XML and Java technology can change the way people perceive database query.

4.1 Summary of Contributions

In this thesis WISH XML Query Composer was designed and implemented to assist users to compose and execute form-based queries and generate associated reports for relational databases.

In order to achieve this goal, XML and Java technology were used jointly (through the use of JAXB) to enable seamless translation between XML and database, with a user-friendly GUI interface. The advanced users who use the Form Composer will benefit from the visual aid of the Schema Browser with dynamic help documents and tips. The form-based Query Executer interface presents an easy and intuitive interaction style, as compared to command-line SQL query scripting. It simplifies query composition by showing all available

options, defaults and current values, therefore very little user training is required and recognition memory can be used to generate meaningful queries. In summary, WISH XML Query Composer is a wish-come-true for novice users with no programming or database background. It helps them easily query relational databases and generate reports according to their wish.

In completion of this thesis, I have intensively studied several knowledge areas and methodologies, including database systems and query languages, framework and interface development, GUI design principles, object-oriented design and UML, evolving XML standards and Java programming, as well as technical documentation. As a result, my knowledge and skills have been greatly enhanced.

4.2 Future Work

Following are some possible feature enhancements of WISH XML Query Composer in the future:

1. Add Connection Manager component to support any databases that can be accessed through JDBC or ODBC, multiple database connections and multiple database schemas.

2. Retrieve database metadata to generate database schema definition XML. After a connection has been established to a database, schema metadata can be retrieved using the JDBC DatabaseMetaData class methods, such as getSchema(), getTables(), getTableTypes(), getColumns(). This JDBC interface

is implemented to let users know the capabilities of a DBMS in combination with the driver based on JDBC technology. Our DBMS-independent database tool can use this API because most JDBC drivers of major DBMS support it. Another related API ResultSetMetaData class can also be used to retrieve resultset metadata. The XML Database Schema definition can include more data types, such as all MIME types supported in Java. Support SYSTEM TABLE and VIEW table types in metadata browsing and reporting; data type information with precision and scale; primary keys, imported and exported keys in metadata browsing and reporting, if database driver provides the information.

3. Create Form-based Query Wizard for building complex queries, and automatically creating hierarchical output in XML and HTML format. Or support drag-and-drop creation of reports for database metadata, table contents and query results in XML and HTML formats

4. Present or store database column information in XML documents either as elements or attributes

5. Support object-oriented databases compliant with ODMG and OQL. Based on the numerous researches and studies [SOD00, VOO99, WQL00, XTS00], XML to object database translation is a matter of mapping JAXB derived objects to objects in the ODBC. To achieve object database compatibility, about 80% of the current design can be reused. We will need to enhance the QueryProcessor class to handle XML translation into OQL (instead of SQL), to connect to the object database and get the result objects.

Bibliography

[ADS03] John L.H., Aqua Data Studio, AquaFold Inc., 2003

[DCC00] ATG, Dynamo Control Center, 2000

[DQT01] Ling Chen, "User Interface Design for a Diagrammatic Query Tool", Concordia University, 2001.

[DSI00] Hector Garcia-Molina, Jeffery D. Ullman and Jennifer Widom, "Database System Implementation", ISBN 0-13-040264-8, 2000.

[DXV02] DataMirror, DB-XML Vision, 2002

[FBA94] Adam, Nabil R., Gangopadhyay, Aryya and Clifford, James "A Form-Based Approach to Natural Language Query Processing", Proc. Journal of Management Information Systems Vol. 11 No. 2, pp. 109 – 136, Fall 1994

[JAX03] Ed Ort and Bhakti Mehta, "Java Architecture for XML Binding(JAXB)", Technical Article on java.sun.com, 2003.

[JHP99] H.M. Deitel and P.J. Deitel, *Java: How to Program*, Prentice Hall, ISBN 0-13-012507-5, 1999.

[JWS03] Eric Armstrong, Stephanie Bodoff, Debbie Carson, Maydene Fisher, Scott Fordin, Dale Green, Kim Haase and Eric Jendrock, "The Java Web Services Tutorial", February 2003

[KIA02] G. Butler, L. Chen, X. Chen, A. Gaffar, J. Li, and L. Xu, "The Know-It-All Project: A Case Study in Framework Development and Evolution", To appear in Domain Oriented Systems Development – Practices and Perspectives, K. Itoh and S. Kumagai (eds.), 2002.

[QBE77] Zloof M., Query By Example: A Database Language, IBM System Journal, Vol. 16, n°4, 1977

[SFU99] Larry L. Constantine and Lucy A.D. Lockwood, "Software for Use", ISBN 0-201-924781, 1999

[SOD00] Raymond Wong, "Semistructured Object Database", Version 2, University of New South Wales, Australia, 2000

[SQA00] Microsoft, SQL Server 2000 Query Analyzer, 2000.

[TSP99] Kevin Kline, Lee Gould and Andrew Zanevsky, "Transact-SQL Programming", 1st Edition, ch.1, ISBN 1-56592-401-0, March 1999.

[UML99] James Rumbaugh, Ivar Jacobson and Grady Booch, The Unified Modeling Language Reference Manual", ISBN 0-201-30998-X, 1999

[USD99] Ivar Jacobson, Grady Booch and James Rumbaugh, The Unified Software Development Process", ISBN 0-201-57169-2, 1999

[VOO99] Leonidas Fegaras, "A Visual Object-Oriented Database language for ODMG OQL", University of Texas at Arlington, 1999

[WQL00] Eric Andonoff, Christian Sallaberry and Nicolas Belloir, "WDBQL: A Web-Based Query Language for Remote Relational and Object-Oriented Databases" Proc. 10th International Conference on Computing and Information, Kuwait City, Kuwait, November 18-21, 2000.

[XDT00] Liam Quin, "Open Source XML Database Toolkit", ISBN 0-471-37522-5, 2000.

[XLE01] IBM, "XML Lightweight Extractor", Feb 2001. http://alphaworks.ibm.com/tech/xle

[XML00] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, *"Extensible Markup Language (XML) 1.0 (Second Edition)"* W3C Recommendation, October 2000. http://www.w3.org/TR/REC-xml

[XQL03] XQuery 1.0: An XML Query Language, W3C Working Draft 02, May 2003. http://www.w3.org/TR/xquery/

[XSL99] James Clark, *"XSL Transformations (XSLT) Version 1.0"* W3C Recommendation, November 1999. http://www.w3.org/TR/xslt

[XSP01] David C. Fallside, *"XML Schema Part 1,2,3"* W3C Recommendation, May 2001. http://www.w3.org/TR/REC-xml

[XTS00] Matthias Geiss, "An XML to Object Oriented Database Translation System", University of Newcastle Upon Tyne, U.K., April 2000

Appendix A: XML Schema for Query Form

(PreQuery.xsd)

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:annotation>
   <xsd:documentation xml:lang="en">
     XML Schema for form-based Query in WISH XML Query Composer.
   </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="QUERY" type="WishQuery"/>

 <xsd:complexType name="WishQuery">
   <xsd:sequence>
     <xsd:element name="OUTPUT" type="WishOutput"
maxOccurs="unbounded"/>
     <xsd:element name="SOURCE" type="WishSource"
maxOccurs="unbounded"/>
     <xsd:element name="CONDITION" type="WishCondition" minOccurs="0"
maxOccurs="unbounded"/>
     <xsd:element name="GROUP" type="WishOutput" minOccurs="0"
maxOccurs="unbounded"/>
     <xsd:element name="HAVING" type="WishCondition" minOccurs="0"
maxOccurs="unbounded"/>
     <xsd:element name="SORT" type="WishSort" minOccurs="0"
maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="ID" type="xsd:string" use="required"/>
   <xsd:attribute name="DESCRIPTION" type="xsd:string"/>
   <xsd:attribute name="SUBQUERY" type="xsd:boolean"/>
 </xsd:complexType>

 <xsd:complexType name="WishOutput">
      <xsd:attribute name="NAME" type="xsd:string" use="required"/>
      <xsd:attribute name="COLUMN" type="xsd:string" use="required"/>
      <xsd:attribute name="TABLE" type="xsd:string"/>
      <xsd:attribute name="DISPLAY" type="xsd:string"/>
 </xsd:complexType>

 <xsd:complexType name="WishSource">
      <xsd:sequence>
           <xsd:element name="KEY" type="WishKey" minOccurs="0"
maxOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute name="NAME" type="xsd:string" use="required"/>
      <xsd:attribute name="TABLE" type="xsd:string" use="required"/>
      <xsd:attribute name="DISPLAY" type="xsd:string"/>
      <xsd:attribute name="JOIN" type="WishJoin"/>
      <xsd:attribute name="PARENT" type="xsd:string"/>
 </xsd:complexType>

 <xsd:complexType name="WishCondition">
   <xsd:sequence>
```

```
      <xsd:element name="EXPRESSION" type="WishExpression"/>
      <xsd:element name="OPERATOR" type="WishOperator"/>
      <xsd:element name="INPUT" type="WishInput" minOccurs="0"/>
      <xsd:element name="QUERY" type="WishQuery" minOccurs="0"/>
   </xsd:sequence>
   <xsd:attribute name="MORE" type="WishMore"/>
 </xsd:complexType>

 <xsd:complexType name="WishSort">
      <xsd:attribute name="DISPLAY" type="xsd:string" use="required"/>
      <xsd:attribute name="TABLE" type="xsd:string"/>
      <xsd:attribute name="COLUMN" type="xsd:string" use="required"/>
      <xsd:attribute name="ORDER" type="WishOrder"/>
 </xsd:complexType>

 <xsd:complexType name="WishKey">
   <xsd:attribute name="COLUMN" type="xsd:string" use="required"/>
   <xsd:attribute name="PARENT-TABLE" type="xsd:string"
use="required"/>
   <xsd:attribute name="PARENT-COLUMN" type="xsd:string"
use="required"/>
 </xsd:complexType>

 <xsd:complexType name="WishOperator">
   <xsd:sequence>
      <xsd:element name="OPTION" type="WishOption" maxOccurs="15" />
   </xsd:sequence>
   <xsd:attribute name="NAME" type="xsd:string" use="required"/>
   <xsd:attribute name="SELECTED" type="xsd:positiveInteger"
use="required"/>
 </xsd:complexType>

 <xsd:complexType name="WishOption">
   <xsd:attribute name="SEQNUM" type="xsd:positiveInteger"
use="required"/>
   <xsd:attribute name="DISPLAY" type="xsd:string" use="required"/>
   <xsd:attribute name="SQLOP" type="WishSqlOp" use="required"/>
 </xsd:complexType>

 <xsd:complexType name="WishExpression">
   <xsd:attribute name="DISPLAY" type="xsd:string"/>
   <xsd:attribute name="TABLE" type="xsd:string"/>
   <xsd:attribute name="COLUMN" type="xsd:string" use="required"/>
 </xsd:complexType>

 <xsd:complexType name="WishInput">
   <xsd:attribute name="NAME" type="xsd:string" use="required"/>
   <xsd:attribute name="SIZE" type="xsd:positiveInteger"/>
   <xsd:attribute name="VALUE" type="xsd:string"/>
   <xsd:attribute name="FIXED" type="xsd:boolean"/>
   <xsd:attribute name="TYPE" type="WishType"/>
 </xsd:complexType>

 <xsd:simpleType name="WishJoin">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="inner"/>
```

```
      <xsd:enumeration value="left"/>
      <xsd:enumeration value="right"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="WishMore">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="and"/>
      <xsd:enumeration value="or"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="WishType">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="string"/>
      <xsd:enumeration value="integer"/>
      <xsd:enumeration value="float"/>
      <xsd:enumeration value="datetime"/>
      <xsd:enumeration value="bit"/>
      <xsd:enumeration value="boolean"/>
      <xsd:enumeration value="subquery"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="WishSqlOp">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="none"/>
      <xsd:enumeration value="and"/>
      <xsd:enumeration value="or"/>
      <xsd:enumeration value="is"/>
      <xsd:enumeration value="is not"/>
      <xsd:enumeration value="is null"/>
      <xsd:enumeration value="is not null"/>
      <xsd:enumeration value="like"/>
      <xsd:enumeration value="not like"/>
      <xsd:enumeration value="="/>
      <xsd:enumeration value="&lt;&gt;"/>
      <xsd:enumeration value="&lt;"/>
      <xsd:enumeration value="&lt;="/>
      <xsd:enumeration value="&gt;"/>
      <xsd:enumeration value="&gt;="/>
      <xsd:enumeration value="&lt;&gt;"/>
      <xsd:enumeration value="in"/>
      <xsd:enumeration value="not in"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="WishOrder">
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="ASC"/>
      <xsd:enumeration value="DESC"/>
    </xsd:restriction>
  </xsd:simpleType>

</xsd:schema>
```

Appendix B: 13 Bookstore Example Queries in SQL

```
Q1. Return author named Diana Palmer.
   SELECT * FROM Authors WHERE FirstName='Diana' AND LastName='Palmer'

Q2. Return authors born earlier than 1960 sorted by year of birth and
last name.
   SELECT * FROM Authors WHERE YearBorn<'1960'
   ORDER BY YearBorn, LastName

Q3. Return the names of authors.
   SELECT FirstName + ' ' + LastName AS Name FROM Authors

Q4. Return all the information related to title 'A Perfect Evil'.
   SELECT * FROM ((((Titles
   LEFT JOIN Publishers ON Titles.PublisherID = Publishers.PublisherID)
   LEFT JOIN TitleAuthor ON Titles.ProductID=TitleAuthor.ProductID)
   LEFT JOIN Authors ON TitleAuthor.AuthorID=Authors.AuthorID
   LEFT JOIN TitleCategory ON Titles.ProductID=TitleCategory.ProductID)
   LEFT JOIN Categories ON TitleCategory.CategoryID=
            Categories.CategoryID)
   WHERE Title = 'A Perfect Evil'

Q5. Return titles published by Harlequin in May 2003.
   SELECT * FROM Titles
   INNER JOIN Publishers ON Titles.PublisherID = Publishers.PublisherID
   WHERE PublisherName = 'Harlequin'
   AND DatePubished = '5/1/2003 12:00:00'

Q6. Return titles priced higher than $10 after discount IN descending
order of price.
   SELECT *, ListPrice * (1 - DiscountRate) FROM Titles
   LEFT JOIN Publishers ON Titles.PublisherID = Publishers.PublisherID
   WHERE ListPrice * (1 - DiscountRate) > 10
   ORDER BY ListPrice * (1 - DiscountRate) DESC

Q7. Return available titles IN the same category as "A Perfect Evil"
   SELECT * FROM Titles
   LEFT JOIN Publishers ON Titles.PublisherID = Publishers.PublisherID
   INNER JOIN TitleCategory ON Titles.ProductID=TitleCategory.ProductID
   WHERE Available<>0 AND CategoryID IN
      (SELECT CategoryID FROM TitleCategory
      INNER JOIN Titles ON TitleCategory.ProductID=Titles.ProductID
      WHERE Title = 'A Perfect Evil')

Q8. Return titles written by author Diana Palmer.
   SELECT * FROM Titles
   LEFT JOIN Publishers ON Titles.PublisherID = Publishers.PublisherID
   WHERE ProductID IN
      (SELECT ProductID FROM TitleAuthor
      INNER JOIN Authors ON TitleAuthor.AuthorID=Authors.AuthorID
      WHERE FirstName='Diane' AND LastName='Palmer')

Q9. Return authors in alphabetical order of last name AND titles
written by them.
```

```
   SELECT * FROM Authors
   LEFT JOIN TitleAuthors ON Authors.AuthorID=TitleAuthor.AuthorID
   LEFT JOIN Titles ON TitleAuthor.ProductID=Title.ProductID
   LEFT JOIN Publishers ON Titles.PublisherID = Publishers.PublisherID
   ORDER BY LastName

Q10. Return authors and available titles written by them.
   SELECT * FROM Authors
   LEFT JOIN TitleAuthors ON Authors.AuthorID=TitleAuthor.AuthorID
   LEFT JOIN Titles ON TitleAuthor.ProductID=Title.ProductID
   LEFT JOIN Publishers ON Titles.PublisherID = Publishers.PublisherID
   WHERE Available<>0

Q11. Return authors writing titles in Love INspired category.
   SELECT * FROM Authors WHERE AuthorID IN
      (SELECT AuthorID FROM TitleAuthor
       INNER JOIN TitleCategory ON
                TitleAuthor.ProductID=TitleCategory.ProductID
       INNER JOIN Categories ON
                TitleCategory.CategoryID=Categories.CategoryID
       WHERE DisplayName = 'Love INspired')

Q12. Return authors only writing titles in Love Inspired category.
     SELECT * FROM Authors WHERE AuthorID IN
      (SELECT AuthorID FROM (
       (TitleAuthor INNER JOIN TitleCategory
        ON TitleAuthor.ProductID=TitleCategory.ProductID)
        INNER JOIN Categories
        ON TitleCategory.CategoryID=Categories.CategoryID))
       AND AuthorID NOT IN
        (SELECT AuthorID FROM (
        (TitleAuthor INNER JOIN TitleCategory
          ON TitleAuthor.ProductID=TitleCategory.ProductID)
         INNER JOIN Categories
          ON TitleCategory.CategoryID=Categories.CategoryID)
      WHERE DisplayName <> 'Thrillers')

Q13. Return average price of books IN each category sorted by average
price.
      SELECT DisplayName, AVG(ListPrice) AS AvgPrice
      FROM (Categories INNER JOIN TitleCategory
       ON TitleCategory.CategoryID=Categories.CategoryID)
      INNER JOIN Titles ON TitleCategory.ProductID=Titles.ProductID
      GROUP BY Categories.DisplayName
      HAVING AVG(ListPrice) NOT ISNULL
      ORDER BY AvgPrice
```

Appendix C: XML Files for 13 Example Queries

Query1.xml

```
<?xml-stylesheet type="text/xsl" href="WishQuery.xsl"?>
<QUERY ID="1" DESCRIPTION="Return author whose name ...">
      <OUTPUT NAME="authorID" COLUMN="AuthorID" />
      <OUTPUT NAME="firstName" COLUMN="FirstName" />
      <OUTPUT NAME="lastName" COLUMN="LastName" />
      <OUTPUT NAME="yearBorn" COLUMN="YearBorn" DISPLAY="year of birth"
/>
      <OUTPUT NAME="webSite" COLUMN="WebSite" DISPLAY="personal web
site" />
      <OUTPUT NAME="biography" COLUMN="Biography" />
      <OUTPUT NAME="interview" COLUMN="Interview" />
      <OUTPUT NAME="introduction" COLUMN="Introduction" />
      <OUTPUT NAME="photo" COLUMN="Photo" />
      <SOURCE NAME="author" TABLE="Authors" />
      <CONDITION>
          <EXPRESSION DISPLAY="first name" COLUMN="FirstName" />
              <OPERATOR NAME="OP1" SELECTED="4">
              <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none" />
              <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not
null" />
              <OPTION SEQNUM="3" DISPLAY="is not defined" SQLOP="is
null" />
              <OPTION SEQNUM="4" DISPLAY="is equal to" SQLOP="=" />
              <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="5" />
              <OPTION SEQNUM="6" DISPLAY="contains" SQLOP="like" />
              <OPTION SEQNUM="7" DISPLAY="does not contain" SQLOP="not
like" />
          </OPERATOR>
          <INPUT NAME="Input1" TYPE="string" SIZE="30" VALUE="Diane" />
      </CONDITION>
      <CONDITION MORE="and">
          <EXPRESSION DISPLAY="last name" COLUMN="LastName" />
          <OPERATOR NAME="OP2" SELECTED="4">
              <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none" />
              <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not
null" />
              <OPTION SEQNUM="3" DISPLAY="is not defined" SQLOP="is
null" />
              <OPTION SEQNUM="4" DISPLAY="is equal to" SQLOP="=" />
              <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="5" />
              <OPTION SEQNUM="6" DISPLAY="contains" SQLOP="like" />
              <OPTION SEQNUM="7" DISPLAY="does not contain" SQLOP="not
like" />
          </OPERATOR>
          <INPUT NAME="Input2" TYPE="string" SIZE="30" VALUE="Palmer"
/>
      </CONDITION>
</QUERY>
```

Query2.xml

```
<?xml-stylesheet type="text/xsl" href="WishQuery.xsl"?>
<QUERY ID="2" DESCRIPTION="Return author whose year of birth ... in
order of age">
      <OUTPUT NAME="authorID" COLUMN="AuthorID" />
      <OUTPUT NAME="firstName" COLUMN="FirstName" />
      <OUTPUT NAME="lastName" COLUMN="LastName" />
      <OUTPUT NAME="yearBorn" COLUMN="YearBorn" />
      <OUTPUT NAME="webSite" COLUMN="WebSite" />
      <OUTPUT NAME="biography" COLUMN="Biography" />
      <OUTPUT NAME="interview" COLUMN="Interview" />
      <OUTPUT NAME="introduction" COLUMN="Introduction" />
      <OUTPUT NAME="photo" COLUMN="Photo" />
      <SOURCE NAME="authors" TABLE="Authors" />
      <CONDITION>
        <EXPRESSION DISPLAY="year of birth" COLUMN="YearBorn" />
        <OPERATOR NAME="OP1" SELECTED="4">
            <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none" />
            <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"
/>
            <OPTION SEQNUM="3" DISPLAY="is undefined" SQLOP="is null"
/>
            <OPTION DISPLAY="is earlier than" SQLOP="&lt;" SEQNUM="4"
/>
            <OPTION DISPLAY="is no earlier than" SQLOP="&gt;="
SEQNUM="5" />
            <OPTION SEQNUM="6" DISPLAY="is equal to" SQLOP="=" />
            <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="7" />
            <OPTION DISPLAY="is later than" SQLOP="&gt;" SEQNUM="8" />
            <OPTION DISPLAY="is no later than" SQLOP="&lt;=" SEQNUM="9"
/>
            <OPTION SEQNUM="10" DISPLAY="contains" SQLOP="like" />
            <OPTION SEQNUM="11" DISPLAY="does not contain" SQLOP="not
like" />
        </OPERATOR>
        <INPUT NAME="Input1" TYPE="string" SIZE="4" VALUE="1960" />
      </CONDITION>
      <SORT DISPLAY="year of birth" COLUMN="YearBorn" ORDER="DESC" />
      <SORT DISPLAY="lastName" COLUMN="LastName" />
</QUERY>
```

Query3.xml

```
<?xml-stylesheet type="text/xsl" href="WishQuery.xsl"?>
<QUERY ID="3" DESCRIPTION="Return names of all authors order by
LastName.">
      <OUTPUT NAME="authorName" COLUMN="FirstName + ' ' + LastName" />
      <SOURCE NAME="authors" TABLE="Authors"/>
      <SORT DISPLAY="Last Name" COLUMN="LastName" />
```

```
</QUERY>
```

Query4.xml

```
<?xml-stylesheet type="text/xsl" href="WishQuery.xsl"?>
<QUERY ID="4" DESCRIPTION="Return all information related to
title ...">
      <OUTPUT NAME="productID" COLUMN="ProductID" TABLE="Titles"/>
      <OUTPUT NAME="isbn" COLUMN="ISBN" />
      <OUTPUT NAME="title" COLUMN="Title" />
      <OUTPUT NAME="editionNumber" COLUMN="EditionNumber" />
      <OUTPUT NAME="datePublished" COLUMN="DatePublished" />
      <OUTPUT NAME="bookDescription" COLUMN="Description"
TABLE="Titles" />
      <OUTPUT NAME="available" COLUMN="Available" />
      <OUTPUT NAME="formats" COLUMN="Formats" />
      <OUTPUT NAME="coverImage" COLUMN="CoverImage" />
      <OUTPUT NAME="listPrice" COLUMN="ListPrice" />
      <OUTPUT NAME="discountRate" COLUMN="DiscountRate" />
      <OUTPUT NAME="publisherID" COLUMN="PublisherID" TABLE="Titles" />
      <OUTPUT NAME="publisherName" COLUMN="PublisherName" />
      <OUTPUT NAME="authorID" COLUMN="AuthorID" TABLE="Authors" />
      <OUTPUT NAME="firstName" COLUMN="FirstName" />
      <OUTPUT NAME="lastName" COLUMN="LastName" />
      <OUTPUT NAME="yearBorn" COLUMN="YearBorn" DISPLAY="year of birth"
/>
      <OUTPUT NAME="webSite" COLUMN="WebSite" DISPLAY="personal web
site" />
      <OUTPUT NAME="biography" COLUMN="Biography" />
      <OUTPUT NAME="interview" COLUMN="Interview" />
      <OUTPUT NAME="introduction" COLUMN="Introduction" />
      <OUTPUT NAME="photo" COLUMN="Photo" />
      <OUTPUT NAME="categoryID" COLUMN="CategoryID" TABLE="Categories"
/>
      <OUTPUT NAME="startDate" COLUMN="StartDate" />
      <OUTPUT NAME="endDate" COLUMN="EndDate" />
      <OUTPUT NAME="displayName" COLUMN="DisplayName" />
      <OUTPUT NAME="categoryDescription" COLUMN="Description"
TABLE="Categories" />
      <SOURCE NAME="titles" TABLE="Titles" />
      <SOURCE NAME="ti_pu" TABLE="Publishers" JOIN="left"
PARENT="titles">
       <KEY COLUMN="PublisherID" PARENT-TABLE="Titles" PARENT-
COLUMN="PublisherID"/>
      </SOURCE>
      <SOURCE NAME="ti_pu_a" TABLE="TitleAuthor" JOIN="left"
PARENT="ti_pu">
       <KEY COLUMN="ProductID" PARENT-TABLE="Titles" PARENT-
COLUMN="ProductID"/>
      </SOURCE>
      <SOURCE NAME="ti_pu_au" TABLE="Authors" JOIN="left"
PARENT="ti_pu_a">
       <KEY COLUMN="AuthorID" PARENT-TABLE="TitleAuthor" PARENT-
COLUMN="AuthorID"/>
```

```
      </SOURCE>
      <SOURCE NAME="ti_pu_au_c" TABLE="TitleCategory" JOIN="left"
PARENT="ti_pu_au">
       <KEY COLUMN="ProductID" PARENT-TABLE="Titles" PARENT-
COLUMN="ProductID"/>
      </SOURCE>
      <SOURCE NAME="ti_pu_au_ca" TABLE="Categories" JOIN="left"
PARENT="ti_pu_au_c">
       <KEY COLUMN="CategoryID" PARENT-TABLE="TitleCategory" PARENT-
COLUMN="CategoryID"/>
      </SOURCE>
      <CONDITION>
        <EXPRESSION DISPLAY="title" COLUMN="Title" TABLE="Titles" />
        <OPERATOR NAME="OP1" SELECTED="4">
            <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none" />
            <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"
/>
            <OPTION SEQNUM="3" DISPLAY="is undefined" SQLOP="is null"
/>
            <OPTION SEQNUM="4" DISPLAY="is equal to" SQLOP="=" />
            <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="5" />
            <OPTION SEQNUM="6" DISPLAY="contains" SQLOP="like" />
            <OPTION SEQNUM="7" DISPLAY="does not contain" SQLOP="not
like" />
        </OPERATOR>
        <INPUT NAME="Input1" TYPE="string" SIZE="255" VALUE="A Perfect
Evil" />
      </CONDITION>
</QUERY>
```

Query5.xml

```
<?xml-stylesheet type="text/xsl" href="WishQuery.xsl"?>
<QUERY ID="5" DESCRIPTION="Return titles published by ... in
month ...">
      <OUTPUT NAME="productID" COLUMN="ProductID" />
      <OUTPUT NAME="isbn" COLUMN="ISBN" />
      <OUTPUT NAME="title" COLUMN="Title" />
      <OUTPUT NAME="editionNumber" COLUMN="EditionNumber" />
      <OUTPUT NAME="datePublished" COLUMN="DatePublished" />
      <OUTPUT NAME="bookDescription" COLUMN="Description"
TABLE="Titles" />
      <OUTPUT NAME="available" COLUMN="Available" />
      <OUTPUT NAME="formats" COLUMN="Formats" />
      <OUTPUT NAME="coverImage" COLUMN="CoverImage" />
      <OUTPUT NAME="listPrice" COLUMN="ListPrice" />
      <OUTPUT NAME="discountRate" COLUMN="DiscountRate" />
      <OUTPUT NAME="publisherID" COLUMN="PublisherID" TABLE="Titles" />
      <OUTPUT NAME="publisherName" COLUMN="PublisherName" />
      <SOURCE NAME="titles" TABLE="Titles" />
      <SOURCE NAME="ti_pu" TABLE="Publishers" JOIN="inner"
PARENT="titles">
       <KEY COLUMN="PublisherID" PARENT-TABLE="Titles" PARENT-
```

```
COLUMN="PublisherID"/>
      </SOURCE>
      <CONDITION>
        <EXPRESSION DISPLAY="publisher's name" COLUMN="PublisherName"
TABLE="Publishers"/>
        <OPERATOR NAME="OP1" SELECTED="4">
            <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none" />
            <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"
/>
            <OPTION SEQNUM="3" DISPLAY="is undefined" SQLOP="is null"
/>
            <OPTION SEQNUM="4" DISPLAY="is equal to" SQLOP="="   />
            <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="5" />
            <OPTION SEQNUM="6" DISPLAY="contains" SQLOP="like" />
            <OPTION SEQNUM="7" DISPLAY="does not contain" SQLOP="not
like" />
          </OPERATOR>
          <INPUT NAME="Input1" TYPE="string" SIZE="100"
VALUE="Harlequin" />
      </CONDITION>
      <CONDITION MORE="and">
        <EXPRESSION DISPLAY="date of publish" COLUMN="DatePublished" />
        <OPERATOR NAME="OP2" SELECTED="5">
            <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
            <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"
/>
            <OPTION SEQNUM="3" DISPLAY="is not defined" SQLOP="is null"
/>
            <OPTION DISPLAY="is earlier than" SQLOP="&lt;" SEQNUM="4"
/>
            <OPTION DISPLAY="is no earlier than" SQLOP="&gt;="
SEQNUM="5" />
            <OPTION SEQNUM="6" DISPLAY="is equal to" SQLOP="=" />
            <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="7" />
            <OPTION DISPLAY="is later than" SQLOP="&gt;" SEQNUM="8" />
            <OPTION DISPLAY="is no later than" SQLOP="&lt;=" SEQNUM="9"
/>
          </OPERATOR>
          <INPUT NAME="Input2" TYPE="datetime" SIZE="30"
VALUE="5/1/2003" />
      </CONDITION>
      <CONDITION MORE="and">
        <EXPRESSION DISPLAY="date of publish" COLUMN="DatePublished" />
        <OPERATOR NAME="OP3" SELECTED="9">
            <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
            <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"
/>
            <OPTION SEQNUM="3" DISPLAY="is not defined" SQLOP="is null"
/>
            <OPTION DISPLAY="is earlier than" SQLOP="&lt;" SEQNUM="4"
/>
            <OPTION DISPLAY="is no earlier than" SQLOP="&gt;="
SEQNUM="5" />
            <OPTION SEQNUM="6" DISPLAY="is equal to" SQLOP="=" />
```

```
            <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="7" />
            <OPTION DISPLAY="is later than" SQLOP="&gt;" SEQNUM="8" />
            <OPTION DISPLAY="is no later than" SQLOP="&lt;=" SEQNUM="9"
/>
          </OPERATOR>
          <INPUT NAME="Input3" TYPE="datetime" SIZE="30"
VALUE="6/1/2003"/>
      </CONDITION>
</QUERY>
```

Query6.xml

```
<?xml-stylesheet type="text/xsl" href="WishQuery.xsl"?>
<QUERY ID="6" DESCRIPTION="Return titles sorted by price with discounted
price ... ">
      <OUTPUT NAME="productID" COLUMN="ProductID" TABLE="Titles" />
      <OUTPUT NAME="isbn" COLUMN="ISBN" />
      <OUTPUT NAME="title" COLUMN="Title" />
      <OUTPUT NAME="editionNumber" COLUMN="EditionNumber" />
      <OUTPUT NAME="datePublished" COLUMN="DatePublished" />
      <OUTPUT NAME="bookDescription" COLUMN="Description" TABLE="Titles"
/>
      <OUTPUT NAME="available" COLUMN="Available" />
      <OUTPUT NAME="formats" COLUMN="Formats" />
      <OUTPUT NAME="coverImage" COLUMN="CoverImage" />
      <OUTPUT NAME="listPrice" COLUMN="ListPrice" />
      <OUTPUT NAME="discountRate" COLUMN="DiscountRate" />
      <OUTPUT NAME="discountPrice" COLUMN="ListPrice * (1 -
DiscountRate)" />
      <OUTPUT NAME="publisherID" COLUMN="PublisherID" TABLE="Titles" />
      <OUTPUT NAME="publisherName" COLUMN="PublisherName" />
      <SOURCE NAME="titles" TABLE="Titles" />
      <SOURCE NAME="ti_pu" TABLE="Publishers" JOIN="left"
PARENT="titles">
       <KEY COLUMN="PublisherID" PARENT-TABLE="Titles" PARENT-
COLUMN="PublisherID"/>
      </SOURCE>
      <CONDITION>
        <EXPRESSION DISPLAY="on sale price" COLUMN="ListPrice * (1-
DiscountRate)"/>
        <OPERATOR NAME="OP1" SELECTED="5">
            <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
            <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"
/>
            <OPTION SEQNUM="3" DISPLAY="is not defined" SQLOP="is null"
/>
            <OPTION DISPLAY="is lower than" SQLOP="&lt;" SEQNUM="4" />
            <OPTION DISPLAY="is no lower than" SQLOP="&gt;=" SEQNUM="5"
/>
            <OPTION SEQNUM="6" DISPLAY="is equal to" SQLOP="=" />
            <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="7" />
            <OPTION DISPLAY="is higher than" SQLOP="&gt;" SEQNUM="8"/>
```

```
            <OPTION DISPLAY="is no higher than" SQLOP="&lt;=" SEQNUM="9"
/>
          </OPERATOR>
          <INPUT NAME="Para1" TYPE="float" SIZE="8" VALUE="10" />
      </CONDITION>
      <CONDITION MORE="and">
        <EXPRESSION DISPLAY="on sale price" COLUMN="ListPrice * (1-
DiscountRate)" />
        <OPERATOR NAME="OP2" SELECTED="9">
            <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
            <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"
/>
            <OPTION SEQNUM="3" DISPLAY="is not defined" SQLOP="is null"
/>
            <OPTION DISPLAY="is lower than" SQLOP="&lt;" SEQNUM="4" />
            <OPTION DISPLAY="is no lower than" SQLOP="&gt;=" SEQNUM="5"
/>
            <OPTION SEQNUM="6" DISPLAY="is equal to" SQLOP="=" />
            <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="7" />
            <OPTION DISPLAY="is higher than" SQLOP="&gt;" SEQNUM="8"/>
            <OPTION DISPLAY="is no higher than" SQLOP="&lt;=" SEQNUM="9"
/>
          </OPERATOR>
          <INPUT NAME="Para2" TYPE="float" SIZE="8" VALUE="100" />
      </CONDITION>
      <SORT DISPLAY="on sale price" COLUMN="ListPrice * (1 -
DiscountRate)" TABLE="Titles" ORDER="DESC" />
</QUERY>
```

Query7.xml

```
<?xml-stylesheet type="text/xsl" href="WishQuery.xsl"?>
<QUERY ID="7" DESCRIPTION="Return available titles in the same category
as ...">
      <OUTPUT NAME="productID" COLUMN="ProductID"  TABLE="Titles" />
      <OUTPUT NAME="isbn" COLUMN="ISBN" />
      <OUTPUT NAME="title" COLUMN="Title" />
      <OUTPUT NAME="editionNumber" COLUMN="EditionNumber" />
      <OUTPUT NAME="datePublished" COLUMN="DatePublished" />
      <OUTPUT NAME="bookDescription" COLUMN="Description" TABLE="Titles"
/>
      <OUTPUT NAME="available" COLUMN="Available" />
      <OUTPUT NAME="formats" COLUMN="Formats" />
      <OUTPUT NAME="coverImage" COLUMN="CoverImage" />
      <OUTPUT NAME="listPrice" COLUMN="ListPrice" />
      <OUTPUT NAME="discountRate" COLUMN="DiscountRate" />
      <OUTPUT NAME="publisherID" COLUMN="PublisherID" TABLE="Titles" />
      <OUTPUT NAME="publisherName" COLUMN="PublisherName" />
      <SOURCE NAME="titles" TABLE="Titles" />
      <SOURCE NAME="ti_pu" TABLE="Publishers" JOIN="left"
PARENT="titles">
       <KEY COLUMN="PublisherID" PARENT-TABLE="Titles" PARENT-
COLUMN="PublisherID"/>
```

```
      </SOURCE>
      <SOURCE NAME="ti_pu_c" TABLE="TitleCategory" JOIN="inner"
PARENT="ti_pu">
          <KEY COLUMN="ProductID" PARENT-TABLE="Titles" PARENT-
COLUMN="ProductID" />
      </SOURCE>
      <CONDITION>
        <EXPRESSION DISPLAY="available" COLUMN="Available"
TABLE="Titles" />
        <OPERATOR NAME="Op1" SELECTED="5">
            <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
            <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"
/>
            <OPTION SEQNUM="3" DISPLAY="is undefined" SQLOP="is null"
/>
            <OPTION SEQNUM="4" DISPLAY="is equal to" SQLOP="=" />
            <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="5" />
        </OPERATOR>
        <INPUT NAME="Input1" TYPE="bit"  SIZE="1" VALUE="0"
FIXED="true"/>
      </CONDITION>
      <CONDITION MORE="and">
        <EXPRESSION DISPLAY="category" COLUMN="CategoryID"
TABLE="TitleCategory" />
        <OPERATOR NAME="Op2" SELECTED="4">
            <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
            <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"
/>
            <OPTION SEQNUM="3" DISPLAY="is not defined" SQLOP="is null"
/>
            <OPTION SEQNUM="4" DISPLAY="is the same as" SQLOP="in" />
            <OPTION SEQNUM="5" DISPLAY="is different than" SQLOP="not
in" />
        </OPERATOR>
        <QUERY SUBQUERY="true" ID="7-1" DESCRIPTION="return category of
book ...">
          <OUTPUT NAME="INTERNAL" COLUMN="CategoryID"
TABLE="TitleCategory" />
          <SOURCE NAME="titleCategory" TABLE="TitleCategory" />
          <SOURCE NAME="title" TABLE="Titles" JOIN="inner"
PARENT="TitleCategory">             <KEY COLUMN="ProductID" PARENT-
TABLE="TitleCategory" PARENT-COLUMN="ProductID"/>
          </SOURCE>
          <CONDITION>
            <EXPRESSION DISPLAY="title" COLUMN="Title" TABLE="Titles" />
            <OPERATOR NAME="Op3" SELECTED="4">
                <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
                <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not
null"   />
                <OPTION SEQNUM="3" DISPLAY="is undefined" SQLOP="is
null"   />
                <OPTION SEQNUM="4" DISPLAY="is equal to" SQLOP="=" />
                <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="5" />
                <OPTION SEQNUM="6" DISPLAY="contains" SQLOP="like" />
```

```
                <OPTION SEQNUM="7" DISPLAY="does not contain" SQLOP="not
like" />
            </OPERATOR>
            <INPUT NAME="Input2" TYPE="string" SIZE="255" VALUE="A
Perfect Evil" />
          </CONDITION>
        </QUERY>
      </CONDITION>
</QUERY>
```

Query8.xml

```
<?xml-stylesheet type="text/xsl" href="WishQuery.xsl"?>
<QUERY ID="8" DESCRIPTION="Return titles written by author ...">
      <OUTPUT NAME="productID" COLUMN="ProductID" />
      <OUTPUT NAME="isbn" COLUMN="ISBN" />
      <OUTPUT NAME="title" COLUMN="Title" />
      <OUTPUT NAME="editionNumber" COLUMN="EditionNumber" />
      <OUTPUT NAME="datePublished" COLUMN="DatePublished" />
      <OUTPUT NAME="bookDescription" COLUMN="Description" TABLE="Titles"
/>
      <OUTPUT NAME="available" COLUMN="Available" />
      <OUTPUT NAME="formats" COLUMN="Formats" />
      <OUTPUT NAME="coverImage" COLUMN="CoverImage" />
      <OUTPUT NAME="listPrice" COLUMN="ListPrice" />
      <OUTPUT NAME="discountRate" COLUMN="DiscountRate" />
      <OUTPUT NAME="publisherID" COLUMN="PublisherID" TABLE="Titles" />
      <OUTPUT NAME="publisherName" COLUMN="PublisherName" />
      <SOURCE NAME="titles" TABLE="Titles" />
      <SOURCE NAME="ti_pu" TABLE="Publishers" JOIN="left"
PARENT="titles">
       <KEY COLUMN="PublisherID" PARENT-TABLE="Titles" PARENT-
COLUMN="PublisherID"/>
      </SOURCE>
      <CONDITION>
        <EXPRESSION DISPLAY="which" COLUMN="ProductID" TABLE="Titles" />
        <OPERATOR NAME="Op1" SELECTED="4">
            <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
            <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"
/>
            <OPTION SEQNUM="3" DISPLAY="is not defined" SQLOP="is null"
/>
            <OPTION SEQNUM="4" DISPLAY="is written by" SQLOP="in" />
            <OPTION SEQNUM="5" DISPLAY="is not written by" SQLOP="not
         in" />
        </OPERATOR>
       <QUERY SUBQUERY="true" ID="8-1" DESCRIPTION="return productID of
titles written by ...">
           <OUTPUT NAME="INTERNAL" COLUMN="ProductID"
TABLE="TitleAuthor"/>
           <SOURCE NAME="titleAuthor" TABLE="TitleAuthor" />
           <SOURCE NAME="authors" TABLE="Authors" JOIN="inner"
PARENT="TitleAuthor">
             <KEY COLUMN="AuthorID" PARENT-TABLE="TitleAuthor" PARENT-
```

```
COLUMN="AuthorID" />
            </SOURCE>
            <CONDITION>
              <EXPRESSION DISPLAY="first Name" COLUMN="FirstName" />
              <OPERATOR NAME="Op2" SELECTED="6">
                <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
                <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not
null"   />
                <OPTION SEQNUM="3" DISPLAY="is not defined" SQLOP="is
null"   />
                <OPTION SEQNUM="4" DISPLAY="is equal to" SQLOP="=" />
                <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="5" />
                <OPTION SEQNUM="6" DISPLAY="contains" SQLOP="like" />
                <OPTION SEQNUM="7" DISPLAY="does not contain" SQLOP="not
like" />
              </OPERATOR>
              <INPUT NAME="Input1" TYPE="string" SIZE="30" VALUE="Dian"
/>
            </CONDITION>
            <CONDITION MORE="and">
              <EXPRESSION DISPLAY="last name" COLUMN="LastName" />
              <OPERATOR NAME="Op3" SELECTED="4">
                <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
                <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not
null"   />
                <OPTION SEQNUM="3" DISPLAY="is not defined" SQLOP="is
null"   />
                <OPTION SEQNUM="4" DISPLAY="is equal to" SQLOP="=" />
                <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="5" />
                <OPTION SEQNUM="6" DISPLAY="contains" SQLOP="like" />
                <OPTION SEQNUM="7" DISPLAY="does not contain" SQLOP="not
like" />
              </OPERATOR>
              <INPUT NAME="Input2" TYPE="string" SIZE="30"
VALUE="Palmer" />
            </CONDITION>
          </QUERY>
      </CONDITION>
</QUERY>
```

Query9.xml

```
<?xml-stylesheet type="text/xsl" href="WishQuery.xsl"?>
<QUERY ID="9" DESCRIPTION="Return authors in alphabetical order of last
name and titles written by them.">
      <OUTPUT NAME="authorID" COLUMN="AuthorID" TABLE="Authors" />
      <OUTPUT NAME="firstName" COLUMN="FirstName" />
      <OUTPUT NAME="lastName" COLUMN="LastName" />
      <OUTPUT NAME="yearBorn" COLUMN="YearBorn" DISPLAY="year of birth"
/>
      <OUTPUT NAME="webSite" COLUMN="WebSite" DISPLAY="personal web
site" />
```

```
      <OUTPUT NAME="biography" COLUMN="Biography" />
      <OUTPUT NAME="interview" COLUMN="Interview" />
      <OUTPUT NAME="introduction" COLUMN="Introduction" />
      <OUTPUT NAME="photo" COLUMN="Photo" />
      <OUTPUT NAME="productID" COLUMN="ProductID" TABLE="Titles" />
      <OUTPUT NAME="isbn" COLUMN="ISBN" />
      <OUTPUT NAME="title" COLUMN="Title" />
      <OUTPUT NAME="editionNumber" COLUMN="EditionNumber" />
      <OUTPUT NAME="datePublished" COLUMN="DatePublished" />
      <OUTPUT NAME="bookDescription" COLUMN="Description" TABLE="Titles"
/>
      <OUTPUT NAME="available" COLUMN="Available" />
      <OUTPUT NAME="formats" COLUMN="Formats" />
      <OUTPUT NAME="coverImage" COLUMN="CoverImage" />
      <OUTPUT NAME="listPrice" COLUMN="ListPrice" />
      <OUTPUT NAME="discountRate" COLUMN="DiscountRate" />
      <OUTPUT NAME="publisherID" COLUMN="PublisherID" TABLE="Titles" />
      <OUTPUT NAME="publisherName" COLUMN="PublisherName" />
      <SOURCE NAME="authors" TABLE="Authors" />
      <SOURCE NAME="au_t" TABLE="TitleAuthor" JOIN="left"
PARENT="authors">
       <KEY COLUMN="AuthorID" PARENT-TABLE="Authors" PARENT-
COLUMN="AuthorID" />
      </SOURCE>
      <SOURCE NAME="au_ti" TABLE="Titles" JOIN="left" PARENT="au_t" >
       <KEY COLUMN="ProductID" PARENT-TABLE="TitleAuthor" PARENT-
COLUMN="ProductID" />
      </SOURCE>
      <SOURCE NAME="au_ti_pu" TABLE="Publishers" JOIN="left"
PARENT="titles">
       <KEY COLUMN="PublisherID" PARENT-TABLE="Titles" PARENT-
COLUMN="PublisherID"/>
      </SOURCE>
      <SORT DISPLAY="last name" COLUMN="LastName" ORDER="ASC" />
</QUERY>
```

Query10.xml

```
<?xml-stylesheet type="text/xsl" href="WishQuery.xsl"?>
<QUERY ID="10" DESCRIPTION="Return authors and available titles written
by them.">
      <OUTPUT NAME="authorID" COLUMN="AuthorID" TABLE="Authors" />
      <OUTPUT NAME="firstName" COLUMN="FirstName" />
      <OUTPUT NAME="lastName" COLUMN="LastName" />
      <OUTPUT NAME="yearBorn" COLUMN="YearBorn" DISPLAY="year of birth"
/>
      <OUTPUT NAME="webSite" COLUMN="WebSite" DISPLAY="personal web
site" />
      <OUTPUT NAME="biography" COLUMN="Biography" />
      <OUTPUT NAME="interview" COLUMN="Interview" />
      <OUTPUT NAME="introduction" COLUMN="Introduction" />
      <OUTPUT NAME="photo" COLUMN="Photo" />
      <OUTPUT NAME="productID" COLUMN="ProductID" TABLE="Titles"/>
      <OUTPUT NAME="isbn" COLUMN="ISBN" />
```

```
      <OUTPUT NAME="title" COLUMN="Title" />
      <OUTPUT NAME="editionNumber" COLUMN="EditionNumber" />
      <OUTPUT NAME="datePublished" COLUMN="DatePublished" />
      <OUTPUT NAME="bookDescription" COLUMN="Description" TABLE="Titles"
/>
      <OUTPUT NAME="available" COLUMN="Available" />
      <OUTPUT NAME="formats" COLUMN="Formats" />
      <OUTPUT NAME="coverImage" COLUMN="CoverImage" />
      <OUTPUT NAME="listPrice" COLUMN="ListPrice" />
      <OUTPUT NAME="discountRate" COLUMN="DiscountRate" />
      <OUTPUT NAME="publisherID" COLUMN="PublisherID" TABLE="Titles" />
      <OUTPUT NAME="publisherName" COLUMN="PublisherName" />
      <SOURCE NAME="authors" TABLE="Authors" />
      <SOURCE NAME="au_t" TABLE="TitleAuthor" JOIN="left"
PARENT="authors">
          <KEY COLUMN="AuthorID" PARENT-TABLE="Authors" PARENT-
COLUMN="AuthorID" />
      </SOURCE>
      <SOURCE NAME="au_ti" TABLE="Titles" JOIN="left" PARENT="au_t">
       <KEY COLUMN="ProductID" PARENT-TABLE="TitleAuthor" PARENT-
COLUMN="ProductID" />
      </SOURCE>
      <SOURCE NAME="au_ti_pu" TABLE="Publishers" JOIN="left"
PARENT="titles">
       <KEY COLUMN="PublisherID" PARENT-TABLE="Titles" PARENT-
COLUMN="PublisherID"/>
      </SOURCE>
      <CONDITION>
        <EXPRESSION DISPLAY="available" COLUMN="Available"
TABLE="Titles" />
        <OPERATOR NAME="Op1" SELECTED="5">
            <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
            <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"
/>
            <OPTION SEQNUM="3" DISPLAY="is undefined" SQLOP="is null"
/>
            <OPTION SEQNUM="4" DISPLAY="is equal to" SQLOP="=" />
            <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="5" />
        </OPERATOR>
        <INPUT NAME="Input1" TYPE="bit" SIZE="1" VALUE="0"
FIXED="true"/>
      </CONDITION>
</QUERY>
```

Query11.xml

```
<?xml-stylesheet type="text/xsl" href="WishQuery.xsl"?>
<QUERY ID="11" DESCRIPTION="Return author writing titles in
category ...">
      <OUTPUT NAME="authorID" COLUMN="AuthorID" />
      <OUTPUT NAME="firstName" COLUMN="FirstName" />
      <OUTPUT NAME="lastName" COLUMN="LastName" />
      <OUTPUT NAME="yearBorn" COLUMN="YearBorn" DISPLAY="year of birth"
```

```
/>
      <OUTPUT NAME="webSite" COLUMN="WebSite" DISPLAY="personal web
site" />
      <OUTPUT NAME="biography" COLUMN="Biography" />
      <OUTPUT NAME="interview" COLUMN="Interview" />
      <OUTPUT NAME="introduction" COLUMN="Introduction" />
      <OUTPUT NAME="photo" COLUMN="Photo" />
      <SOURCE NAME="authors" TABLE="Authors" />
      <CONDITION>
        <EXPRESSION DISPLAY="who" COLUMN="AuthorID" TABLE="Authors" />
        <OPERATOR NAME="OP1" SELECTED="4">
            <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
            <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"
/>
            <OPTION SEQNUM="3" DISPLAY="is not defined" SQLOP="is null"
/>
            <OPTION SEQNUM="4" DISPLAY="writes in" SQLOP="in" />
            <OPTION SEQNUM="5" DISPLAY="does not write in" SQLOP="not
in" />
        </OPERATOR>
        <QUERY ID="11-1" DESCRIPTION="category whose ..."
SUBQUERY="true">
          <OUTPUT NAME="INTERNAL" COLUMN="AuthorID" TABLE="TitleAuthor"
/>
          <SOURCE NAME="au_t" TABLE="TitleAuthor" />
          <SOURCE NAME="au_t_c" TABLE="TitleCategory" JOIN="inner"
PARENT="au_t">
            <KEY COLUMN="ProductID" PARENT-TABLE="TitleAuthor" PARENT-
COLUMN="ProductID"/>
          </SOURCE>
          <SOURCE NAME="au_t_ca" TABLE="Categories" JOIN="inner"
PARENT="au_t_c">
            <KEY COLUMN="CategoryID" PARENT-TABLE="TitleCategory"
PARENT-COLUMN="CategoryID" />
          </SOURCE>
          <CONDITION>
            <EXPRESSION DISPLAY="categoryName" COLUMN="DisplayName"
TABLE="Categories"/>
            <OPERATOR NAME="OP2" SELECTED="4">
                <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
                <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not
null"   />                  <OPTION SEQNUM="3" DISPLAY="is not defined"
SQLOP="is null"   />
                <OPTION SEQNUM="4" DISPLAY="is equal to" SQLOP="=" />
                <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="5" />
                <OPTION SEQNUM="6" DISPLAY="contains" SQLOP="like" />
                <OPTION SEQNUM="7" DISPLAY="does not contain" SQLOP="not
like" />
            </OPERATOR>
            <INPUT NAME="Input1" TYPE="string" SIZE="30"
VALUE="Thrillers" />
          </CONDITION>
        </QUERY>
      </CONDITION>
</QUERY>
```

Query12.xml

```
<?xml-stylesheet type="text/xsl" href="WishQuery.xsl"?>
<QUERY ID="12" DESCRIPTION="Return author only writing titles in
category ...">
      <OUTPUT NAME="authorID" COLUMN="AuthorID" DISPLAY="Author ID"/>
      <OUTPUT NAME="firstName" COLUMN="FirstName" DISPLAY="First Name"/>
      <OUTPUT NAME="lastName" COLUMN="LastName" DISPLAY="Last Name"/>
      <OUTPUT NAME="yearBorn" COLUMN="YearBorn" DISPLAY="Year of Birth"
/>
      <OUTPUT NAME="webSite" COLUMN="WebSite" DISPLAY="Personal Web
Site" />
      <OUTPUT NAME="biography" COLUMN="Biography" DISPLAY="Biography"/>
      <OUTPUT NAME="interview" COLUMN="Interview" DISPLAY="Interview"/>
      <OUTPUT NAME="introduction" COLUMN="Introduction"
DISPLAY="Introduction"/>
      <OUTPUT NAME="photo" COLUMN="Photo" DISPLAY="Photo"/>
      <SOURCE NAME="authors" TABLE="Authors" />
      <CONDITION>
        <EXPRESSION DISPLAY="who" COLUMN="AuthorID" TABLE="Authors" />
        <OPERATOR NAME="Operator1" SELECTED="4">
            <OPTION SEQNUM="4" DISPLAY="writes in" SQLOP="in"  />
            <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
            <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"
/>
            <OPTION SEQNUM="3" DISPLAY="is not defined" SQLOP="is null"
/>
            <OPTION SEQNUM="5" DISPLAY="does not write in" SQLOP="not
in" />
        </OPERATOR>
        <QUERY ID="12-1" DESCRIPTION="any category" SUBQUERY="true">
          <OUTPUT NAME="INTERNAL" COLUMN="AuthorID" TABLE="TitleAuthor"
/>
          <SOURCE NAME="au_t" TABLE="TitleAuthor" />
          <SOURCE NAME="au_t_c" TABLE="TitleCategory" JOIN="inner"
PARENT="au_t">
            <KEY COLUMN="ProductID" PARENT-TABLE="TitleAuthor" PARENT-
COLUMN="ProductID"/>
          </SOURCE>
          <SOURCE NAME="au_t_ca" TABLE="Categories" JOIN="inner"
PARENT="au_t_c" >
            <KEY COLUMN="CategoryID" PARENT-TABLE="TitleCategory"
PARENT-COLUMN="CategoryID" />
          </SOURCE>
        </QUERY>
      </CONDITION>
      <CONDITION MORE="and">
        <EXPRESSION DISPLAY="" COLUMN="AuthorID" TABLE="Authors" />
        <OPERATOR NAME="Operator2" SELECTED="5">
            <OPTION SEQNUM="5" DISPLAY="does not write in" SQLOP="not
in"  />
            <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
            <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not null"
```

```
/>
            <OPTION SEQNUM="3" DISPLAY="is not defined" SQLOP="is null"
/>
            <OPTION SEQNUM="4" DISPLAY="writes in" SQLOP="in" />
        </OPERATOR>
        <QUERY ID="12-2" DESCRIPTION="any category whose ..."
SUBQUERY="true">
          <OUTPUT NAME="INTERNAL" COLUMN="AuthorID" TABLE="TitleAuthor"
/>
          <SOURCE NAME="au_t" TABLE="TitleAuthor" />
          <SOURCE NAME="au_t_c" TABLE="TitleCategory" JOIN="inner"
PARENT="au_t">
            <KEY COLUMN="ProductID" PARENT-TABLE="TitleAuthor" PARENT-
COLUMN="ProductID" />
          </SOURCE>
          <SOURCE NAME="au_t_ca" TABLE="Categories" JOIN="inner"
PARENT="au_t_c">
            <KEY COLUMN="CategoryID" PARENT-TABLE="TitleCategory"
PARENT-COLUMN="CategoryID" />
          </SOURCE>
          <CONDITION>
            <EXPRESSION DISPLAY="categoryName" COLUMN="DisplayName"
TABLE="Categories"/>
            <OPERATOR NAME="Operator3" SELECTED="5">
                <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="5"/>
                <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
                <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not
null"   />
                <OPTION SEQNUM="3" DISPLAY="is not defined" SQLOP="is
null"   />
                <OPTION SEQNUM="4" DISPLAY="is equal to" SQLOP="=" />
                <OPTION SEQNUM="6" DISPLAY="contains" SQLOP="like" />
                <OPTION SEQNUM="7" DISPLAY="does not contain" SQLOP="not
like" />
            </OPERATOR>
            <INPUT NAME="text1" SIZE="30" VALUE="Thrillers"
TYPE="string"/>
          </CONDITION>
        </QUERY>
      </CONDITION>
</QUERY>
```

Query13.xml

```
<?xml-stylesheet type="text/xsl" href="WishQuery.xsl"?>
<QUERY ID="13" DESCRIPTION="Return average price of books in each
category sorted by average price.">
      <OUTPUT NAME="CategoryName" COLUMN="DisplayName"/>
      <OUTPUT NAME="AvgPrice" COLUMN="AVG(ListPrice)" />
      <SOURCE NAME="categories" TABLE="Categories" />
      <SOURCE NAME="ca_t" TABLE="TitleCategory" JOIN="inner"
PARENT="categories">
          <KEY COLUMN="CategoryID" PARENT-TABLE="Categories" PARENT-
```

```
COLUMN="CategoryID"/>
      </SOURCE>
      <SOURCE NAME="ca_ti" TABLE="Titles" JOIN="inner" PARENT="ca_t" >
          <KEY COLUMN="ProductID" PARENT-TABLE="TitleCategory" PARENT-
COLUMN="ProductID"/>
      </SOURCE>
      <GROUP NAME="CategoryName" COLUMN="DisplayName"/>
      <HAVING>
        <EXPRESSION DISPLAY="average price" COLUMN="Avg(ListPrice)" />
        <OPERATOR NAME="Op1" SELECTED="2">
                  <OPTION SEQNUM="1" DISPLAY="is any" SQLOP="none"   />
                  <OPTION SEQNUM="2" DISPLAY="is defined" SQLOP="is not
null"   />
                  <OPTION SEQNUM="3" DISPLAY="is undefined" SQLOP="is
null"   />
                  <OPTION DISPLAY="is lower than" SQLOP="&lt;"
SEQNUM="4" />
                  <OPTION DISPLAY="is no lower than" SQLOP="&gt;="
SEQNUM="5" />
                  <OPTION SEQNUM="6" DISPLAY="is equal to" SQLOP="=" />
                  <OPTION DISPLAY="is not equal to" SQLOP="&lt;&gt;"
SEQNUM="7" />
                  <OPTION DISPLAY="is higher than" SQLOP="&gt;"
SEQNUM="8"/>
                  <OPTION DISPLAY="is no higher than" SQLOP="&lt;="
SEQNUM="9" />
        </OPERATOR>
        <INPUT NAME="Input1" TYPE="float" VALUE="0"/>
      </HAVING>
      <SORT DISPLAY="average price" COLUMN="AVG(ListPrice)"/>
</QUERY>
```