Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

CHAPTER 26

XML Support

Abstract

This chapter begins with an overview of XML. It then discusses how SQL provides XML support, including functions for creating XML documents from stored database data, and using the XML data type.

Keywords

XML
SQL
SQL XML extensions
XML schemas

Extensible Markup Language (XML) is a way of representing data and data relationships in text files. Because such UNICODE[1] text files have become widespread and are usually platform independent, XML is being used heavily for data transfer, especially between databases and the Web.

As you will see, XML shares some characteristics with HTML. However, the two markup languages have very different goals. HTML is designed to give instructions to a browser about how to display a page. XML, in contrast, is designed to describe the structure of data and to facilitate moving that data from one place to another.

XML has been accepted by the Word Wide Web Consortium (W3C), and has become a de facto standard for cross-platform data transfers. Because it is an open source specification, a number of application programming languages have been built on top of it, such as XHTML, RSS, MathML, GraphML, Scalable Vector Graphics, and MusicXML.

Relational databases can use XML in three ways:

- Import XML documents and store the data in one or more relations: At this time, such functionality is not provided within the SQL standard. In practice, it requires program code that is specific to a particular DBMS and therefore is beyond the scope of this book.
- Format data stored in relations as XML for output: SQL/XML, which first appeared in the SQL:2003 standard, provides a group of functions for performing this output. The standard also includes detailed instructions for mapping SQL elements to XML.
- Store entire XML documents (or portions of documents) in a column of type XML. SQL provides this capability with the XML data type.

This chapter covers the basics of the structure of XML documents. It then looks at the major SQL/XML functions, and the XML data type.

XML Basics

XML is a *markup language*. In other words, you place special coding in the text to identify data elements. It shares some contents with HTML, but at the same time is both more and less flexible. (As you will see, this is not a contradiction!)

Note: The following is not intended to be a complete primer on XML. It will, however, give you more than enough background to understand what SQL/XML can and cannot do.

XML Structure

XML data are organized in hierarchies. As an example, consider the hierarchies diagrammed in Figure 26.1, both of which are taken from the rare book store database. Notice that it takes two hierarchies to represent the entire database because a hierarchy does not allow a child element—an element at the "many" end of a 1:M relationship—to have two parent elements (elements at the "one" end of a 1:M relationship). Therefore, we can represent the relationship between a customer, a sale, and a volume, *or* the relationship between author, work, book, and volume, but not both within the same hierarchy.

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

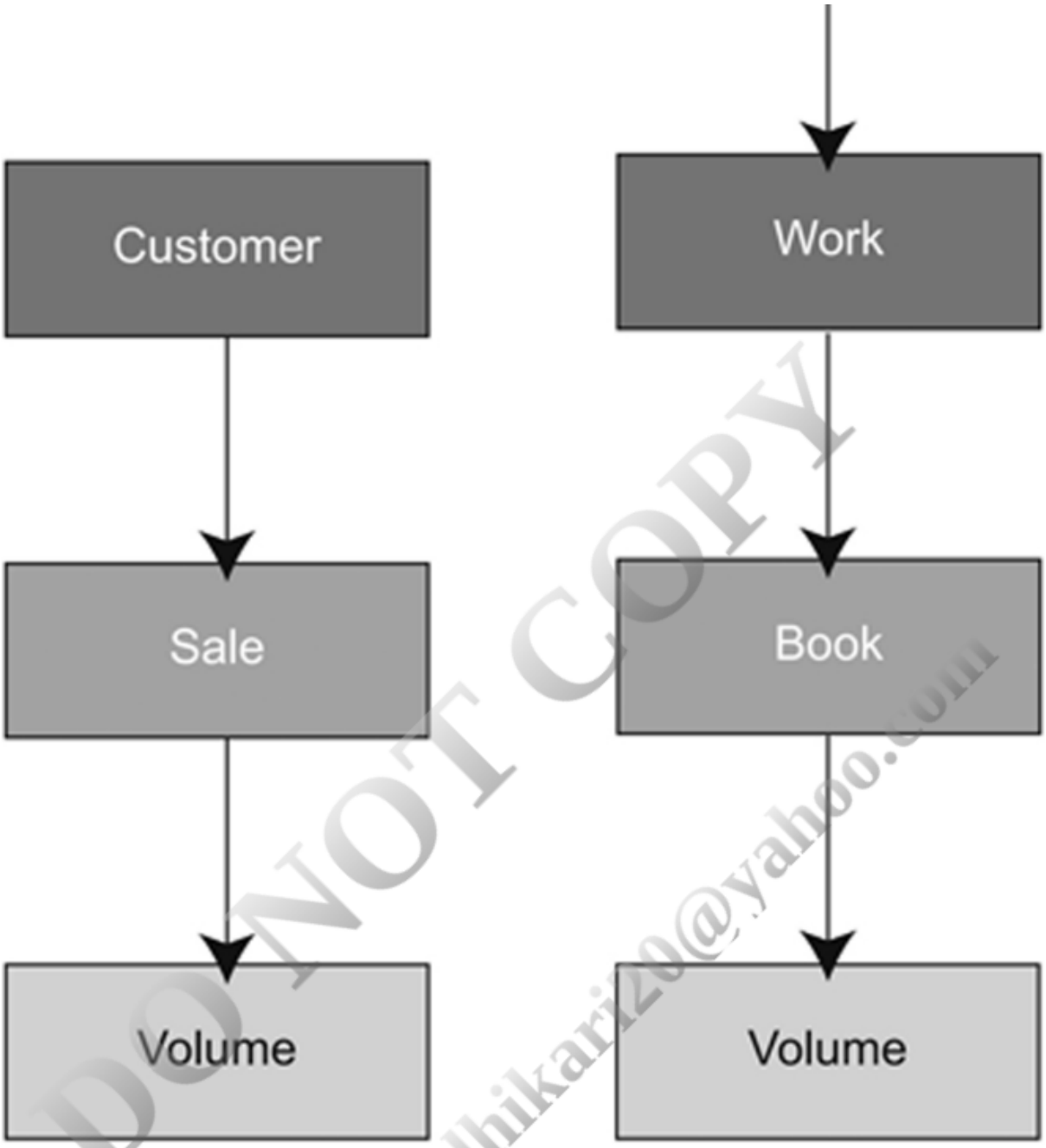
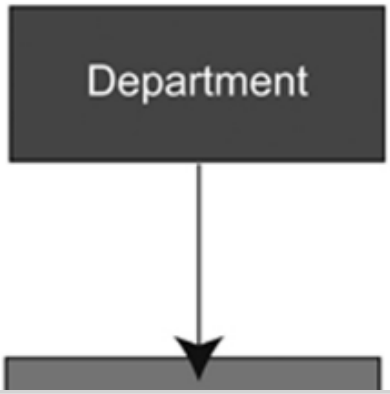


FIGURE 26.1 **Hierarchies for the rare book store database for use in an XML document.**

Parent elements may have more than one type of child entity. As an example, consider the hierarchy in Figure 26.2. Each department has many salespeople working for it. A salesperson has many contacts (potential customers) but also makes many sales. Each sale is for one or more customers, and contains one or more items. This entire hierarchy can be represented as a whole in an XML document.



Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

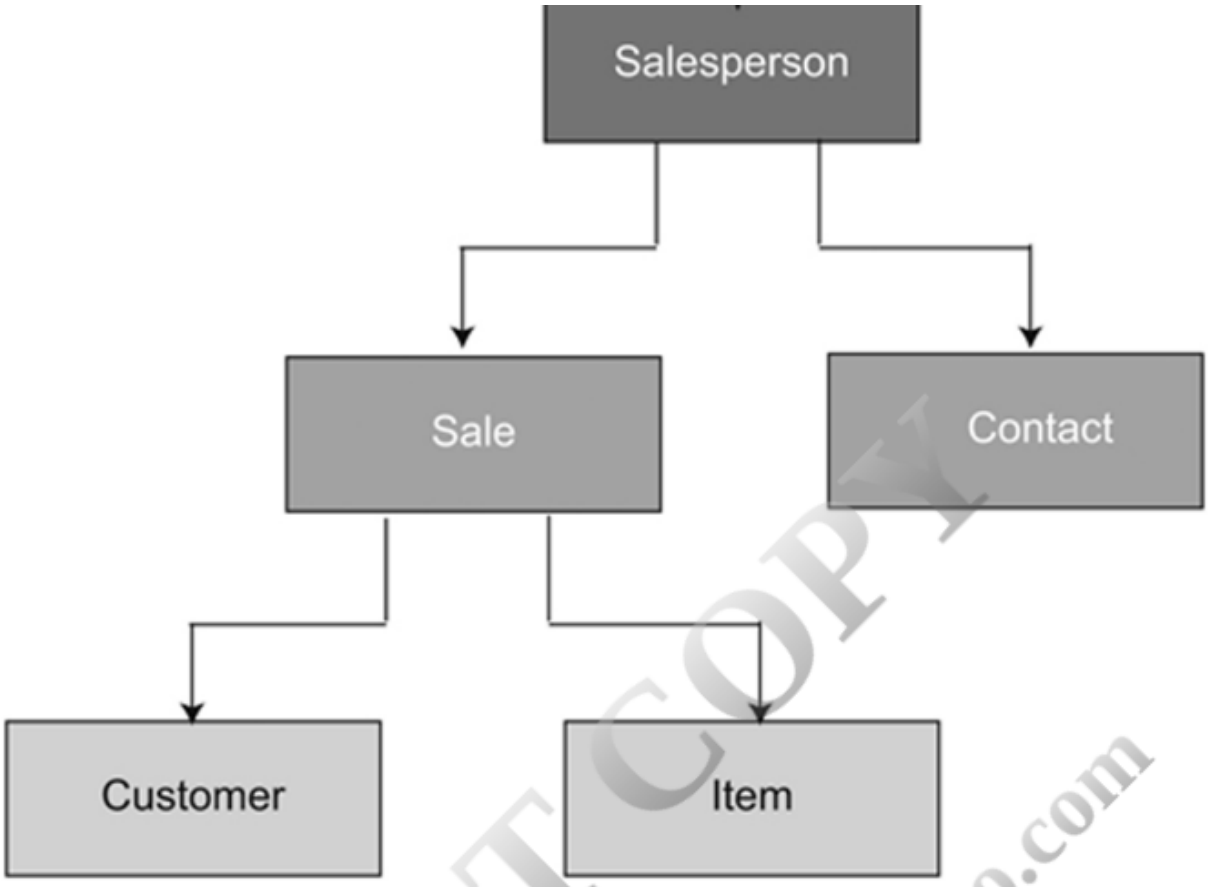


FIGURE 26.2 **A hierarchy that includes parent elements with multiple child element types.**

Because of their hierarchical structure, XML documents are said to be organized in tree structures. (You will discover later in this chapter that this botanical terminology turns up in other places in XML.)

Each hierarchy has a *root* element, the top element in the hierarchy. In Figure 26.1, there are two root elements—Author and Customer; Figure 26.2 has only one root element, Department.

Attributes

As well as having child elements, an element may have "attributes." An attribute is a single data value that describes an element. For example, in Figure 26.2, a department might have its name and the name of its manager as attributes. In contrast, salespeople and the data that describe them are represented as child elements.

Attributes do have some limitations. First, they are single-valued; elements can be multivalued. Because they are a single value attached to an element, attributes can't be related to other attributes or elements. Attributes also make the structure of the XML more difficult to modify should the underlying structure of the data change.

XML Document Structure

An XML document really has a very simple structure: It is made up of nested pairs of tags. With the exception of the first line in the file, that's all there is to it. (If this were an e-mail rather than a book, a smiley face would go here.)

Tags

Elements and their data are identified by *tags* in the text file. These tags are similar to HMTL tags, but don't come from a fixed set of tags like those available for HTML. Each element begins with an opening tag—

```
<element_name>
```

—and ends with a closing tab:

```
</element_name>
```

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Unlike HTML, which has some tags that don't come in pairs (for example,

-), all XML tags must be paired. Between its tags, an element has either a data value or other elements. Nested elements may, in turn, have data values or other elements.

As mentioned earlier, XML elements may have attributes, as well as elements, that are nested beneath them. An attribute is a single occurrence of a value that describes an element. Their values are placed within the element's opening tag:

```
<element_name attribute_name = "attribute_value" …>
```

Attribute values must be quoted.

If you have a pair of tags that have attributes but no nested elements (an *empty tag*), then you can get away with combining the two tags into one, placing the closing symbol (/) just before the closing symbol (>):

```
<element_name element_data/>
```

You can find a portion of an XML document that describes some of the data for the hierarchy in Figure 26.2 in Figure 26.3.

```
<?xml version "1.0" encoding="ISO-8859-1"?>
<department dept_name = "Hannover" manager_last_name = "Benson" manager_first_name
= "James">
      <salesperson>
      <first>John</first>
      <last>Doe</last>
      <ext>1234</ext>
         <sale>
            <sale_date>1-12-2012<sale_date>
            <sale_total>152</sale_total>
            <sale_numb>65</sale_numb>
            <customer>
               <cust_numb>12</cust_numb>
            </customer>
            <item>
               <item_numb>109</item_numb>
               <quantity>1</quantity>
               <sale_numb>65</sale_numb>
            </item>
            <item>
               <item_numb>85</item_numb>
               <quantity>1</quantity>
               <sale_numb>65</sale_numb>
            </item>
         </sale>
         <contact>
            <c_first>Mary</c_last>
            <c_last>Jones</c_last>
            <c_phone>555-525-1111</c_phone>
         </contact>
         <contact>
            <c_first>Sam</c_first>
            <c_last>Smith</c_last>
            <c_phone>555-999-1212</ c_phone >
         </contact>
      </salesperson>
</department>
```

FIGURE 26.3 **A portion of an XML document.**

DO NOT COPY
rituadhikari20@yahoo.com

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

Note: The indentation is not required and is included to make the code easier to read here. In fact, the indentation and spacing in some of the longer listings in this chapter have been changed in the interest of space preservation. Nonetheless, the structure of the documents has been maintained through the proper use of nested tags.

Declarations (Prologs)

Each XML document begins with a declaration (or prolog) that identifies the version of XML being used and optionally a character encoding scheme. The particular encoding specified in Figure 26.3 identifies a Western European (and thus also North American) scheme.

The declaration also optionally includes an attribute named STANDALONE. When its value is "yes," the document has no external references needed to understand its content. If the value is "no," then an application processing the XML document will need to consult outside resources to get complete information for understanding the document. Although some of the examples of declarations that you will see include this attribute, in practice it is not widely used.

The declaration is followed by the opening tag for the root element; the last line in the document is the closing tag for the root element. In between the root element tags, the document contains the elements describing the remainder of the data.

Being "Well-Formed"

As you read about XML and XML documents, especially in the context of interactions with databases, you will find references to *well-formed XML documents*. An XML document is well-formed if it meets all the XML syntax rules. Specifically, that means that:

- It has one and only one root element.
- Every element has a closing tag, or is an empty element.
- Tags are case sensitive.
- All elements are nested properly.
- All attribute values are quoted.

A well-formed XML document is much more likely to be acceptable by many different types of software running on many different platforms.

XML Schemas

To help validate the structure of an XML document, you can create an *XML schema*, a special type of XML document that contains definitions of document structure. An XML schema also defines data types for elements and attributes. It can also contain a number of domain constraints.

Note: XML schemas are a relatively recent addition to XML. The older description of the structure of an XML file is a DTD (Document Type Definition).

Unlike XML documents, XML schemas use pre-defined element tags and element attributes. An example can be found in Figure 26.4, which contains a schema that describes the structure of the XML in Figure 26.3.

```
<?xml version = "1.0"?>
<xs:schema>
   <xs:complexType name = "department">
      <xs:sequence>
         <xs:element name = "dept_name" type = "xs:string"/>
         <xs:element name = "manager_first_name" type = "xs:string"/>
      </xs:sequence>
      <xs:complexType name = "salesperson">
         <xs:sequence>
            <xs:element name = "first" type = "xs:string"/>
            <xs:element name = "last" type = "xs:string"/>
            <xs:element name = "ext" type = "xs:string"/>
         </xs:sequence>
         <xs:complexType name = "sale" mixed = "true">
            <xs:all>
               <xs:element name = "sale_date" type = "xs:date"/>
               <xs:element name = "sale_total" type = "xs:decimal"/>
               <xs:element name = "sale_numb" type = "xs:string"/>
            </xs:all name = "item" mixed = "true">
               <xs:sequence>
                  <xs:element name = "item_numb" type = "xs:integer"/>
                  <xs:element name = "quantity"  type = "xs:integer"/>
                  <xs:element name = "sale_numb" type = "xs:string"/>
               </xs:sequence>
            </xs:complexType>
            <xs:complexType name = "customer" mixed = "true">
               <xs:element name = "cust_numb" type = "xs:integer"/>
```

DO NOT COPY
rituadhikari20@yahoo.com

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
                <xs:element name = "cust_numb" type = "xs:integer"/>
            </xs:complexType>
        </xs:complexType>
    <xs:complexType name = "contact">
        <xs:sequence>
            <xs:element name = "c_first" type = "xs:string"/>
            <xs:element name = "c_last" type = "xs:string"/>
            <xs:element name = "c_phone" type = "xs:string"/>
        </xs:sequence>
    </xs:complexType>
  </xs:complexType>
</xs:schema>
```

FIGURE 26.4 **An XML schema that describes the XML document in Figure 17.3.**

This schema contains the following elements:

- *xs:schema*: the root element.
- *xs:element*: a declaration of a root element. Each element has a name and a data type. XML includes string, integer, decimal, Boolean, date, and time types.
- *xs:complexType*: a group of elements that contain multiple data types.
- *xs:sequence* (an *indicator):* a group of elements that must appear in the order specified within the pair of sequence tags.
- *xs:all* (an indicator): a group of elements that can appear in any order specified within the pair of sequence tags. Each element can appear only once within the group.

SQL's ability to use XML schemas for validation is strongly implementation-dependent. You may find that you will have to use standalone software to determine whether an XML document is well-formed.

Note: There are a number of free Web sites that will check your XML for you, including http://www.w3schools.com/xml/xml_validator.asp, http://www.validome.org/xml/, and http://xmlgrid.net/validator.html.

SQL/XML

SQL/XML, the extensions that provide XML support from within the SQL language, entered the standard in 2003. This section covers those that are used frequently.

Note: XML standards now include a language called XQuery. It is separate from SQL, and appeals primarily to XML programmers who need to interact with data stored in a relational database. In contrast, SQL/XML tends to appeal to SQL programmers who need to generate XML documents.

XML Publishing Functions

SQL/XML's publishing functions extract data from tables and produce XML. You can therefore use a series of the functions—most usually within an embedded SQL application—to generate an entire XML document.

XMLCOMMENT

XML documents can include any comments needed. The XMLCOMMENT function has the general syntax:

```
XMLCOMMENT (source_string);
```

The source string can be a literal string or a string stored in a host language variable. The SQL statement

```
SELECT XMLCOMMENT ('High priced volumes');
```

produces the output

```
xmlcomment
-----------------------------
```

DO NOT COPY
rituadhikari20@yahoo.com

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
<!--High priced volumes-->
```

Notice that the output string is now surrounded by the XML comment indicators.

XMLPARSE

The XMLPARSE function turns a string of text into XML. It is particularly useful when you need to store a string of text in a column of type XML. The function has the general format:

```
XMLPARSE (type_indicator content_string whitespace_option)
```

The type indicator takes one of two values: DOCUMENT (for a complete XML document) or CONTENT (for a small chunk of XML that doesn't represent an entire document). Most of the time, the DOCUMENT type is used in an embedded SQL program where an entire XML document can be stored in a host language string variable.

The content string cannot be generated with a SELECT within the XMLPARSE statement. The content string therefore must be either a literal or a host language variable. There is no reason, however, that the host language variable can't be loaded with a long string created by embedded SELECTs and/or other host language string manipulations.

The whitespace option tells SQL what to do with any empty space at the end of a string. Its value can be either STRIP WHITESPACE or PRESERVE WHITESPACE.

As a simple example, consider the following:

```
SELECT XMLPARSE
    (CONTENT 'Converting a literal to XML' STRIP WHITESPACE);
```

The output looks just like regular text because the interactive SQL command processor strips off the XML tags before displaying the result:

```
xmlparse
--------------------------------
Converting a literal to XML
```

XMLROOT

The XMLROOT function is a bit of a misnomer. Its definition says that it modifies an XML root node, but what it really does is create or modify an XML declaration at the very beginning of an XML document. The function has the following general syntax:

```
XMLROOT (XML_value, VERSION version, STANDALONE standalone_property)
```

The first parameter—*xml_value*—is the XML data for which you will be creating or modifying the prolog. You can then set its version (usually supplied as a literal string) and its standalone property.

The standalone property indicates whether all declarations needed for the XML document are contained within the document or whether declarations are contained in an external document. In most cases when you are using XML with a relational database, the standalone property will be set to *yes*.

The following example adds the prolog at the beginning of the XML, sets the version to 1.1, and the standalone property to *yes*—

```
SELECT XMLROOT (XMLPARSE (CONTENT '<content>abc</content>'),
      VERSION '1.1', STANDALONE YES);
```

—and produces the output

DO NOT COPY
rituadhikari20@yahoo.com

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
                    xmlroot
-----------------------------------------------------------
<?xml version="1.1" standalone="yes"?><content>abc</content>
```

XMLELEMENT

XMLELEMENT is one of two ways to generate content for an XML document. It has the following general syntax:

```
XMLELEMENT (NAME name_of_element,
     XMLATTRIBUTES (attribute_value AS attribute_name, …), content…)
```

Each element has a name that becomes the element's tag. Attributes are optional. The content may be

- A literal value (in quotes)
- Data from a database table
- Another XML element, typically generated with an embedded call to XMLELEMENT or XMLFOREST

As an example, let's create some XML that contains data about books (author, title, and ISBN):

DO NOT COPY
rituadhikari20@yahoo.com

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
SELECT XMLELEMENT (NAME "Books",
     XMLELEMENT (NAME "Author", author.author_last_first),
     XMLELEMENT (NAME "Title",work.title),
     XMLELEMENT (name "ISBN",book.isbn))
FROM author JOIN book JOIN work;
```

The function call is placed in a SELECT statement as one of the values that SELECT should return. The FROM clause identifies the tables from which data will be drawn. This particular example creates an XML element named *Books*. The contents of the element include three other elements—*Author*, *Title*, *ISBN*—each of which is created with an embedded call to XMLELEMENT. The SQL statement produces one element for each row retrieved by the query. In this case, the SELECT has no WHERE clause and therefore generates an element for every row in the joined table created by the FROM clause.

The output of this command can be found in Figure 26.5. Note that the spacing of the output has been adjusted by putting multiple tags on the same line so the output will take up less space. This isn't a problem because XML is text only; any spacing between tags is purely cosmetic.

```
<Books>     <Author>Bronte, Charlotte</Author>
     <Title>Jane Eyre</Title>
     <ISBN>978-1-11111-111-1</ISBN>    </Books>
<Books>     <Author>Bronte, Charlotte</Author>
     <Title>Jane Eyre</Title>
    <ISBN>978-1-11111-112-1</ISBN>    </Books>
<Books>     <Author>Bronte, Charlotte</Author>
     <Title>Villette</Title>
     <ISBN>978-1-11111-113-1</ISBN>     </Books>
<Books>     <Author>Doyle, Sir Arthur Conan</Author>
     <Title>Hound of the Baskervilles</Title>
     <ISBN>978-1-11111-114-1</ISBN>     </Books>
<Books>     <Author>Doyle, Sir Arthur Conan</Author>
     <Title>Hound of the Baskervilles</Title>
     <ISBN>978-1-11111-115-1</ISBN>     </Books>
<Books>     <Author>Doyle, Sir Arthur Conan</Author>
     <Title>Lost World, The</Title>
     <ISBN>978-1-11111-116-1</ISBN>     </Books>
<Books>     <Author>Doyle, Sir Arthur Conan</Author>
     <Title>Complete Sherlock Holmes</Title>
     <ISBN>978-1-11111-117-1</ISBN>     </Books>
<Books>     <Author>Doyle, Sir Arthur Conan</Author>
     <Title>Complete Sherlock Holmes</Title>
     <ISBN>978-1-11111-118-1</ISBN>     </Books>
<Books>     <Author>Twain, Mark</Author>
     <Title>Tom Sawyer</Title>
     <ISBN>978-1-11111-120-1</ISBN>     </Books>
<Books>     <Author>Twain, Mark</Author>
     <Title>Connecticut Yankee in King Arthur's Court, A</Title>
     <ISBN>978-1-11111-119-1</ISBN>     </Books>
```

```
<Books>     <Author>Twain, Mark</Author>
     <Title>Tom Sawyer</Title>
     <ISBN>978-1-11111-121-1</ISBN>     </Books>
<Books>     <Author>Twain, Mark</Author>
     <Title>Adventures of Huckleberry Finn, The</Title>
     <ISBN>978-1-11111-122-1</ISBN>     </Books>
<Books>     <Author>Ludlum, Robert</Author>
     <Title>Matarese Circle, The</Title>
```

DO NOT COPY
rituadhikari20@yahoo.com

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
     <Title>Matarese Circle, The</Title>
     <ISBN>978-1-11111-123-1</ISBN>     </Books>
<Books>     <Author>Ludlum, Robert</Author>
     <Title>Bourne Supremacy, The</Title>
     <ISBN>978-1-11111-124-1</ISBN>     </Books>
<Books>     <Author>Rand, Ayn</Author>
     <Title>Fountainhead, The</Title>
     <ISBN>978-1-11111-125-1</ISBN>     </Books>
<Books>     <Author>Rand, Ayn</Author>
     <Title>Fountainhead, The</Title>
     <ISBN>978-1-11111-126-1</ISBN>     </Books>
<Books>     <Author>Rand, Ayn</Author>
     <Title>Atlas Shrugged</Title>
     <ISBN>978-1-11111-127-1</ISBN>     </Books>
<Books>     <Author>Stevenson, Robert Louis</Author>
     <Title>Kidnapped</Title>
     <ISBN>978-1-11111-128-1</ISBN>     </Books>
<Books>     <Author>Stevenson, Robert Louis</Author>
     <Title>Kidnapped</Title>
     <ISBN>978-1-11111-129-1</ISBN>     </Books>
<Books>     <Author>Stevenson, Robert Louis</Author>
     <Title>Treasure Island</Title>
     <ISBN>978-1-11111-130-1</ISBN>     </Books>
<Books>     <Author>Barth, John</Author>
     <Title>Sot Weed Factor, The</Title>
     <ISBN>978-1-11111-131-1</ISBN>     </Books>
<Books>     <Author>Barth, John</Author>
     <Title>Lost in the Funhouse</Title>
     <ISBN>978-1-11111-132-1</ISBN>     </Books>
<Books>     <Author>Barth, John</Author>
     <Title>Giles Goat Boy</Title>
     <ISBN>978-1-11111-133-1</ISBN>     </Books>
<Books>     <Author>Herbert, Frank</Author>
     <Title>Dune</Title>
     <ISBN>978-1-11111-134-1</ISBN>     </Books>
<Books>     <Author>Asimov, Isaac</Author>
     <Title>Foundation</Title>
     <ISBN>978-1-11111-135-1</ISBN>     </Books>
<Books>     <Author>Asimov, Isaac</Author>
     <Title>Foundation</Title>
     <ISBN>978-1-11111-136-1</ISBN>     </Books>
<Books>     <Author>Asimov, Isaac</Author>
     <Title>Last Foundation</Title>
     <ISBN>978-1-11111-137-1</ISBN>     </Books>
```

```
<Books>     <Author>Asimov, Isaac</Author>
     <Title>Foundation</Title>
     <ISBN>978-1-11111-138-1</ISBN>     </Books>
<Books>     <Author>Asimov, Isaac</Author>
     <Title>I, Robot</Title>
     <ISBN>978-1-11111-139-1</ISBN>     </Books>
<Books>     <Author>Funke, Cornelia</Author>
     <Title>Inkheart</Title>
     <ISBN>978-1-11111-140-1</ISBN>     </Books>
<Books>     <Author>Funke, Cornelia</Author>
```

DO NOT COPY
rituadhikari20@yahoo.com

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
     <Title>Inkdeath</Title>
     <ISBN>978-1-11111-141-1</ISBN>     </Books>
<Books>     <Author>Stephenson, Neal</Author>
     <Title>Anathem</Title>
     <ISBN>978-1-11111-142-1</ISBN>     </Books>
<Books>     <Author>Stephenson, Neal</Author>
     <Title>Snow Crash</Title>
     <ISBN>978-1-11111-143-1</ISBN>     </Books>
<Books>     <Author>Rand, Ayn</Author>
     <Title>Anthem</Title>
     <ISBN>978-1-11111-144-1</ISBN>     </Books>
<Books>     <Author>Rand, Ayn</Author>
     <Title>Anthem</Title>
     <ISBN>978-1-11111-145-1</ISBN>     </Books>
<Books>     <Author>Stephenson, Neal</Author>
     <Title>Cryptonomicon</Title>
     <ISBN>978-1-11111-146-1</ISBN>     </Books>
```

FIGURE 26.5 **An XML fragment created with calls to XMLELEMENT.**

XMLFOREST

The XMLFOREST function can be used to create elements that are part of a higher-level element. Its results are very similar to XMLELEMENT, although its syntax can be simpler than using multiple embedded XMLELEMENT calls. However, the result of XMLFOREST alone is not a valid XML document. We therefore often wrap a call to XMLELEMENT around XMLFOREST.

By itself, XMLFOREST has the following general syntax:

```
XMLFOREST (content AS element_name, ...)
```

As an example, assume that we want to create an XML element for an inventory item, including the ISBN, asking price, and selling price. One way to code the element would be as follows:

```
SELECT XMLELEMENT (NAME "Inventory_item",
     XMLFOREST (volume.isbn, volume.asking_price,
          volume.selling_price)) "Volumes"
FROM volume
WHERE selling_price > 75;
```

Notice that the external function call is to XMLELEMENT to create the element named *inventory_item*. The content of the element is produced by a single call to XMLFOREST, which contains the three data values that are part of the inventory item element. Because there is no AS clause, the function uses the column names as the names of the data elements. You can find the output of the sample query in Figure 26.6.

```
<Inventory_item>
     <isbn>978-1-11111-111-1</isbn>
     <asking_price>175.00</asking_price>
     <selling_price>175.00</selling_price>     </Inventory_item>
<Inventory_item>
     <isbn>978-1-11111-133-1</isbn>
     <asking_price>300.00</asking_price>
<Inventory_item>
     <isbn>978-1-11111-144-1</isbn>
     <asking_price>80.00</asking_price>
```

DO NOT COPY
rituadhikari20@yahoo.com

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
     <asking_price>80.00</asking_price>
     <selling_price>76.10</selling_price>      </Inventory_item>
<Inventory_item>
     <isbn>978-1-11111-121-1</isbn>
     <asking_price>110.00</asking_price>
     <selling_price>110.00</selling_price>     </Inventory_item>
<Inventory_item>
     <isbn>978-1-11111-121-1</isbn>
     <asking_price>110.00</asking_price>
     <selling_price>110.00</selling_price>     </Inventory_item>
<Inventory_item>
     <isbn>978-1-11111-130-1</isbn>
     <asking_price>150.00</asking_price>
     <selling_price>120.00</selling_price>     </Inventory_item>
<Inventory_item>
     <isbn>978-1-11111-126-1</isbn>
     <asking_price>110.00</asking_price>
     <selling_price>110.00</selling_price>     </Inventory_item>
<Inventory_item>
     <isbn>978-1-11111-139-1</isbn>
     <asking_price>200.00</asking_price>
     <selling_price>170.00</selling_price>     </Inventory_item>
<Inventory_item>
     <isbn>978-1-11111-133-1</isbn>
     <asking_price>125.00</asking_price>
     <selling_price>125.00</selling_price>     </Inventory_item>
<Inventory_item>
     <isbn>978-1-11111-130-1</isbn>
     <asking_price>200.00</asking_price>
     <selling_price>150.00</selling_price>     </Inventory_item>
```

FIGURE 26.6 **The results of using XMLFOREST to generate the contents of an XML attribute.**

XMLATTRIBUTES

As you will remember from earlier in this chapter, an XML element can have attributes, data values that are part of the element tag. The XMLATTRIBUTES function is used to specify those attributes. Like XMLFOREST, it is most commonly used as part of an XMLELEMENT function call.

The function has the following general syntax:

```
XMLATTRIBUTES (value AS attribute_name)
```

If the attribute's value is a column in a database table, then the AS and the attribute name are optional. SQL will then use the column name as the attribute name.

As an example, let's create an XML element for books with selling prices of more than $75, the results of which appear in Figure 26.7. Notice that text values (eg, the ISBN) are in quotes.

```
SELECT XMLELEMENT (NAME "High_Priced",
     XMLATTRIBUTES (volume.isbn AS ISBN),
     XMLELEMENT (NAME "Asking_price", volume.asking_price),
     XMLELEMENT (NAME "Selling_price", volume.selling_price))
FROM volume
WHERE selling_price > 75;
```

DO NOT COPY
rituadhikari20@yahoo.com

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
<High_Priced isbn="978-1-11111-111-1">
     <Asking_price>175.00</Asking_price>
     <Selling_price>175.00</Selling_price>
</High_Priced>
<High_Priced isbn="978-1-11111-133-1">
     <Asking_price>300.00</Asking_price>
     <Selling_price>285.00</Selling_price>
</High_Priced>
<High_Priced isbn="978-1-11111-144-1">
     <Asking_price>80.00</Asking_price>
     <Selling_price>76.10</Selling_price>
</High_Priced>
<High_Priced isbn="978-1-11111-121-1">
     <Asking_price>110.00</Asking_price>
     <Selling_price>110.00</Selling_price>
</High_Priced>
<High_Priced isbn="978-1-11111-121-1">
     <Asking_price>110.00</Asking_price>
     <Selling_price>110.00</Selling_price>
</High_Priced>
```

```
<High_Priced isbn="978-1-11111-130-1">
     <Asking_price>150.00</Asking_price>
     <Selling_price>120.00</Selling_price></High_Priced>
<High_Priced isbn="978-1-11111-126-1">
     <Asking_price>110.00</Asking_price>
     <Selling_price>110.00</Selling_price>
</High_Priced>
<High_Priced isbn="978-1-11111-139-1">
     <Asking_price>200.00</Asking_price>
     <Selling_price>170.00</Selling_price>
</High_Priced>
<High_Priced isbn="978-1-11111-133-1">
     <Asking_price>125.00</Asking_price>
     <Selling_price>125.00</Selling_price>
</High_Priced>
<High_Priced isbn="978-1-11111-130-1">
     <Asking_price>200.00</Asking_price>
     <Selling_price>150.00</Selling_price>
</High_Priced>
```

FIGURE 26.7 **The results of using XMLATTRIBUTES to add attributes to an XML element.**

DO NOT COPY
rituadhikari20@yahoo.com

XMLCONCAT

The XML functions we have been discussing generate fragments of XML documents. To paste them together, you use the XMLCONCAT function. It has a relatively simple general syntax:

```
XMLCONCAT (XML_value, XML_value, …)
```

As an example, let's put a comment in front each of the elements that contain data about books with high selling prices:

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
SELECT XMLCONCAT (XMLCOMMENT ('This is a high-priced book'),
     XMLELEMENT (NAME "High_Priced",
        XMLATTRIBUTES (volume.isbn AS ISBN),
        XMLELEMENT (NAME "Asking_price", volume.asking_price),
        XMLELEMENT (NAME "Selling_price", volume.selling_price)))
FROM volume
WHERE selling_price > 75;
```

The results can be found in Figure 26.8. Line breaks have been added to make the result readable. Note, however, that SQL views each occurrence of the comment and the entire element as a single string of text and therefore inserts a line break only at the end of each element.

```
                                    xmlconcat
----------------------------------------------------------------------
 <!--This is a high-priced book-->
     <High_x0020_Priced isbn="978-1-11111-111-1">
     <Asking_x0020_Price>175.00</Asking_x0020_Price>
     <Selling_x0020_Price>175.00</Selling_x0020_Price>
</High_x0020_Priced>
 <!--This is a high-priced book-->
     <High_x0020_Priced isbn="978-1-11111-133-1">
     <Asking_x0020_Price>300.00</Asking_x0020_Price>
     <Selling_x0020_Price>285.00</Selling_x0020_Price>
</High_x0020_Priced>
 <!--This is a high-priced book-->
     <High_x0020_Priced isbn="978-1-11111-144-1">
     <Asking_x0020_Price>80.00</Asking_x0020_Price>
     <Selling_x0020_Price>76.10</Selling_x0020_Price>
</High_x0020_Priced>
 <!--This is a high-priced book-->
     <High_x0020_Priced isbn="978-1-11111-121-1">
     <Asking_x0020_Price>110.00</Asking_x0020_Price>
     <Selling_x0020_Price>110.00</Selling_x0020_Price>
</High_x0020_Priced>
 <!--This is a high-priced book-->
     <High_x0020_Priced isbn="978-1-11111-121-1">
     <Asking_x0020_Price>110.00</Asking_x0020_Price>
     <Selling_x0020_Price>110.00</Selling_x0020_Price>
</High_x0020_Priced>
 <!--This is a high-priced book-->
     <High_x0020_Priced isbn="978-1-11111-130-1">
     <Asking_x0020_Price>150.00</Asking_x0020_Price>
     <Selling_x0020_Price>120.00</Selling_x0020_Price>
</High_x0020_Priced>
 <!--This is a high-priced book-->
     <High_x0020_Priced isbn="978-1-11111-126-1">
     <Asking_x0020_Price>110.00</Asking_x0020_Price>
     <Selling_x0020_Price>110.00</Selling_x0020_Price>
</High_x0020_Priced>
 <!--This is a high-priced book-->
     <High_x0020_Priced isbn="978-1-11111-139-1">
     <Asking_x0020_Price>200.00</Asking_x0020_Price>
     <Selling_x0020_Price>170.00</Selling_x0020_Price>
</High_x0020_Priced>
```

DO NOT COPY
rituadhikari20@yahoo.com

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
 <!--This is a high-priced book-->
     <High_x0020_Priced isbn="978-1-11111-133-1">
     <Asking_x0020_Price>125.00</Asking_x0020_Price>
     <Selling_x0020_Price>125.00</Selling_x0020_Price>
</High_x0020_Priced>
 <!--This is a high-priced book-->
     <High_x0020_Priced isbn="978-1-11111-130-1">
     <Asking_x0020_Price>200.00</Asking_x0020_Price>
     <Selling_x0020_Price>150.00</Selling_x0020_Price>
</High_x0020_Priced>
```

FIGURE 26.8 **The result of using XMLCONCAT to concatenate XML fragments.**

Note: The x0020 that appears frequently in the output is the ASCII code for a blank.

The XML Data Type

You can declare a column in a table to be of type XML, just as you would with any other data type:

```
CREATE TABLE xmlstuff
       (seq_numb INT,
       xml_text XML,
       PRIMARY KEY (seq_numb));
```

The XML column can then be used to store fragments of XML or entire XML documents. However, doing so has several drawbacks:

- The contents of the column are not searchable with standard SQL commands.[2]
- The contents of the column cannot be used in predicates that require comparison operators such as > or =.
- The column cannot be indexed.

For that reason, tables that have XML columns need at least a unique sequence number to identify each row. You may also want to include a table that assigns keywords to each document, so there is some type of search capability. Such a table might be created with

```
CREATE TABLE keywords
     (seq_numb int,
     keyword char (30),
     PRIMARY KEY (seq_numb, keyword),
     FOREIGN KEY keywords2xmlstuff
        (seq_numb) REFERENCES xmlstuff);
```

As mentioned earlier in the discussion of XMLPARSE, you need to use that function to convert text into XML to store in an XML column. Because you can't generate an input string with a SELECT, the interactive INSERT is limited to XML fragments:

```
INSERT INTO xmlstuff
   VALUES (1, XMLPARSE
      (CONTENT 'This is a test' STRIP WHITE SPACE));
```

DO NOT COPY
rituadhikari20@yahoo.com

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
(CONTENT 'This is a test' STRIP WHITE SPACE);
```

For complete document input, you will generally be working with an embedded SQL application.
If you want to look at the contents of an XML column, you can use an interactive SELECT. SQL strips the XML tags for output. The query

```
SELECT * FROM xmlstuff;
```

produces

```
 sequ_numb |     xml_text
-----------+-----------------
         1 | this is a test
```

Note: You could store XML in a text column, tags and all. However, when you use an XML column, SQL will check the XML to see that it is well-formed.

XMLSERIALIZE

The XMLSERIALIZE function is essentially the opposite of XMLPARSE: it takes the contents of an XML column and converts it to a text string:

```
XMLSERIALIZE (type_indicator column_name AS character_type)
```

For example,

```
SELECT XMLSERIALIZE
     (DOCUMENT xmltext AS VARCHAR (256))
    FROM sql_stuff
    WHERE seq_numb = 16;
```

would extract the document from the row with the sequence number of 16, convert it to plain text (removing the tags), and display it on the screen. Because SQL removes the tags from interactive SELECT output, this function is particularly useful in an embedded SQL program.

For Further Reading

Fawcett J. *Beginning XML*. fifth ed. Wrox; 2012.

Key S. *XML Programming Success in a Day: Beginner's Guide to Fast, Easy and Efficient Learning of XML Programming*. CreativeSpace Independent Publishing Platform; 2015.

Ray ET. *Learning XML*. second ed. O'Reilly Media; 2003.

Walmsley P. *Definitive XML Schema*. second ed. Prentice Hall; 2012.

Walmsley P. *XQUERY: Search Across a Variety of XML Data*. second ed. O'Reilly Media; 2015.

[1] UNICODE is the character encoding scheme that has replaced ASCII in many computing platforms. It includes ASCII, but uses two-byte rather than single-byte codes, allowing it to support characters from non-Roman alphabets.

[2] Some current DBMSs support XQuery, a SQL extension that can be used to search XML data.

DO NOT COPY
rituadhikari20@yahoo.com

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

CHAPTER 27

Object-Relational Databases

Abstract

This chapter introduces readers to the object-oriented paradigm and how the relational data model has been extended to support a hybrid object–relational data model. The chapter covers the basic principles of object-orientation and the nature of pure object-oriented databases. It then discusses the object–relational data model in depth, followed by how SQL provides support for that model. Topics include SQL data types for object-oriented features, user-defined data types, typed tables, and methods that are a part of user-defined data types.

Keywords

object-oriented paradigm
object-oriented databases
object-oriented DBMS
SQL
object-relational data model
SQL object-oriented features
user-defined data types
SQL user-defined data types
UDT
object-oriented inheritance
SQL typed tables

The relational data model has been a mainstay of business data processing for more than 30 years. Nothing has superseded it in the way the relational data model superseded the simple network data model. However, a newer data model—the object-oriented data model—has come into use as an alternative for some types of navigational data processing.

Note: To be completely accurate, the relational data model is the only data model that has a formal specification. The hierarchical data model and the OO data model do not. The closest thing the simple network data model has is the CODASYL specifications.

This chapter begins with an overview of some object-oriented concepts for readers who aren't familiar with the object-oriented paradigm. If you have object-oriented programming experience, then you can skip over the first parts of this chapter and begin reading with the section *The Object-Relational Data Model*.

The object-oriented paradigm was the brainchild of Dr Kristen Nygaard, a Norwegian who was attempting to write a computer program to model the behavior of ships, tides, and fjords. He found that the interactions were extremely complex and realized that it would be easier to write the program if he separated the three types of program elements and let each one model its own behavior against each of the others.

The object-oriented programming languages in use today (most notably C + +, Java, and JavaScript) are a direct outgrowth of Nygaard's early work. The way in which objects are used in databases today is an extension of object-oriented programming.

Note: This is in direct contrast to the relational data model, which was designed specifically to model data relationships, although much of its theoretical foundations are found in mathematical set theory.

Getting Started: Object-Orientation without Computing

To understand the role of objects in relational databases, you must first understand the object-oriented paradigm as it is used in object-oriented programming and pure object-oriented databases. The easiest way to do so is to begin with an example that has absolutely nothing to with programming at all.

Assume that you have a teenage daughter (or sister, whichever is more appropriate) named Jane and that your family is going to take a long car trip. Like many teens, Jane is less than thrilled about a trip with the family and, in particular, with spending so much time with her 12-year-old brother, especially since her parents have declared the trip a holiday from hand-held electronics. In self-defense, Jane needs something to keep her brother busy so he won't bother her as she reads while her parents are driving. She therefore decides to write up some instructions for playing solitaire games for him.

The first set of instructions is for the most common solitaire game, Klondike. As you can see in Figure 27.1, the deal involves seven piles of cards of increasing depth, with the top card turned over. The rest of the deck remains in the draw pile. Jane decides to break the written instructions into two main parts: information about the game and questions her brother might ask. She therefore produces instructions that look something like Figure 27.2. She also attached the illustration of the game's deal.

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

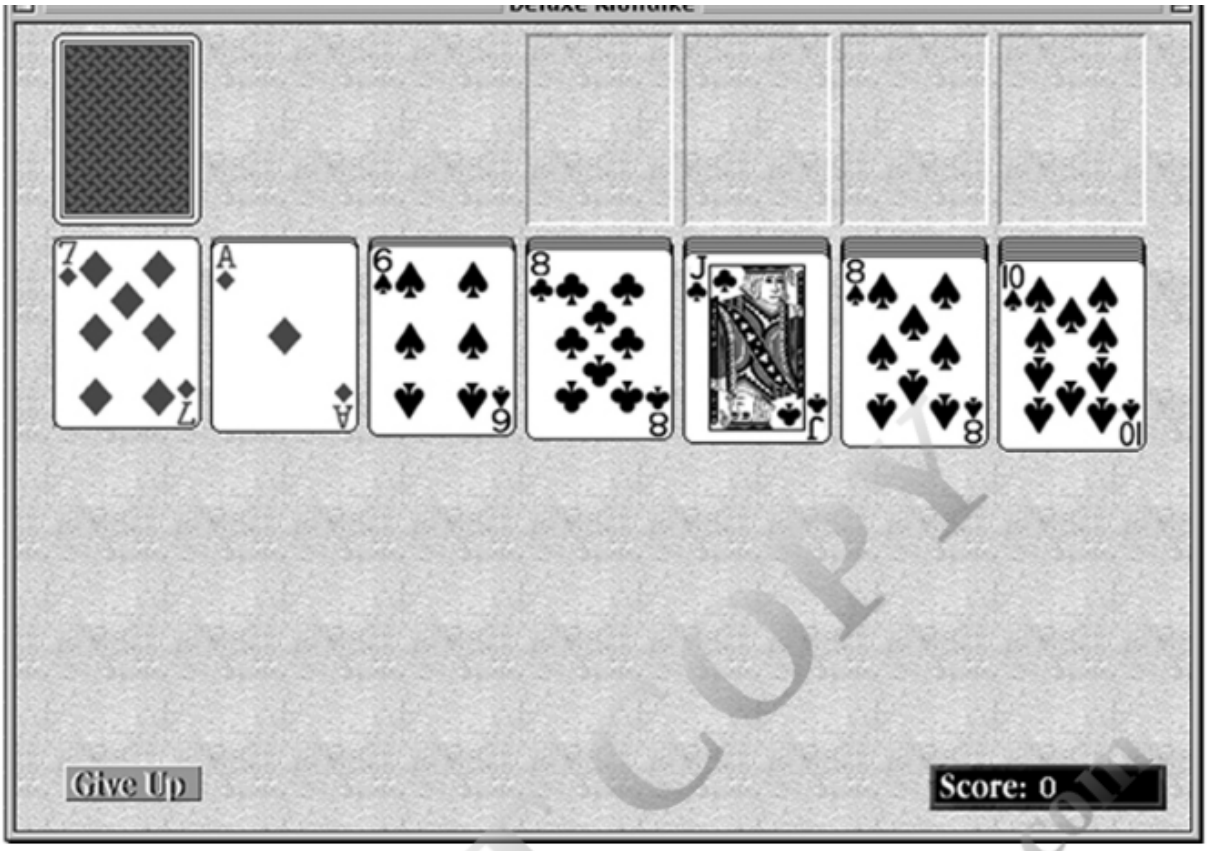


FIGURE 27.1 The starting layout for Klondike.

```
Information about the game
      Name: Klondike
      Illustration: See next page
      Decks: One
      Dealing: Deal from left to right
            First pass: First card face up six cards down.
            Second pass: First card face up on top of pile #2, five
                  cards down on remaining piles.
            Third pass: First card face up on top of pile #3; four
                  cards down on remaining piles.
            …repeat pattern for total of seven passes.
            Place remaining cards in draw pile, face down.
      Playing: One or two cards can be turned from the draw pile
            at a time. As encountered, put aces above layout. Build up
            from aces in suits. Build down on the deal, opposite suit
            colors. Can move from the middle of a stack moving card
            and all cards built below it.
            Move only kings into empty spots on the layout.
            If turning one card, make only one pass through the draw
                  Pile.
            If turning three cards, make as many passes as you like
                  through the draw pile.
      Winning: All cards built on top of their aces.
Questions to Ask
      What is the name of the game?
            Read Name section.
      How many decks do I need?
```

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
How many decks do I need?
      Read Decks section.
What does the layout look like?
      Read Illustration section.
How do I deal the game?
      Read Dealing section.
How do I play the game?
      Read Playing section.
How do I know when I've won?
      Read Winning section.
```

FIGURE 27.2 **Instructions for playing Klondike.**

The next game she tackles is Canfield. Like Klondike, it is played with one deck, but the deal and play are slightly different (see Figure 27.3). Jane uses the same pattern for the instructions as she did for Klondike because it cuts down the amount of writing she has to do (see Figure 27.4).

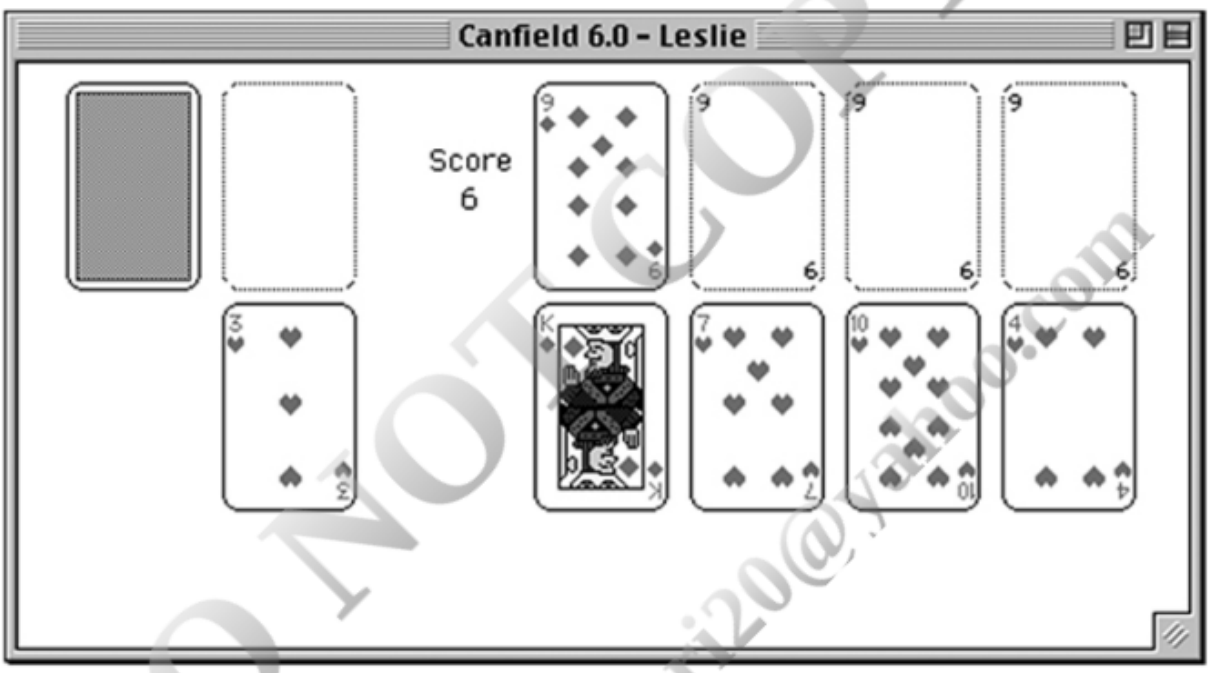


FIGURE 27.3 **The starting deal for Canfield.**

```
Information about the game
      Name: Canfield
      Illustration: See next page
      Decks: One
      Dealing: Deal four cards face up.
            Place one additional card above the first four as the
.                 starting card for building suits.
            The remaining cards stay in the draw pile.
      Playing: Turn one card at a time, going through the deck as many
                  times as desired.
            Build down from deal, opposite suit colors.
            Can move cards from the middle of stack, moving card and
                  all cards built below it.
            Place cards of the same value as the initial foundation
                  card above the deal as encountered.
            Build up in suits from the foundation cards.
            Any card can be placed in any empty space in the deal.
```

Printed by: rituadhikari20@yahoo.com. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.

```
            Any card can be placed in any empty space in the deal.
      Winning: All cards built on top of the foundation cards.
Questions to Ask
      What is the name of the game?
            Read Name section.
      How many decks do I need?
            Read Decks section.
      What does the layout look like.
            Read Illustration section.
      How do I deal the game?
            Read Dealing section.
      How do I play the game?
            Read Playing section.
      How do I know when I've won?
            Read Winning section.
```

FIGURE 27.4 **Instructions for playing Canfield.**

And finally, just to make sure her brother doesn't get too bored, Jane prepares instructions for Forty Thieves (see Figure 27.5). This game uses two decks of cards and plays in a very different way from the other two games (see Figure 27.6). Nonetheless, preparing the instructions for the third game is fairly easy because she has the template for the instructions down pat.

DO NOT COPY
rituadhikari20@yahoo.com