4

The Path to DB2 9 XML Capabilities

In this chapter, you'll learn more about XQuery, XPath, and the XPath Data Model and receive an introduction to DB2 9's XML components and native XML support. The chapter continues to build on XML as the foundation for effective understanding and use of XML in DB2 9 and introduces the DB2 9 XML components and architecture.

XQuery and XPath

Just as SQL provides a query language for relational databases, a native query language for XML was needed due to the synchronization problem between XML and relational data that we discussed in Chapter 3. To address this need, the W3C organization developed XQuery, the query language for XML, defining the language in the XQuery 1.0 specification. As of the writing of this book, the XQuery 2.0 draft is in the works.

XQuery is the main entry point for accessing and processing XML data in DB2 9. It is a domain-specific language designed to work with XML data that is highly variable, unstructured, and unpredictable. XQuery provides the flexibility to work with this kind of data. For example, you can use XML queries to perform the following operations:

- Return results that have mixed types
- Search XML data for objects that are at unknown levels of the hierarchy
- Perform structured transformations on the data (e.g., invert a hierarchy)

Note: DB2 supports storing and retrieving well-formed XML data in a column of a table. XML instances (documents) consisting of XML that conforms to XML's syntax rules are legal and are considered to be well-formed.

XQuery queries use expressions written in the XML Path (XPath) language to navigate through XML trees and extract XML fragments as well as to create, sort, aggregate, combine, and iterate over sequences (examine or manipulate each item in a sequence) and construct new XML data. At the heart of the XPath language is the *path expression*, which provides for the hierarchical addressing of nodes in an XML tree.

The following excerpt from the *XML Path Language (XPath) 2.0* W3C recommendation describes the XPath language in detail.

"XPath 2.0 is an expression language that allows the processing of values conforming to the data model defined in [the] XQuery/XPath Data Model (XDM) [specification]. The data model provides a tree representation of XML documents as well as atomic values such as integers, strings, and booleans, and sequences that may contain both references to nodes in an XML document and atomic values. The result of an XPath expression may be a selection of nodes from the input documents, or an atomic value, or more generally, any sequence allowed by the data model."

Note: XPath is an expression language that allows the processing of values conforming to the data model defined in the XQuery/ XPath Data Model W3C specifications. DB2 9 supports these specifications.

An XQuery 1.0 expression takes one or more XDM instances as input and returns an XDM instance as a result. The current XQuery 1.0 specification does not support updating instances of the XDM. However, DB2 9 provides facilities to accomplish this; you'll learn how to do this in Chapter 5.

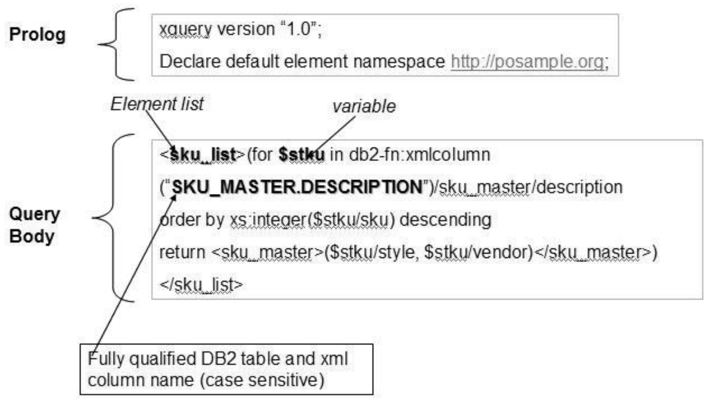
Note: The XQuery 1.0 and XPath 2.0 Data Model does not allow for updating of nodes. As of the writing of this book, extensions to XQuery 1.0 are being considered that would provide for an XQuery update facility.

Like most data models, XDM defines the format of the data but not the application programming interfaces (APIs) to access the data. That is where specific implementations such as DB2 9 come into play. In a nutshell, the XQuery language is the fundamental query language for querying XML data, and it is guided by the XDM for data format rules.

Because XQuery is a reference-based language, subsequent expressions on the result of a path expression may traverse the document in both forward and reverse direction. With XQuery and XPath, you can store, compose, and decompose XML documents.

Creating Queries in XQuery

You form queries in XQuery by making declarations in the form of expressions. Queries consist of an optional *prolog* and a *query body*. Figure 4.1 shows a sample DB2 9 XQuery structure.


Prolog
Element list
variable
Query Body
Fully qualified DB2 table and xml column name (case sensitive)

```
xquery version "1.0";
Declare default element namespace http://posample.org;
```

```
<sku_list>(for $stku in db2-fn:xmlcolumn
("SKU_MASTER.DESCRIPTION")/sku_master/description
order by xs:integer($stku/sku) descending
return <sku_master>($stku/style, $stku/vendor)</sku_master>)
</sku_list>
```

Figure 4.1: Basic XQuery structure

The prolog defines the processing environment for the query. It is followed by the query body, which consists of one or more expressions that define the query results.

Expressions can consist of multiple XQuery expressions that are combined using various XQuery operators or keywords. In the example, the prolog contains two declarations: a version declaration, which specifies the version of the XQuery syntax to apply to the query, and a default namespace declaration, which specifies the namespace Uniform Resource Identifier (URI) to use for unprefixed element and type names.

The sample query body contains an expression that constructs a `sku_list` element. The content of the `sku_list` element is a list of `sku_master` elements. The sample query uses the `for` expression along with the `$stku` variable to iterate through the description subelements of the `sku_master` elements contained in the `SKU_MASTER.DESCRIPTION` column. The `/sku_master/description` path expression iterates through every node in the expression on the left (the *context*) and for each such node performs the selection on the right (the *step*). Last, the `return` expression indicates that for each iteration, the value of `$stku` is returned. The constructed `sku_list` element is a list of `sku_master` elements sorted in descending order.

Note: An XML sequence returned by the `db2-fn:xmlcolumn` or `db2-fn:sqlquery` function can contain any XML values, including atomic values and nodes.

DB2 9 supports queries written in SQL, XQuery, or a combination of SQL and XQuery. Both languages are supported as primary query languages, and each provides functions for invoking the other language.

Before we delve into specific aspects of DB2 9 support for XML, it will be helpful to take a closer look at details of the XPath Data Model.

XPath Data Model

The *XQuery 1.0 and XPath 2.0 Data Model (XDM)* W3C recommendation (23 January 2007) defines the data model used by XQuery 1.0, XPath 2.0, and Extensible Stylesheet Language Transformations (XSLT) 2.0. as follows:

"The XQuery 1.0 and XPath 2.0 Data Model . . . serves two purposes. First, it defines the information contained in the input to an XSLT or XQuery processor. Second, it defines all permissible values of expressions in the XSLT, XQuery, and XPath languages."

XDM provides support for XML Schema types and extends the Infoset model by providing precise type information and supporting the representation of collections of documents and complex values.

Every value in the XDM is the *sequence* of zero or more *items*. An item may be a node or an atomic value. A sequence may contain nodes, atomic values, or any mixture of nodes and atomic values. A *node* is one of the seven kinds of nodes defined by the XPath 2.0 specification:

- *Document nodes* encapsulate XML documents and have the following properties, which can be empty:
  - » base-uri
  - » children
  - » unparsed-entities
  - » document-uri
- *Element nodes* encapsulate elements and have the following properties:
  - » base-uri
  - » node-name
  - » parent (can be empty)
  - » type
  - » children (can be empty)
  - » attributes (can be empty)
  - » namespaces (can be empty)
  - » nilled
- *Attribute nodes* represent XML attributes and have the following properties:
  - » node-name
  - » string value
  - » parent (can be empty)
  - » type
- *Text nodes* contain XML character content and have the following properties:
  - » content
  - » parent
- *Namespace nodes* contain information about XML namespaces:
  - » prefix (can be empty)
  - » uri
  - » parent (can be empty)

- *Processing instruction nodes* contain XML processing instructions and have the following properties:
  - » target
  - » content
  - » base-uri (can be empty)
  - » parent (can be empty)
- *Comment nodes* contain XML comments and have the following properties:
  - » content
  - » parent

Although namespace nodes are part of the XPath 2.0 specification, support for them by applications (in this case, by DB2 9) is application-dependent. In DB2 9, namespace nodes are not supported; instead, IBM has chosen to represent namespaces in an internal format that complies with the specification. However, I've included namespace nodes in this section to give you an overall understanding of node types.

Table 4.1 provides a complete list of the rules defined by the W3C for the seven types of nodes.

Table 4.1: Rules for XPath Data Model node types

Document nodes
1. Every document must have a unique identity, distinct from all other nodes.
2. The children must consist exclusively of element, processing instruction, comment, and text nodes if the property is not empty. Attribute, namespace, and document nodes can never appear as children.
3. The sequence of nodes in the children is ordered and must be in document order.
4. The children property must not contain two consecutive text nodes.
5. If node N is a child of a document node D, the parent of N must be D.
6. If N has a parent document node D, N must be among the children of D.
7. The children property must not contain two nodes with the same identity.

Table 4.1: Rules for XPath Data Model node types (contiuned)
Element nodes
1. Every element node must have a unique identity, distinct from all other nodes.
2. The children must consist exclusively of element, processing instruction, comment, and text nodes if the property is not empty. Attribute, namespace, and document nodes can never appear as children.
3. The sequence of nodes in the children property is ordered and must be in document order.
4. The children property must not contain two consecutive text nodes.
5. The children property must not contain two consecutive nodes with the same identity.
6. The attributes of an element must have distinct xs:QNames.
7. The namespace nodes of an element must have distinct names. At most one of the namespace nodes of an element has no name (this is the default namespace).
8. If node N is a child of element E, the parent of N must be E.
9. Exclusive of attribute and namespace nodes, if a node N has a parent element E, N must be among the children of E. (Attribute and namespace nodes have a parent, but they do not appear among the children of their parent.)
10. If an attribute node A has a parent element E, A must be among the attributes of E.
11. If a namespace node has a parent element E, N must be among the namespaces of E.
Attribute nodes
1. Every attribute node must have a unique identity, distinct from all other nodes.
2. If an attribute node A has a parent element E, A must be among the attributes of E. Attribute nodes are permitted without parents, but such attributes must not appear among the attributes of any element node.
Text nodes
1. A text node must not contain the empty string as its content. Two consecutive text nodes cannot appear as adjacent siblings.
Namespace nodes
1. Every namespace node must have a unique identity, distinct from all other nodes.
2. If a namespace node N has a parent element E, N must be among the namespaces of E. Namespace nodes without parents are permitted in special cases.
Processing instruction nodes
1. Every processing instruction node must have a unique identity, distinct from all other nodes.
2. The target must be an NCName.
Comment nodes
1. Every comment node must have a unique identity, distinct from all other nodes.
2. The string “--” must not occur within the content.

Figure 4.2 shows how you can browse XML documents, elements, and attributes using the DB2 9 Control Center. You can also use the Control Center to drill down to each node type and to view entire documents.

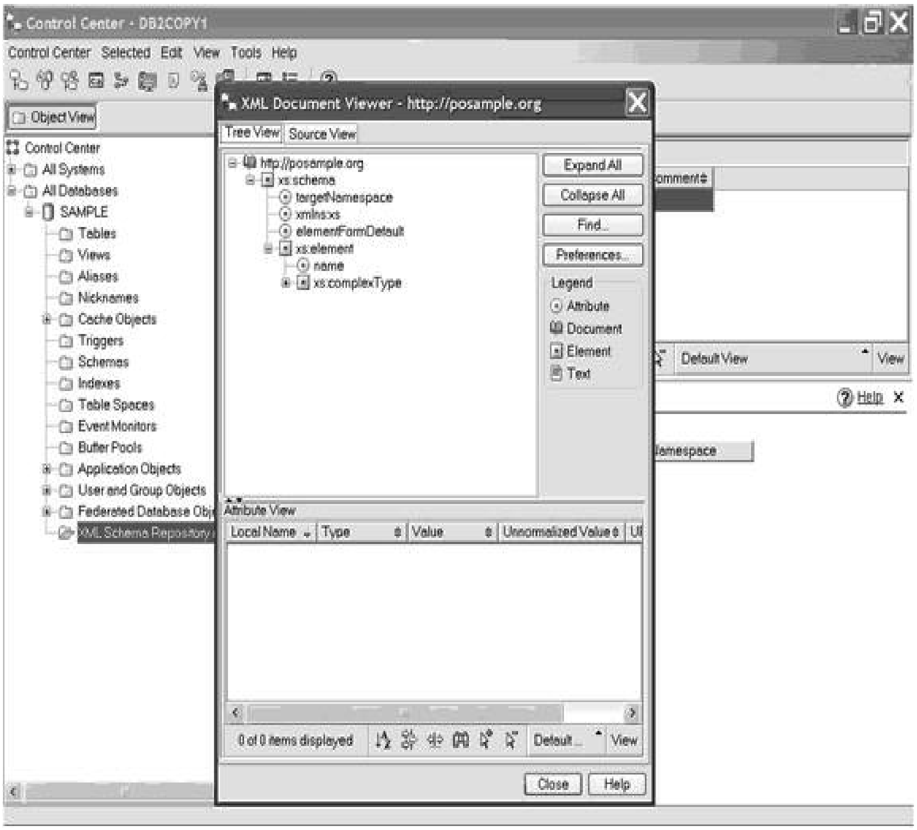


Figure 4.2: Browsing XML documents with the DB2 Control Center

DB2 9 Hybrid Architecture

Key to DB2 9's hybrid architecture is the integration of the native DB2 XML data type into the existing relational engine. This integration enables a company to store both relational and XML data in the DB2 9 hybrid data server. This integration of XML data in the DB2 9 database engine and seamless XML support is packaged as IBM's *pureXML* technology.

Although DB2 9 provides complete XML support, this support is offered as an option and does not come with the base DB2 9 product. This approach is in keeping with today's flexible software configuration and pricing movement. With this flexibility, you can customize your DB2 installations to use just the features you need. This capability can help to reduce annual database licensing costs. So, even though pureXML is tightly integrated into DB2 9, the feature is optional.

Components of DB2 9 XML Support

Figure 4.3 provides an overview of the DB2 9 pureXML architecture.

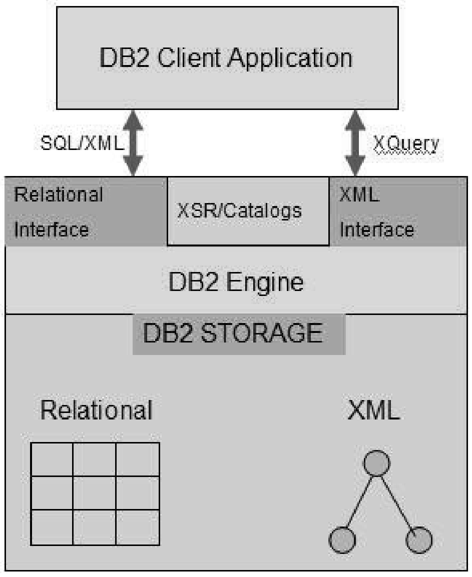


Figure 4.3: DB2 hybrid data server architecture

As the diagram depicts, DB2 9's XML support can be broken down into several areas. DB2 9 supports the XQuery language through the XML interface and supports SQL through existing relational interfaces. The hybrid architecture supports combinations of SQL, SQL/XML, and XQuery. The DB2 9 architecture supports the W3C XQuery 1.0 and XPath 2.0 specifications.

DB2 9 Optimizer Extensions

Chapter 3 outlined how IBM integrated DB2 9 native XML support using the existing DB2 architecture. Key to this integration was the fact that the optimizer component of DB2 was designed from the beginning in a modular fashion. Thanks to this modularity, DB2 9 developers were able to "extend" the optimizer components to support XML expressions. In DB2 9, XQuery is represented with an internal query graph model. The optimizer Query Graph Model (QGMX) has been modified to incorporate the internal data flow model, with native constructs that are specific to XML and represent complex navigation of XQuery and XPath.

Figure 4.4 provides an overview of the hybrid query compiler.

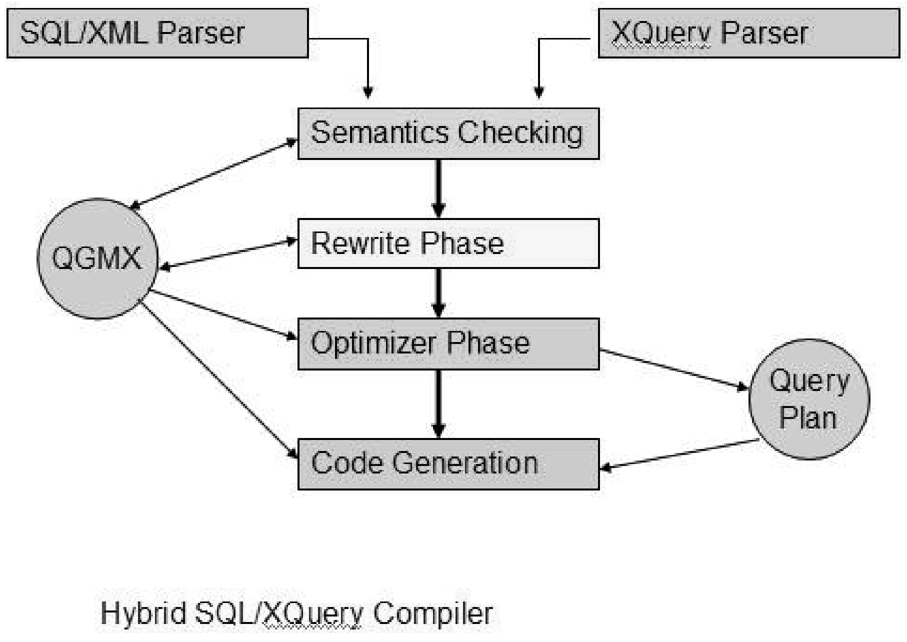


Figure 4.4: Hybrid SQL/XQuery compiler

The XQuery parser is new in DB2 9; all other components were extended to support the XQuery language and the XPath Data Model. As the figure illustrates, an SQL statement or XQuery expression is compiled into an internal data flow graph. In the next step, rewrite transformations are applied to optimize the data flow. The optimizer then uses the data flow graph to generate a physical query plan, which the code-generation step translates into executable code.

XML Schema Repository

The XML Schema Repository (XSR) is a DB2 9 repository for all XML artifacts used to validate and process XML instance documents stored in XML columns. The XSR is unique to DB2 9 and is based on the W3C XML Schema specification.

DB2 9 uses the XSR to validate XML instance documents pointed to by a URI associated with an XML schema, DTD, or other external entity. The XSR enables DB2 9 well-formed documents to be validated without requiring external resources that could be unavailable when needed. For example, if an XML instance document contained a reference to an external URI and that external URI wasn't available to complete the validation, the situation would cause validations to fail and would adversely impact DB2 9 processing and be beyond the control of the DB2 9 database system. For this reason, the XSR is internal to DB2, and there is an XSR for each database.

Each DB2 9 database contains an XSR that is composed of catalog tables, five catalog views, and system-provided stored procedures to enable schemas to be defined to the XSR. Each XML schema, DTD, or external entity registered with the XSR is represented as an XSR object. The XSR object is used to validate and process DB2 XML instance documents stored in XML columns.

You can register an XSR object using any of the following methods:

- DB2 9–provided stored procedures
- A Java application
- The DB2 command line processor

Using stored procedures, XSR object registration is a three-step process:

1. Register the primary XML schema document by calling the SYSPROC.XSR_REGISTER stored procedure.
2. Add any additional XML schema documents to be included with the primary XML schema by using the SYSPROC.XSR_ADDSCHEMADOC stored procedure.
3. Complete the registration by calling the SYSPROC.XSR_COMPLETE stored procedure.

Using the DB2 command line processor involves a similar three-step process:

1. Register the primary XML schema document using the REGISTER XMLSCHEMA command.
2. Optionally add any additional XML schema documents to be included with the primary XML schema using the ADD XMLSCHEMA DOCUMENT command.
3. Complete the registration by issuing the COMPLETE XMLSCHEMA command.

To register using a Java application, call the previously mentioned stored procedures. In addition, the following methods let you perform the same operations from a Java application program.

- DB2Connection.registerDB2XMLSchema registers an XML schema using one or more XML schema documents.
- DB2Connection.deregisterDB2XMLObject removes an XML schema definition from DB2.

Note that by default the XSR samples aren't installed in DB2 9; to create them, run the db2sample_XML command line processor script contained in the DB2 9 samples subdirectory. You'll see a complete example of the XML Schema registration process using the Developer Workbench (DWB) in Chapter 5.

DB2 9 Native XML Storage Architecture

Together, native XML data-type integration into the database engine and integration of XML support into the DB2 Storage Model form the pillars of the XML support provided by the DB2 9 hybrid data server. Figure 4.5 gives a high-level overview of the DB2 9 native XML storage architecture. The diagram illustrates how both relational and XML storage are integrated into the DB2 Storage Model.

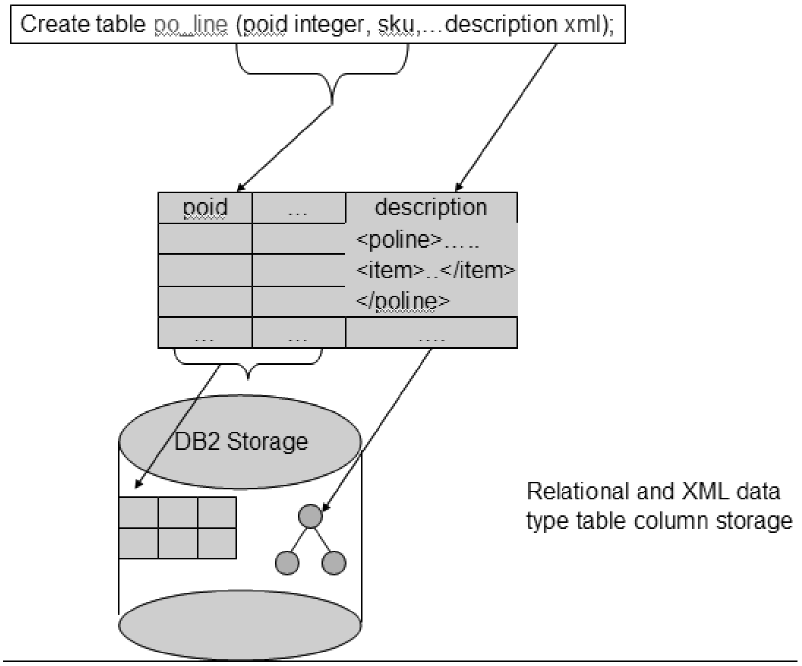

Create table po_line (poid integer, sku,…description xml);
poid
…
description
<poline>…..
<item>..</item>
</poline>
…
…
….
DB2 Storage
Relational and XML data type table column storage

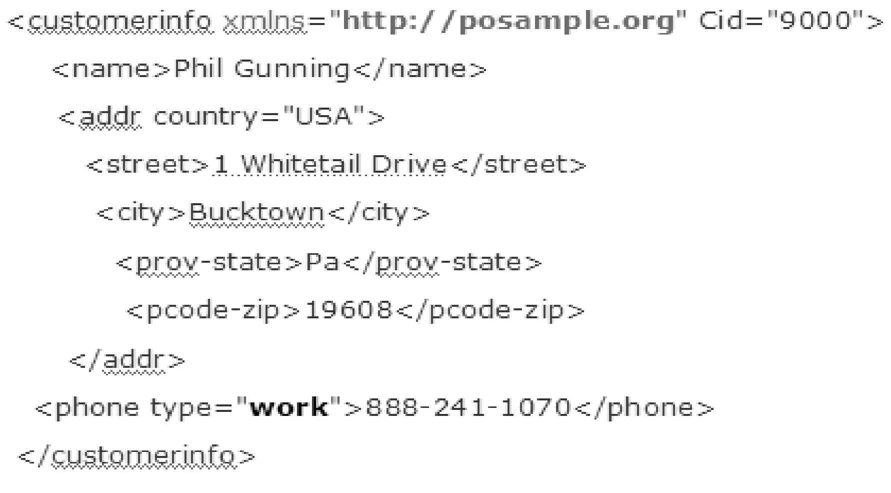
Figure 4.5: DB2 9 integrated storage architecture

Although DB2 9 stores the two types of data differently, it uses the same DB2 storage “subsystem” to manage both types of storage, providing seamless integration for applications accessing both relational and XML data. This powerful solution increases productivity because developers don’t have to spend time writing interfaces to access XML documents stored elsewhere.

DB2 stores XML instance documents in columns of type XML, and their hierarchical structure and meaning are preserved. The amount and nature of XML data can vary widely, from small XML documents to large XML documents that can be many gigabytes in length. Retrieval and update frequency can also vary widely. In such an environment, documents must be able to span disk pages because a single text node could be larger than a page.

The DB2 9 solution to this problem is to implement a storage model that stores XML documents as instances of the XDM in a structured, type-annotated tree. DB2 9 stores the binary representation of type-annotated XML trees, which avoids the repeated parsing and validation of documents. This approach also enables digital signatures to be preserved, a capability that is increasingly important in the financial sector, where digital signatures are used to provide proof of authorization for sensitive financial transactions, such as buy and sell orders.

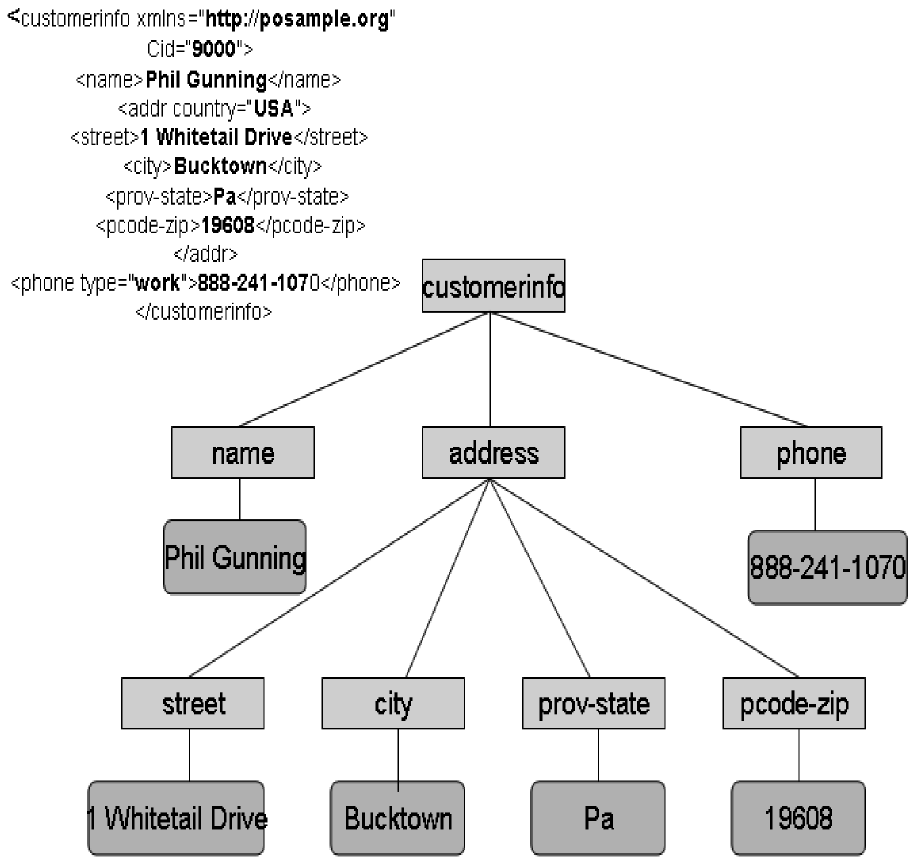
In most cases, you won't want to concern yourself with how DB2 9 internally manages XML storage; however, to help you understand what DB2 9 is doing under the covers, I'll briefly discuss how XML data is managed internally using the sample document shown in Figure 4.6 as an example.



```
<customerinfo xmlns="http://posample.org" Cid="9000">
   <name>Phil Gunning</name>
    <addr country="USA">
      <street>1 Whitetail Drive</street>
       <city>Bucktown</city>
         <prov-state>Pa</prov-state>
          <pcode-zip>19608</pcode-zip>
     </addr>
  <phone type="work">888-241-1070</phone>
 </customerinfo>
```

Figure 4.6: Customerinfo document

As the figure shows, the customerinfo document contains multiple elements represented in a hierarchy. The customerinfo root element contains subelements and attributes such as name, address, city, and state. Figure 4.7 is a representation of the customerinfo document in hierarchical form.



```
<customerinfo xmlns="http://posample.org"
                  Cid="9000">
        <name>Phil Gunning</name>
              <addr country="USA">
        <street>1 Whitetail Drive</street>
              <city>Bucktown</city>
            <prov-state>Pa</prov-state>
          <pcode-zip>19608</pcode-zip>
                      </addr>
<phone type="work">888-241-1070</phone>
              </customerinfo>
```

Figure 4.7: Hierarchical representation of the customerinfo document

When this document is stored, DB2 preserves its internal structure and converts its tag names and other information into integer values. Figure 4.8 shows how DB2 assigns *StringIDs* internally to nodes.

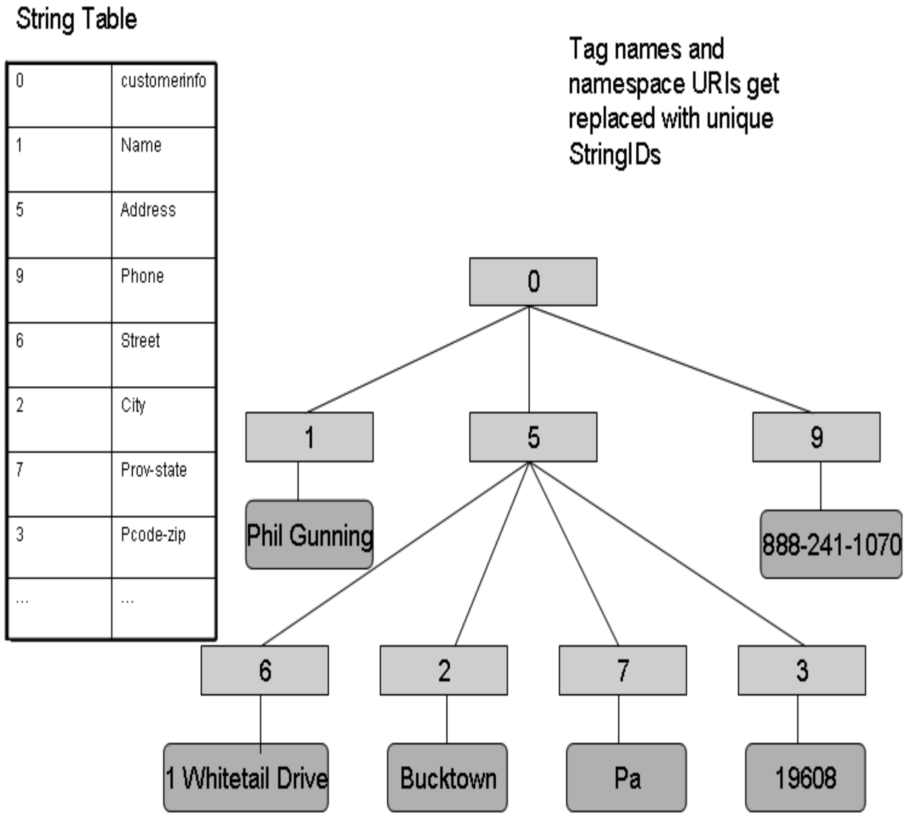


Figure 4.8: Hierarchical representation of customerinfo document with StringIDs

Replacing tags with StringIDs enables DB2 to provide excellent query performance. StringIDs let DB2 perform node comparisons using integers instead of strings. DB2 9 also stores extra information with each node. Child slots are associated with element nodes and have hints within them to provide information about what the child represents, enabling non-qualifying children to be skipped. This mechanism helps reduce I/O by allowing navigation across child nodes.

A unique identifier is assigned to each node and provides for logical and physical addressability that may be used for indexing and query optimization. If a document tree doesn't fit on one page, DB2 9 will split it into *regions* containing subtrees of nodes. Regions are connected by a *region index*, which is created automatically for every table that contains one or more XML columns. Region indexes enable DB2 9 to use efficient prefetching techniques. Finally, because logical node identifiers are independent of physical node locations, it's much easier to insert, update, and delete nodes or subtrees. Similarly, documents or page reorganization is facilitated.

Querying DB2 9 XML Data

DB2 9 contains two XQuery functions for obtaining input XML data from DB2 9 databases. The db2-fn:xmlcolumn function (which you saw used in a sample query earlier in the chapter) retrieves an entire XML column. The db2-fn:sqlquery function retrieves XML values based on an SQL fullselect.

The db2-fn:xmlcolumn function takes as input a string literal that identifies an XML column (defined as a DB2 XML data type) and returns a sequence consisting of all document nodes in the specified column. The following example shows how you use this function.

```
for $c in db2-fn:xmlcolumn ("ORDERS.PO_ORDER")
where $c/order/customer/custid = 4388
return $c/tot_qty
```

You can use db2-fn:xmlcolumn multiple times in a single XQuery to reference different XML columns in the same or separate tables or to reference the same XML column several times.

The db2-fn:sqlquery function lets you restrict the input to an XQuery based on conditions placed on relational columns in the same or related tables. This function accepts any select statement that returns a single XML column. The following example filters the set of input documents to XQuery by using a predicate and a join on another relational table.

```
for $c in db2_fn:sqlquery ('select po_order from purchases, ship-
ping where purchases.cust_id = shipping.SORDER and purchases.sku
="12345"')/order/
customer
where $c/cust_ID =4388
return $C/tot_qty
```

DB2 9 Methods of Querying XML Data

DB2 9 provides four ways to query and/or manipulate DB2 XML data:

- XQuery-only
- XQuery that invokes SQL
- SQL-only
- SQL/XML functions that execute XQuery expressions

The preceding examples demonstrated retrieving XML data via the XQuery-only method. Which method you choose depends on the type of data to be accessed and the processing to be performed, whether by the DB2 application or an application (e.g., a Web service) downstream.

XQuery-Only

A query that invokes XQuery directly in DB2 9 must begin with the keyword XQUERY. By specifying this keyword, you indicate to DB2 that XQuery is being used and that the query must be processed in accordance with the case-sensitivity rules that apply to the XQuery language.