Modifying XML Documents

BDB XML allows you to modify documents already stored in its containers using XQuery Update statements. This section provides a brief introduction to update statements so as to help you get going with them.

Note that if you use update statements on a document stored in a whole document container, then you might lose some of your document's formatting. This is because update statements reparse the documents they operate upon and then ultimately store them back in the container in the format used for node storage containers. For this reason, if the formatting of your XML documents are very important to you, you should avoid using XQuery Update Statements on your documents.

XQuery Update Introduction

XQuery Update allows you to insert, delete, replace and rename nodes using built-in keywords (insert, delete, replace and rename, respectively). You can also perform a node update by declaring an update function.

XQuery Update does not perform updates (node insertion, deletion, and so forth) until after the query has completed. This means a couple of things. First, you cannot perform an update and return the results in the same query.

Also, update statements are order independent, although in some cases conflicting updates are performed in an order defined by the XQuery Update statement specification).

Finally, updates are generally expected to be performed in isolation from other queries. You can not, for example, perform a search and then in a subsequent statement perform an update, all in the same query.

Note

XQuery Update is described in the W3C specification, *XQuery Update Facility 1.0*. This specification is currently a working draft. BDB XML implements the version of the specification dated 28 August 2007

Inserting Nodes Using XQuery Update

To insert a node into an existing document, you must identify the node that you want to insert, and the location in the document where you want the insertion to be performed. You indicate that you are performing an insertion operation using the XQuery `insert` keyword.

The general format of this expression is:

```
insert nodes nodes keyword position
```

where

- *nodes* is the content that you want to insert. This can be a string, or it can be an XQuery selection statement.
- *keywords* indicates how you would like the new content to be inserted.
- *position* indicates the document and the location in that document where the insertion is to occur.

```
<a>
   <b1>first child</b1>
   <b2>second child</b2>
   <b3>third child</b3>
</a>
```

Assuming this document is called 'mydoc.xml', then you can insert a node, `b4` after node `b2` using the following query expression:

```
insert nodes <b4>inserted child</b4> after
doc("dbxml:/con.dbxml/mydoc.xml")/a/b2
```

The above expression applied to the XML document results in a document like this:

```
<a>
   <b1>first child</b1>
   <b2>second child</b2><b4>inserted child</b4>
   <b3>third child</b3>
</a>
```

Note that if the query expression provided above happens to match more than one node, then the query will fail. For multiple node insertions, use an XQuery FLWOR expression. For example if our original working document is:

```
<a>
   <b1>first child</b1>
   <b2>second child</b2>
   <b2>another second child</b2>
   <b3>third child</b3>
</a>
```

Then to insert a node after every `<b2>` node in the document, use this:

```
for $i in doc('dbxml:/con.dbxml/mydoc.xml')/a/b2 return
insert nodes <b4>inserted child</b4> after $i
```

This results in the document:

```
<a>
   <b1>first child</b1>
   <b2>second child</b2><b4>inserted child</b4>
   <b2>another second child</b2><b4>inserted child</b4>
   <b3>third child</b3>
</a>
```

Position Keywords

XQuery Update expressions that add content to a document must first select the location in the document where the content is to be added, and then it must identify where the content is to be added relative to the selected location. You do this by specifying the appropriate keywords to the update expression.

Valid keywords are:

- `before`

  The new content precedes the target node.

- `after`

  The new content follows the target node.

- `as first into`

  The new content becomes the first child of the target node.

not used in an update expression that also makes use of the keywords noted above. It that happens, the node is inserted so that it does not interfere with the indicated position of the other new nodes.

Note that the behavior described here is an artifact of BDB XML's current implementation of the XQuery Update specification. The specification does not require the inserted node to be placed as the last child of the target node, so this behavior may change for some future release of the product.

Insertion Rules

When inserting elements, the selection expression must be non-updating, and it must not result in an empty set.

If any form of the `into` keyword is specified, the selection expression must result in a single element or document node. Also, if `before` or `after` is provided, the selection expression result must be a single element, text, comment or processing instruction node.

If an attribute node is selected, then the new content must provide an attribute.

Deleting Nodes Using XQuery Update

You can delete zero or more nodes using a `delete nodes` query. For example, given the document named "mydoc.xml" in container "con.dbxml":

```
<a>
   <b1>first child</b1>
   <b2>second child</b2><b4>inserted child</b4>
   <b3>third child</b3>
</a>
```

The following query deletes the `b4` node:

```
  delete nodes doc("dbxml:/con.dbxml/mydoc.xml")/a/b4
```

Note that if the document had more than one `<b4>` node, then they all would be deleted by this query.

The selection expression that you provide must be a non-updating expression, and the result must be a sequence of zero or more nodes. If the selection expression selects a node that has no parent, then the result is to delete the entire document from the container.

Replacing Nodes Using XQuery Update

You can use XQuery Update statements to either replace an entire node, or a node's value. To replace a node, use the `replace node` query. For example, given the document named "mydoc.xml" in container "con.dbxml":

```
<a>
   <b1>first child</b1>
   <b2>second child</b2>
   <b3>third child</b3>
</a>
```

You can replace node `b2` with a different node such as `<r1>replacement child</r1>` using the following query:

```
  replace node doc("dbxml:/con.dbxml/mydoc.xml")/a/b2
  with <z1>replacement node</z1>
```

The result of this replacement query is:

```
<a>
   <b1>first child</b1>
   <z1>replacement node</z1>
```

The replacement value can also be a selection expression. For example, suppose you had a second document named `replace.xml`:

```
<a>
   <rep>more replacement data</rep>
</a>
```

Then you can replace node `z1` with the `rep` node using the following query:

```
replace node doc("dbxml:/con.dbxml/mydoc.xml")/a/z1
with doc("dbxml:/con.dbxml/replace.xml")/a/rep
```

Or, as an XQuery FLWOR expression:

```
for $i in doc("dbxml:/con.dbxml/mydoc.xml")/a/z1 return
replace node $i with doc("dbxml:/con.dbxml/replace.xml")/a/rep"
```

Either expression results in the document:

```
<a>
   <b1>first child</b1>
   <rep>more replacement data</rep>
   <b3>third child</b3>
</a>
```

In addition to the `replace node ... with ...` form, you can also replace node values. Do this using `replace value of node ... with ...` queries.

For example, to replace the value of the `rep` node, above, use:

```
replace value of node doc("dbxml:/con.dbxml/mydoc.xml")/a/rep
  with "random replacement text".
```

The results of this query is:

```
<a>
   <b1>first child</b1>
   <rep>random replacement text</rep>
   <b3>third child</b3>
</a>
```

Replacement Rules

When replacing elements, the selection expression used to select the target must be non-updating, and it must not result in an empty set.

Selection results must consist of a single element, text, comment or processing instruction. In addition, the selection expression must not select a node without a parent node.

Finally, If you replace an attribute node, its replacement value must not have a namespace property that conflicts with the `namespaces` property of the parent node.

Renaming Nodes Using XQuery Update

You can rename a node using `rename node` query. For example, given the document named "mydoc.xml" in container "con.dbxml":

```
<a>
   <b1>first child</b1>
   <b2>second child</b2>
   <b3>third child</b3>
</a>
```

You can rename node `b3` to `z1` using the following query:

The selection expression that you provide must be a non-updating expression, and the result must be non-empty and consist of a single element, attribute, or processing instruction node.

Updating Functions

You can create a function that performs an update, so long as it is declared to be an updating function. In addition, this function must not have a return value, and the argument passed to the function cannot be an update query.

For example, the following query creates a function that renames any element node passed to it, to the value passed in the second argument. The function is then called for `b1` in document `mydoc.xml`, which is stored in container `con.dbxml`:

```
declare updating function
    local:renameNode($elem as element(),
                     $rep as xs:string)
    {
        rename node $elem as $rep
    };

    local:renameNode(doc("dbxml:/con.dbxml/mydoc.xml")/a/b1, "aab1")
```

If the prior query is called on a document such as this:

```
<a>
   <b1>first child</b1>
   <b2>second child</b2>
   <b3>third child</b3>
</a>
```

then that document becomes:

```
<a>
   <aab1>first child</aab1>
   <b2>second child</b2>
   <b3>third child</b3>
</a>
```

Transform Functions

While it is true that you cannot run an update query and simultaneously return the results, there is a way to almost do the same thing. You do this by making a copy of the nodes that you want to modify, then perform the modifications against that copy. The result of the modification is returned to you. This type of an operation is called a *transformation*.

Note that when you perform a transformation, the original nodes that you copied are *not* modified. For this reason, transformations are often limited only to situations where you want to modify a query result — for reporting purposes, for example.

To run a transformation, use the

1. `copy` keyword to copy the nodes of interest
2. `modify` keyword to perform the XQuery Update against the newly copied nodes
3. `return` keyword to return the result of the transformation.

For example, given the following XML document (which is identified as document `mydoc.xml`, and is stored in container `con.dbxml`):

```
<a>
   <aab1>first child</aab1>
   <b2>second child</b2>
   <b3>third child</b3>
</a>
```

results in the following document:

```
<a>
   <b2>replacement value</b2>
   <b3>third child</b3>
</a>
```

Resolving Conflicting Updates

Modifications that you specify as a part of an update query are not actually made until after the query is completed. The order in which update statements are made may or may not be relevant when it comes time to apply the update. As a result, it's possible to request an update that on its own is acceptable, but when used with other update statement may result in an error.

Keep the following rules in mind as you use update expressions:

1. And exception is raised if:
    a. Two or more `rename` expressions target the same node.
    b. Two or more `replace` expressions or `replace value of` expressions target the same node.
2. The following expressions are made effective, in the following order:
    a. All `insert into`, `insert attributes` and `replace value` expressions in the order they are supplied.
    b. All `insert before`, `insert after`, `insert as first`, and `insert as last` expressions in the order they are supplied.
    c. All `replace` expressions.
    d. All `replace value of` expressions.
    e. All `delete` expressions.

Note that atomicity of the expression is guaranteed; either the entire expression is made effective with regard to the original document, or no aspect of the expression is made effective.