Technical note— XTABLES: Bridging relational technology and XML

by J. E. Funderburk
G. Kiernan
J. Shanmugasundaram
E. Shekita
C. Wei

Additional work completed after the publication of "XTABLES: Bridging relational technology and XML" (*IBM Systems Journal* **41**, No. 4, 2002) indicated that a number of figures containing XQUERY and XML commands in that paper require modifications or additions in order to be correct and complete. We present the modified figures and other queries, along with SQL commands that can be used to generate the sample data described in the original paper and those produced by the modified queries.

The paper, "XTABLES: Bridging relational technology and XML" described the design and implementation of the XTABLES middleware system, which was intended to act as a bridge between legacy relational database systems and the emerging number of XML (Extensible Markup Language) -based applications. XTABLES uses relational databases for storing and querying XML documents.

An example carried throughout that paper concerned a simple purchase order database. Views of this database, both default and user-defined, could be queried by the use of XQUERY expressions. The sample data shown in Figure 1 (Figure 2 of the original paper) can be generated by the Structured Query Language (SQL) commands shown in this technical note in Figure 2.

Figure 1 (Figure 2 in the original paper) shows the default XML view for the purchase order database. This view can be generated by the following query:

```
namespace xp =
  "http://www.ibm.com/2001/12/xquery-functions"
⟨db⟩
{xp:table("EPURCHASE","ORDER")}
{xp:table("EPURCHASE","ITEM")}
{xp:table("EPURCHASE","PAYMENT")}
⟨/db⟩;
```

The user-defined XML view (a "create view" called "orders") shown here in an updated Figure 3 (Figure 4 in the original paper) transforms the default view into an XML format as desired by the user.

Queries can be issued against this user-defined view. The following two options for queries produce the same results, extracting a list of "item" elements from this view for a customer whose name begins with "Smith." In the original paper the "like" operator, undefined in XQUERY, was used.

Query using the XQUERY "starts-with" function:

```
for $order in view("orders")
let $items := $order/items
where starts-with(data($order/customer), "Smith") eq 'true'
return $items;
```

Query using the XQUERY "contains" function:

```
for $order in view("orders")
let $items := $order/items
where contains(data($order/customer), "Smith")
return $items;
```


0018-8670/03/$5.00 © 2003 IBM

Figure 1 A purchase order database and its default XML view

ORDER

id	custname	custnum
10	Smith Construction	7734
9	Western Builders	7725

ITEM

oid	desc	cost
10	generator	8000
10	backhoe	24000

PAYMENT

oid	due	amount
10	01/10/01	20000
10	06/10/01	12000

```
<db>
  <ORDER>
    <row> <id>10 </id> <custname> Smith Construction </custname> <custnum> 7734 </custnum> </row>
    <row> <id> 9 </id> <custname> Western Builders </custname> <custnum> 7725 </custnum> </row>
  </ORDER>
  <ITEM>
     <row> <oid> 10 </oid> <desc> generator </desc> <cost> 8000 </cost> </row>
     <row> <oid> 10 </oid> <desc> backhoe </desc> <cost> 24000 </cost> </row>
  </ITEM>
  <PAYMENT>
    ... (similar to <order> and <item>)
  </PAYMENT>
</db>
```

Figure 2 SQL commands for purchase order database sample data

```
drop table epurchase.order;
create table epurchase.order("id" int not null primary key, "custname" varchar(40) not null, "custnum" int not null);

drop table epurchase.item;
create table epurchase.item("oid" int not null, "desc" varchar(40) not null, "cost" int not null);

drop table epurchase.payment;
create table epurchase.payment("oid" int not null, "due" date not null, "amount" int not null);

insert into epurchase.order values(10,'Smith Construction', 7734);
insert into epurchase.order values(9, 'Western Builders', 7725);
insert into epurchase.item values(10, 'generator', 8000);
insert into epurchase.item values(10, 'backhoe',24000);
insert into epurchase.payment values(10, '2001-01-10', 20000);
insert into epurchase.payment values(10, '2001-06-10', 12000);
```

These queries can be parsed and converted to XQGM (XML Query Graph Model), then translated to SQL. For the "starts-with" query, the SQL produced is shown in Figure 4, including the "with" clause for common subexpressions. Note that the "starts-with" function is translated into a user-defined function xperanto."starts-with", which is implemented by XTABLES.

For the "contains" query, the SQL produced (including the "with" clause for the common subexpressions) is shown in Figure 5. Note that the "contains" function is translated into the SQL "locate" function.

Figure 3 User-defined XML view

```
1.  create view orders as (
2.    namespace xp = "http://www.ibm.com/2001/12/xquery-functions"
3.    for $order in xp:table("EPURCHASE","ORDER")/ORDER/row
4.     return
5.        <order>
6.           <customer>{ data($order/custname) }</customer>
7.           <items>{
8.            for $item in xp:table("EPURCHASE","ITEM")/ITEM/row
9.                where $order/id = $item/oid
10.               return
11.                   <item><description>{ data($item/desc) }</description>{$item/cost} </item>}
12.          </items>
13.          <payments>{
14.               for $payment in xp:table("EPURCHASE","PAYMENT")/PAYMENT/row
15.               where $order/id = $payment/oid
16.               return
17.                   <payment due={ data($payment/due) }>{ $payment/amount }</payment>
18.                   sortby(@due)}
19.          </payments>
20.       </order>
21.    sortby(customer)
22. );
```

Figure 4 SQL produced by XQUERY using “starts-with” function

```
WITH Q6 (c1) as (select q9."id" from EPURCHASE.ORDER AS q9
   where(xperanto."starts-with"(q9."custname", 'Smith') = 'true'))
select q1.c1, q1.c2, q1.c3, q1.c4
from  table(
            select q2.c1, 0, cast (null as VARCHAR(40)), cast (null as INTEGER)
            from Q6 AS q2
            union all
            select q3.c3, 1, q3.c1, q3.c2
            from table(
                        select q7."desc", q7."cost", q5.c1
                        from EPURCHASE.ITEM AS q7, Q6 AS q5
                        where (q5.c1 = q7."oid")
                       ) AS q3(c1, c2, c3)
) AS q1(c1, c2, c3, c4)
order by q1.c1, q1.c2;
```

Figure 5 SQL produced by XQUERY using "contains" function

```
WITH Q6 (c1) as (select q9."id" from EPURCHASE.ORDER AS q9
    where(locate('Smith', q9."custname") <> 0 ))
 select q1.c1, q1.c2, q1.c3, q1.c4
 from  table(
              select q2.c1, 0, cast (null as VARCHAR(40)), cast (null as INTEGER)
              from Q6 AS q2
              union all
              select q3.c3, 1, q3.c1, q3.c2
              from table(
                           select q7."desc", q7."cost", q5.c1
                           from EPURCHASE.ITEM AS q7, Q6 AS q5
                           where (q5.c1 = q7."oid")
                          ) AS q3(c1, c2, c3)
 ) AS q1(c1, c2, c3, c4)
 order by q1.c1, q1.c2;
```

General references

J. E. Funderburk, G. Kiernan, J. Shanmugasundaram, E. Shekita, and C. Wei, "XTABLES: Bridging relational technology and XML," *IBM Systems Journal* **41,** No. 4, 616–641 (2002).

Accepted for publication December 12, 2002.

John E. Funderburk *IBM Software Group, Silicon Valley Laboratory, 555 Bailey Avenue, San Jose, California 95141 (jfund@us.ibm.com).* Mr. Funderburk is a software developer at IBM's Silicon Valley Lab. He previously worked on the XML Extender for DB2 and is currently working on XTABLES.

Gerald Kiernan *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (kiernan@almaden.ibm.com).* Dr. Kiernan is a senior software engineer at IBM's Almaden Research Center. He previously worked on IBM's Object Broker, as well as the research version of XTABLES. He is currently doing research on privacy preserving databases.

Jayavel Shanmugasundaram *Cornell University, Department of Computer Sciences, Ithaca, New York 14853 (jai@cs.cornell.edu).* Dr. Shanmugasundaram is an assistant professor of computer science at Cornell University. He previously worked on the research version of XTABLES while he was a visiting scientist at IBM's Almaden Research Center. He is currently doing research on P2P indexing systems and query processing for unstructured data.

Eugene Shekita *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, Califoirnia 95120 (shekita@almaden.ibm.com).* Mr. Shekita is a research staff manager at IBM's Almaden Research Center. He previously worked on DB2's query optimizer, as well as the research version of XTABLES. He is currently doing research on query processing.

Catalina Wei *IBM Software Group, Silicon Valley Laboratory, 555 Bailey Avenue, San Jose, California 95141 (fancy@us.ibm.com).* Ms. Wei is a senior software developer at IBM's Silicon Valley Lab. She previously worked on IBM's Object Broker and is currently working on XTABLES.