-- XML IN ORACLE

-- This example shows how to create a table that stores XML data.

drop table addr_book;

create table addr_book(
    name       VARCHAR2(20),
    card       sys.xmltype,
    creationDate DATE
)
/

-- This shows that Objects of Type XML can be stored in Tables.

insert into addr_book values ( 'James',
    sys.XMLType.createXML( -- CONSTRUCTOR
        '<acard createdby="Jim">'
            '<email>geller@njit.edu</email>
            <wphone>596-3383</wphone>
            <wphone>596-3366</wphone>
            <cell>794-4399</cell>
        '</acard>'),
    sysdate)
/

-- The constructor takes the string that contains XML code and stores it as an XML-object.

select * from addr_book
/

-- This shows a more complicated version with deeper nested XML.

insert into addr_book values ( 'James',
    sys.XMLType.createXML( 'acard createdby="Jim">
        <email>geller@njit.edu</email>
        <wphone>596-3383</wphone>
        <wphone>596-3366</wphone>
        <cell>794-4399</cell>
        <address>
            <line1>323 Dr. King Blvd</line1>
            <city>Newark</city>
            <state>NJ</state>
        </address>
    '</acard>'),
    sysdate)
/
In order to do any decent querying, we need a larger database.
-- I had an unexplainable error. I fixed it by deleting and retyping.

```sql
insert into addr_book values ('James',
sys.XMLType.createXML(''acard createdby="Jim">
  <email>geller@njit.edu</email>
  <wphone>596-3383</wphone>
  <wphone>596-3366</wphone>
  <cell>794-4399</cell>
  <address>
    <line1>323 Dr. King Blvd</line1>
    <city>Newark</city>
    <state>NJ</state>
    <zip>07102</zip>
  </address>
</acard>'
),
sysdate)
/

insert into addr_book values ('Roger',
sys.XMLType.createXML(''acard createdby="Jim">
  <email>roger12@yahoo.com</email>
  <wphone>111-1234</wphone>
  <wphone>111-5678</wphone>
  <cell>111-3342</cell>
  <address>
    <line1>123 Main St.</line1>
    <city>Atlana</city>
    <state>Georgia</state>
    <zip>33333</zip>
  </address>
</acard>'
),
sysdate)
/

insert into addr_book values ('Bobby',
sys.XMLType.createXML(''acard createdby="Jim">
  <email>bob@aol.com</email>
  <wphone>111-2900</wphone>
  <address>
    <line1>200 Oak St.</line1>
    <city>Atlana</city>
    <state>Georgia</state>
  </address>
</acard>')
)
insert into addr_book values (  
'Tommy',  
sys.XMLType.createXML(  
'<acard createdby="Jim">  
<email>tom200@yahoo.com</email>  
<wphone>333-8000</wphone>  
<cell>111-9000</cell>  
</acard>' ),  
sysdate)  
/  
insert into addr_book values (  
'Abe',  
sys.XMLType.createXML(  
'<acard createdby="Jim">  
<email>abe1212@yahoo.com</email>  
<wphone>111-2000</wphone>  
<cell>111-3344</cell>  
<address>  
<line1>125 Main St.</line1>  
<city>Savannah</city>  
<state>GA</state>  
<zip>33300</zip>  
</address>  
<comment>Kids: John, Jamie</comment>  
</acard>' ),  
sysdate)  
/  
insert into addr_book values (  
'Tony',  
sys.XMLType.createXML(  
'<acard createdby="Raj">  
<email>anthony1@yahoo.com</email>  
<wphone>111-1000</wphone>  
<cell>111-3000</cell>  
<address>  
<line1>123 Elm St.</line1>  
<city>Savannah</city>  
<state>GA</state>  
<zip>33300</zip>  
</address>  
<comment>Works for CNN</comment>  
</acard>' ),  
sysdate)  
/  
insert into addr_book values (  
'Frank',  
sys.XMLType.createXML(  
'<acard createdby="Jim">  
<email>frank13@yahoo.com</email>  
<wphone>111-5000</wphone>  
<cell>111-5000</cell>  
<address>  
<line1>123 Elm St.</line1>  
<city>Savannah</city>  
<state>GA</state>  
<zip>33300</zip>  
</address>  
<comment>Works for CNN</comment>  
</acard>' ),  
sysdate)  
/
-- AS OF FALL 2008, in SQL Developer I did **NOT** have to use an alias for this query. 
-- HOWEVER in more complicated queries I still needed the alias!!!

-- CARD was the name of the column that contains XML objects. 
-- You MUST use an alias. Otherwise it does not work for more complicated queries.

-- We added a standard WHERE clause. 
-- This does NOT use the XML data.

-- To get subtrees out of an XML expression, we use the method extract(). 
-- extract() takes as argument a path expression. 

-- Now we have to learn a whole new language. It is called XPath. 
-- It is a query language for XML.

-- (1) A '/' extracts the whole XML object. 
-- This is called the ROOT object. 
-- Because XML forms trees.
-- And in UNIX '/' stands for root.
-- So that name makes sense.

-- As in UNIX, / is also used as a path separator.
-- The "right thing" is a child of the "left thing":

-- Like this:     left_thing/right_thing

select Name, a.card.extract('/').getstringVal()
from addr_book a;

-- "Give me the name and all the roots in the column card
-- and all descendants."

NOTE: Many times this works without the "getstringVal()"
method. However, putting this method is always safe (unless you
expect a number as a result).

p41

-- In the middle column I can put anything. Not just addresses

-- I did this.
insert into addr_book values ('Joe',
sys.XMLType.createXML('<test>here</test>'),
sysdate)
/

-- The last select statement will extract the also:

-- select Name, a.card.extract('')
-- from addr_book a;

-- Now try these two:
select Name, a.card.extract('/acard').getstringVal()
from addr_book a
/

select Name, a.card.extract('/test').getstringVal()
from addr_book a
/

-- The first one means: Return the root expressions
-- that are with all descendants.

-- The second one means: Return the roots with the root tag
-- test and all descendants.

p42
-- Now we refine this:
-- We need a new function. A boolean. It's like extract().
-- But it returns true (1) if something is there.
-- It is called existsNode().
-- It does NOT return the the XML structure, only the truth value.

select Name, a.card.extract('/test').getstringval()
from GELLER.addr_book a
where a.card.existsNode('/test') = 1
/

-- Now every row except "Joe" is gone completely!
-- --------------------------------------------------

-- This drops the row with Joe completely.
select Name, a.card.extract('/acard').getstringval()
from addr_book a
where a.card.existsNode('/acard') = 1;
p43

-- Now we will study how to get different parts of an /acard with XPath
-- Get the elements that are under roots.

select Name, a.card.extract('/acard/email').getstringval()
from addr_book a
where a.card.existsNode('/acard') = 1
/

-- Joe has no email address. So, try this again:
select Name, a.card.extract('/').getstringval()
from addr_book a
where a.card.existsNode('/acard/email') = 1;

-- Show name and all XML for people that have
-- an email tag inside of an acard tag. 0 = false,
p44

-- We don't want the tags. How do we get rid of them?

-- There is a function text() that goes INTO the XPath!!
-- NOT after it. text() works only for ONE LEVEL.

select Name, a.card.extract('/acard/email/text()').getstringval()
from addr_book a
where a.card.existsNode('/acard') = 1

p45
-- James and Roger have two phone numbers.
-- By default both are returned. Without Blank in between!

```sql
select Name, a.card.extract('/acard/wphone/text()').getstringval()
from addr_book a
where name='James' or name='Roger'
/
```

-- Not good...

```
select Name,
    a.card.extract('/acard/wphone[1]/text()').getstringval(),
    a.card.extract('/acard/wphone[2]/text()').getstringval()
from addr_book a
where name='James' or name='Roger'
/
```

-- What if we want both phone numbers?

```sql
select Name,
    a.card.extract('/acard/wphone[1]/text()').getstringval(),
    a.card.extract('/acard/wphone[2]/text()').getstringval(),
    a.card.extract('/acard/wphone[3]/text()').getstringval()
from addr_book a
where name='James' or name='Roger'
/
```

-- If out of bound, no error. Just nothing returned.

```sql
select Name, a.card.extract('/acard/wphone[1]/text()').getstringval(),
    a.card.extract('/acard/wphone[3]/text()').getstringval(),
    a.card.extract('/acard/wphone[2]/text()').getstringval()
from addr_book a
where name='James' or name='Roger'
/
```

```
-- What if we want both phone numbers nicely separated?

select Name,
    a.card.extract('/acard/wphone[1]/text()').getstringval() || ' ** ' ||
    a.card.extract('/acard/wphone[2]/text()').getstringval()
from addr_book a
where Name = 'Roger'
/
```

```
-- A longer path:

select Name, a.card.extract('/acard/address/zip/text()').getstringval()
from addr_book a
where Name = 'Bobby'
/
```

```
-- * matches one element.
```
-- This stands for ONE LEVEL.
-- Note that below I skip the "address"

select Name, a.card.extract('/acard/*/zip/text()').getstringval()
from addr_book a
where Name = 'Bobby'
/

-- This does not give the zip code. Can't skip two levels.

select Name, a.card.extract('/*/zip/text()').getstringval()
from addr_book a
where Name = 'Bobby'
/

This DOES work.

select Name, a.card.extract('/*//*/zip/text()').getstringval()
from addr_book a
where Name = 'Bobby'
/

p50

-- You can use wildcards in paths. // matches all descendants.
-- That means.... everything under the //
-- Note that below I skip the "address"

select Name, a.card.extract('/acard//zip/text()').getstringval()
from addr_book a
where Name = 'Bobby'
/

-- Or even this:

select Name, a.card.extract('//zip/text()').getstringval()
from addr_book a
where Name = 'Bobby'
/

-- Why? Because I don't want to remember where in the tree
-- zip is stored. But notice: Search time goes up!

select Name, a.card.extract('//wphone/text()').getstringval()
from addr_book a
where Name = 'Roger'
/

p51

-- You can use something like an implicit WHERE statement in the path.
select Name,
a.card.extract('/acard[email="roger12@yahoo.com"]//zip/text()').getstringval()
from addr_book a
/

-- The above gets the ZIP code. But... only for an acard that
-- has under it an email as given.
-- Note: The above only works with DOUBLE QUOTES.
-- Also note, as usual it returns all the rows anyway.
-- The extraction does not affect the where clause.

select Name,
a.card.extract('/acard[email="roger12@yahoo.com"]//zip/text()').getstringval()
from addr_book a
where
a.card.existsNode('/acard[email="roger12@yahoo.com"]//zip/text()') = 1
/

-- You can even have a path inside of the [ ] !!!!

This below says: Give me the Name and email address of everybody who lives in Atlana, but show me all the names of people with email addresses.

select Name,
a.card.extract('/acard[address/city="Atlana"]/email/text()').getstringval()
from addr_book a
where
a.card.existsNode('/acard/email') = 1
/

p52

-- You can extract in a where clause also.

select Name, a.card.extract('/acard/email/text()').getstringval()
from addr_book a
where a.card.extract('/acard/address/city/text()').getstringval() = 'Atlana'

-- It's wrong in the database, so I need it wrong here too!!!!

-- Give me the email address of people who live in Atlanta.
-- If you want a string, you have to now append the method
-- getstringval()
-- If you want a number, you use .getnumberval()

-- This is almost the end of processing XML within Oracle.
-- Of course there is much more to learn about XPath...
-- REMEMBER XML PRESERVES WHITESPACE.

THUS
234
IS NOT THE SAME AS
 234
AND

234

p52b

-- You can extract attributes also, with an @ sign:

select Name, a.card.extract('/acard/email/text()').getstringval()
from addr_book a
where a.card.extract('/acard/@createdby').getstringval() = 'Jim'
/

select Name, a.card.extract('/acard/@createdby').getstringval()
from addr_book a
/