INSTITUTIONEN FÖR DATA-
OCH SYSTEMVETENSKAP
SU / KTH

DB2 & XML LABORATION

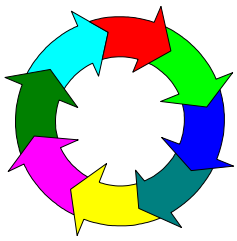
v. 2.0

IS4

Modeller och språk för objekt- och relationsdatabaser

HÖSTTERMINEN 2001

http://L238.dsv.su.se/courses/IS4/



nikos dimitrakas

Table of contents

Table of figures

1 Introduction

This compendium contains the following:

- An introduction to XML
- An introduction to DB2's facilities for handling XML data
- Exercises on using DB2 for managing XML data

1.1 Homepage

Information about this compendium can be found here:
http://L238.dsv.su.se/courses/IS4

The following can be found at this address:

- Feedback form

Use this form to send comments/questions about the compendium to the author.

- FAQ

Here there is a list of corrections and explanations.

- Links

Internet resources that can be helpful when working with the compendium.

- Files

The newest version of the compendium and all the files needed to complete the exercises in the compendium.

1.2 The environment

- IBM DB2 Universal Database version 6, with XML extender
  The following facilities of DB2 will be used:
  - DB2 Command Window
  - DB2 Command Center
  - DB2 Information Center
  - Editor (of your choice)

More information on DB2 and its facilities can be found in the "Introduktion till DB2 v. 6"-compendium.

2 XML & DB2

This chapter introduces XML and DB2's facilities for working with XML. This is not a complete reference of either XML or DB2's XML extender. The following sections only present the aspects of XML and DB2 that are needed to complete the exercises that follow.

2.1 XML

XML (eXtensible Markup Language) is a language with many uses. One of them is to transport data between different systems.

XML consists of two languages, one language for the actual XML documents and one language for specifying how the XML documents[1] should be structured, called DTD (Document Type Definition). Not all XML documents are associated to DTDs. Here is an example of an XML document and its DTD:

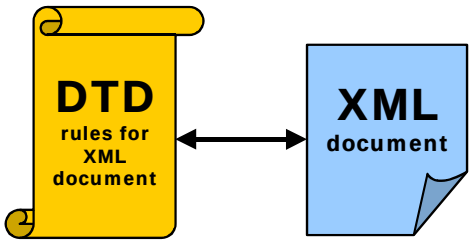
XML Document **(saved in a file called "book.xml")**

```
<?xml version="1.0"?>
<!DOCTYPE book SYSTEM "c:\dxx\samples\dtd.book.dtd">
<book>
    <chapter id="1" date="07/01/1997">
        <section>This is a section in Chapter One.</section>
    </chapter>
    <chapter id="2" date="01/02/1997">
        <section>This is a section in Chapter Two.</section>
        <footnote>A footnote in Chapter Two is here.</footnote>
    </chapter>
    <price date="12/22/1998" time="11.12.13" timestamp="1998-12-22-11.12.13">
   38.281
  </price>
</book>
```

DTD **(file "book.dtd")**

```
<?xml encoding="US-ASCII"?>
<!ELEMENT book (author*,chapter*,price)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT chapter (section*, footnote*)>
<!ATTLIST chapter id   (1|2|3) #REQUIRED
  date CDATA #IMPLIED>
<!ELEMENT price (#PCDATA)>
<!ATTLIST price date CDATA #IMPLIED
  time CDATA #IMPLIED
  timestamp CDATA #IMPLIED>
<!ELEMENT section (#PCDATA)>
<!ELEMENT footnote (#PCDATA)>
```

ⓘ Both languages are case sensitive!



An XML document can refer to a DTD file.
A DTD file can be associated with many XML documents. When an XML document refers to a DTD file then the XML documents content is supposed to follow the rules in the DTD file.

Figure 1 XML and DTD

2.1.1 XML Explanation

Elements:

In the previous example **chapter** is an element. Everything from the **<chapter>** to the **</chapter>** constitutes an element **chapter.**

Every XML document must have a root element, an element that has its start tag in the beginning of the XML document and its end tag at the end of the XML document. This element may appear only once in the XML document.

Attributes:

The element **chapter** has an attribute **id** and an attribute **date.** All attributes of an element appear within the starting tag of the element. Attributes have a value that is within double quotation marks (").

[1] **The term XML document refers to a file with the extension .xml.**

Structure:

<element attribute1=”value” attribute2=”value2”>
element content
</element>

The element content can be empty, text or other elements.

XML declaration & DOCTYPE element

The first two lines of any XML document are always the XML declaration & the DOCTYPE element:

XML declaration:
<?xml version="1.0" standalone=”no”?>
In the XML declaration we define the XML version and whether there is a DTD file with rules for the XML structure or not

DOCTYPE element:
<!DOCTYPE Book SYSTEM "c:\ dtd\book.dtd">
The DOCTYPE points out the root element of the XML document and the SYSTEM points out the DTD file for the XML document.

2.1.2 DTD Explanation

The DTD file contains rules to be followed when constructing an XML document.

It defines the elements that can appear in the XML document:
<!ELEMENT element-name>

It defines the elements that can appear within an element:
<!ELEMENT element-name (element2-name)>
or the type of the element content:
<!ELEMENT element-name (#PCDATA)>

It also defines the attributes that an element can have, with the appropriate rules (the type of the attribute, whether it has to be there or not, etc.) :
<!ATTLIST element-name
attribute1-name CDATA #IMPLIED
attribute2-name CDATA #IMPLIED>

For more help on how to construct an XML document visit one of the following tutorial sites (tutorials for both XML and DTD):

- http://L238.dsv.su.se/tutorial
- http://www.w3schools.com/xml/default.asp
- http://www.spiderpro.com/bu/buxmlm001.html

2.2 XML in DB2

DB2 provides two ways for working with XML documents and XML data[2]:

- XML collection
- XML column

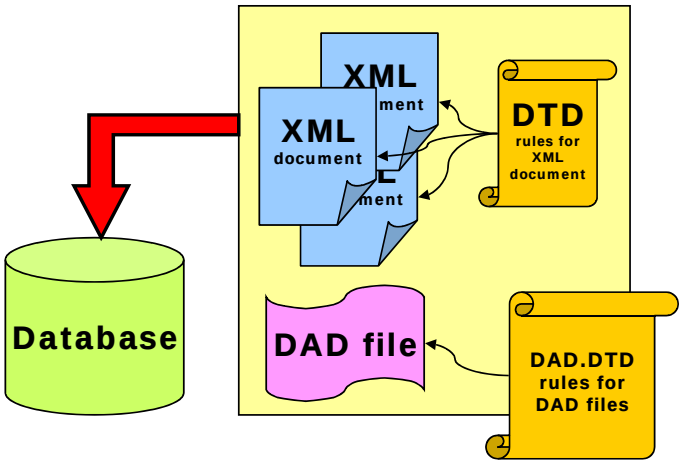
2.2.1 XML collection

When XML data is stored in a relational database, then this database is called an XML collection. DB2 XML extender provides functions for decomposing XML documents into relational data to be stored in the XML collection, and functions for composing XML documents from XML data stored in the XML collection.
Since XML documents are based on hierarchical models and relational databases are based on relational models, it is important to have a mapping between the two models. This mapping can then be used for transformations in both directions. The mapping is defined in DAD (Document Access Definition) files. A DAD file is an XML document that has the extension .dad and follows the rules defined in the file dad.dtd[3]. The DAD file is then used when enabling the XML collection. At that time DB2 verifies that the tables referred in the DAD file exist, otherwise they are created.
In chapter 4 there is a more detailed description of how to do all this in practice.

2.2.2 XML column

XML column is a different approach than XML collection. XML column is an XML enabled database that contains intact XML documents. Those XML documents are stored in a certain table that has a column of one of these three types: XMLCLOB, XMLVARCHAR, XMLFile. That column has to be enabled and associated with a DAD file. In the DAD file there can be reference to a DTD file for validating the incoming XML documents, and rules for creating side tables[4] and storing XML data in them. The DTD file must have been registered in the DTD_REF table that is created when a database is being enabled for XML.
There are more details about this in chapter 4.

[2] With the term XML data we refer to the contents of XML documents, even when the data has been transformed. Data that is going to become the content of an XML document can also be referred to as XML data
[3] The file dad.dtd can be found in the following directory:
c:\dxx\dtd on all the machines that have the DB2 XML extender installed.
[4] A side table is a table that contains data from the XML document. The side tables are used to improve performance when searching through the XML documents. Usually, only some of the XML data is placed in the side tables – the data that is used most frequently when searching.



All the XML components are stored in the database. The XML documents, DTD files and DAD files are stored in user tables, while the DAD.DTD file is stored in the database manager.

The database can of course contain other non XML specific components too. Those components are not represented in Figure 2.

Figure 2 Main components of XML in DB2

3 Database

For the exercises that follow we will use a database about horse-riding. The database consists of five tables. The tables are connected with foreign keys as shown in Figure 3.

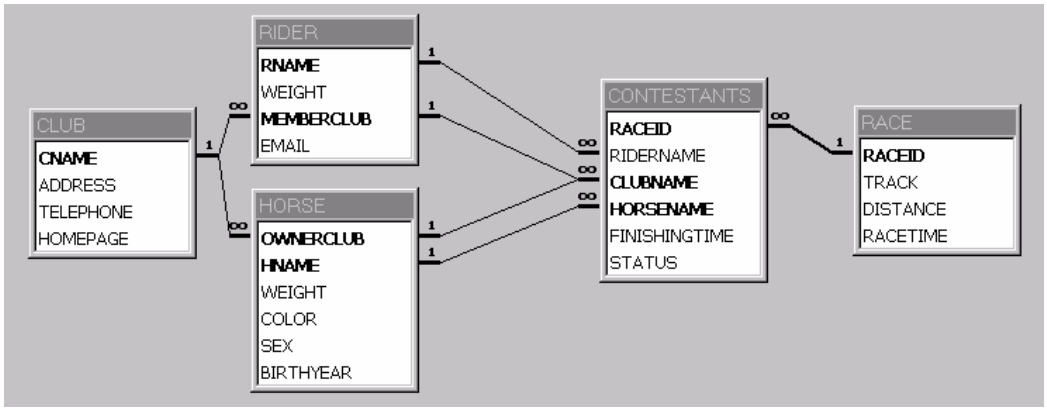


Figure 3 Database model

Scripts for creating and populating the database can be found here:

- \\DB-SRV-1\StudKursInfo\IS4 ht2001\DB2-XML\Scripts, or
- http:\\L238.dsv.su.se\courses\is4

4 Exercises

This chapter is divided into two sections. In the first section we go through an exercise step by step from the beginning to the end. In the second section a similar exercise is described and has to be completed based on the knowledge acquired from the first section.

4.1 Step-by-step exercise

In this exercise we will do the following:

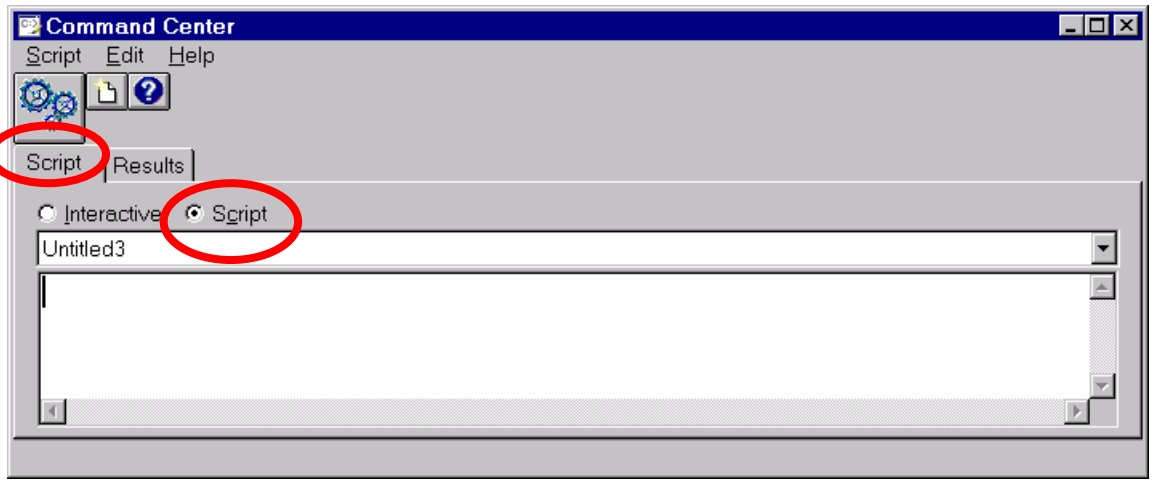
- Create a database

- Enable the database for XML (as an XML collection) and compose XML documents from the data in the XML collection
- Extract XML documents into XML files
- Store XML documents in an XML column
- Run queries against the XML column

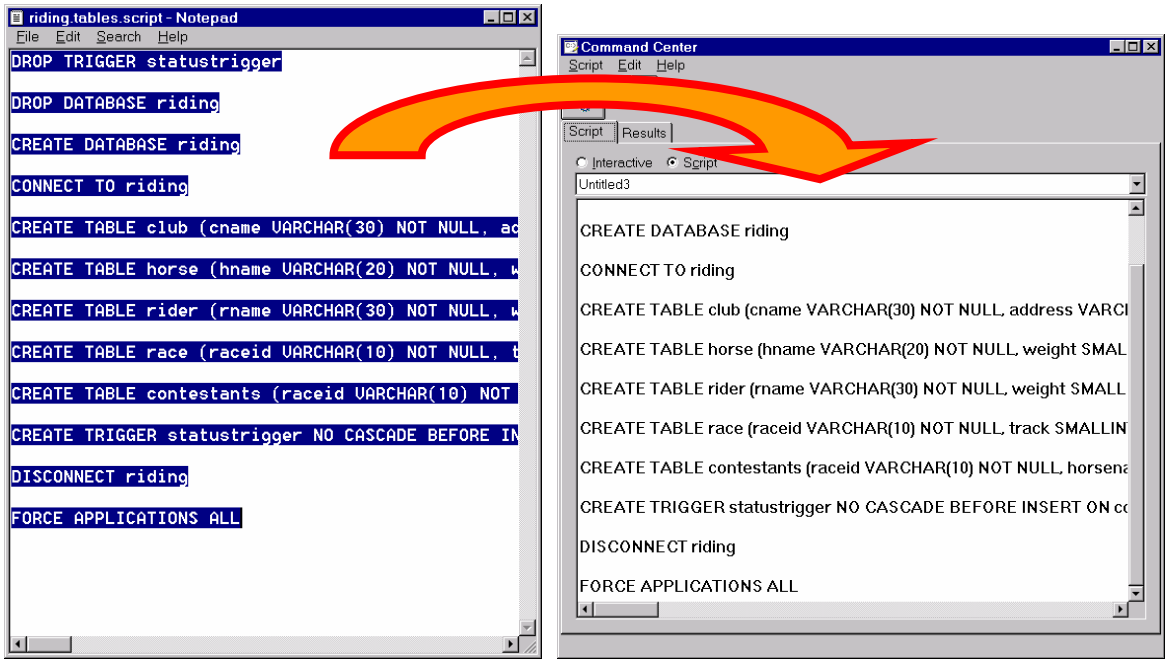
4.1.1 Create a database

The database can easily be created and populated by using the scripts (see chapter 3). To run the scripts follow these steps:

- Open the command center (Start – Programs – DB2 for Windows NT – Command Center)!
- Go to the Script tab and activate the script radio button!



- Copy and then paste the contents of the first script file (riding.tables.script) into the command center!



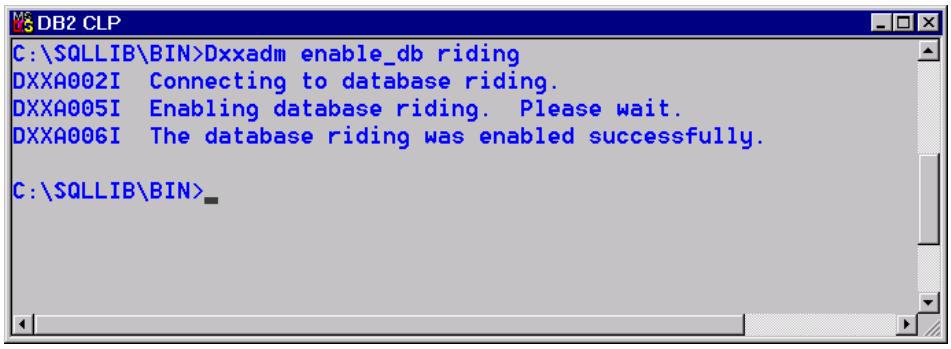
- Execute the script by pressing the execute button 

When the execution of the script is finished, your database has been created. For populating the database use the second script file (riding.insert.all.script).

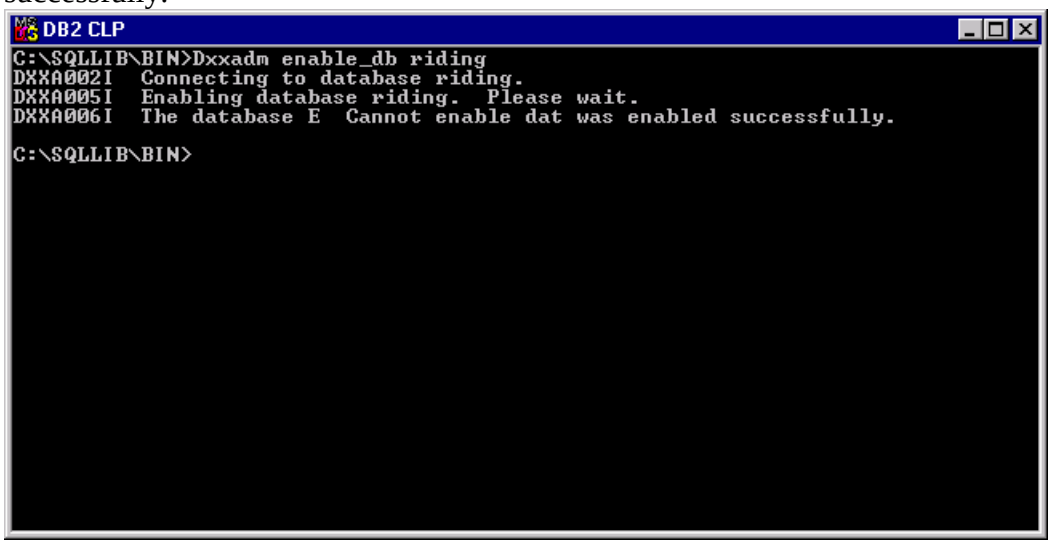
4.1.2 Enable the database for XML (as an XML collection) and compose XML documents

When the database has been created, it is just an ordinary relational database. If the database is going to be used as an XML collection then it has to be enabled for XML. That is done by using the following:

- Start the Command Window (Start > Programs > DB2 for Windows NT > Command Window)
- Execute this command in the Command Window:
  Dxxadm enable_db riding



Sometimes DB2 responds with a funny/confusing message (see image that follows). If that message comes up then the operation has gone well and the database has been enabled successfully.



When that is done there should be a few more tables in the database. Those tables are used by the XML extender. For example the table DTD_REF contains information about DTD files.

The next step is to enable the XML collection. That is not a necessary step. To enable the XML collection we need to have a DAD file. The DAD file is specified when enabling an XML collection. The DAD file can contain information on how to compose XML documents from the XML collection and how to decompose XML documents into the XML collection. If the XML collection is not enabled, then the DAD file must be specified every time an XML document is to be composed or decomposed.

In this exercise we will just specify rules for composition of XML documents in the DAD file and we will enable the XML collection.

First we need to create a DAD file. To do that we need to know how we want the XML document to be structured and where all the XML data are stored in the database. In other words we need to define the XML document structure and map it to the XML collection tables and columns.

Here is the structure for the XML documents that we want to compose:

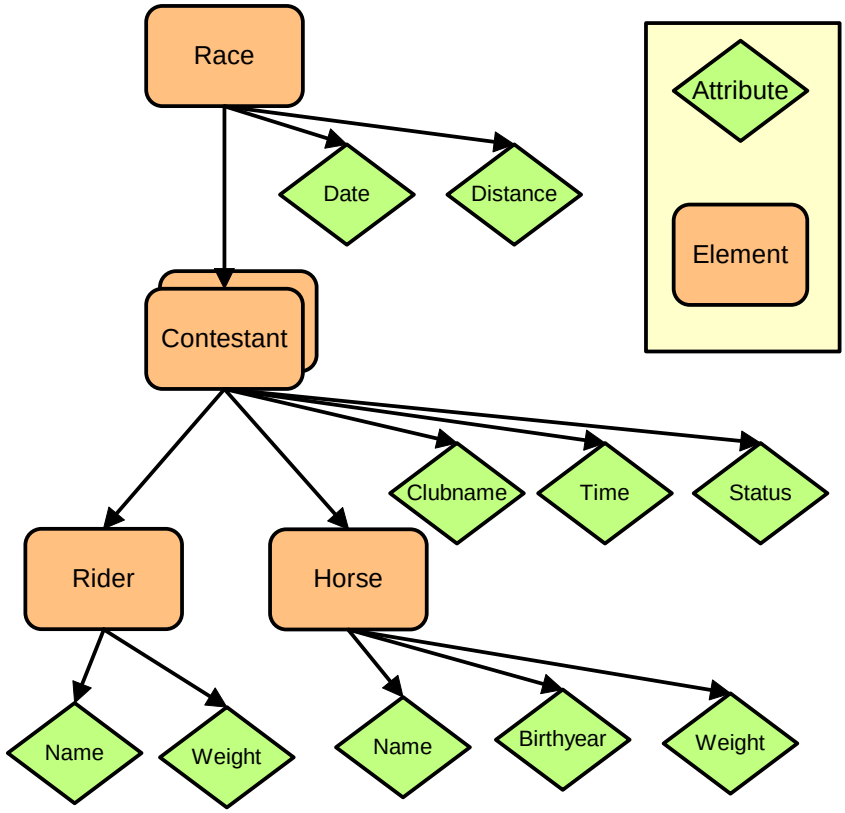


Figure 4 Structure of elements and attributes for the XML documents

The Race element will be the root element of the XML documents. The Race element consists of two attributes (Date, Distance) and one element (Contestant). The Contestant element can appear several times within a Race element. Each Contestant element has three attributes (Clubname, Time, Status) and two elements (Rider, Horse), which in turn have two and three attributes respectively.

An XML document with that structure wold look like this:

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE Race SYSTEM "">
<Race Date="2001-06-05" Distance="1000">
  <Contestant Clubname="Appaloosa Horse Club" Status="finished" Time="00:02:02">
    <Rider Name="Bill Spawr" Weight="48"></Rider>
    <Horse Name="Lake William" Weight="461" Birthyear="1993"></Horse>
  </Contestant>
  <Contestant Clubname="Horseriders" Status="finished" Time="00:02:02">
    <Rider Name="Warren Stute" Weight="55"></Rider>
    <Horse Name="Magellan" Weight="471" Birthyear="1995"></Horse>
  </Contestant>
  <Contestant Clubname="Wild Horse Club" Status="walkover">
    <Rider Name="Simon Bray" Weight="53"></Rider>
    <Horse Name="Spinelessjellyfish" Weight="493" Birthyear="1989"></Horse>
  </Contestant>
</Race>
```

Creating a DAD file, with the mapping for the transformation from XML data stored in the XML collection into XML documents, is a little more complicated. In the DAD file that we will create we will use SQL mapping. SQL mapping works as follows:
"SQL mapping allows simple and direct mapping from relational data to XML documents through a single SQL statement… SQL mapping is used for composition; it is not used for decomposition…The SQL_stmt maps the columns in the SELECT clause to XML elements or attributes that are used in the XML document. When defined for composing XML documents, the column names in the SQL statement's SELECT clause are used to define the value of an attribute_node or a content of text_node. The FROM clause defines the tables containing the data; the WHERE clause specifies the join and search condition." (XML Extender Administration and Programming).
In addition to that, the SQL statement must contain an ORDER BY clause, where the columns that identify the rows uniquely must be listed. The column names listed in the SELECT clause must be unique, if two columns have the same name then one of them must be renamed using the AS statement (example: SELECT address, address AS address2 …).

Before we start with the structure we defined above, let's look at a simpler case!

Here is a simple example of a valid SQL statement:

```
SELECT cname, address FROM club ORDER BY cname
```

Cname is the primary key of the club table, therefore it appears in the ORDER BY clause.

It is then possible to place the values of the columns into elements or attributes of the XML document. Here is how it's done:

To get an element Club we define (in the DAD file) the following tag:

In the DAD file:	Will produce in the XML document:
`<element_node name="Club">`	`<Club>`
`</element_node>`	`</Club>`

To get an attribute address in the Club element:

In the DAD file:

```
<element_node name="Club">
  <attribute_node name="address">
  </attribute_node>
</element_node>
```

Will produce in the XML document:

```
<Club address: "">
</Club>
```

To add a value to the address attribute from the SQL statement:

In the DAD file:

```
<element_node name="Club">
  <attribute_node name="address">
    <column name="address"/>
  </attribute_node>
</element_node>
```

Will produce in the XML document:

```
<Club address: "my address">
</Club>
```

To add a value to the Club element from the SQL statement:

In the DAD file:

```
<element_node name="Club">
  <attribute_node name="address">
    <column name="address"/>
  </attribute_node>
  <text_node>
    <column name="cname"/>
  </text_node>
</element_node>
```

Will produce in the XML document:

```
<Club address: "my address">
  My club
</Club>
```

So if we put all this (and a little more) together, we should have a DAD file:

First we start with two XML lines. DAD files are also XML files, that follow the rules specified in a DTD file (dad.dtd).

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
```

The DAD element is the root element of any DAD file.

```
<DAD>
```

Validation is applicable only when the DAD file is used for decomposition, therefore we set it to NO.

```
<validation>NO</validation>
```

The Xcollection element is where all our code is placed.

```
<Xcollection>
```

The SQL statement is placed within an element called SQL_stmt

```
<SQL_stmt> SELECT cname, address FROM club ORDER BY cname </SQL_stmt>
```

These lines make sure that the resulting XML document contains standard XML lines. The DOCTYPE has to always match the root element of the XML document, therefore we set it to Club.

```
<prolog>?xml version="1.0"?</prolog>
<doctype>!DOCTYPE Club SYSTEM ""</doctype>
```

This is to define the root element of the resulting XML document

```
<root_node>
```

This is the structure of elements and attributes that we have defined

```
<element_node name="Club">
  <attribute_node name="address">
    <column name="address"/>
  </attribute_node>
  <text_node>
    <column name="cname"/>
  </text_node>
</element_node>
```

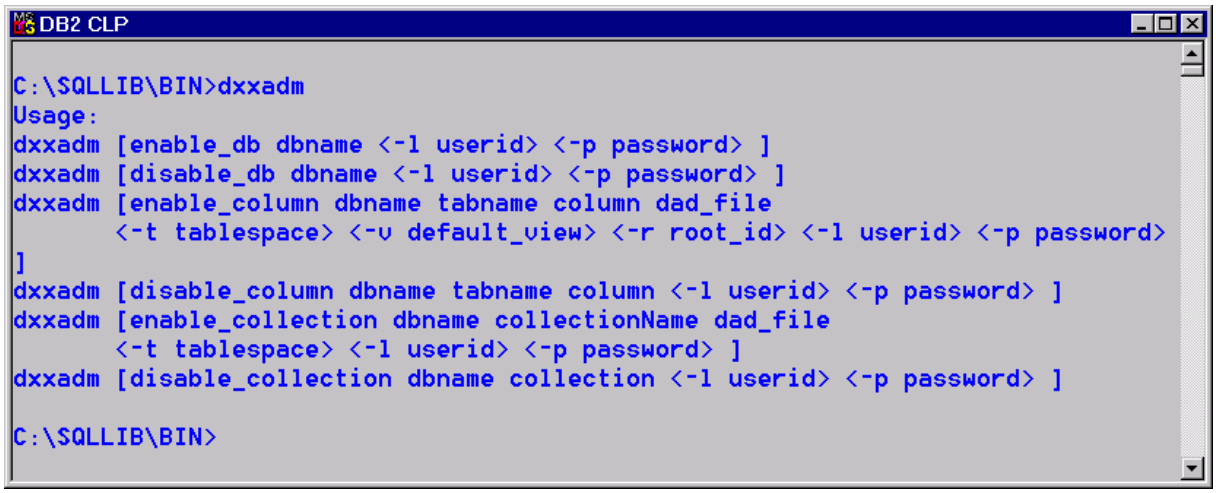
These are the end tags of all the elements

```
</root_node>
</Xcollection>
</DAD>
```

Now that the DAD file is ready we can enable the XML collection. The DAD file must be saved as a file with the extension DAD (for example as D:\temp\club.dad). In the DB2 Command Window we can execute the following command.

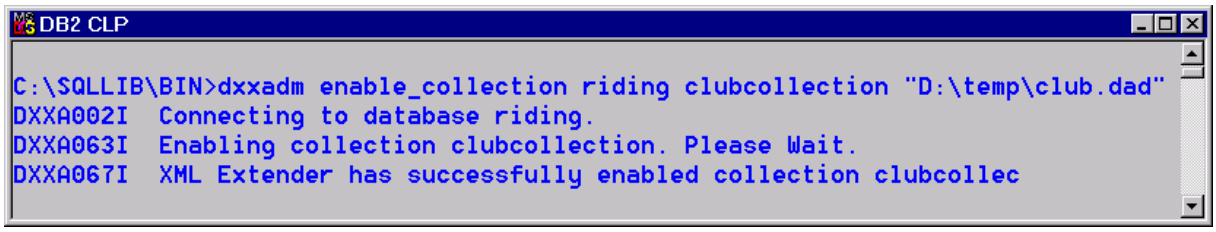
dxxadm

A response with the correct syntax of the dxxadm command comes up:



Now for the complete command that enables an XML collection:

dxxadm enable_collection riding clubcollection “D:\temp\club.dad”



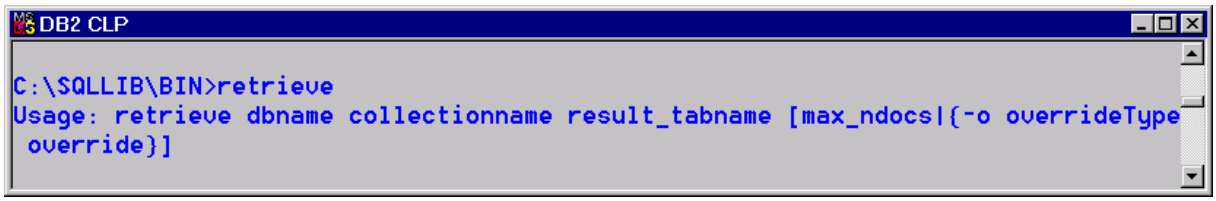
Clubcollection is the collection’s name, there can be more than one collection enabled on the same database.
D:\temp\club.dad is the location of the DAD file.

When the XML collection was enabled, a new row was created in the XML_USAGE table. The new row contains information about the XML collection (the collection name, the DAD file etc).

Note that if for some reason the DAD file needs to be altered, it is not enough to change the file. The XML Collection should be disabled (dxxadm disable command) and then enabled with the altered DAD file. It is only then that the XML Collection sees the changes!

Extracting XML documents can be done with the retrieve command. Try to execute the following command in the Command Window to get more information about the retrieve command:

retrieve



The retrieve command requires a result_tablename argument. It is this table that the XML document(s) are going to be stored in. Before we can execute the retrieve command successfully, we have to define a new table to receive the results. Here is a table definition:

CREATE TABLE results(xmldoc db2xml.XMLVARCHAR)

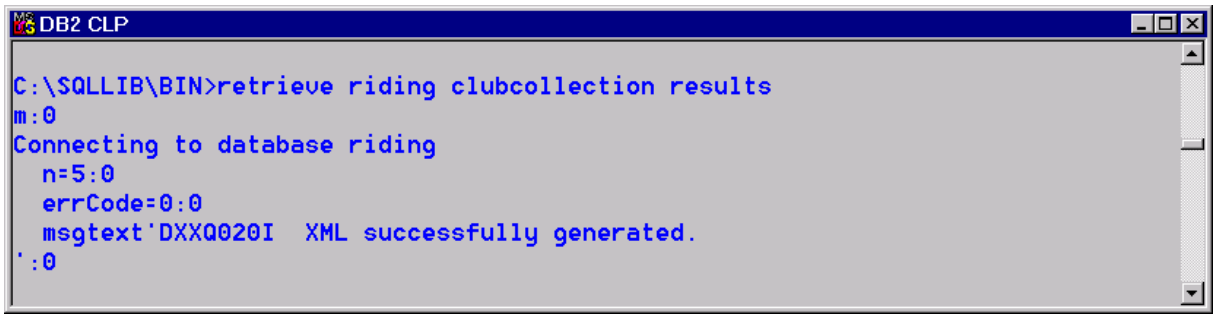
db2xml.XMLVARCHAR is a user defined type that comes with XML extender. This type is similar to VARCHAR. We use this type because it is compatible with XML extender user defined functions that we will use later.

- Create a table according to the definition above! You may need to connect to the database first with the command connect to riding.

When this table has been created, it can be used as a result table for the retrieve command.

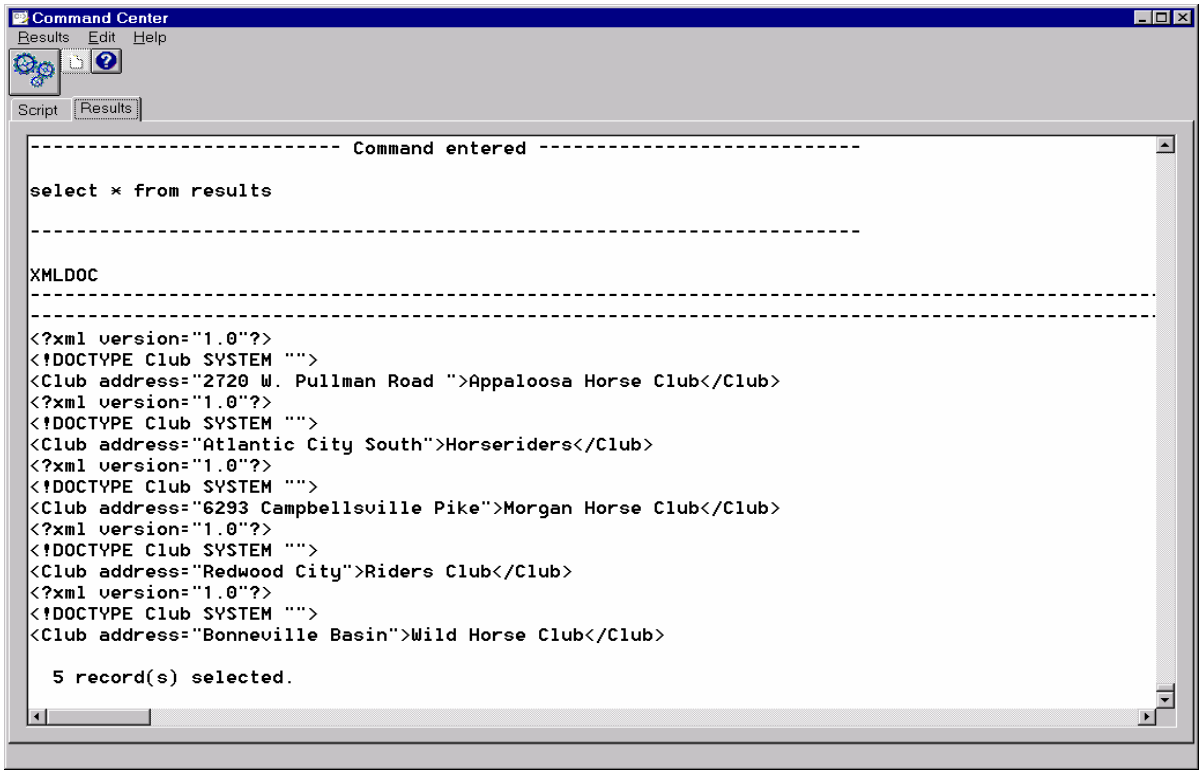
Here is the complete retrieve command:

retrieve riding clubcollection results



The XML documents that have been composed should be stored in the results table. You can easily check the contents of the results table by executing the following SQL statement:

SELECT * FROM results



Let's go back now to the more complicated structure (from page 10), and create a DAD file.

First we must have an SQL statement that returns all the columns that we need for the XML elements and attributes. The following SQL statement returns those columns:

```
SELECT date(race.racetime) as racedate, race.distance, clubname, finishingtime,
status, rname, r.weight AS rweight, hname, h.weight AS hweight, birthyear
FROM race, horse AS h, rider AS r, contestants AS c
WHERE race.raceid = c.raceid
AND ridername = rname
AND clubname = Memberclub
AND clubname = ownerclub
AND horsename = hname
```

This is a valid SQL statement but it is not valid as a DAD SQL statement. A DAD SQL statement requires an ORDER BY clause that should contain the column that can identify uniquely each entity. That is, one column for each entity. This facility of DB2 XML extender is quite new and it may appear to behave inconsistently. Not all entities' identifiers need to be part of the ORDER BY clause, only the ones that lead to a level where many elements of the same type can appear. To make that more understandable we can look at our structure and the entities that exist:

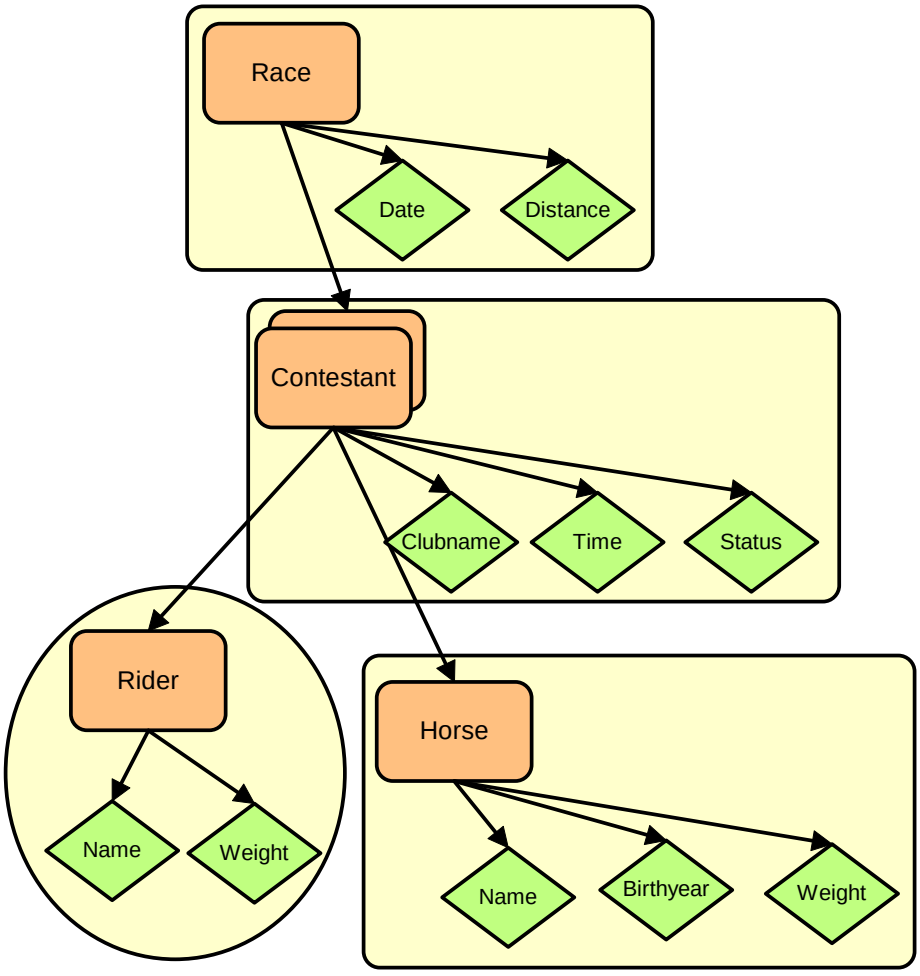


Figure 5 Entities of the XML structure

In this structure each entity is associated with one table. So the unique identifier for each entity is the primary key (or a candidate key) of the associated table. Now there is one problem remaining. There can only be one column that identifies uniquely an entity, but the tables contestant, rider and horse require more than one column to identify a row uniquely. (Of course we only need to include the unique identifier of the tables race and contestant. The tables rider and horse produce only one entry per contestant, while there can be several contestants per race.) One way to solve this problem is to use the table expression and the generate_unique() function to produce a single column unique identifier[5]. After making all these changes in the SQL statement, it should look like this:

```
SELECT race.raceid, date(race.racetime) as racedate, race.distance, c.cid,
clubname, finishingtime, status, rid, rname, r.weight AS rweight, hname, h.weight AS
hweight, birthyear
FROM race, horse AS h, rider AS r, table(SELECT generate_unique() as cid, raceid,
ridername, clubname, horsename, finishingtime, status FROM contestants) AS c
WHERE race.raceid = c.raceid
AND ridername = rname
AND clubname = Memberclub
AND clubname = ownerclub
AND horsename = hname
ORDER BY raceid, cid
```

Creating the element and attribute structure of the XML document is not different from before.

We start with the root element and we continue deeper into the structure.

The root element is the Race element.

Definition in DAD file	Produces in XML document
<element_node name="Race"> </element_node>	<Race> </Race>

Now for the attributes of the Race element.

Definition in DAD file	Produces in XML document
<element_node name="Race"> <attribute_node name="Date"> </attribute_node> <attribute_node name="Distance"> </attribute_node> </element_node>	<Race Date="" Distance=""> </Race>

[5] **In certain cases this technique may not work. In those cases we may need to create a unique identifier for an entity in a different way, for example by concatenating the components of the primary key.**

Now for the Contestant element which can exist several times within a Race element.

Definition in DAD file

```
<element_node name="Race">
  <attribute_node name="Date">
  </attribute_node>
  <attribute_node name="Distance">
  </attribute_node>
  <element_node name="Contestant"
multi_occurrence=”YES”>
  </element_node>
</element_node>
```

Produces in XML document

```
<Race Date="" Distance="">
  <Contestant>
  </Contestant>
  <Contestant>
  </Contestant>
  .
  .
  .
</Race>
```

After adding the rest of the elements and attributes of the structure we should have the following:

```
<element_node name="Race">
  <attribute_node name="Date">
  </attribute_node>
  <attribute_node name="Distance">
  </attribute_node>
  <element_node name="Contestant" multi_occurrence="YES">
    <attribute_node name="Clubname">
    </attribute_node>
    <attribute_node name="Status">
    </attribute_node>
    <attribute_node name="Time">
    </attribute_node>
    <element_node name="Rider">
      <attribute_node name="Name">
      </attribute_node>
      <attribute_node name="Weight">
      </attribute_node>
    </element_node>
    <element_node name="Horse">
      <attribute_node name="Name">
      </attribute_node>
      <attribute_node name="Weight">
      </attribute_node>
      <attribute_node name="Birthyear">
      </attribute_node>
    </element_node>
  </element_node>
</element_node>
```

The last thing to do is to place the values from the SQL statement in the structure. It is important that the order that the values appear in the SQL statement is the same with the order

that they appear in the XML structure (even though there can be columns in the SQL statement that do not appear in the XML structure). When that is done, all the parts of the DAD file are done. By putting them together (and changing the XML declaration and the DOCTYPE element of the resulting XML document) we should get this:

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
<validation>NO</validation>
<Xcollection>
<SQL_stmt>
SELECT race.raceid, date(race.racetime) as racedate, race.distance, cid, clubname, status, finishingtime, rname, r.weight AS rweight, hname, h.weight AS hweight, birthyear FROM race, table(SELECT generate_unique() as cid, raceid, ridername, clubname, horsename, finishingtime, status FROM contestants) AS c, rider AS r, horse AS h WHERE race.raceid = c.raceid AND ridername = rname AND clubname = memberclub AND clubname = ownerclub AND horsename = hname ORDER BY raceid, cid
</SQL_stmt>
<prolog>?xml version="1.0" standalone="no"?</prolog>
<doctype>!DOCTYPE Race SYSTEM "d:\temp\race.dtd"</doctype>
<root_node>
<element_node name="Race">
  <attribute_node name="Date">
    <column name="racedate"/>
  </attribute_node>
  <attribute_node name="Distance">
    <column name="distance"/>
  </attribute_node>
  <element_node name="Contestant" multi_occurrence="YES">
    <attribute_node name="Clubname">
      <column name="clubname"/>
    </attribute_node>
    <attribute_node name="Status">
      <column name="status"/>
    </attribute_node>
    <attribute_node name="Time">
      <column name="finishingtime"/>
    </attribute_node>
    <element_node name="Rider">
      <attribute_node name="Name">
        <column name="rname"/>
      </attribute_node>
      <attribute_node name="Weight">
        <column name="rweight"/>
      </attribute_node>
    </element_node>
    <element_node name="Horse">
```

```
      <attribute_node name="Name">
        <column name="hname"/>
      </attribute_node>
      <attribute_node name="Weight">
        <column name="hweight"/>
      </attribute_node>
      <attribute_node name="Birthyear">
        <column name="birthyear"/>
      </attribute_node>
    </element_node>
  </element_node>
</element_node>
</root_node>
</Xcollection>
</DAD>
```

The DAD file contains information about the XML declaration and the DOCTYPE element of the XML documents to be composed. This information is the following:

The XML document is composed according to XML version 1.0 and it is not standalone (it is associated to a DTD file):

<prolog>?xml version="1.0" standalone="no"?</prolog>

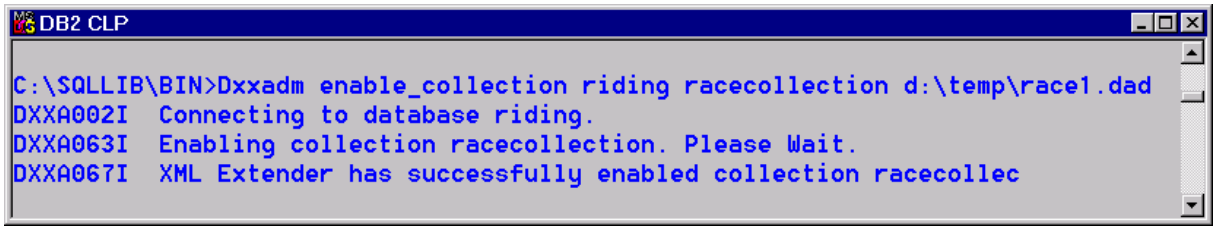
The DOCTYPE of the XML document is Race. That means that the root element of the XML document is an element called Race. The SYSTEM specifies that the XML document is supposed to follow the rules in the DTD file d:\temp\race.dtd:

<doctype>!DOCTYPE Race SYSTEM "d:\temp\race.dtd"</doctype>

The file d:\temp\race.dtd does not exist yet. In chapter 4.1.4 we will create this DTD file and we will use the XML documents composed with this DAD file.

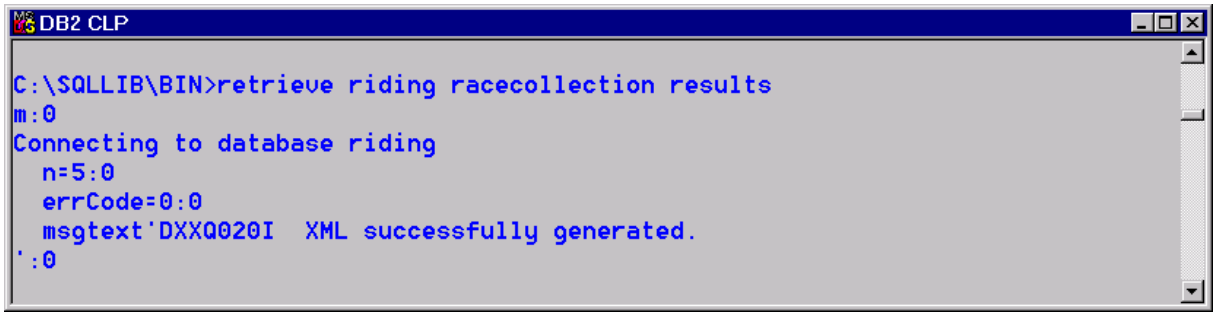
Assuming that the DAD file has been saved as d:\temp\race1.dad we can enable an XML collection called racecollection by submitting the following command in the Command Window:

dxxadm enable_collection riding racecollection d:\temp\race1.dad



When the new XML collection has been enabled use the retrieve command to compose XML documents and place them in the results table (You may want to remove the previous XML documents from the results table first) :

```
retrieve riding racecollection results
```



The XML documents are now stored in the table results.

4.1.3 Extract XML documents into XML files

So far we have composed XML documents and stored them in a table. It can be desired to extract these XML documents from the database and keep them as separate files. To do that we will use the XML extender's Content function.

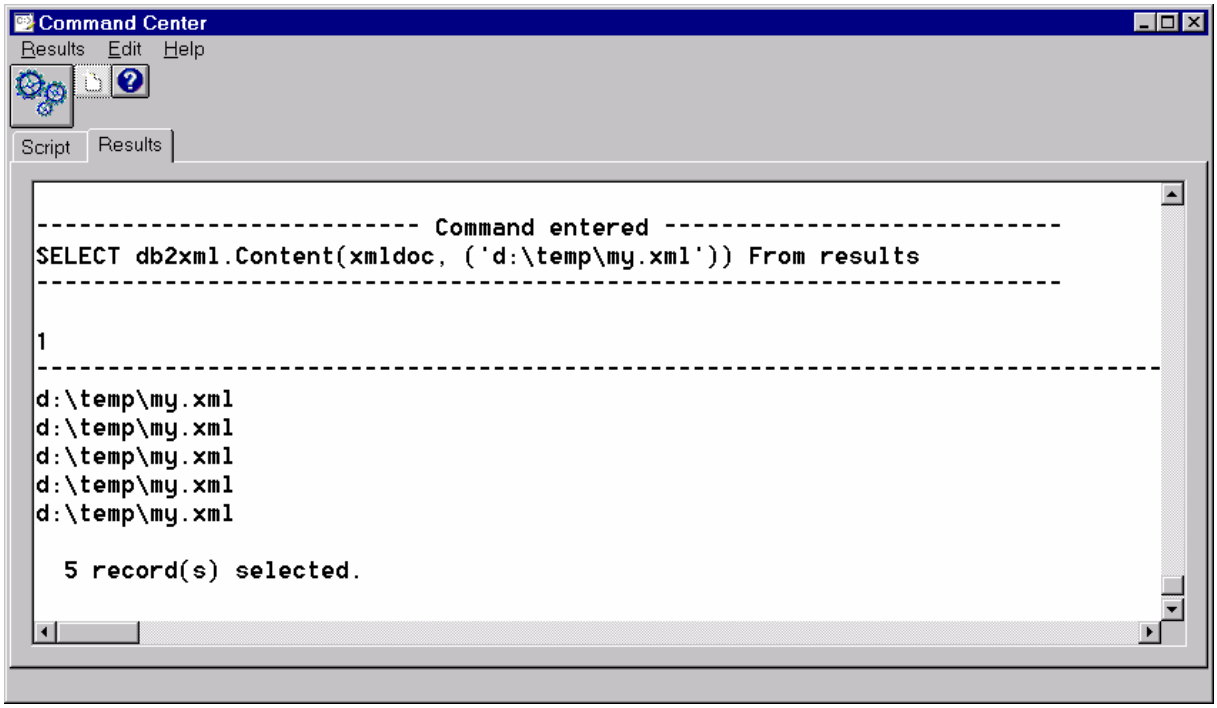
Like all other functions, the Content function can be used in a SELECT statement. The Content function has three different sets of parameters. The one that we will use is the following:

Content(xmlobj, filename)
xmlobj is the XML document as an XMLVARCHAR.
Filename is a string with the fully qualified filename and location of the file where the XML document will be saved.
When this function is executed it returns the filename to where the XML document was saved.

Here is an example of how to use this function:

```
SELECT db2xml.Content(xmldoc, 'd:\temp\my.xml') From results
```

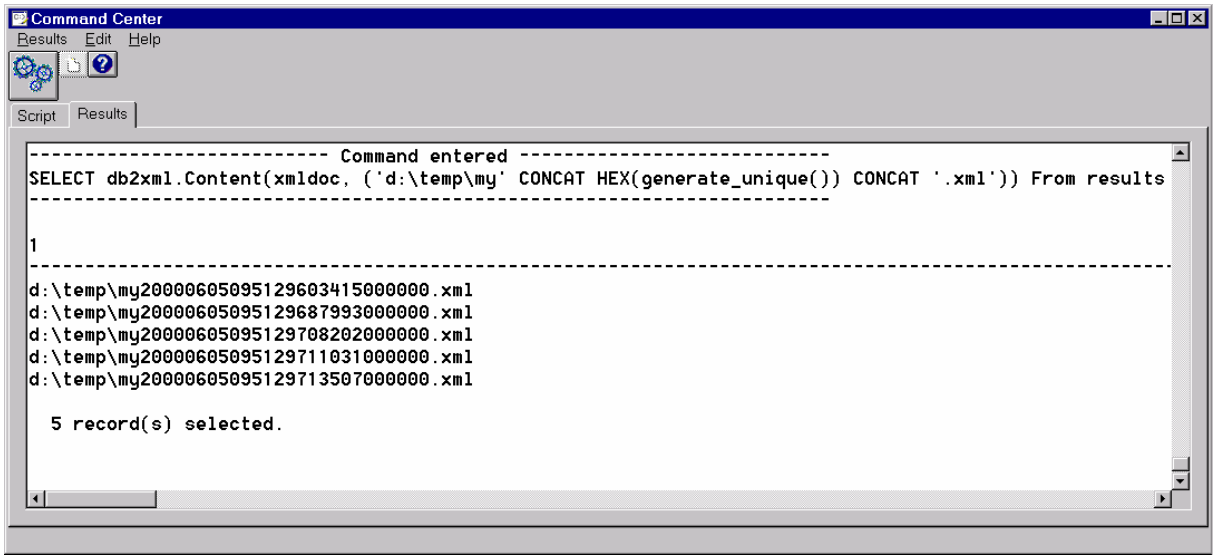
This command produces a file called d:\temp\my.xml which contains the XML document that is stored in the xmldoc column of the results table. The problem with this command is that it tries to save each and every XML document from the xmldoc column as a file called d:\temp\my.xml. Consequently only the last XML document gets saved. The next figure shows what this command returns:



An easy way to produce unique names for all the XML files saved, is to use the generate_unique() function to produce the filename:

```
SELECT db2xml.Content(xmldoc, ('d:\temp\my' CONCAT HEX(generate_unique()) CONCAT '.xml')) From results
```

This command will produce a unique key for every row in the results table, and then concatenate a hexadecimal representation of that unique key into the filename. The next figure shows a result of this command:



4.1.4 Store XML documents in an XML column

In this section we will create an XML column and store in it the XML documents that we generated before. To accomplish that we have to do the following:

1. Create a database with a table where the XML documents will be stored
2. Enable the database for XML
3. Prepare a DTD for controlling the incoming XML documents
4. Store the DTD in the DTD_REF table
5. Prepare a DAD file for the XML column
6. Enable the XML column
7. Insert XML documents into the XML column

- Start by creating a database! You will need to disconnect from the other database if you are still connected. Use the command disconnect riding.
  Here is a command that creates a database:

  ```
  CREATE DATABASE myxmlcol
  ```

The database is now ready to be enabled for XML.

- Enable the database for XML by issuing the following command in the Command Window:

  ```
  dxxadm enable_db myxmlcol
  ```

- Connect to the new database and create a table for the XML documents! The table should have a column of one of the three XML extender types (XMLVARCHAR, XMLCOLB, XMLFILE). Here we use XMLVARCHAR.

  ```
  CONNECT TO myxmlcol
  CREATE TABLE xmlcol (xmldoc DB2XML.XMLVARCHAR)
  ```

  Note that this table can contain many other columns. Those columns do not interfere with the XML column.

When an XML document is inserted into the database, it has to be controlled. If there is no control of incoming XML documents, the database will soon become corrupt. To control an XML document we need a set of rules of what is and is not allowed. Those rules can be defined in a DTD file.

Before defining a DTD, we must know the exact structure of the XML documents that we want the DTD file to control (and accept). The XML documents that we want to insert into the XML column, are the ones we created earlier from the XML data in the XML collection. So the structure is already defined.

Now let's create a DTD file to represent that structure.

First we have a Race element.

```
<!ELEMENT Race>
```

The Race element has a sub-element called Contestant, that can occur zero or more times (denote this with an asterisk after the element name).

```
<!ELEMENT Race (Contestant*)>
```

The Race element has two attributes (Date and Distance).

```
<!ELEMENT Race (Contestant*)>
<!ATTLIST Race
   Date CDATA #REQUIRED
   Distance CDATA #REQUIRED>
```

We continue with the Contestant element.

```
<!ELEMENT Contestant>
```

The Contestant element has two sub-elements called Rider and Horse, that can occur once and only once within a Contestant element.

```
<!ELEMENT Contestant (Rider, Horse)>
```

The Contestant element has three attributes (Clubname, Status and Time) The first two have to be there, the third can be missing. Status can only be one of four predefined values: finished, walkover, disqualified and dropout.

```
<!ELEMENT Contestant (Rider, Horse)>
<!ATTLIST Contestant
   Clubname CDATA #REQUIRED
   Status (finished | walkover | disqualified
| dropout) #REQUIRED
   Time CDATA #IMPLIED>
```

The Rider element. The Rider element has no content.

```
<!ELEMENT Rider EMPTY>
```

The Rider element has two attributes (Name and Weight). Name is required, Weight is not.

```
<!ELEMENT Rider EMPTY>
<!ATTLIST Rider
   Name CDATA #REQUIRED
   Weight CDATA #IMPLIED>
```

The Horse element. The Horse element has no content

```
<!ELEMENT Horse EMPTY>
```

The Horse element has three attributes (Name, Weight and Birthyear). Only Name is required

```
<!ELEMENT Horse EMPTY>
<!ATTLIST Horse
   Name CDATA #REQUIRED
   Weight CDATA #IMPLIED
   Birthyear CDATA #IMPLIED>
```

- Put all the elements together and save the file, for example as d:\temp\Race.dtd

Here is the content of the file Race.dtd:

```
<!ELEMENT Race (Contestant*)>
<!ATTLIST Race
   Date CDATA #REQUIRED
   Distance CDATA #REQUIRED>
<!ELEMENT Contestant (Rider, Horse)>
<!ATTLIST Contestant
   Clubname CDATA #REQUIRED
   Status (finished | walkover | disqualified | dropout) #REQUIRED
   Time CDATA #IMPLIED>
<!ELEMENT Rider EMPTY>
<!ATTLIST Rider
   Name CDATA #REQUIRED
   Weight CDATA #IMPLIED>
<!ELEMENT Horse EMPTY>
<!ATTLIST Horse
   Name CDATA #REQUIRED
   Weight CDATA #IMPLIED
   Birthyear CDATA #IMPLIED>
```

Now we can insert the DTD file into the DTD_REF table (which was created when we enabled the database for XML).

Execute the following INSERT statement, to insert the DTD file into the DTD_REF table of the database:

```
INSERT INTO db2xml.DTD_REF VALUES ('d:\temp\Race.dtd',
db2xml.XMLClobFromFile('d:\temp\Race.dtd'), 0, 'userX', 'userZ', 'userY')
```

The first value specifies a name for the inserted DTD file, this is also the primary key of the DTD_REF table. It is usual to set the fully qualified name of the file as this value.
The second value is the DTD file itself. This value has to be of XMLCLOB type, hence we use the XML extender's function XMLClobFromFile to import the DTD file into an XMLCLOB.
The third value (called USAGE_COUNT) shows how many DAD files refer to this DTD file. It has to always be set to 0 when a DTD file is first being inserted.
The rest of the parameters are optional and specify the following: AUTHOR, CREATOR, UPDATOR.

When a DTD file has been inserted into the DTD_REF table, it can be referenced by DAD files associated with XML columns or XML collections in the database in question.

Note that as with DAD files, if the DTD file has to be altered then it is not enough to change the file. The row for the old DTD has to first be removed from the DTD_REF table. If the DTD is in use then XML Column or Collection that is using it has to first be disabled. It is always possible to see if a DTD in the DTD_REF table is in use by checking the usage_count value for a specific DTD.

We can now define the DAD file for the XML column. The DAD file will contain a reference to the DTD file and information about the side tables. It is not important to have side tables but we will use one side table to illustrate how this feature works. We will have a side table with two columns: Date and Distance.

The DAD file starts, as before, with the following lines:

```
<?xml version="1.0"?>
<!DOCTYPE DAD SYSTEM "c:\dxx\dtd\dad.dtd">
<DAD>
```

Then we have an element called dtdid, where we define the DTD to be used to control incoming XML documents:

```
<dtdid>d:\temp\Race.dtd</dtdid>
```

Then the validation element, in this case we set the validation to YES. This activates the control of the incoming XML documents:

```
<validation>YES</validation>
```

Now we have the Xcolumn element:

```
<Xcolumn>
```

Within this element we can specify the side tables (in this case only one side table), and the mapping between elements or attributes and the columns of the side tables. In this way the side tables will be automatically updated every time a new XML documents is inserted. Here is the content of the Xcolumn element:

A table element with a name attribute. That is the name of the side table.

```
<table name="race_st">
```

A column element for each column of the side table. The name attribute indicates the name of the column, the type attribute indicates the data-type of the column, the path attribute indicates where in the XML document's structure to get the value from, the multi_occurrence attribute indicates whether or not the specified path can appear many times within an XML document. (Note that an empty element can be closed with a "/" in the end of the opening tag)

```
<column name="Racedate"
        type="date"
        path="/Race/@Date"
        multi_occurrence="NO"/>
<column name="Racedistance"
        type="integer"
        path="/Race/@Distance"
        multi_occurrence="NO"/>
```

And the closing tag of the table element.

```
</table>
```

And of course the closing tags of the Xcolumn element and the DAD element:

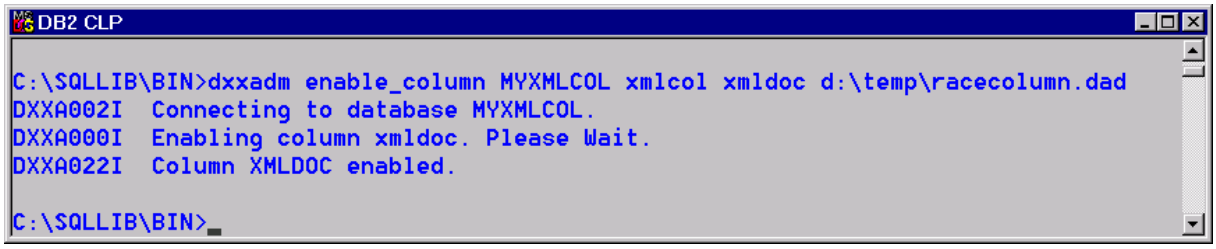
```
</Xcolumn>
</DAD>
```

- Now save the DAD file (for example d:\temp\racecolumn.dad)!

- Enable the XML column! Here is the command:

```
dxxadm enable_column myxmlcol xmlcol xmldoc d:\temp\racecolumn.dad
```

where myxmlcol is the database name, xmlcol is the name of the table and xmldoc is the name of the column in the table.



Now that the XML column has been enabled, we can insert XML documents into it. To insert an XML document we can execute an INSERT statement. When inserting an XML document into a column of a table, we must always think of the data type of the column. The column, to which we will insert the XML documents is of the following type: DB2XML.XMLVARCHAR. Fortunately, there is a set of functions for transforming XML documents to and from all the different XML data types. One of those functions is this: DB2XML.XMLVarcharFromFile(). This function takes one argument: the full filename as a string and returns the content of that file (the XML document) as an DB2XML.XMLVARCHAR. Here is an example of an INSERT statement:

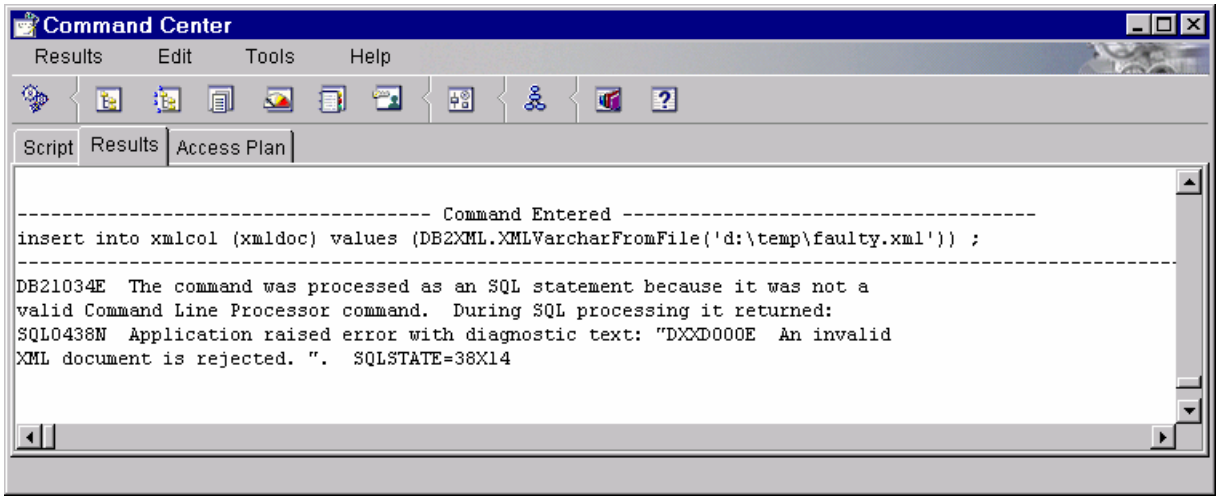
```
INSERT INTO xmlcol (xmldoc) VALUES
(DB2XML.XMLVarcharFromFile('d:\temp\my20000603132654013484000000.xml'))
```

The file d:\temp\my20000603132654013484000000.xml is just one of the files we generated before (see chapter 4.1.3). The filenames are random, so the files that you have, have different filenames from the filenames that appear in chapter 4.1.3.

When the XML document has been inserted into the database, the side tables have also been updated. In our case there should be one new record in the race_st table.

If the XML document does not comply with the DTD file, specified in the DAD file, then it will be rejected. That can easily be tested; try to insert an XML document with the wrong type of elements or attributes.

Here is what happened when a faulty XML document was inserted into the XML column:



An XML document is rejected when:

- The element structure is not as specified in the DTD file
- The attributes of the elements are not following the rules of the DTD file
- The SYSTEM of the XML document (specified in the DOCTYPE element) is not the same as the one in the DAD file. Both should point to the same DTD file.

On the other hand an XML document that is defined as standalone (in the XML declaration) can be accepted if it does not break any of the rules above.

4.1.5 Run queries against the XML column

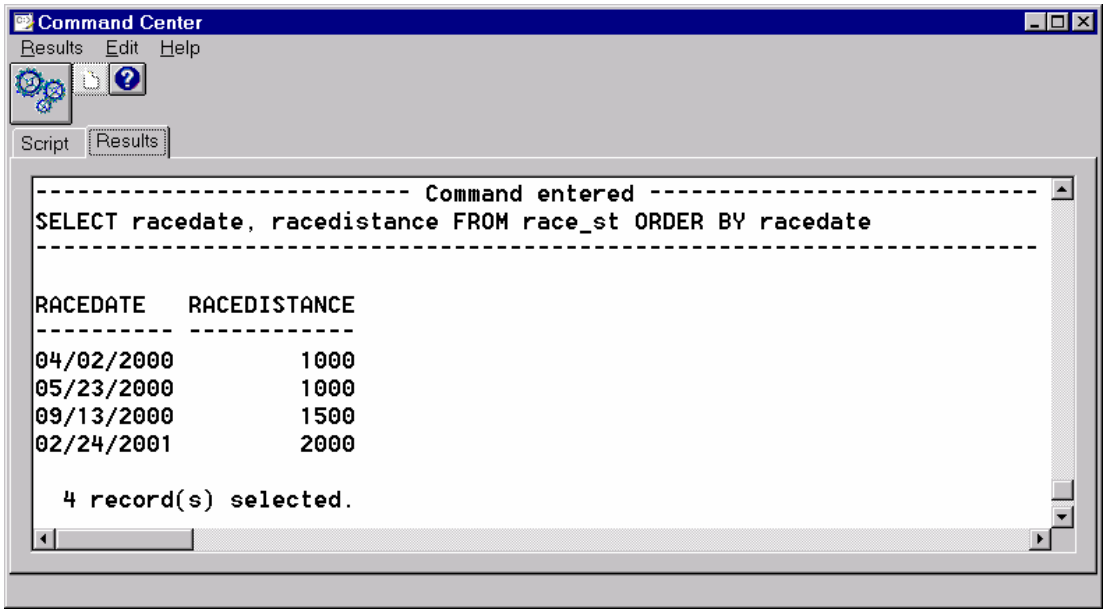
After having inserted a number of XML documents into the XML column, we can run a few queries against the XML column and its side tables.
Data on the side tables can be accessed much faster than data in the XML column. When a query requires data that have been stored in side tables then it is better to run the query against the side tables and not the XML column. The next example illustrates this situation:

Run a query that returns the date and the distance of all the races ordered by date!

Both the date and the distance are columns in the race_st side table. The following SQL statement returns the date and distance of all the races ordered by date:

```
SELECT racedate, racedistance FROM race_st ORDER BY racedate
```

Here is a possible result:



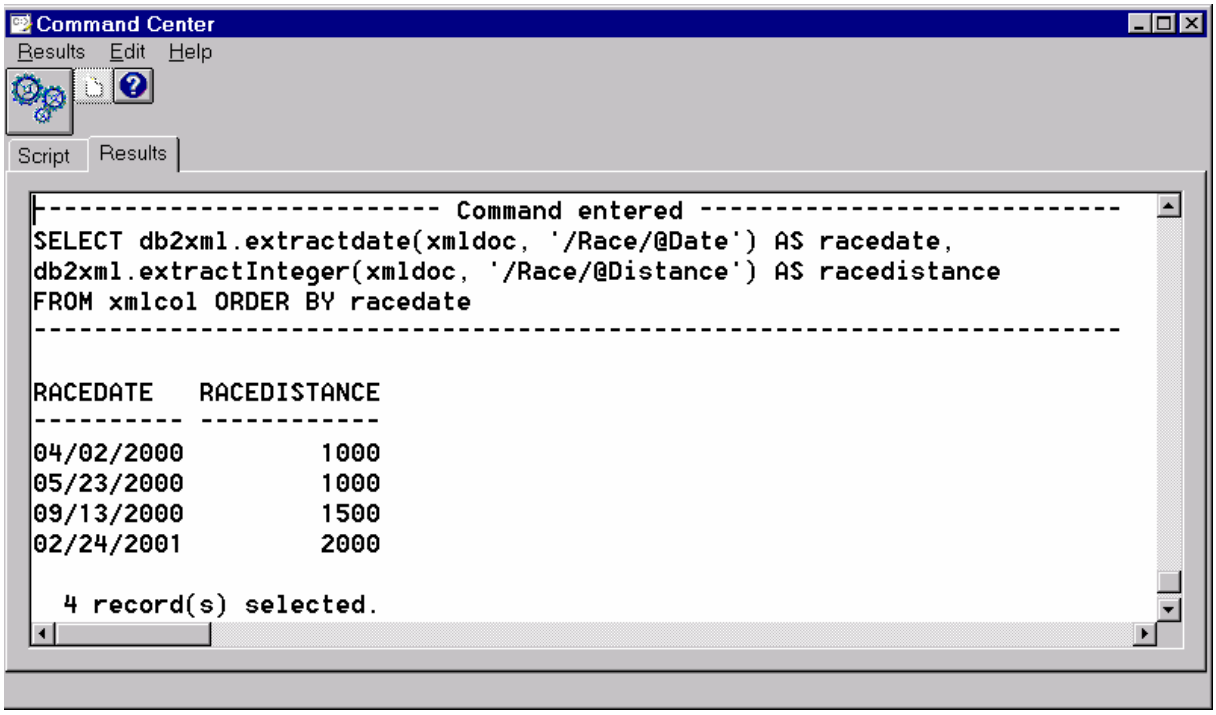
An other way to retrieve the date and distance of all the races is to extract them directly from the XML column. Here is an SQL statement that does that:

```
SELECT db2xml.extractdate(xmldoc, '/Race/@Date') AS racedate,
db2xml.extractInteger(xmldoc, '/Race/@Distance') AS racedistance
FROM xmlcol ORDER BY racedate
```

Db2xml.extractDate() and db2xml.extractInteger() are two of the functions that the XML extender provides for extracting data from XML documents. Other functions are db2xml.extractVarchar(), db2xml.extractLong(), db2xml.extractChar(), etc. All these functions have a plural version: db2xml.extractIntegers() etc. All the functions take two parameters, the first one is the XML document and the second one is the location path of the wanted element or attribute in the specified XML document. The location path is a sequence

of element names separated by a single slash (/). An attribute is denoted with an at sign (@) followed by the name of the attribute.

Here is the result of the SQL statement:



The results of both SQL statements are the same, but the execution time is much longer for the second statement.

When the requested data exist only in the XML column the db2xml.extract functions have to be used. If the value that we want to extract appears more than once in an XML document then we have to use the plural version of the db2xml.extract functions. Those functions cannot be used directly in a SELECT statement, because they return more than one value. Instead they have to be used in the FROM clause of the SELECT statement combined with the table() function. Here is an example that uses the plural function db2xml.extractVarchars():

List all the Riders (name), sorted and without any duplicates!

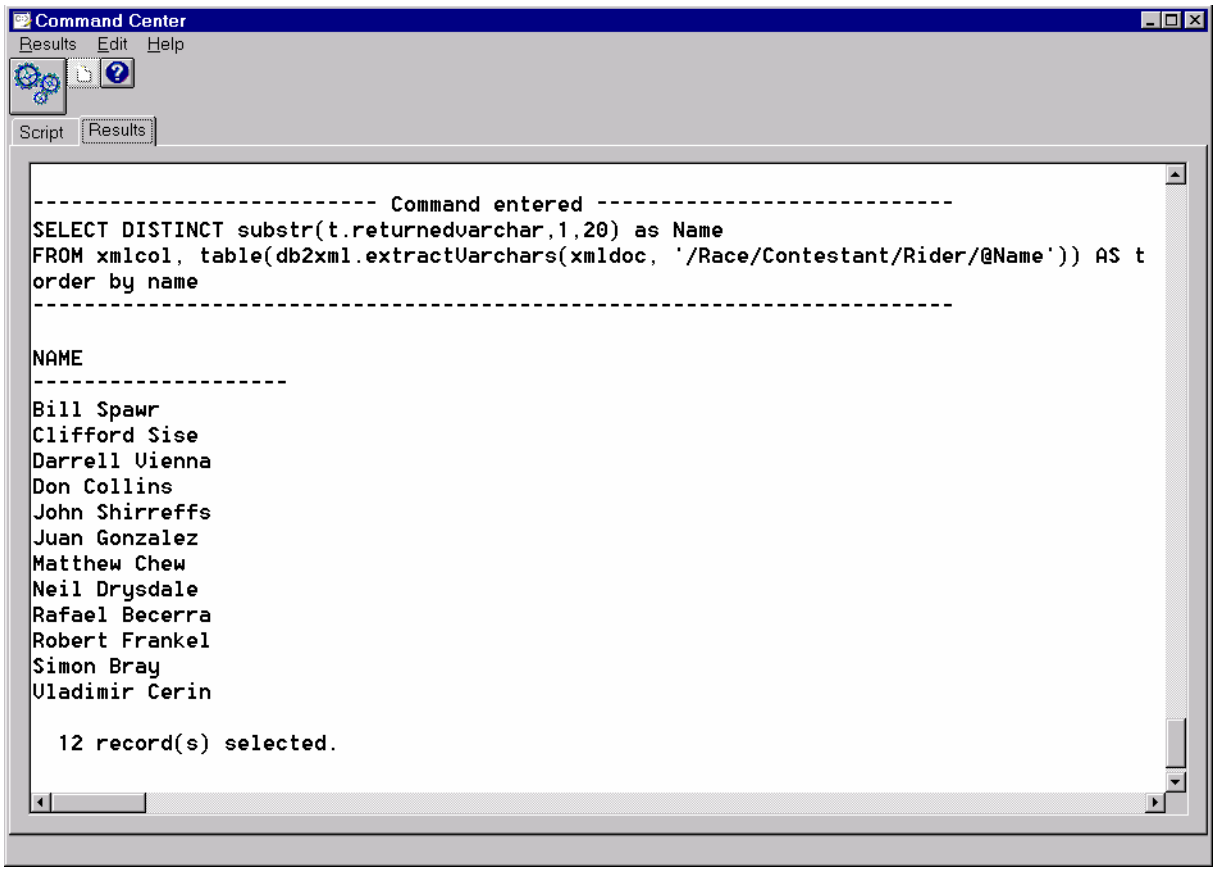
Here is an SQL statement that returns just that:

```
SELECT DISTINCT substr(t.returnedvarchar,1,20) as Name FROM xmlcol,
table(db2xml.extractVarchars(xmldoc, '/Race/Contestant/Rider/@Name')) AS t
order by name
```

In the FROM clause we have the table where the XML documents are (xmlcol) and within a table() function we call the db2xml.extractVarchars() function. In this way the returned values are transformed into a table called t. The column name for the returned values is

returnedvarchar. Similarly the db2xml.extractReals() function returns a column called returnedreal.
The substr() function returns a string 20 characters long, the returnedvarchar is by default 4000 characters long. To be able to use the DISTINCT function and the ORDER BY clause, the length of each column cannot exceed 255 characters.

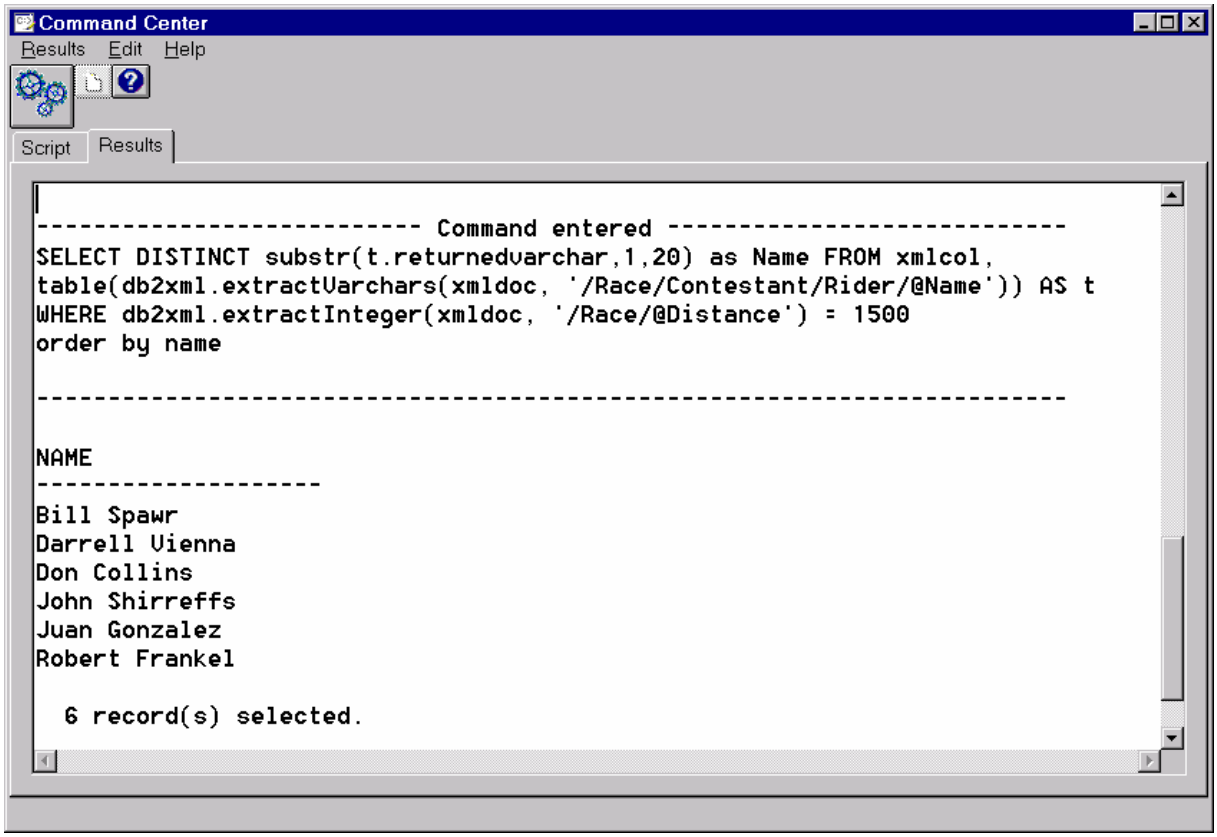
Here is the result:



All the db2xml.extract functions can also be used in the WHERE clause. We can easily modify the previous query to only show the riders that participated in races with distance equal to 1500. This is the new SQL statement:

```
SELECT DISTINCT substr(t.returnedvarchar,1,20) as Name FROM xmlcol,
table(db2xml.extractVarchars(xmldoc, '/Race/Contestant/Rider/@Name')) AS t
WHERE db2xml.extractInteger(xmldoc, '/Race/@Distance') = 1500
order by name
```

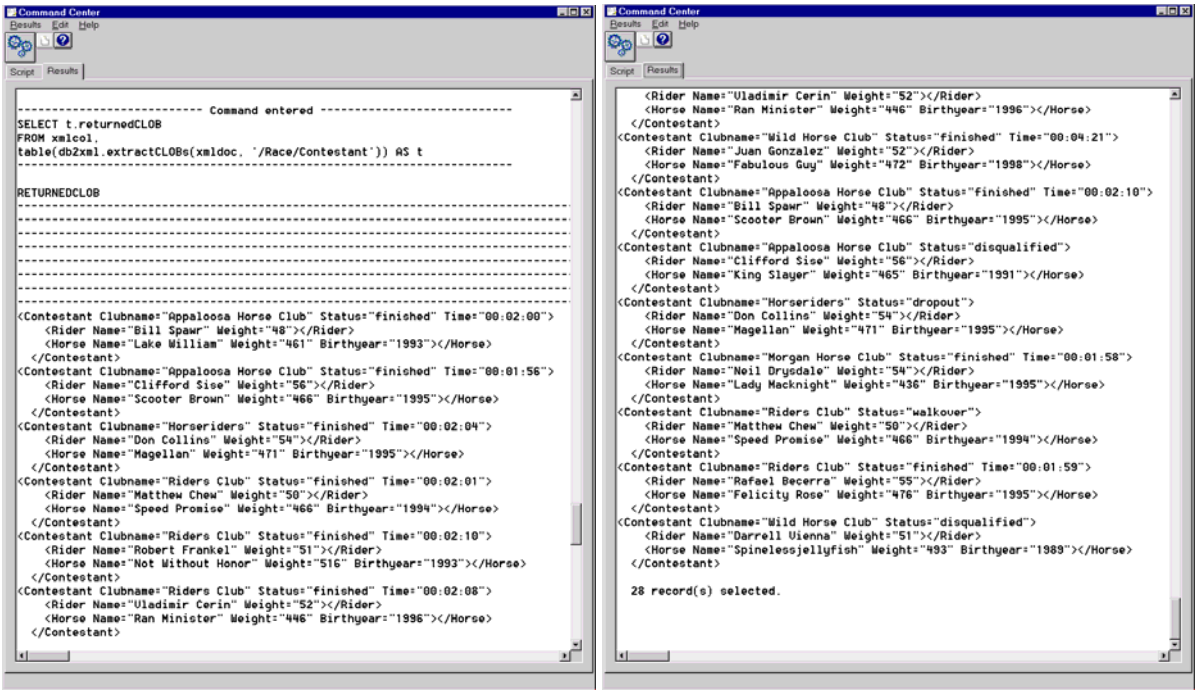
Here is the result:



A very useful function is the db2xml.extractCLOBs() function. This function can be used to extract parts of an XML document as new XML documents. To illustrate how this function works we will try to extract each Contestant element as a new XML document. Here is the SQL statement:

```
SELECT t.returnedCLOB
FROM xmlcol, table(db2xml.extractCLOBs(xmldoc, '/Race/Contestant')) AS t
```

And here is the result:



Now lets try to combine this technique with the rest of the functions. Here is a possible demand:

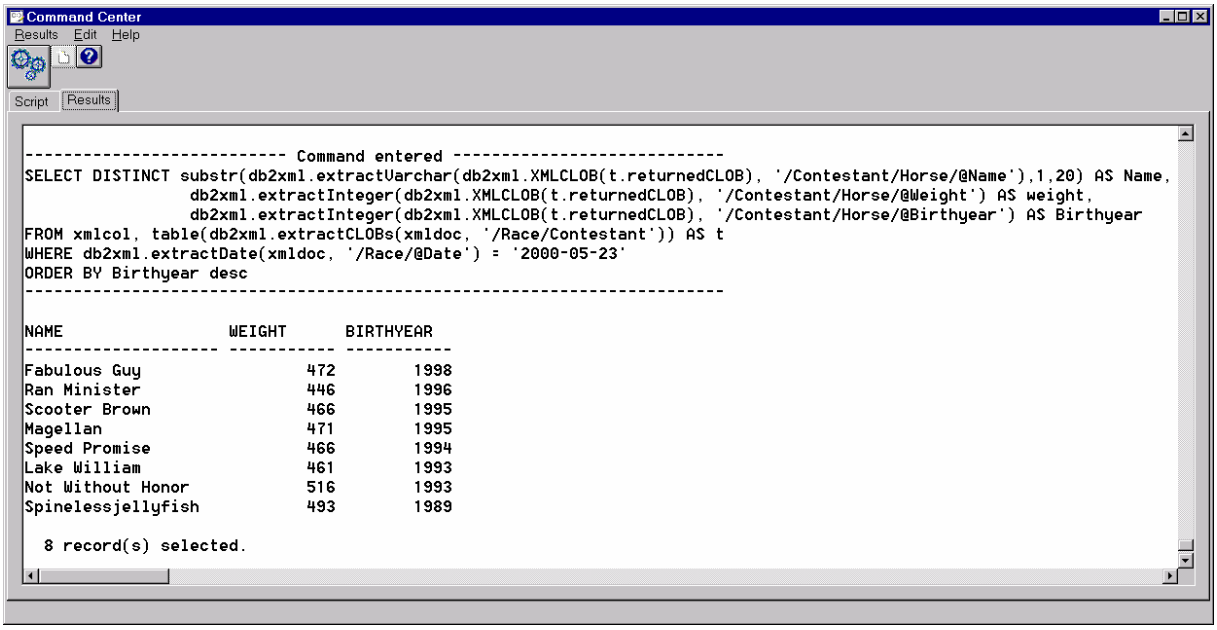
List all the horses that participated in a race on 2000-05-23. List their names, year of birth and weight. Order them according to age.

To solve this problem, we need to extract one contestant at a time and then extract the horse's information from the contestant. Here is an SQL statement that does exactly that:

```
SELECT DISTINCT
 substr(db2xml.extractVarchar(db2xml.XMLCLOB(t.returnedCLOB),
'/Contestant/Horse/@Name'),1,20) AS Name,
 db2xml.extractInteger(db2xml.XMLCLOB(t.returnedCLOB),
'/Contestant/Horse/@Weight') AS Weight,
 db2xml.extractInteger(db2xml.XMLCLOB(t.returnedCLOB),
'/Contestant/Horse/@Birthyear') AS Birthyear
FROM xmlcol, table(db2xml.extractCLOBs(xmldoc, '/Race/Contestant')) AS t
WHERE db2xml.extractDate(xmldoc, '/Race/@Date') = '2000-05-23'
ORDER BY Birthyear desc
```

The db2xml.extractCLOBs() function returns a CLOB value under the name returnedCLOB. To use this returnedCLOB in another db2xml.extract function, we have to transform it to one of the three XML datatypes (XMLVARCHAR, XMLCLOB, XMLFILE). The db2xml.XMLCLOB() function does exactly that: It takes a normal CLOB as an argument and returns it as an XMLCLOB.

Here is the result:



Here is an even more complex example:

For every contestant that finished a race of distance 1000, list the rider's name and weight, the horse's name and birth year, as well as the club name! Order the results by Club and weight of rider (heaviest first)!

```
SELECT DISTINCT Clubname,
substr(db2xml.extractvarchar(riderxml, '/Rider/@Name'),1,20) AS RiderName,
db2xml.extractinteger(riderxml, '/Rider/@Weight') AS RiderWeight,
substr(db2xml.extractvarchar(horsexml, '/Horse/@Name'),1,20) AS HorseName,
db2xml.extractinteger(horsexml, '/Horse/@Birthyear') AS HorseBirthYear
FROM (SELECT db2xml.XMLCLOB(tr.returnedCLOB) AS riderxml,
        db2xml.XMLCLOB(th.returnedCLOB) AS horsexml,
        substr(db2xml.extractvarchar(contestantxml, '/Contestant/@Clubname'),1,30) AS Clubname
        FROM (SELECT db2xml.XMLCLOB(t.returnedCLOB) AS contestantxml
                FROM xmlcol, table(db2xml.extractCLOBs(xmldoc, '/Race/Contestant')) AS t
                WHERE db2xml.extractinteger(xmldoc, '/Race/@Distance') = 1000) as contestant,
        table(db2xml.extractCLOBs(contestantxml, '/Contestant/Rider')) AS tr,
        table(db2xml.extractCLOBs(contestantxml, '/Contestant/Horse')) AS th
        WHERE db2xml.extractvarchar(contestantxml, '/Contestant/@Status') = 'finished') AS horserider
ORDER BY Clubname, RiderWeight desc
```

This works as follows:

1. Start with table xmlcol
2. extract CLOBs setting Contestant as root in a temporary table called t
3. extract an Integer corresponding to path /Race/@Distance
4. compare it to 1000
5. Select the CLOBs from t and call them contestantxml
6. Call the entire result contestant
7. Second level start with contestant as input table in FROM clause

8. extract CLOBs from column contestantxml setting Rider as root in a temporary table called tr
9. extract CLOBs from column contestantxml setting Horse as root in a temporary table called th
10. extract varchar for path /Contestant/@Status
11. compare it to 'finished'
12. Select the CLOBs from tr and call them riderxml
13. Select the CLOBs from th and call them horsexml
14. extract varchar for path /Contestant/@Clubname and cut it down to 30 characters
15. select the 30 character long varchar as Clubname
16. Call the entire result horserider
17. third level starts with horserider as source table
18. extract clubname from horserider
19. extract varchar for path /Rider/@Name and cut it down to 20 characters
20. call it RiderName
21. extract integer for path /Rider/@Weight
22. call it RiderName
23. extract varchar for path /Horse/@Name and cut it down to 20 characters
24. call it HorseName
25. extract integer for path /Horse/@Birthyear
26. call it HorseBirthYear
27. sort results on ClubName and RiderWeight
28. remove duplicates (they would occur if the same rider with the same horse had finished a race of the right distance!)

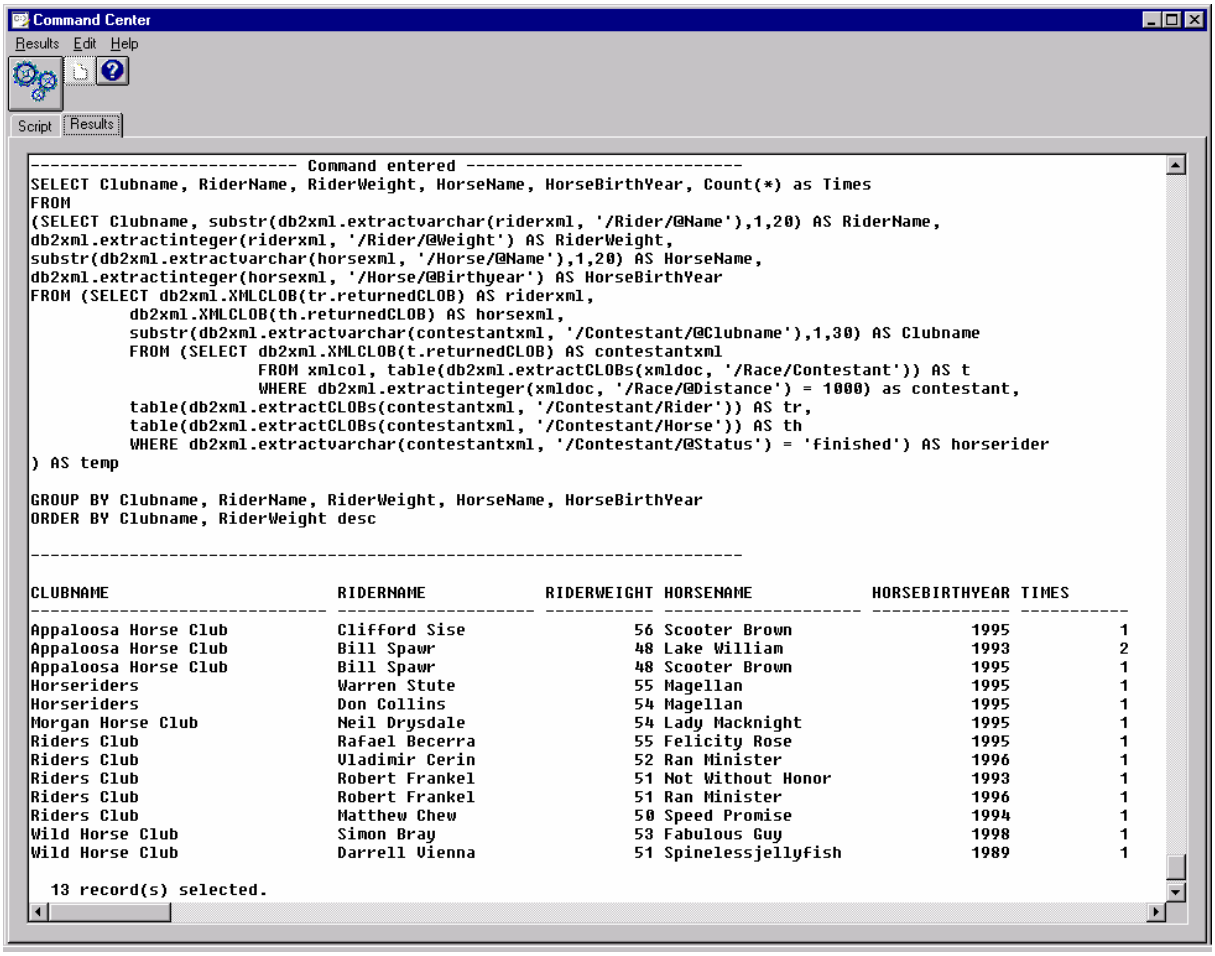
Alternatively we could add a GROUP BY clause and count the amount of times the same rider with the same horse has finished a race of the given distance. To do this we need to add one more level to the SQL statement:

```
SELECT Clubname, RiderName, RiderWeight, HorseName, HorseBirthYear, Count(*) as Times
FROM

(SELECT Clubname,
substr(db2xml.extractvarchar(riderxml, '/Rider/@Name'),1,20) AS RiderName,
db2xml.extractinteger(riderxml, '/Rider/@Weight') AS RiderWeight,
substr(db2xml.extractvarchar(horsexml, '/Horse/@Name'),1,20) AS HorseName,
db2xml.extractinteger(horsexml, '/Horse/@Birthyear') AS HorseBirthYear
FROM (SELECT db2xml.XMLCLOB(tr.returnedCLOB) AS riderxml,
        db2xml.XMLCLOB(th.returnedCLOB) AS horsexml,
        substr(db2xml.extractvarchar(contestantxml, '/Contestant/@Clubname'),1,30) AS Clubname
        FROM (SELECT db2xml.XMLCLOB(t.returnedCLOB) AS contestantxml
                FROM xmlcol, table(db2xml.extractCLOBs(xmldoc, '/Race/Contestant')) AS t
                WHERE db2xml.extractinteger(xmldoc, '/Race/@Distance') = 1000) as contestant,
        table(db2xml.extractCLOBs(contestantxml, '/Contestant/Rider')) AS tr,
        table(db2xml.extractCLOBs(contestantxml, '/Contestant/Horse')) AS th
        WHERE db2xml.extractvarchar(contestantxml, '/Contestant/@Status') = 'finished') AS horserider
) AS temp

GROUP BY Clubname, RiderName, RiderWeight, HorseName, HorseBirthYear
ORDER BY Clubname, RiderWeight desc
```

And here is the result:



4.2 More to do

Now that we have gone through DB2's facilities for XML, it is time to test what you have learned. In this section you will find a description of an XML structure and a few queries. The new XML structure is based on the same XML collection as before. What you have to do is this:

- Create XML documents according to the new XML structure (from the XML data stored in the XML collection)!
- Create a new XML column and store the new XML documents in it! (Don't forget to validate the XML documents.)
- Write SQL statements that solve the given queries! The SQL statements should extract the data from the XML documents in the XML column and not from possible side tables.

Here is the XML structure:

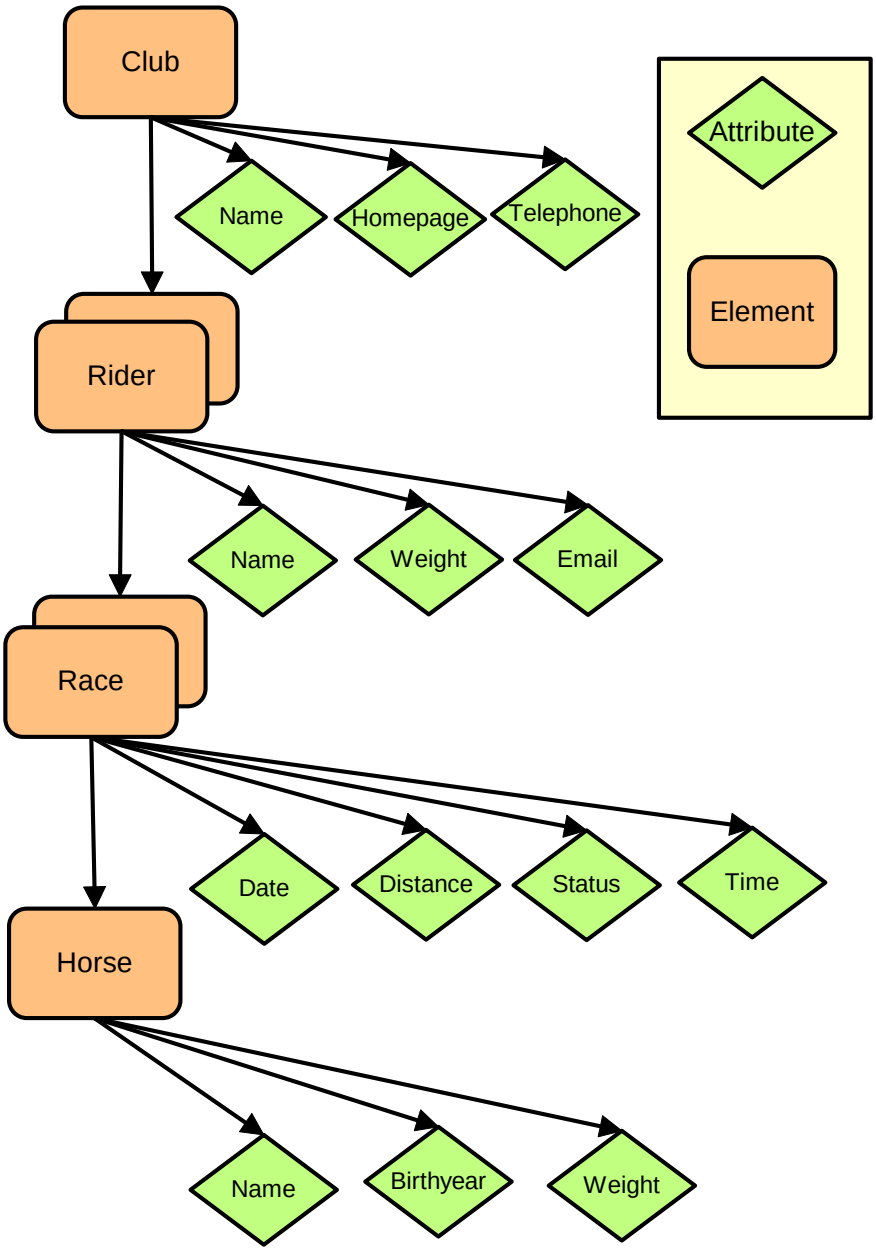


Figure 6 XML structure to be implemented

And here are a few queries:

- List all the clubs (name and telephone) in alphabetical order!
- List all the riders that are members of the club “Riders Club”!
- List the date, distance and status of all the races that Robert Frankel participates in!
- List all the riders (name, email), their club (name) and the amount of races that they have finished!
- For every rider of the club "Horseriders" list the name of the rider and the amount of different horses the rider rides!

5 Completed Lab requirements

All the exercises in chapter 4 are compulsory. For the exercises in section 4.2 you have to send in electronically (use the conference called "OBJDB Inluppar" in Firstclass) the following documents:

1. **DAD file for the composition of XML documents from the XMLcollection**
2. **DAD file for the XML column**
3. **DTD file for the XML documents.**
4. **SQL statements for all the queries, with execution results.**

The results of the exercises should also be presented to nikos in a short interactive session per group. All the members of the group should be present.

The deadline for this lab is the 12th of October 2001.

6 Internet Resources

XML & DTD Tutorials

http://L238.dsv.su.se/tutorial
http://www.w3schools.com/xml/default.asp
http://www.spiderpro.com/bu/buxmlm001.html

DB2 XML extender

http://www-4.ibm.com/software/data/db2/extenders/xmlext/

7 Epilogue

When all this is done, you should have quite a good understanding of how to use DB2 to manage XML documents and XML data.

I hope you have enjoyed this compendium. Please give me feedback!

The Author

nikos dimitrakas