

# Baze de date și XML în Perl

## Migrarea de la bazele de date relaționale la documentele XML

– Sabin Corneliu Buraga

**A**rticolul de față dorește să realizeze o introspecție în diferite maniere de prelucrare, pentru Web, a bazelor de date prin prima XML, folosindu-se scripturi CGI scrise în Perl.

Limbajul Perl poate fi folosit, mai ales prin intermediul modulelor puse la dispoziția programatorilor, pentru prelucrarea facilă a bazelor de date via modulelor și *driver*-elor SQL. După cum se știe, bazele de date relaționale sunt implementări practice, optimizate, ale modelului relațional propus de *E.F. Codd* în anii 1970, utilizând algebra relațională (cei interesați de fundamentele teoretice ale bazelor de date relaționale pot consulta cartea V.Felea, *Baze de date relaționale. Dependente*, Editura Universității „A.I. Cuza”, Iași, 1996). În limbajul Perl se pot concepe foarte ușor diverse scripturi CGI pentru a asigura, pe partea de server, conectivitatea cu serverele de baze de date (e.g. Oracle, PostgreSQL, MySQL etc.). De asemenea, există posibilitatea de a utiliza un *driver* generic ODBC destinat conectării la orice server de baze de date care respectă acest standard (în speță, unul pe platforma Windows). Înainte de a putea prelucra bazele de date în Perl, trebuie să ne asigurăm că avem instalat pe serverul Web un server de baze de date și diferite module Perl utilizate mai jos.

### Utilizarea bazelor de date relaționale

Pentru început vom utiliza vechea paradigmă, bazată de modelul client/server, în care clientul joacă un rol predeterminat, pasiv, primind datele într-un format *a-priori* stabilit de către server (via câmpul *Content-type* din antetul HTTP, de cele mai multe ori *text/html*). Fiecare aplicație poate stabili un model propriu de marcare a datelor, model bazat în ultima vreme pe XML, programatorii trebuind să se concentreze asupra marcării structurale și a validității datelor reprezentate. Astfel, datele stocate în formatul specific unui server de baze de date se vor regăsi ca documente XML, independente de platformă și de o reprezentare particulară (de cele mai multe ori, proprietară).

Vom considera, drept exemplificare, o bază de date Biblioteca având în componență trei tabele: *utilizatori*, *carti* și *imprumuturi* și unul sau mai multe scripturi CGI care rezolvă interogările (formulate într-un dialect SQL) posibile asupra datelor stocate în tabelele prezentate. Pentru toate exemplele de mai jos, am presupus că interpretorul Perl este localizat (natural, pe o mașină UNIX/Linux) în directorul */usr/bin*. Vom presupune, de asemenea, că sunt instalate unul dintre serverele de baze de date MySQL sau PostgreSQL și modulele *DBI* (*DataBase Interface*) pentru Perl.

Cele două *listing*-uri (*script.pl* și *rez.tmp1*) prezentate mai jos relevă modul în care se poate realiza independența datelor de reprezentarea lor, folosindu-se două module Perl de la *CPAN* (*Comprehensive Perl Archive Network*) *DBI* și *Text::Template*. De asemenea, pentru procesarea comodă a marcatorilor HTML și pentru conectivitatea cu serverul și clientul Web se folosește modulul Perl CGI.

În primul caz, se realizează independența relativ la baza de date (modulul *DBI* va încărca un *driver* particular serverului de baze de date ales), iar în al doilea caz relativ la reprezentarea datelor în afara bazei de

date, întrucât formatul tabelii extrase se schimbă ușor cu un nou șablon (*template*) care poate fi transmis direct ca parametru scriptului CGI. Astfel este posibil, de exemplu, să se realizeze un *feedback* din partea clientului. Mai putem observa, de asemenea, că datele de ieșire vor fi marcate în XHTML.

```
#!/usr/bin/perl

# script.pl
use strict;
use CGI;
use CGI::Carp;
use DBI;
use Text::Template;

my $q = new CGI;
$q->import_names('R');

# incercam conectarea la serverul de baze de date
my $dbh = DBI->connect('dbi:Pg:dbname=biblioteca', 'ciu',
    '')
    || die $DBI::errstr;

# ne pregatim sa trimitem comanda SQL
my $sth = $dbh->prepare(q{
    SELECT c.titlu, i.data
    FROM utilizatori u , carti c, imprumuturi i
    WHERE u.ume = ? and u.prenume = ?
        and u.cod = i.cod_utilizator
        and c.cod = i.cod_carte
});
$sth->execute($R::nume, $R::prenume) || die $DBI::errstr;

# colectam rezultatele trimise de server
my $ref;
while( defined($ref = $sth->fetchrow_arrayref) ) {
    push @R::rows, [ @$ref ];
}
# afisarea paginii XHTML
print $q->header('text/html');

# afisam datele la iesirea standard,
# folosind sablonul stocat in 'rez.tmp1'
my $t = new Text::Template (TYPE => 'FILE', SOURCE =>
    'rez.tmp1',
    PREPEND => q{use strict;use CGI qw/:standard/;});
$t->fill_in(OUTPUT => \*STDOUT);

$sth->finish;
$dbh->disconnect;
```

Urmează șablonul de afișare a datelor (memorat în fișierul rez.tpl, folosind facilitățile oferite de modulul CGI):

```
{
  $OUT .= start_html();
  my @rows;
  foreach (@R::rows) {
    push @rows, td($_);
  }

  # datele vor fi afisate intr-un tabel,
  # continutul celulelor fiind centrat
  $OUT .= table({-border=>undef},
    caption('Rezultate'),
    Tr({-align=>'center',- valign=>'top'},
      [
        th(['Titlul cartii', 'Data imprumutului']),
          @rows
        ]
      )
    );
  $OUT .= end_html();
}
```

După cum se poate remarca parcurgând sursele de mai sus, datele furnizate de serverul de baze de date (s-a recurs în acest caz la serverul PostgreSQL) vor fi trimise în format XHTML, către clientul Web, de scriptul CGI.

### Extragerea de informații ca documente XML

Următorul pas este să vedem cum putem extrage din baza de date informațiile dorite direct în XML, cu ajutorul modulului DBIx::XML\_RDB, ilustrând prin intermediul unui exemplu nu foarte complicat cum clientul poate degreva serverul de generarea unei reprezentări particulare a datelor. După cum se va remarca, modulul DBIx::XML\_RDB este răspunzător pentru transformare oricărei aserțiunii SQL SELECT în document XML.

Vom folosi transformările XSLT (*Extensible Stylesheet Language Transformations*), slujindu-ne de funcționalitățile oferite de modulul XML::XSLT. Pentru aceasta, vom scrie următorul script CGI (programul va realiza o interogare SQL, iar rezultatul primit de la server va fi un fișier XML pe care îl vom transforma în format XHTML prin intermediul unei foi de stiluri denumite rezult.xml):

```
#!/usr/bin/perl

# extract.pl
use strict;
use CGI;
use CGI::Carp;
use DBIx::XML_RDB; # modificat pentru a suporta foi de
  stiluri

my $q = new CGI;
$q->import_names('R');

# incercam sa ne conectam
my $xmlout = new DBIx::XML_RDB(
  'http://localhost/styles/rezult.xml',
  'dbname=biblioteca', 'Pg', 'ciu', '')
  || die "nu ma pot conecta...";

# lansam interogarea SQL
$xmlout->DoSql(qq{
```

```
SELECT c.titlu, i.data
FROM utilizatori u , carti c, imprumuturi i
WHERE u.ume = '$R::nume' and u.prenume = '$R::prenume'
      and u.cod = i.cod_utilizator
      and c.cod = i.cod_carte
});
```

```
# trimitem Content-type: text/xml
print $q->header('text/xml');
```

```
# apoi continutul fisierului XML generat
print $xmlout->GetData;
```

Documentul XML returnat respectă structura de mai jos. Drept exercițiu util, cititorul este îndemnat să construiască DTD-ul sau schema XML pentru validarea acestui fișier:

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl"
  href="http://.../styles/rezult.xsl" ?>
<biblioteca>
  <RESULTSET statement="
    SELECT c.titlu, i.data
    FROM utilizatori u , carti c, imprumuturi i
    WHERE u.ume = 'Ciubotariu' and u.prenume = 'Vlad'
      and u.cod = i.cod_utilizator
      and c.cod = i.cod_carte">
    <ROW>
      <titlu>In spatele usilor inchise</titlu>
      <data>2000-10-10</data>
    </ROW>
    <ROW>
      <titlu>Greața</titlu>
      <data>2001-11-02</data>
    </ROW>
  </RESULTSET>
</biblioteca>
```

Acest document va conține datele rezultate în urma interogării în cadrul elementelor <ROW> (în cazul nostru, valorile câmpurilor titlu și data).

Foaia de stiluri XSL rezult.xml care va fi folosită pentru afișarea comodă a acestor date este următoarea (recent, din octombrie 2001, XSL 1.0 a devenit recomandare a Consorțiului Web):

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/TR/XSL">
<!-- aplicam sablonul pentru elementul radacina al documentului -->
<xsl:template match="/">
  <xsl:apply-templates select="biblioteca" />
</xsl:template>

<xsl:template match="biblioteca">
  <html>
    <head><title>Rezultatele interogarii</title></head>
    <body bgcolor="white" text="blue">
      <!-- inceput de pagina -->
      <table align="center" width="633">
        <!-- antetul tabelului -->
        <tr align="center">
          <th>Titlu</th>
```

```
        <th>Data imprumutului</th>
    </tr>
    <!-- aplicam un alt sablon pentru
        a construi rindurile de tabel
        corespunzatoare datelor din <ROW> ->
    <xsl:apply-templates select='RESULTSET/ROW' />
</table>
<!-- final de pagina ->
</body>
</html>
</xsl:template>

<!-- sablonul pentru afisarea informatiilor din <ROW> ->
<xsl:template match='ROW'>
    <tr align="center">
        <!-- titlul cartii scris ingrosat ->
        <td>
            <b> <xsl:value-of select='titlu' /> </b>
        </td>
        <td>
            <xsl:value-of select='data' />
        </td>
    </tr>
</xsl:template>

</xsl:stylesheet>
```

Un ultim script Perl care realizează transformarea datelor marcate în XML în document XHTML, folosind foaia de stiluri prezentată mai sus. Acest program va funcționa ca un „proxy“ Web trimițând o interogare asupra bazei de date, primind documentul XML cu rezultatele și apoi transformându-l în format XHTML (avem, în fapt, o structură 3-tier). Pentru acest script, vom apela la modulele HTTP::Request (util pentru rezolvarea cererilor HTTP) și LWP::User-Agent (folosit pentru a putea „simula“ dialogul dintre serverul și navigatorul Web).

```
#!/usr/bin/perl

# xslt.pl
use strict;
use XML::XSLT;
use LWP::UserAgent;
use HTTP::Request;

die 'usage: xslt.pl <nume> <prenume>' unless @ARGV == 2;
my ($nume, $prenume) = @ARGV;

# formulam cererea GET catre server
# (se invoca scriptul CGI prezentat mai sus)
my $req = new HTTP::Request(GET => qq{

    http://localhost/~ciu/extract.pl?nume=$nume&prenume=$prenume
});
# vom accepta doar documente XML
$req->header(Accept => 'text/xml');

my $ua = new LWP::UserAgent;
my $res = $ua->request($req);
die $res->code . $res->message unless $res->is_success;

# verificam daca a fost furnizata o foaie de stiluri,
# adica daca exista directiva de procesare
```

```
# <?xml-stylesheet ... ?>
my $xmlDoc = $res->content;
my $xslref;
die "fara xsl... nu putem vizualiza documentul"
  unless ($xslref) = $xmlDoc =~ /<?xml-stylesheet\b
    [^>]+\b
    href
    \s*=\s*
    (?
    "([^"]*)" |
    '([^']*)' |
    )/gx;

# preluam URI-ul foii de stiluri XSL si o incarcam
$req->uri($xslref);
$req->header(Accept => 'text/xsl');
$res = $ua->request($req);
die $res->code . $res->message unless $res->is_success;

# instantiem procesorul XSLT
my $parser = XML::XSLT->new ($res->content, 'STRING');
# transformam documentul XML utilizand foaia de stiluri XSL
$parser->transform_document ($xmlDoc, 'STRING');

# afisam rezultatul XHTML
print $parser->result_string;
$parser->dispose ();
```

În cazul de față am înlocuit „dialectul” impus de `Text::Template` cu o foaie de stiluri (care poate fi procesată direct pe partea client, de exemplu de Internet Explorer 5 sau o versiune superioară). Rezultatul final va fi un document XHTML. Modificând numai foaia de stiluri XSL, aceleași date le putem afișa diferit pe alte dispozitive – e.g. pe telefonul

## Resurse

- ✓ Sabin-Corneliu Buraga – *Tehnologii Web*, Matrix Rom, București, 2001: <http://www.infoiasi.ro/~busaco/books/web.html>
- ✓ Marin Fotache – *SQL – Dialecte DB2, Oracle, Visual FoxPro*, Polirom, Iași, 2001
- ✓ Mircea Sârbu – *PostgreSQL practic*, NET Report, nr. 108, anul 10, sep. 2001
- ✓ Parand Tony Daruger – *Manipulating XML documents with Perl and other scripting languages*, IBM developerWorks, feb. 2000
- ✓ Ștefan Trăușan-Matu, Claudia Răibuleț, Ovidiu Constantin – *Prelucrarea documentelor folosind XML și Perl*, Matrix Rom, București, 2001
- ✓ Alligator Descartes, Tim Bunce – *Programming the Perl DBI*, O'Reilly and Associates, 2000
- ✓ Lincoln Stein – *CGI.pm – a Perl5 CGI Library*, 1998
- ✓ \*\*\* – *DBI – A Database Interface Module for Perl 5*:  
<http://dbi.symbolstone.org/index.html>
- ✓ \*\*\* – *DBIx::XML\_RDB*:  
[http://theoryx5.uwinnipeg.ca/CPAN/data/DBIx-XML\\_RDB/XML\\_RDB.html](http://theoryx5.uwinnipeg.ca/CPAN/data/DBIx-XML_RDB/XML_RDB.html)
- ✓ \*\*\* – *Perl CPAN*: <http://www.perl.com/CPAN/>
- ✓ \*\*\* – *Perl XML FAQ*:  
<http://www.perlxml.com/faq/perl-xml-faq.html>
- ✓ \*\*\* – *World-Wide Web Consortium's Technical Reports*:  
<http://www.w3.org/TR/>

celular, utilizându-se, drept limbaje de marcare, *WML* (*Wireless Markup Language*) sau mai vechiul *HDML* (*Handheld Device Markup Language*). O aplicație sugerată de autorul modulului `DBIx::XML_RDB` folosit mai sus este cea de a realiza transferul de date între baza de date heterogene prin intermediul unui format neutru (datele binare vor putea fi codificate prin UTF-8). După cum am văzut mai sus, formatul XML omogenizează reprezentarea datelor pe diferite platforme, favorizând integrarea unor medii heterogene și, nu în ultimul rând, independența logică a datelor.

## Realizarea de interogări prin XQL

XML permite marcarea structurii logice a datelor. Rezultă de aici că meta-limbajul XML poate fi formatul nativ pentru stocarea de date, având și avantajul că se comprimă ușor și eficient (este în fapt un format text). Limbajul *XQL* (*Extensible Query Language*) extinde natural capacitatea XSL de a identifica clase de noduri, adăugând operatori logici, filtre, indecși etc. Este conceput special pentru documentele XML, având o sintaxă concisă și eficientă, inspirată întrucâtva de XPath. Avizăm cititorul că există și alte propuneri de limbaje de interogare pentru datele structurate ca documente XML, ca de exemplu *XQuery* sau *XML-QL*.

Studiul nostru de caz cuprinde o rescriere în XML a schemei de bibliotecă folosită în exemplele anterioare și care era inerent optimizată pentru modelul relațional. Pentru simplitate, s-a renunțat la definirea formală a tipului de document prin DTD sau XML Schema.

Vom prezenta în continuare o propunere de structură de document XML care va stoca informațiile referitoare la împrumuturile de cărți. Acest fișier îl vom denumi `imprumuturi.xml`:

```
<?xml version="1.0" ?>
<imprumuturi>
  <RECORD>
    <carte>
      <titlu>Greata</titlu>
      <autor>J.P. Sartre</autor>
    </carte>
    <utilizator>
      <nume>Ciubotariu</nume>
      <prenume>Vlad</prenume>
    </utilizator>
    <data>2001-11-02</data>
  </RECORD>
  <RECORD>
    <carte>
      <titlu>Muntele vrajtit</titlu>
      <autor>Th. Mann</autor>
    </carte>
    <utilizator>
      <nume>Dumitriu</nume>
      <prenume>Daniel</prenume>
    </utilizator>
    <data>2001-10-12</data>
  </RECORD>
  <RECORD>
    <carte>
      <titlu>Lupul de stepa</titlu>
      <autor>H. Hesse</autor>
    </carte>
    <utilizator>
      <nume>Tarhon-Onu</nume>
      <prenume>Victor</prenume>
    </utilizator>
    <data>2001-12-12</data>
```

```
</RECORD>  
</imprumuturi>
```

Pentru procesarea în Perl a unei interogări XQL care găsește toate cărțile împrumutate de o anumită persoană vom folosi *DOM (Document Object Model)* și modulul `XML::XQL`. Codul sursă al scriptului CGI (denumit `xql.pl`) este următorul:

```
#!/usr/bin/perl  
  
# xql.pl  
use strict;  
use XML::XQL;  
use XML::XQL::DOM;  
  
die 'usage: xql.pl <nume> <prenume>' unless @ARGV == 2;  
my ($nume, $prenume) = @ARGV;  
  
# instantiem analizorul DOM  
# pentru a realiza procesarea documentului XML  
my $parser = new XML::DOM::Parser;  
my $imprumuturi = $parser->parsefile("imprumuturi.xml");  
  
# formulam interogarea XQL  
my $q1 =  
    '/*/RECORD[utilizator/nume = ? $and$ utilizator/prenume =  
        ?]';  
  
# se substituie primul '?' din interogare cu numele  
# furnizat de utilizator se substituie al doilea '?' cu  
# prenumele dat de utilizator  
$q1 =~ s/(.*)\?(.*)\?(.*)/$1$nume'$3'$prenume'$5/;  
  
# afisam rezultatele ca sir de caractere  
foreach (XML::XQL::solve($q1, $imprumuturi)) {  
    print $_->xql_toString . "\n";  
}
```

Pentru scriptul de mai sus, dacă utilizatorul l-a apelat cu parametrii *Tarhon-Onu Victor*, cererea XQL va fi `/*/RECORD[utilizator/nume = 'Tarhon-Onu' $and$ utilizator/prenume = 'Victor']`. Se va căuta în documentul XML, parcurgând arborele de elemente XML, orice potrivire a conținutului sub-elementului `<nume>` și a sub-elementului `<prenume>` ale elementului `<utilizator>` cu numele și prenumele specificate la intrare. Cititorul poate transforma acest program Perl în script CGI pentru a putea fi folosit pentru Web.

Așadar, ne putem dispensa de baza de date relațională în cazul unei aplicații simple. Obținând resursa XML dorită, putem crea și extrage datele (într-o multitudine de forme) într-un mod flexibil, în funcție de necesități.

## Concluzii

Am văzut mai sus câteva modalități practice de a realiza, pe Web, interogări asupra datelor stocate sub formă de baze de date relaționale sau în documente XML folosind tehnologii *open-source*. Cititorul interesat poate aprofunda domeniul și poate experimenta aspecte mai sofisticate, prin scrierea de scripturi Perl mai complexe (și mai utile) decât cele din acest articol. Înainte de a încheia, trebuie să mulțumim călduros absolventului *Vlad Ciubotariu* pentru suportul acordat în conceperea programelor prezentate, urându-i succes oriunde ar fi.

---

*Sabin-Corneliu Buraga este doctorand în Computer Science la Facultatea de Informatică, Universitatea „Al.I.Cuza” din Iași și poate fi contactat la [busaco@infoiasi.ro](mailto:busaco@infoiasi.ro) sau <http://www.infoiasi.ro/~busaco/>. ■ 98*