Ch. 4 Query Languages

```
4.1 Path Expressions

Semistructured Data Model: edge-labeled graph

Path Expression: l1.l2. ... ln  a sequence of edge labels

A path expression may be viewed as a simple query whose
result is a set of nodes in the edge-labeled graph where
the path expression ends.

Consider the data in Figure 4.1 (Page 56)

  biblio.book  results in the set of nodes {n1, n2}
  biblio.book.author results in the nodes associated
    with the strings: {"Combalusier", "Roux", "Smith"}
```

```
In general, the result of the path expression l1.l2. ... .ln
on a data graph is the set of nodes vn such that there exists
edges (r,l1,v1), (v1,l2,v2), ..., (vn-1,ln,vn) where r is the root.

The path expressions can be expressed in terms of some properties
they satisfy; To accomplish this, we use Regular Expressions
(both on the alphabet of edge labels and on the alphabet of
characters that form the edge labels) to describe such properties.

ex.  biblio.(book | paper).author

    -- biblio followed by book or paper followed by author

     biblio._.author

    -- biblio followed by any one edge followed by author

       biblio._*.author

    -- biblio followed by zero or more edges followed by author
```

The general syntax for regular expressions on paths is

```
e :== l | e | _ | e.e | (e) | (e|e) | e* | e+ | e?
```

To specify more complex label patterns, we use grep patterns, for example

```
((s|S)ection|paragraph)(s)?
```

matches any one of six patterns:
section, Section, sections, Sections, paragraph, paragraphs

To avoid ambiguity between regular expressions for labels and those for path expressions, the regular expressions on labels are enclosed within quotes:

```
biblio._*.section.("[tT]itle" | paragraph.".*heading.*")
```

matches any path that starts with a biblio label and ends with a section label followed by either title or Title edge or paragraph edge followed by an edge whose label contains the string "heading"

For data graphs that have cycles in them, it is possible to specify path expressions of arbitrary length. For example,

```
cities.state-of.capital.state-of.capital.state-of
```

4.2 A Core Language

Path expressions produce as their result a set of nodes of the Data Graph.

They cannot produce semi-structured data
(which requires joining ability).

Query language features will be necessary for this.

4.2.1 The Basic Syntax

based on OQL (Object Query Language)

```
% Query q1
select author: X
from   biblio.book.author X
```

computes the set of book authors and forms a ssd-expression out of the nodes:

```
{author: "Roux", author: "Combalusier",  author: "Smith"}
```

```
% Query q2
select row: X
from    biblio._ X
where  "Smith" in X.author
```

computes the answer:

```
  {row: {author: "Smith", date: 1999, title: "Database Systems"},
   ... }
```

The "in" predicate tests for set membership.
Here X.author is a path expression whose root is taken to
be the node represented by X.

Assume a matches predicate exists which matches
strings to regular expressions.

The following query collects all authors of publications
whose title consists of the word "database".

```
select author:Y
from   biblio._ X,
           X.author Y,
           X.title Z
where matches(".*(D|d)atabase.*", Z)
```

```
Semantics of query

  select E from B where C

Step 1: Find the set of all bindings of the variables
        that appear in B (assume 3 variables X, Y, Z)
        Each binding maps the variables to oids in the
        data graph.

Step 2: Filter the bindings that satisfy C; Let the resulting
        set of bindings be (x1,y1,z1), ..., (xn,yn,zn)

Step 3: Construct the ssd-expression
           { E(x1,y1,z1), ..., E(xn,yn,zn) } where
        E(xi,yi,zi) denotes the expression E in which the
        variables X, Y, Z are replaced by xi, yi, zi
```

Some queries create more than one new node:

```
select row: { title:Y, author:Z}
from    biblio.book X, X.title Y, X.author Z
```

The result will be constructed as follows:

```
  { row: {title:y1, author:z1}, ...,
    row: {title:yn, author:zn} }
```

Another means of creating many new nodes is by nesting subqueries in the select clause.

```
% Query q3
select row: (select author: Y
                      from   X.author Y)
from    biblio.book X
```

The output of this query is:

```
  {row: {author: "Roux", author: "Combalusier"},
    row: {author: "Smith"}}
```

See Figure 4.1 (d) for a graphical representation of the query output; Compare with 4.1 (b), answer to query q1.

Another nested-select query.

```
% Query q4
select row: (select author:Y, title: T
                      from   X.author Y, X.title T)
from   biblio.book X
where "Roux" in X.author
```

Output of q4:

```
  {row: {author: "Roux", title: "Database Systems"},
    row: {author: "Combalusier", title: "Database Systems"}}
```

shown in graphical form in Figure 4.1 (e)

Join Examples:

```
r1(a,b)  r2(b,c)

{r1: {row: {a:1, b:2}, row: {a:1, b:3}},
 r2: {row: {b:2, c:4}, row: {b:2, c:3}} }

project(a,c)(r1 join r2)

% Query q-join
select a:A, c:C
from    r1.row X, r2.row Y, X.a A, X.b B, Y.b B', Y.c C
where  B=B'
```

Observation: If multiple B values were allowed in r1 and r2, join will take place if the two sets of B values in the rows have at least one common value.

Another Join example:

Get authors who are referred to at least twice in some paper with "Database" in the title.

```
select row: W
from    biblio.paper X,  X.refers-to Y,
        Y.author W,  X.refers-to Z
where  NOT (Y=Z) and
            W in Z.author and
            matches(".*Database.*", X.title)
```

4.3 More on Lorel (Lore Language; Query Language for Lore)
Lore: Lightweight Object REpository

Core language plus syntactic shortcuts.

Omission of labels:

```
select X
from    biblio.book.author X
```

default label in Lore is answer

So, the answer to above query will be

```
{answer: "Roux", answer: "Combalusier", answer: "Smith" }
```

Use of Path Expressions in select clause:

```
% Query q3'
select X.author
from    biblio.book X
```

X.author can be understood as the nested query

```
 (select author: Y
   from  X.author Y)
```

In general, an expression of the form:

```
X.p.l,
```

where p is an arbitrarily complex path expression,
is understood as the nested query

```
select l:Y
from   X.p.l Y
```