MỤC LỤC

DANH SÁCH CÁC HÌNH

LỜI CẢM ƠN

Trong lời đầu tiên của báo cáo đồ án tốt nghiệp "Tìm hiểu các thuật toán truy xuất dữ liệu XML" này, em muốn gửi những lời cám ơn và biết ơn chân thành nhất của mình tới tất cả những người đã hỗ trợ, giúp đỡ em về kiến thức và tinh thần trong quá trình thực hiện đồ án.

Trước hết, em xin chân thành cám ơn Thầy Giáo - Ths. Nguyễn Trịnh Đông, Giảng viên Khoa Công Nghệ Thông Tin, Trường ĐHDL Hải Phòng, người đã trực tiếp hướng dẫn, nhận xét, giúp đỡ em trong suốt quá trình thực hiện đồ án.

Xin chân thành cảm ơn các thầy cô trong Khoa Công Nghệ Thông Tin và các phòng ban nhà trường đã tạo điều kiện tốt nhất cho em cũng như các bạn khác trong suốt thời gian học tập và làm tốt nghiệp.

Cuối cùng em xin gửi lời cảm ơn đến gia đình, bạn bè, người thân đã giúp đỡ động viên em rất nhiều trong quá trình học tập và làm Đồ án Tốt Nghiệp.

Do thời gian thực hiện có hạn, kiến thức còn nhiều hạn chế nên Đồ án thực hiện chắc chắn không tránh khỏi những thiếu sót nhất định. Em rất mong nhận được ý kiến đóng góp của thầy cô giáo và các bạn để em có thêm kinh nghiệm và tiếp tục hoàn thiện đồ án của mình.

Em xin chân thành cảm ơn!

Hải Phòng, ngày 25 tháng 12 năm2012

Sinh viên

Lê Anh Tuấn

MỞ ĐẦU

Giới thiệu đồ án

- XML là ngôn ngữ đánh dấu mở rộng dựa trên chuẩn SGML (Standard Generalized Markup Language).
- XML rất Linh động, nó đủ mạnh dùng để lưu trữ tất cả các loại dữ liệu.
- Cách truy xuất dữ liệu của XML nhanh, dễ dàng và thích nghi với mọi hệ thống, nên các hệ thống lớn trên thế giới chọn nó làm chuẩn dữ liệu chính dùng để trao đổi thông tin.
- XML là chuẩn định dạng dữ liệu cho cơ sở dữ liệu eXist database hiện nay.
- Chính vì lý do đó em đã chọn đề tài "tìm hiểu các thuật toán truy xuất dữ liệu XML" để có thể hiểu nhiều hơn, nắm chắc hơn, nâng cao hiểu biết đồng thời tổng hợp lại được các kiến thức đã học được ở trường trong thời gian qua để có thể sử dụng thành thạo và khai thác được hết tiềm năng to lớn trong khoa học kỹ thuật của XML

Mục tiêu đồ án

- Đồ án nghiên cứu tìm hiểu các thuật toán truy xuất dữ liệu XML.
- Ứng dụng minh họa sử dụng cơ sở dữ liệu mã nguồn mở eXist để tổ chức lưu trữ tài liệu XML.

CHƯƠNG 1: XML

1.1. GIỚI THIỆU VỀ XML

Trong thời đại Công nghệ thông tin hiện nay XML (*Extensible Markup Language*) chiếm một vị trí rất quan trọng trong việc chuyển tải, trao đổi dữ liệu và liên lạc giữa các ứng dụng.Điều này càng được khẳng định khi trong các hệ điều hành từ *WindowsXP* trở đi, bên trong nó chứa đầy XML.Hơn nữa khi bộ.*Net* ra đời thì càng làm cho XML trở nên thịnh hành.Sử dụng kỹ thuật XML không chỉ có tập đoàn *Microsoft* mà ngay cả *Sun, IBM, Oracles* điều hỗ trợ XML và dùng nó trong các ứng dụng.

1.1.1. Giới thiệu

XML (eXtensible Markup Language) là ngôn ngữ đánh dấu mở rộng dựa theo chuẩn SGML (*Standard Generalized Markup Languague*). SGML được phát triển cho việc định dạng cấu trúc và nội dung tài liệu điện tử, do tổ chức ISO (*International Organization for Standards*) chuẩn hóa năm 1986.

SGML là do IBM (*International Business Machines*: tập đoàn công nghệ máy tính đa quốc gia có trụ sở tại *Armonk, New York, Mỹ*) đưa ra, song không thể không kể đến những đóng góp của nhiều công ty khác. XML được W3C (*World Wide Web Consortium*: tổ chức độc lập định ra tiêu chuẩn cho trình duyệt Web, máy chủ và ngôn ngữ) phát triển, nhưng đặc tả XML lại do *Netscape, Microsoft* và các thành viên của dự án *Text Encoding Initiative* (TEI) xây dựng. Tổ chức W3C XML *Special Interest Group* có đại diện từ hơn 100 công ty cùng nhiều chuyên gia khác được mời.

Lý do ra đời của XML vì SGML rất phức tạp, còn HTML có nhiều hạn chế nên năm 1996 tổ chức W3C thiết kế XMLnhằm mục đích đơn giản hóa việc chia sẻ dữ liệu giữa các hệ thống khác nhau, đặc biệt là các hệ thống được kết nối *Internet*.

Điểm quan trọng của kỹ thuật mô tả XML là nó không thuộc riêng về một công tynào vì XML thuộc về cả thế gới, nó là một tiêuchuẩn được mọi người công nhận vì được tạo ra bởi W3C (*World Wide WebConsortium).*

Do XML rất là đơn giản cho nên các công cụ chuẩn được tạo ra đểlàm việc với XML như *Document Object Model - DOM, Xpath, XSL,* v.v..rất hữu hiệu, và chính các chuẩn này được phát triển không ngừng.

XML cũng giống như HTML đều là ngôn ngữ đánh dấu, nhưng sự ra đời của XML là để khắc phục cho một số yếu kém của HTML.HTML và XML đều sử dụng các thẻ (*tag*) nhưng các thẻ của HTML là một bộ dữliệu thẻ được xây dựng và định nghĩa trước, tức là người lập trình phải tuânthủ theo các thẻ đã định nghĩa của HTML, hiện HTML có khoản hơn 400 thẻ,để nhớ hết 400 thẻ này cũng không có gì khó khăn đối với người lập trình *Web*chuyên nghiệp nhưng thật khó đối với những người không chuyên. Hơn nữacác thẻ của HTML không nói lên được mô tả dữ liệu trong đó. Nhưng đối vớiXML thì hoàn toàn khác bởi vì tag trong XML là do người lập trình định nghĩavà mỗi thẻ là một mô tả dữ liệu mà người lập trình muốn truyền đạt.

1.1.2. Lợi ích về của XML

Lợi ích về thương mại

Chia sẻ dữ liệu: XML cho phép các doanh nghiệp định nghĩa chuẩn dữ liệu của mình, từ đó dễ dàng xây dựng các công cụ để đọc, viết và trao đổi dữ liệu. Điều này cho phép các doanh nghiệp có thể tự xây dựng một chuẩn định dạng dữ liệu XML cho mình.Do đó dữ liệu của một ứng dụng có thể dễ dàng chia sẻ với nhiều ứng dụng khác. Chẳng hạn dữ liệu về khách hàng của một siêu thị có thể được chia sẻ với một công ty tiếp thị sử dụng cùng tiêu chuẩn định dạng.

Mô tả dữ liệu phức tạp: XML là một ngôn ngữ mềm dẻo cho việc mô tả các dữ liệu phức tạp. Chẳng hạn trong đồ họa *vector*, ký hiệu âm nhạc, toán học, hóa học và nhiều lĩnh vực khác nữa.Vì vậy nó là một công cụ mạnh để xây dựng các ứng dụng mới.

Phân phát nội dung: XML có khả năng hỗ trợ những người dùng và kênh truyền khác nhau vì thế ta có thể xây dựng các ứng dụng có hiệu quả cao. Kênh truyền ở đây bao gồm phân phát thông tin cho các máy móc, cơ chế khác nhau ví dụ như TV kỹ thuật số, điện thoại, web,...Hỗ trợ các kênh truyền khác nhau là một bước quan trọng trong việc phân phát ứng dụng thương mại điện tử (*e-bussinese*).Chẳng hạn một siêu thị điện tử có thể phục vụ cho người dùng sử dụng laptop ở nhà, ở công ty hay ở bất cứ đâu, đang làm bất kỳ việc gì sử dụng điện thoại di động hỗ trợ *WAP*.

Lợi ích về kỹ thuật

XML đơn giản hóa việc trao đổi dữ liệu: Bởi vì những công ty khác nhau hiếm khi cùng làm trên một bộ công cụ định dạng nhất định, điều đó sẽ dẫn đến khó khăn trong việc trao đổi thông tin. Sử dụng XML, mỗi công tycó thể tạo ranhiều ứng dụng riêng đồng thờidễ dàng chuyển đổi những định dạng dữ liệu bên trong chúngđể có thể trao đổi trao đổi dữ liệu một cách đơn giản. Trên hết, đây là cơ hội tốt để nhà cung cấp phần mềm có thể đưa ra các công cụ chuyển đổi những ghi chép cơ sở dữ liệu của họ thành XML và ngược lại.

XML cho phép mã hóa thông minh: Do những văn bản XML được tổ chức để nhận dạng từng thông tin quan trọng, có thể viết mã để xử lí văn bản XML mà không cần con người tác động. Những nhà cung cấp phần mềm đã dành rất nhiều thời gian và tiền bạc xây dựng các công cụ phát triển XML, nên khi sử dụng XML việc viết những mã đó là một quá trình tương đối đơn giản.

XML cho phép tìm kiếm thông minh: Mặc dù công cụ tìm kiếm đã cải thiện dần trong nhiều năm qua, tuy nhiên nhận được những kết quả không chính xác vẫn phổ biến xảy ra. Nếu bạn đang tìm kiếm một cái gì đó mang tên "Shop" trong những trang HTML, bạn sẽ tìm thấy một loạt các trang web về Shopquần áo, shop máy tính, shop đồ gỗ, và rất nhiều thứ vô dụng khác. Tìm kiếm văn bản XML cho <*first-name*> các yếu tố chứa từ Shop sẽ mang lại cho bạn những kết quả tốt hơn rất nhiều.

Sử dụng lại dữ liệu: Khi muốn tính toán lại hay trình bày lại một tập dữ liệu có sẵn. Máy chủ không cần chuyển lại dữ liệu về cho máy trạm mà sử dụng luôn dữ liệu đã được truyền trước đó.Điều này giúp giảm lưu lượng truyền trên mạng. Hoặc dữ liệu của nhà xuất bản có thể được thư viện sử dụng lại vì chúng sử dụng chung định dạng. Bằng cách đó ta không phải xây dựng lại cơ sở dữ liệu cho thư viện.

Chia cắt dữ liệu và trình diễn: Một website sau một thời gian hoạt động cần được thiết kế lại. Nếu website đó sử dụng XML để lưu dữ liệu thì nó chỉ cần thay đổi giao diện còn tầng dữ liệu vẫn được giữ nguyên.

Khả năng mở rộng: Một ứng dụng sử dụng XML có nhiều phiên bản khác nhau. Sau mỗi lần nâng cấp thì các thẻ mới được thêm vào.Điều này sẽ không ảnh hưởng đến việc sử dụng cơ sở dữ liệu mới bằng các ứng dụng cũ khi người dùng muốn thay đổi thói quen làm việc và sử dụng của mình.

Thông tin có ý nghĩa: Khi đưa ra một từ khóa "*Quang Vinh*", thông tin có ý nghĩa sẽ cho phép người đọc có thể lựa chọn đó là một tính từ, tên một cầu thủ, hay

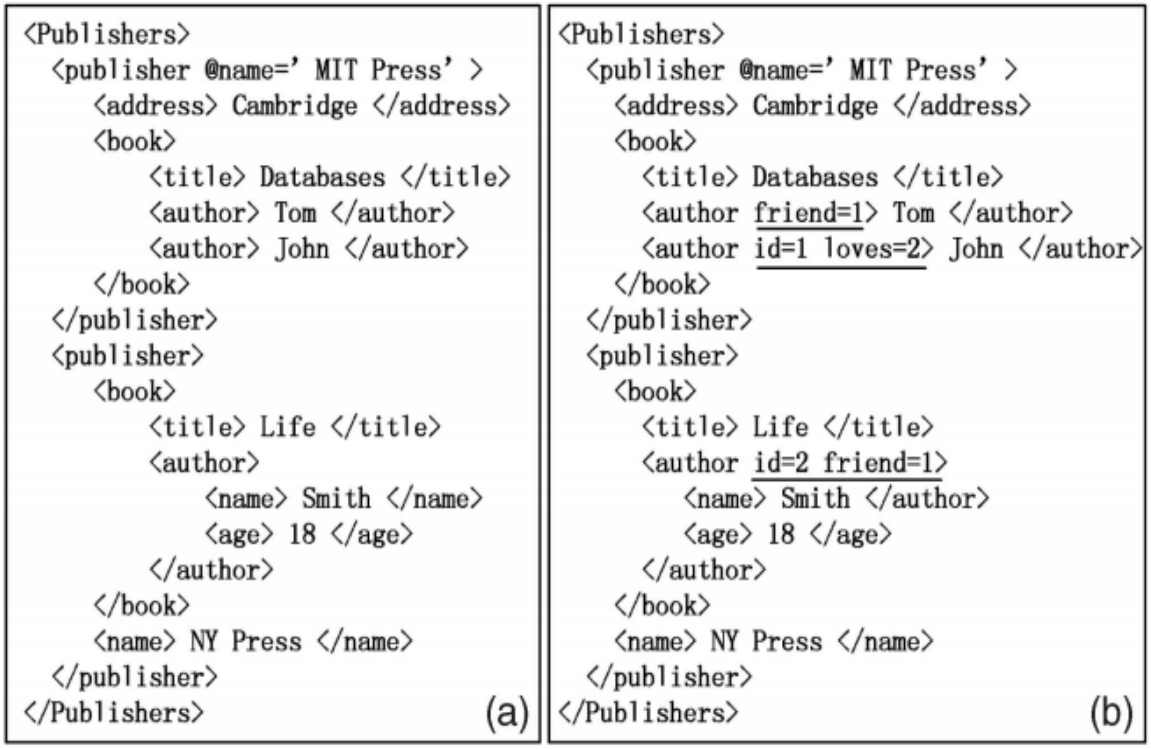
tên một nhà hàng,...Bộ máy tìm kiếm dựa trên HTML không thể làm được điều đó vì không đủ thông tin ý nghĩa trong một trang HTML.Với XML văn bản tự mô tả chính nó vì vậy rất dễ dàng để biết được ý nghĩa của văn bản.

Các lợi ích khác: XML dễ dàng đọc bởi cả máy tính và con người, nó dựa trên cấu trúc cây và rất dễ dàng để tạo ra một văn bản XML (đơn giản nhất là dùng Notepad),...

1.2. MÔ HÌNH DỮ LIỆU CỦA XML

Mô hình cơ bản:Mô hình cây

Mô hình dữ liệu cơ sở của XML là cây gán nhãn.



Hình 1.1(a) Tài liệu XML không có ID/IDREF(b) Tài liệu XML có ID/IDREF

Hình trên biểu diễn cây dữ liệu của một tài liệu XML.*Hình 1.1a* là mô hình cây nhãn nút, *hình 1.1b* là mô hình cây nhãn cung, hai mô hình là tương đương.

Ta xét cây dữ liệu XML theo mô hình cây nhãn nút (và tương tự cho cây nhãn cung).

Có 3 kiểu nút trong cây dữ liệu:

Nút phần tử: tương ứng với thẻ trong tài liệu XML. Ví dụ: "*Publishers*".

Nút thuộc tính: tương ứng với thuộc tính gắn với thẻ trong tài liệu XML. Ví dụ *"@name*". Trái với nút phần tử, nút thuộc tính không được lồng nhau (tức là các thuộc tính không thể có phần tử con), không lặp lại (tức là không có hai thuộc tính cùng tên dưới một phần tử), không thứ tự (tức là các thuộc tính của một phần tử có thể đổi chỗ cho nhau trong cùng một phần tử).

Nút giá trị (nút lá): tương ứng với giá trị của dữ liệu trong tài liệu XML. Ví dụ: "*MIT Press*".

Các cung trong cây dữ liệu biểu diễn quan hệ cấu trúc giữa các phần tử, thuộc tính, và giá trị.Trong một cây dữ liệu XML, một vài nút trùng tên có thể được đặt trong cùng một đường dẫn.Hiện tượng đó được gọi là đệ quy.

Mô hình mở rộng:*Directed Acyclic Graphs* (DAGs) và *General Graphs*

Tài liệu XML cho phép người sử dụng định nghĩa các thuộc tính ID/IDREF của phần tử.Một thuộc tính ID xác định duy nhất một phần tử, và các thuộc tính IDREF trỏ tới phần tử khác, sử dụng thuộc tính ID của nó.Hình 1.1b là tài liệu XML với các thuộc tính ID/IDEF. Các thuộc tính ID/IDEF tăng tính linh hoạt của mô hình dữ liệu XML và mở rộng mô hình cây cơ sở thành đồ thị có hướng phi chu trình – DAGs hay thậm chí thành đồ thị có hướng tổng quát (có chu trình).

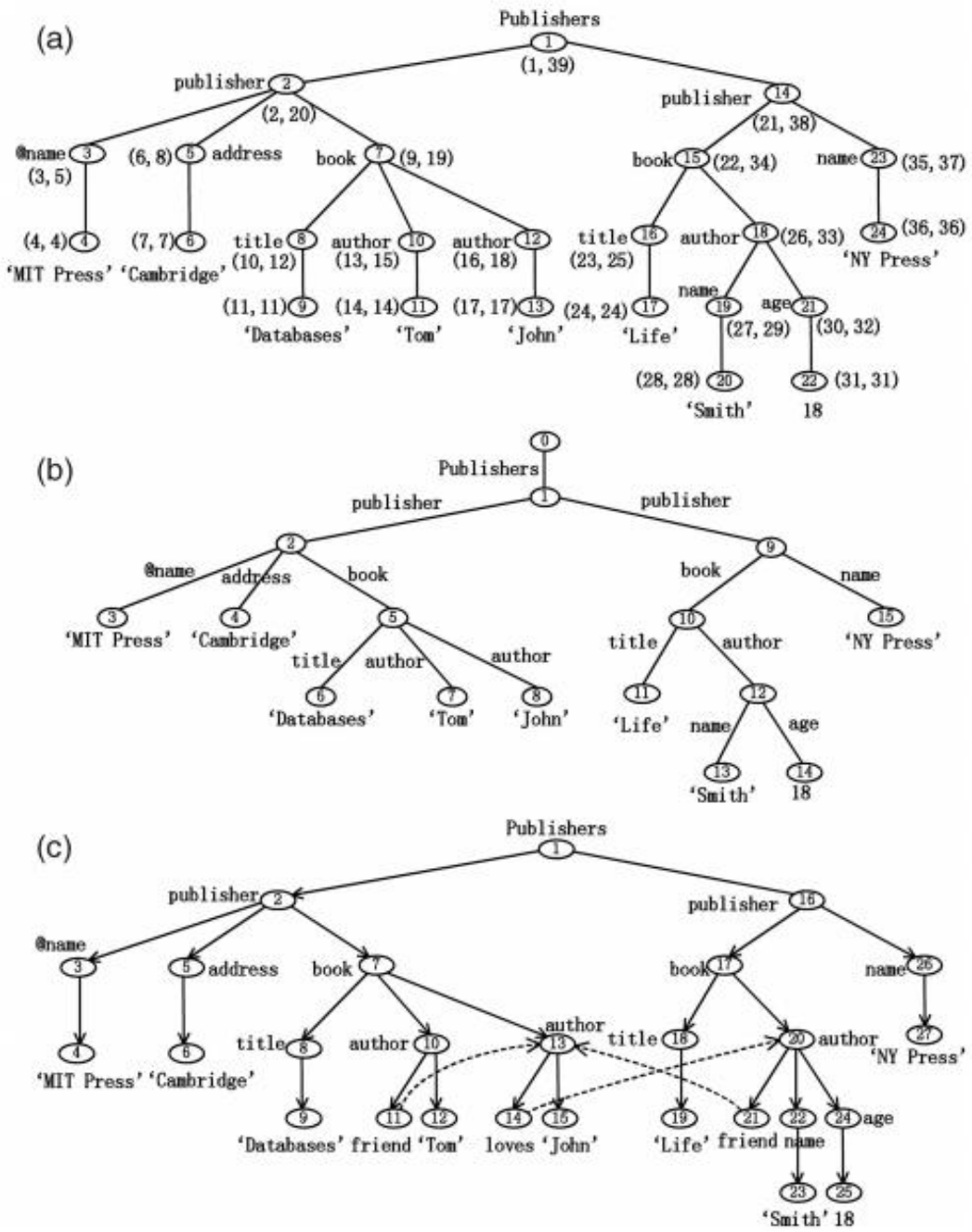
1.3. CÁC TRUY VẤN CỦA XML

Khác với văn bản text, tài liệu XML có cấu trúc lồng nhau nên truy vấn XML liên quan không chỉ nội dung mà còn cả cấu trúc của dữ liệu XML. Về cơ bản, có thể định dạng các truy vấn sử dụng các mẫu dò. Trong đó nút là giới hạn mà người sử dụng quan tâm, tức là phần nội cung của các truy vấn.Còn cung quan hệ cấu trúc giữa các giới hạn, tức là phần cấu trúc của các truy vấn.

Ta phân loại truy vấn XML thành 2 lớp: các truy vấn kiểu cơ sở dữ liệu (*Database-style*) và các truy vấn kiểu phục hồi thông tin (*Information Retrieval Style – TR style*).

Các truy vấn *Database-style* trả về kết quả truy vấn là khớp đúng (nội dung và cấu trúc) các yêu cầu của các truy vấn.Nó tương tự như ngữ nghĩa truy vấn SQL trong cơ sở dữ liệu quan hệ.

Các truy vấn IR cho kết quả truy vấn chính xác, được xếp dựa trên đánh giá thích hợp tới các truy vấn. Chỉ có các kết quả được đánh giá cao được trả về cho người sử dụng. Nó tương tự với ngữ nghĩa tìm kiếm từ khóa trong các truy vấn IR truyền thống.

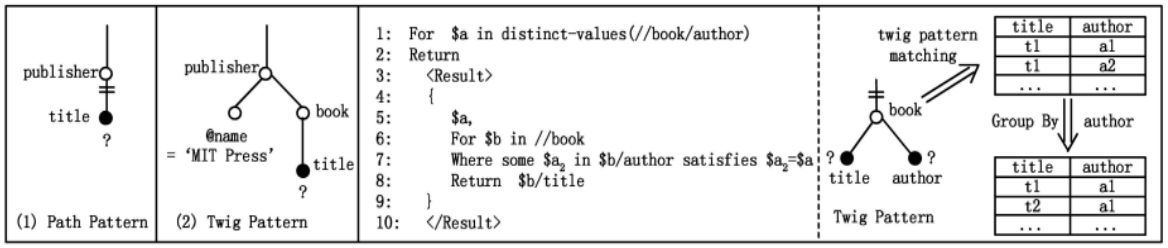


Hình 1.2(a) Cây dữ liệu XML với nút được gán nhãn(b) Cây dữ liệu XML *Edgelabeled*(c) Đồ thị dữ liệu XML với nút được gán nhãn

a) Các truy vấn XML Database-style

XML *Path Language* (*Xpath*)và*Xquery*, là ngôn ngữ truy vấn XML chính. Các mẫu dò đóng vai trò rất quan trọng trong *Xpath*và *Xquery*.

*Xpath:*là ngôn ngữ truy vấn XML cơ sở, chọn nút trong tài liệu XML sao cho đường dẫn từ gốc tới nút được chọn thỏa mãn mẫu đặc biệt. Một truy vấn *Xpath*đơn giản là một chuỗi các trục và thẻ xen kẽ nhau. Hai trục thường dùng nhất là trục con "/" ở đây "A/B" nghĩa là chọn nút con B của nút A, và trục hậu duệ "//" ở đây "A//B" nghĩa là chọn nút hậu thế B của nút A. Xét một ví dụ: truy vấn *Xpath"/publisher//title*" sẽ trả về tất cả các phần tử title ở bên dưới tất cả các phần tử publisher. Kết quả của truy vấn này trên cây dữ liệu hình 1.2a là hai nút title có giá trị là "Databases" và "Life".



Hình 1.3 (a)*Xpath*và (b) *Xquery*

Truy vấn *Xpath*trên có thể hiểu như là một mẫu đường dẫn đơn giản, như hình 1.3a(1), ở đây cung bình thường là trục "/", cung có dấu "=" là trục "//", còn nút tô đậm là ra kết quả. Tổng quát hơn, truy vấn *Xpath*có thể chỉ rõ một mẫu thử phức tạp hơn bằng cách sử dụng các bộ lọc trong biểu thức. Ví dụ *"/publisher[@name = MIT Press] /book/title"* (hình 1.3a(2)), ở đây *"/publisher/book/title"* là đường dẫn chính của truy vấn, còn nội dung nằm trong dấu "[" và "]" là một bộ lọc. Truy vấn này trả về tất cả các tên sách của nhà xuất bản *"MIT Press"*.Một cách tông quát, truy vấn *Xpath*có thể chứa nhiều thuộc tính.

Lõi của một truy vấn *Xpath*là một mẫu thử chỉ có đúng một nút đầu ra. Ta sẽ gọi các nút trong trong truy vấn là nút vấn tin và các nút trong cây dữ liệu là nút dữ liệu. Ngoài trục con và trục hậu duệ, *Xpath*còn định nghĩa 11 kiểu trục khác: cha, tổ tiên, hậu duệ hoặc bản thân, tổ tiên hoặc bản thân, *following*, *preceding*, *following-sibling*, *preceding-sibling*, bản thân, thuộc tính, và không gian tên (*namespace)*. Ta chỉ tập trung vào các trục "/" và "//" vì chúng thường được sử dụng và đáng nghiên cứu nhất.

Xquery: Ngôn ngữ truy vấn *Xquery* diễn cảm hơn *Xpath.*Một truy vấn *Xquery* gồm các mệnh đề *For-Let-Where-Return* (FLWR), có thể lồng nhau, rất tổng quát. Nghĩa là mỗi mệnh đề có thể chứa các truy vấn con *Xquery*.

Điều khoản *For, Let* buộc các nút do biểu thức *Xpath*lựa chọn với các biến nút mà người sử dụng định nghĩa.

Điều khoản *Where* chỉ rõ sự lựa chọn hay vị từ nối đối với các biến nút.

Điều khoản *Return* tác động vào biến nút để định dạng kết quả vấn tin dưới dạng XML.

Hình 1.3b biểu diễn một truy vấn *Xquery* đơn giản, nhóm các tên sách theo tác giả chứ không nhóm tác giả theo sách.

Mặc dù cú pháp phức hợp và lồng nhau của *Xquery* làm cho nó có khả năng biểu diễn cao hơn *Xpath,* sự phong phú ngữ nghĩa này cũng làm tăng độ phức tạp khi tối ưu hóa và đánh giá truy vấn *Xquery*.

Mặc dù vấn tin *Xquery* có thể được đánh giá theo cách đơn giản dùng các thủ tục vòng lặp lồng nhau theo đúng cú pháp. Nhưng cách làm sơ đẳng này sẽ kém hiệu quả. Ví dụ, trong khi truy vấn hình 1.3b gồm 4 biểu thức đường dẫn (dòng 1,6,7 và 8), không lên đánh giá 4 biểu thức đường dẫn đấy riêng biệt. Thật vậy, các biểu thức đường dẫn này có phần chung nhau, và đánh giá chúng riêng biệt sẽ dẫn đến truy cập lặp lại cùng nút dữ liệu.

Nếu phân tích cẩn thận ngữ nghĩa của truy vấn *Xquery* trong ví dụ này, một bộ tối ưu *Xquery* thông minh sẽ làm như hình bên phải hình 1.3b. Lắp ghép cả 4 biểu thức đường dẫn thành chỉ một mẫu dò, tránh được đánh giá lặp lại các biểu thức đường dẫn ấy.Lý do là các vấn tin *Xquery* sử dụng biểu thức *Xpath,* về căn bản đó là đường dẫn hay mẫu so sánh, trong dạng FLWR để ràng buộc hay tác động vào các biến nút.Tuy nhiên, khác với *Xpath,Xquery* làm nhiều hơn là tính toán một mẫu so sánh đơn giản. Nói cụ thể hơn:

- *Trả lời một truy vấn Xquery có thể gồm đánh giá nhiều mẫu dò trong khi truy vấn Xpathchỉ đánh giá một mẫu dò. Để tối ưu, số lượng các mẫu dò trong đánh giá các truy vấn Xquery lên là cực tiểu để tránh lặp lại cùng một biểu thức đường dẫn.*
- *Mẫu dò có trong các truy vấn Xquery thường gồm nhiều hơn một nút ở kết quả ra, trong khi mẫu dò của một truy vấn Xpathchỉ có một nút kết quả ra.*

- *Đánh giá mẫu dò không phải là phép toán duy nhất để đánh giá các truy vấn Xquery. Để nhận được kết quả truy vấn đúng đắn, các phép toán khác cũng phải được thực hiện sau khi so khớp mẫu dò*
- *Cuối cùng, kết quả truy vấn cần phải được định dạng thành XML theo đặc tả trong mệnh đề Return.*

b) Các truy vấn XML kiểu IR

Các truy vấn XML kiểu chủ yếu được sử dụng để truy vấn dữ liệu XML chứa dữ liệu có giá trị các yếu tố thường liên quan đến văn bản dài. Ví dụ, tài liệu XML trong Hình 1.1a là không phải văn bản dài, vì hầu hết các phần tử của nó, ví dụ như "*Database*", "*Tom*" và "*John*" chỉ bao gồm các văn bản rất ngắn. Tuy nhiên, tài liệu này sẽ trở thànhvăn bản dài, nếu xem xét một số phần tử mới có chứa văn bản dài đã được thêm vào dưới cuốn sách như là phần tử con.

Không giống như các kiểu cơ sở dữ liệu truy vấn XML, thường không có thống nhất tiêu chuẩn ngôn ngữ để thể hiện truy vấn XML kiểu IR.

Bây giờ chúng ta một thời gian ngắn giới thiệu hai lớp chính củatruy vấn XML kiểu IR: truy vấn DB+IR và truy vấn IR-only. Không giống như truyền thống truy vấn kiểu IR thực hiện ởmức chi tiết của các tài liệu, có nghĩa là tất cả các tài liệu liên quan đến truy vấn, những truy vấn XML kiểu IR được thực hiện các chi tiết tại các phần tử XML.

Các truy vấn DB+IR. các truy vấn DB+IR là tăng cường từ các truy vấn dữ liệu kiểu XML Database như là các truy vấn *Xpath*và *Xquery* với các đặc tính của IR. Chẳng hạn tăng cường các truy vấn *Xpath,Xquery* với hàm contains tìm kiếm theo từ khóa kiểu IR. Ví dụ *"//publisher[contains ("Database","Tom")] / @ name".* Nó trả về tên của tất cả các nhà xuất bản mà phần tử con (hoặc hậu duệ) của nó chứa "*Database*" "*Tom*".

Các truy vấn IR-only.Khuôn dạng của các truy vấn IR-only không cứng nhắc như các truy vấn DB+IR: các truy vấn IR-only không có phần cấu trúc và đặc tả phần nội dung như là tập hợp các từ khóa K = {K1;K2;...;Kn} (tương tự như các truy vấn tìm kiếm theo từ khóa trong IR truyền thống). Kết quả dự kiến của các truy vấn IR-only là tập hợp các tổ tiên chung gần nhất của các nút dữ liệu ứng với các từ khóa có trong K, vì tập hợp các tổ tiên chung gần nhất là các phần tử trả lời rõ ràng và liên quan nhất tới từ khóa đã cho.Ví dụ: một truy vấn IR-only {"*Database*","*Tom*"} trên cây dữ liệu hình 1.2asẽ trả về nút *book* dưới nút

*publisher*đầu tiên, mà không trả về nút *publisher* đầu tiên cho dù nó cũng là tổ tiên của "*Database*" và "*Tom*".

Xếp hạng. Ngoài việc tìm kiếm truy vấn ứng cử viên kết quả, truy vấn XML kiểu IR(cả hai DB + IR truy vấn và IR chỉ truy vấn) còn phải xếp hạng các kết quả truy vấn ứng cử viên dựa trên sự liên quan của họ với các truy vấn và trả về cho người sử dụng những kết quả hàng đầu giống như trong kiểu IR truyền thống.

1.4. PHÁT BIỂU BÀI TOÁN

So khớp mẫu thử là thiết yếu trong các truy vấn *Xpath* /*Xquery*.Đây là một trong các bài toán quan trọng nhất khi xử lý truy vấn XML.

Hình thức bài toán so khớp mẫu thử là: tìm trong một cây dữ liệu XML D tất cả các phù hợp thỏa mãn mẫu thử Q. Mẫu thử của một truy vấn *Xpath*chỉ có một nút kết quả ra, còn kết quả ra của một mẫu thử phù hợp là một tập hợp nút. Mẫu thử trong một truy vấn *Xquery* thường có nhiều hơn một nút kết quả ra, và kết quả của mẫu thử phù hợp là tập hợp các nút, như minh họa trong hình 1.3b.

Ngoài mẫu thử phù hợp, một bài toán quan trọng khác trong xử lý truy vấn XML là lọc tài liệu XML, xuất hiện trong các ứng dụng SDI*(Selective Dissemination of Information)*. Thành phần lõi của SDI là bộ lọc tài liệu, so khớp mỗi tài liệu XML *D* từ các nhà xuất bản với một sưu tập các truy vấn thử từ các khách hàng, để các định các truy vấn được đặt mua nào có ít nhất một phù hợp trong *D* hơn là tìm tất cả các phù hợp của khách hàng các truy vấn trong *D*, như yêu cầu của mẫu thử phù hợp.

Do tầm quan trọng của mẫu thử phù hợp ta sẽ tập trung xem xét các kỹ thuật mẫu thử phù hợp. Cụ thể hơn là mẫu thử phù hợp kho tài liệu XML được lưu trữ liên tục. Một cách đơn giản để truy vấn tài liệu XML được lưu trữ liên tục là mô hình triển khai dựa trên bộ nhớ: đầu tiên nạp toàn bộ tài liệu XML từ bộ nhớ ngoài vào dưới dạng cây sau đó thực hiện các truy vấn XML trên cây này. Cách làm này là trực tiếp, nhưng chỉ thực hiện các truy vấn XML trên cây này.Cách làm này là trực tiếp, nhưng chỉ cần thực hiện được với kho tài liệu XML cỡ nhỏ.Không hiệu quả khi cỡ lớn. Để hiệu quả cần phát triển thành *Database-style* mức độ tinh vi hơn.

Gần đây, có các kỹ thuật nâng cao đáng kể hiệu năng xử lý truy vấn trong CSDL XML: đánh chỉ mục theo giá trị và đánh chỉ mục theo cấu trúc.

Ta phân loại các kỹ thuật này thành hai lớp: tiếp cận theo hướng quan hệ (*relational approach*) và tiếp cận theo hướng tự nhiên (*native approach*).

Tiếp cận theo hướng quan hệ: sử dụng ngay các hệ CSDL quan hệ đang có sẵn để lưu trữ và truy vấn dữ liệu XML.

Tiếp cận theo hướng tự nhiên: xây dựng từ các hệ thống chuyên dùng để lưu trữ và truy vấn dữ liệu XML một cách không lựa chọn.

Nhiều nghiên cứu về truy vấn dữ liệu XML giả thiết các truy vấn làm việc trên mô hình dữ liệu XML hình cây.

- *Lý do thứ nhất: mô hình tổng quát hơn, dạng đồ thị sẽ tăng độ phức tạp đáng kể.*
- *Lý do thứ hai: tài liệu XML dạng đồ thị với các thuộc tính ID/IDREF không phổ biến trong thực tế như tài liệu dạng cây.*

CHƯƠNG 2:CÁC THUẬT TOÁN TRUY XUẤTDỮ LIỆU XML

2.1. ĐÁNH CHỈ SỐ DỮ LIỆU XML

Nói chung, việc đánh chỉ số sẵn trên các dữ liệu XML giúp cho việc xử lý truy vấn XML thuận tiện hơn rất nhiều vì nó giúp chúng ta có thể định vị nhanh chóng dữ liệu đích thay vì phải quét toàn bộ dữ liệu. Có 2 kiểu đánh chỉ số XML đó là:

- *Đánh chỉ số dữ liệu giá trị trong tài liệu XML.*
- *Đánh chỉ số cấu trúc của tài liệu XML.*

Ta sẽ xét 2 lớp đánh chỉ số cấu trúc quan trọng: *lược đồ đánh số và lược đồ đồ thị chỉ số.*

2.1.1. Lược đồ đánh số

Một vấn đề quan trọng trong mô hình thuật toán *twig*là xác định các mối quan hệ cấu trúc giữa hai nút trong cây dữ liệu. Ví dụ như câu hỏi cặp nút được gán thẻ A, B trong cây dữ liệu hay hai nút (a,b) có khớp với mẫu đường dẫn "A/B" hay không? Trước hết cần biết có hay không đường dẫn từ a đến b trong cây dữ liệu.

Một phương pháp đơn gian để xác định đó là duyệt cây, duyệt từ cây con gốc A xuống tìm B hoặc ngược lại, từ B lên để A. Tuy nhiên, nói chung phương pháp duyệt cây là kém hiệu quả vì phải qua nhiều nút chẳng liên quan, các nút này có thể rải rác nhiều trong ổ đĩa nên sẽ mất nhiều thời gian cho việc truy cập vào ra.

Một cách tiếp cận khác để có thể xác định là tính bao đóng bắc cầu của cây dữ liệu.Tuy nhiên, kích thước của bao đóng bắc cầu là quá lớn để dùng trong thực tế.Vì vậy cần phương pháp biểu diễn cô đọng tính liên quan giữa hai nút trong cây dữ liệu.

PrePost coding:

Năm 1982 *Dietz* đã đưa ra công trình đầu tiên về lược đồ đánh số cho cây. Đề xuất lược đồ đánh số mà ta sẽ gọi là *Prepost coding*. Mỗi nút trong cây được gán cặp số (*pre; post*), là thứ tự xử lý nút khi duyệt cây theo thứ tự trước (*preorder*) và duyệt theo thứ tự sau (*postorder*).

Sau đó *Zhang* và các đồng nghiệp đã đưa *PrePost coding* vào ứng dụng cho XML: gán nhãn mỗi nút trong cây dữ liệu XML một cặp hai số (*start; end*), cho phép suy ra vị trí của thẻ mở <...> và thẻ đóng </...> của phần tử tương ứng với nút trong tài liệu XML. Dễ thấy rằng (*start; end*) và (*pre; post*) về bản chất là một.

PrePost coding có các tính chất sau:

*Tính chất 1***:** quan hệ tổ tiên hậu duệ. Nút a là tổ tiên của nút b trong cây dữ liệu khi và chỉ khi *a:start< b:start < a:end*.

Trong *PrePost coding*: 1) không gian lưu trữ cặp (*start; end*) là rất khiêm tốn. 2) với sự giúp đỡ của cặp (*start; end*) ta có thể xác định quan hệ tổ tiên hậu duệ giữa hai cặp nút với chi phí thời gian không thay đổi bằng cách sử dụng hai phép so sánh. Ngoài ra, chúng ta có thể xác định mối quan hệ cha con bằng việc kết hợp với *level* của nút.

*Tính chất 2***:** quan hệ cha con. Trong cây dữ liệu, nút a là cha của nút b khi và chỉ khi *a:start< b:start < a:end và a:level + 1 = b:level.*

Ngoài hai trục "/" và "//", *PrePost coding* có thể xử lý tất cả các trục khác được định nghĩa trong *Xpath*bằng thêm mã số cha.

PrePost coding cũng gọi là "*interval coding*", vì cặp 2 số có thể coi như một khoảng và việc xác định mối quan hệ giữa hai nút là kiểm tra quan hệ "chứa trong" giữa các khoảng.

Dewey coding:

Một lược đồ đánh số khác là *Dewey coding*. Nó vốn được xây dựng để đánh chỉ số các thông tin phổ biến.

Năm 2002 Tatarinov áp dụng *Dewey coding* và xử lý truy vấn XML.

Mỗi nút được gán với một vector các số, là đường dẫn ID của nút đến nút (hình 1.2a), vector Dewey của nút 2 là 1.2 và nút 10 là 1.2.7.10.nút a là một tổ tiên của nút b trong cây dữ liệu khi và chỉ khi *a:vector* là tiền tố của *b:vector*.

Ưu điểm của *Dewey coding* so với *PrePost coding* là: *Dewey coding* duy trì dễ hơn khi cập nhập cây dữ liệu XML. *IBM System RX*, *Microsoft SQL Server* và *Oracle DB* dùng *Dewey coding* trong xử lý truy vấn XML:

Theo cảm tính, khi thêm nút mới (hay cây con mới) vào cây dữ liệu, chỉ cần cập nhập lại *vector* Dewey của các nút trong cây con gốc là các em (*following sibling*) của nút mới thêm.Hơn nữa, *ORDPATH coding* – một biến thể của *Dewey coding* tích hợp trong *Microsoft SQL Server 2005*, không cần phải cập nhập vector *ORDPATH* khi chèn thêm.

Duy trì *PrePost coding* không minh bạch.Khi thêm nút mới (hay cây con mới) vào cây dữ liệu, cần cập nhập cặp số (*start; end*) của mọi nút, chỉ trừ các nút trong cây con có gốc là anh (*previous sibling*) của nút mới.

Cặp (*start; end*) cần ít không gian nhớ hơn.

PrePost coding hỗ trợ tốt hơn cho kiểm tra mối quan hệ giữa hai nút, vì phép so sánh là rẻ hơn kiểm tra tiền tố giữa hai vector.Chính vì điều này mà*PrePost coding* được dùng rộng rãi trong xử lý truy vấn XML.

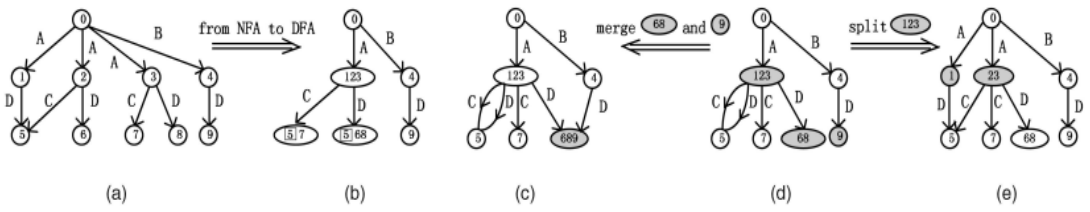
2.1.2. Lược đồchỉ số đồ thị

Lược đồ đồ họa chỉ số còn gọi là "tóm tắt cấu trúc – *structural summaries*".

Tư tưởng cơ bản là: nén đồ thị dữ liệu XML có thể được đánh giá hiệu quả hơn trên *GI* thay cho trên *G*. Khác với lược đồ đánh số, lược đồ chỉ số đồ họa nói chung áp dụng cho đồ thị dữ liệu XML tổng quát. Ta phân loại các lược đồ đồ họa chỉ số thành hai lớp: *P-indexes* dành cho các truy vấn tuyến tính (vấn tin path) và *T-indexes* dành cho các truy vấn twig.

a) Chỉ số đồ thị cho các truy vấn tuyến tính (P-indexes)

Năm 1997 *Goldman* và *Widom* đề xuất Strong *DataGuide*, là một chỉ số đồ họa sớm nhất, tóm tắt tất cả các thông tin đường dẫn trong *G* thành *GI*.*GI* có thể xem như một thiết bị tự động hữu hạn được biến đổi từ *G*. Lưu ý rằng một nút dữ liệu trong *G*, Ví dụ nút 5 trong hình 2.1a, có thể xuất hiện nhiều hơn một nút chỉ số trong *GI* (hình 2.1.b).



Hình 2.1 Từ đồ thị dữ liệu đến đồ thị chỉ số (a)Đồ thị dữ liệu (NFA) (b) Strong DataGuide (DFA) (c) A(1)-index. (d) 1-index (A(2)-index) (e) F&B index.

Nói chung, trong trường hợp xấu nhất kích thước của *GI* dùng Strong *DataGuides* có thể là hàm mũ của kích thước *G*. Tuy nhiên, khi *G* là cây, Strong *DataGuide* rút gọn dùng *1-index* thì kích thước không vượt quá kích thước của *G*.

Năm 2004 *Weigel* tiếp tục mở rộng *Strong DataGuides* thành *Content-Aware DataGuides* để xử lý hiệu quả các truy vấb XML kiểu DB+IR. Đó là tăng cường *DataGuides* với giá trị số làm trước cho giá trị dữ liệu trong tài liệu XML.

Trái với *Strong DataGuides,* nhiều lược đồ chỉ số đồ họa khác đòi hỏi mỗi nút dữ liệu ánh xạ với chỉ một nút chí số.Đặc biệt, nó phân hoạch nút dữ liệu trong *G* thành các lớp tương đương, sau đó nhóm tất cả các nút dữ liệu trong cùng lớp tương đương thành một nút chỉ số. Cuối cùng nó tạo ra một "*tag edge*" giữa mỗi cặp nút chỉ số (I_A,I_B) chỉ khi có 2 nút dữ liệu a thuộc I_A và b thuộc I_B được nối bởi "*tag edge*" như thế trong *G*. Không nút dữ liệu nào ánh xạ với hơn một nút chỉ số. Như vậy, kích thước của *GI* không bao giờ vượt *G*. Kết quả là các truy vấn XML được đánh giá hiệu quả trên *GI*, cho kết quả truy vấn an toàn và chính xác. "An toàn" ở đây có nghĩa là tập hợp các nút trả lời thực sự (tức là các nút dữ liệu được dưa ra nếu xử lý truy vấn trực tiếp trên *G*), "chính xác" có nghĩa là hai tập hợp nút này đúng bằng nhau.

Mỗi lược đồ chỉ số đồ thị sử dụng một chiến lược phân hoạch nút dữ liệu của riêng mình.Năm 1999 *Milo* và *Suciu* đưa ra *1-index,* phân hoạch các nút dữ liệu thành các lớp tương đương dựa trên *B-bisimilarity*: Nếu 2 nút dữ liệu là*B-bisimilarity,* thì chúng có cùng tập hợp đường dẫn gốc (đường dẫn gốc của nút là đường dẫn thẻ từ gốc đến nút – hình 2.1d).Lưu ý rằng nếu *G* là cây thì*1-index* cũng là cây. Ví dụ, xét đồ thị *G* hình cây, hình 2.1a, nhưng gỡ bỏ cạnh (2, 5).*1-index* của *G* sẽ là cây như trong hình 2.1d, nhưng {5} hòa nhập vào {6, 8}.*1-index* hình cây như thế cũng có thể xem như một cây.

Đáng tiếc là mặc dù kích thước của *GI* dùng *1-indexes* không quá kích thước của đồ thị *G*, nhưng trong nhiều trường hợp *GI* vẫn còn quá lớn để áp dụng được trong thực tế.

Để giảm kích thước của *1-index*, năm 2002 *Kaushik* tổng quát hóa *1-index* thành *A(k)-index,* phân hoạch các nút dữ liệu thành các lớp tương đương theo*k-bisimilarity*:

Hai nút dữ liệu là *k-bisimilar,* thì chúng có cùng tập hợp incoming *k-paths,* ở đây *k-path* là đường dẫn thẻ có độ dài $\leq k$. *1-index* là trường hợp riêng của*A(k)-index* khi k đủ lớn.

Ưu điểm của *A(k)-index* so với *1-index* là: *A(k)-index* nói chung có kích thước nhỏ hơn *1-index,* vì *A(k)-index* đưa nhiều nút dữ liệu hơn vào cùng nút chỉ số. Giá trị k càng nhỏ thì kích thước chỉ số càng nhỏ. Tuy nhiên, ưu điểm này

của*A(k)-index* trả giá bằng độ kém chính xác truy vấn: nó chỉ chính xác cho các truy vấn*p-path* với p ≤ k. Các truy vấn *p'-path* p' > k, thì là an toàn nhưng không chính xác. Hình 2.1c minh họa *A(1)-index*. Chú ý rằng *1-index* trong hình 2.1d cũng là *A(2)-index*. Hai nút chỉ số {6, 8} và {9} trong hình 2.1d được hòa nhập thành một nút chỉ số {6, 8, 9} trong hình 2.1c, vì các nút 6, 8, 9 có cùng tập hợp*incoming 1-path* {D}.

Mặc dù các truy vấn *1-path* ví dụ "/A" và "/D" là chính xác trên *A(1)-index*, một số truy vấn *2-pat* như "/A/D" và "/B/D" chỉ an toàn chứ không chính xác. Đánh giá "/A/D" trên A(1)-index sẽ trả về các nút chỉ số {5} và {6, 8, 9}, nhưng nút 9 trong G không khớp "/A/D". Do đó, với các truy vấn *p'-path*, cần thêm bước *postvalidation* nếu yêu cầu kết quả truy vấn phải chính xác. Bước *postvalidation* này kiểm tra các nút dữ liệu dự bị thông qua *incoming path* trong G, vì vậy cần thêm chi phí thời gian. Việc chọn k phù hợp là rất quan trọng trong áp dụng thực tế. K nhỏ sẽ kém hiệu quả với các truy vấn *long-path*, k lớn thì chỉ số quá lớn, tăng chi phí của cả truy vấn *short-path* và *long-path*.

Năm 2003 *Chen* tổng quát hóa *A(k)-index* thành *D(k)-index* tự thích nghi, sử dụng các giá trị k khác nhau (chứ không phải một k duy nhất) cho các nút chỉ số khác nhau.

Năm 2002 *Chung* đề xuất *Adaptive Path index* cho dữ liệu XML (APEX). Đó là một lược đồ chỉ số đồ họa tự thích nghi khác với động cơ và chức năng giống *D(k)-index.*

Năm 2004 *He* và *Yang* tiếp tục cải tiến *D(k)-index,* dùng 2 indexes tinh vi hơn, *M(k)-index* và *M*(k)-index*.

b) Index Graphs bao trùm các truy vấn Twig (T-Indexes)

P-indexes chỉ xử lý các truy vấn tuyến tính.Trả lời các truy vấn *twig* tổng quát đòi hỏi thêm bước nối đường dẫn trên các nút dữ liệu mà truy vấn *path* trên *P-index.*

Năm 2001 *ToXin* đã sử dụng *DataGuide* để đánh chỉ số đường dẫn và sau đó triển khai nối đường dẫn theo tiếp cận *Edge.*

Để tránh bước nối đường dẫn, cần phát triển *T-indexes,* sao cho nó có thể trực tiếp xử lý các truy vấn twig tổng quát, nói chung có kích thước chỉ số lớn hơn *P-indexes.*

c) Tổng kết về lược đồ chỉ số đồ thị

Các lược đồ chỉ số đồ họa đã phát triển từ *P-indexes* chỉ xử lý các truy vấn path thành *T-indexes* kích thước lớn hơn, có thể xử lý các truy vấn twig tổng quát.

P-indexes và *T-indexes* từ chỗ là các chỉ số chính xác, có kích thước lớn (*1-index* với *P-indexes và F&B-index* với *T-indexes*) thành *safe-only* đánh chỉ số có kích thước nhỏ hơn (*A(k), D(k), M(k), APEX idexes* với *P-indexes* và *(F+B)i-index* với *T-indexes*).

Không có lược đồ nào vượt trội các khác vì luôn có cân bằng các yếu tố giữa kích thước chỉ số và khả năng trả lới truy vấn.

Safe-only đánh chỉ số có kích thước chỉ số nhỏ hơn, nhưng lại cần bước *postvalidation* khá tốn kém để nhận được kết quả truy vấn chính xác khi các truy vấn nằm ngoài phạm vi xử lý.

2.1.3. Tổng kết

Lược đồ đánh số và lược đồ đồ họa đánh số không phải là hai công cụ độc lập loại trừ lẫn nhau trong đánh chỉ số cấu trúc của dữ liệu XML. Chúng đóng vai trò khác nhau trong trả lời các truy vấn XML: lược đồ đồ họa đánh số dùng để lựa chọn đường dẫn, còn lược đồ đánh số dùng để nối đường dẫn. Chúng kết hợp với nhau để tăng năng suất xử lý.

2.2. XỬ LÝ TRUY VẤN XML

XML (*Extensible Markup Language*) đang nổi lên như một tiêu chuẩn quốc tế để trao đổi thông tin giữa các ứng dụng *World Wide Web* khác nhau. Hiện nay nhu cầu truy vấn dữ liệu XML ngày càng tăng nhằm phục vụ cho việc lưu trữ lượng dữ liệu lớn một cách hiệu quả. Một trong những vấn đề quan trọng trong xử lý truy vấn XML là chọn cách phù hợp với mô hình, là việc tìm kiếm dữ liệu XML trên cây với tất cả các thuật toán truy vấn *twig* (hoặc đường dẫn) với mô hình Q.

Trong cuộc khảo sát này, chúng em xem xét, phân loại và so sánh chính kỹ thuật *twig* với các thuật toán truy vấn trên XML. Cụ thể, chúng em xem xét hai lớp thuật toán truy vấn XML lớn: cách tiếp cận quan hệ và cách tiếp cận tự nhiên. Cách tiếp cận quan hệ sử dụng trực tiếp hệ thống cơ sở dữ liệu quan hệ hiện có để lưu trữ và truy vấn dữ liệu XML, nó cho phép sử dụng tất cả các kỹ thuật quan trọng đã được phát triển trong các cơ sở dữ liệu có sẵn, còn trong các tiếp cận tự nhiên việc lưu trữ và xử lý truy vấn hệ thống phù hợp cho dữ liệu XML được phát triển từ đầu để cải thiện hơn nữa hiệu suất truy vấn XML.

Ý nghĩa của công việc truy vấn và quản lý dữ liệu XML phát triển theo hướng tích hợp cách tiếp cận theo hướng quan hệ với cách tiếp cận theo hướng tự nhiên sẽ dẫn đến hiệu suất cao hơn và giảm đáng kể chi phí sử dụng dữ liệu trong hệ thống.

2.2.1. Tiếp cận theo hướng quan hệ

Các hệ cơ sở dữ liệu hiện nay phần nhiều là cơ sở dữ liệu quan hệ (*IBM DB2, Microsoft SQL Server, Oracle DB...*). Vì thế có nhiều nghiên cứu về lưu trữ và truy vấn Dữ liệu XML trong CSDL quan hệ.

2.2.1.1. Cách tiếp cận cạnh

a) Cách tiếp cận cạnh cơ sở

Năm 1999 *Florescu* và *Kossmann* đề nghị một cách tiếp cận đơn giản để băm dữ liệu XML vào các quan hệ. Tất cả các cạnh trong cây dữ liệu XML có các cạnh được gán nhãn (*edge-labeled*) được đặt vào chỉ một bảng quan hệ cung (lược đồ hình 2.2a – dữ liệu trong bảng điềntheo cây dữ liệu XML hình1.2b)

Label	Source	Target	Flag	Value
publisher	1	2	Element	null
name	2	3	Attribute	MIT Press
address	2	4	Value	Cambridge
publisher	1	9	Element	null
...	...	...	...	...

(a)

Label	Start	End	Level	Flag	Value
publisher	2	20	1	Element	null
name	3	5	2	Attribute	MIT Press
address	6	8	2	Value	Cambridge
publisher	21	38	1	Element	null
...	...	...	...	...	...

(b)

Hình 2.2 Ví dụ về một (a) Bảng các cạnh và (b) Bảng các nút.

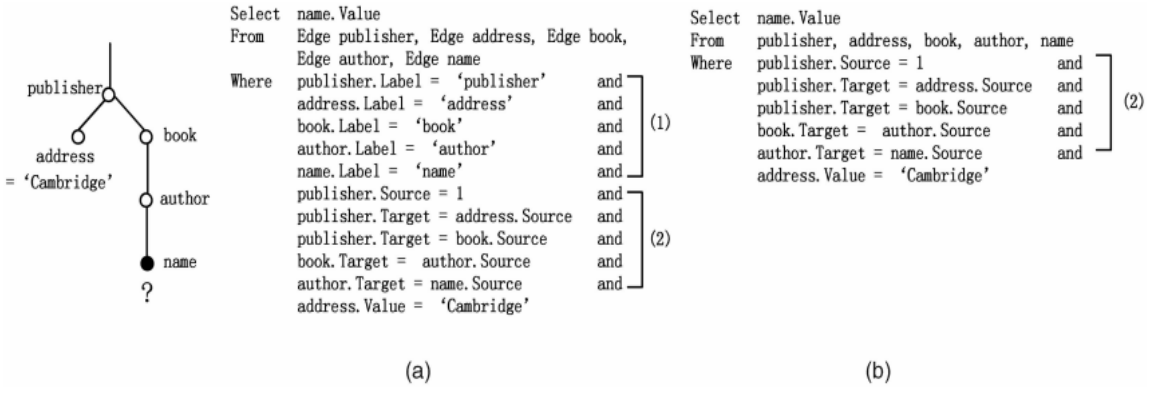
Ý tưởng then chốt sử dụng cặp thuộc tính (*Source; Target*), là hai đầu mút của mỗi cạnh.*Label* là nhãn của cạnh, *Flag* và *Value* là kiểu và giá trị của nút đích của cạnh. Hai cạnhA và B có thể nối khi và chỉ khi *A: Target = B: Source*.

Dựa trên tính chất này, dễ dàng chuyển đổi các truy vấn XML không chứa trục "//" thành các truy vấn SQL (hình 2.3a). Tính toán các truy vấn SQL này bao gồm hai bước chính:

Bước 1: là lựa chọn cạnh (phần1), truy lục các cạnh dữ liệu cho mỗi nhãn trong truy vấn. Một chỉ số cụm (*clustered index*) xây dựng trước trên *Label* có thể tăng đáng kể tốc độ xử lí của bước này. Hơn nữa, chỉ số cụm xây dựng trước trên *Value* làm tăng hiệu quả truy lục của các cạnh dữ liệu, ví dụ cạnh "*address*" với giá trị "*Cambridge*" hình 2.3.

Bước 2: là nối cạnh (phần2), nối các cạnh kề nhau nhận được sau bước 1. Bước này thực thi hiệu quả hơn với các chỉ số trên (*Source ; Target*).

b) Cách tiếp cận nhị phân



Hình 2.3 Cách tiếp cận cạnh: truy vấn SQL cho "/publisher[address = "Cambridge"]/book/author/name"(a) Cách tiếp cận cung cơ bản (b) Cách tiếp cận nhị phân.

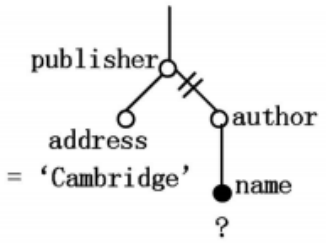
Có thể tránh thực hiện phần 1 trong cách tiếp cận cạnh cơ sở bằng cách tiếp cận nhị phân do *Florescu* và *Kossmann* đề nghị. Đó là nhóm các cạnh trong bảng các cạnh theo*Label* của chúng và tạo ra một bảng cho mỗi *Label* riêng biệt. Mỗi bảng có lược đồ là (*Source, Target, Flag, value*), bỏ Label khỏi lược đồ cạnh.Hình 2.3 là một ví dụ.

Ngoài ra, cách tiếp cận nhị phân còn tiết kiệm không gian lưu trữ vì thuộc tính Label không cần nữa.

Nhược điểm của cách tiếp cận cung là:

1) *Chứa một số phép nối. Số phép nối là số nút truy vấn trong truy vấn twig trừ đi 1. Như vậy, cách tiếp cận này có thể không hiệu quả khi xử lý các truy vấn twig lớn hơn.*
2) *Có thể lỗi khi xử lý các truy vấn với "//" vì khó xác định số tên của các thẻ giữa A và B. Để xác định quan hệ tổ tiên hậu duệ giữa các nút dữ liệu, cần tính một phần bao đóng bắc cầu của cây dữ liệu bằng cách đưa ra các đệ quy SQL các truy vấn. Việc tính trước và biểu diễn sẵn bao đóng bắc cầu có thể tránh chi phí tính toán nhưng mất chi phí cao về không gian lưu trữ.*

2.2.1.2. Cách tiếp cận nút




```
Select  name.Value
From    Node publisher, Node address, Node author, Node name
Where   publisher.Label = 'publisher'              and   ┐
        address.Label = 'address'                  and   │ (1)
        author.Label = 'author'                    and   │
        name.Label = 'name'                        and   ┘
          publisher.level = 1                      and   ┐
        publisher.start < address.start            and   │
          address.start < publisher.end            and   │
          publisher.level + 1 = address.level      and   │
        publisher.start < author.start             and   │ (2)
          author.start < publisher.end             and   │
        author.start < name.start                  and   │
          name.start < author.end                  and   │
          author.level + 1 = name.level            and   ┘
        address.Value = 'Cambridge'
```

Hình 2.4 Cách tiếp cận nút: Truy vấn SQL cho *"/publisher[address="Cambridge"]//author/name"*

Năm 2002 *Zhang* phát triển cách tiếp cận nút, trong đó tất cả các nút (tức là phần tử và nút thuộc tính) trong cây dữ liệu XML nhãn nút, được lưu trữ trong bảng quan hệ nút, có lược đồ trong hình 5b với dữ liệu điền theo dữ liệu XML trong hình 2a. Ý tưởng then chốt ở đây là sử dụng bộ ba thuộc tính (*Start, End, Level*). Các truy vấn chứa trục "//" được trả lời hiệu quả bằng cách sử dụng cặp (*Start, End*). *Level* sẽ kết hợp với (*Start, End*) để trả lời các truy vấn chứa trục "/".

Dựa trên các tính chất 1 và 2 trong mục 2.2, dễ dàng chuyển đổi các truy vấn chứa cả hai trục "/" và "//" thành các truy vấn SQL (xem hình 2.4).

Tượng tự như cách tiếp cận cạnh, tính toán các truy vấn SQL gồm hai bước: chọn nút (phần 1) và nối nút (phần 2).

Phần 2 nối các nút dữ liệu nhận được từ phần 1 thông qua (*Start, End, Level*).Giống như trong cách tiếp cận cạnh, thực thi phần 1 trong cách tiếp cận nút có thể tránh được khi sử dụng sự biến đổi tương tự như cách tiếp cận nhị phân. Không giống cách tiếp cận cạnh, cách tiếp cận nút hỗ trợ các truy vấn có trục "//" một cách hiệu quả. Nhưng giống như cách tiếp cận cung, cách tiếp cận nút hỗ trợ các truy vấn có trục "//" một cách hiệu quả. Nhưng giống như cách tiếp cận cạnh, nó có thể cần một số phép toán nối, làm kém hiệu quả các truy vấn *twig* cỡ lớn. Số phép nối bằng số nút truy vấn trong truy vấn *twig* trừ đi 1.

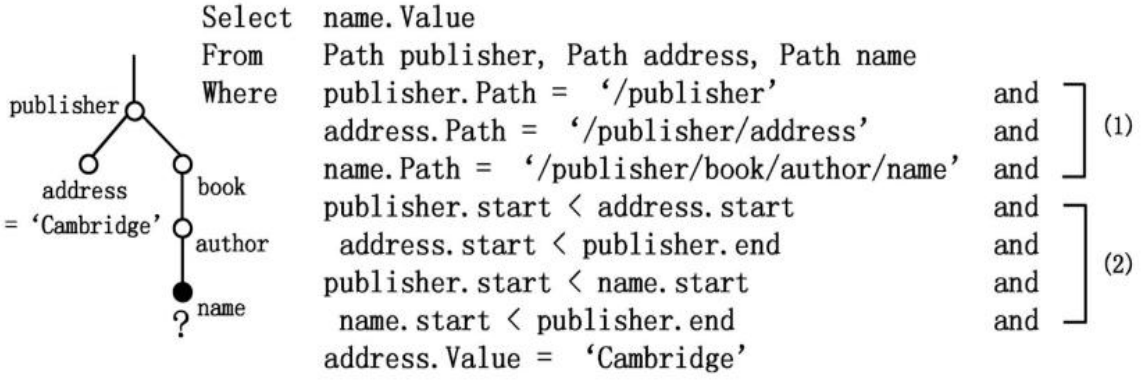
2.2.1.3. Cách tiếp cận cụ thể hoá đường dẫn

a) Cách tiếp cận cụ thể hóa đường dẫn cơ bản

Để giảm số nút nối, *Yoshikawa* đề nghị cách tiếp cận cụ thể hóa đường dẫn, trong đó lưu trữ các nút trong của cây dữ liệu XML nhãn nút vào một bảng quan hệ đường dẫn.Hình 2.2a là lược đồ của bảng quan hệ đường dẫn và điền dữ liệu bằng dữ liệu XML trong hình 2.2.a. Bảng quan hệ đường dẫn rất giống bảng nút.Sự khác nhau là không lưu thẻ của mỗi nút trong thuộc tính Label, cách tiếp cận cụ thể hóa đường dẫn lưu thẻ đường dẫn từ gốc tới mỗi nút (gọi là *root path*) vào thuộc tính đường dẫn.

Sử dụng thuộc tính đường dẫn, cách tiếp cận cụ thể hóa đường dẫn có thể trả lời các truy vấn *twig* hiệu quả theo đơn vị đường dẫn chứ không theo đơn vị cạnh.

Cho một truy vấn *twig*, đầu tiên cách tiếp cận cụ thể hóa đường dẫn phân chia nó thành nhiều truy vấn đường dẫn gốc tới lá sau đó nối kết quả của các truy vấn đường dẫn, như hình minh họa trong hình 2.5.



Hình 2.5 Cách tiếp cận cụ thể hóa đường dẫn cơ sở: truy vấn SQL *"/publisher[address="Cambridge"]/book/author/name"*.

Đánh giá các truy vấn SQL gồm hai bước chính: lựa chọn đường dẫn (phần 1) và nối đường dẫn (phần 2).

Phần 1 sử dụng các đường dẫn gốc của nút là (*address* và *name*) và của nút phân nhánh (*publisher*) trong truy vấn twig để truy lục các nút dữ liệu tương ứng từ cây dữ liệu.

Phần 2 nối các dữ liệu nhận từ phần 1 thông qua (*Start, End, Level*).

Cách tiếp cận cụ thể hóa đường dẫn có hai đặc tính sau:

1) *Nó gồm ít phép nối (phần 2) hơn cách tiếp cận nút vì đường dẫn cụ thể hóa trả lời các truy vấn twig tính theo đơn vị đường dẫn chứ không phải theo đơn vị cạnh. Trong hình 2.5, cách tiếp cận nút cần nối 5 nút truy vấn, trong khi cách tiếp cận cụ thể hóa đường dẫn chỉ cần nối 3 nút truy vấn.*
2) *Tương tự cách tiếp cận nút, cách tiếp cận cụ thể hóa đường dẫn có thể hỗ trợ các truy vấn có trục "//" bằng cách sử dụng hàm Optional String Pattern Matching (OSPM) – hàm "LIKE" có trong SQL. Ví dụ, để trả lời câu truy vấn "/publisher//name", ta có thể dùng "name.Path LIKE "/publisher%name"".*

Bên cạnh đó *Xrel, Xparent* và *MonnetDB* cũng sử dụng phương pháp tiếp cận cụ thể hóa đường dẫn, trong đó phần 1 được thực hiện nhưng phần 2 thực hiên khác. *Xparent* thiết kế đường dẫn thay thế (*Start, End, Level*) trong hình 2.6a với (*Source, Target*) như trong cách tiếp cận canh. Để triển khai hiệu quả phần 2 khi có trục "//", tất cả các cặp nút trong cây dữ liệu cần được tính trước và lưu trữ trong bảng *Ancestor*. Điều này có thể dẫn đến chi phí lưu trữ cao.

Path	Start	End	Level	Flag	Value
/publisher	2	20	1	Element	null
/publisher/@name	3	5	2	Attribute	MIT Press
/publisher/address	6	8	2	Value	Cambridge
/publisher	21	38	1	Element	null
...	...	...	...	...	...

(a)

ReversedPath	ORDPATH	Flag	Value
/publisher	1.1	Element	null
/@name/publisher	1.1.1	Attribute	MIT Press
/address/publisher	1.1.3	Value	Cambridge
/publisher	1.3	Element	null
...	...	...	...

(b)

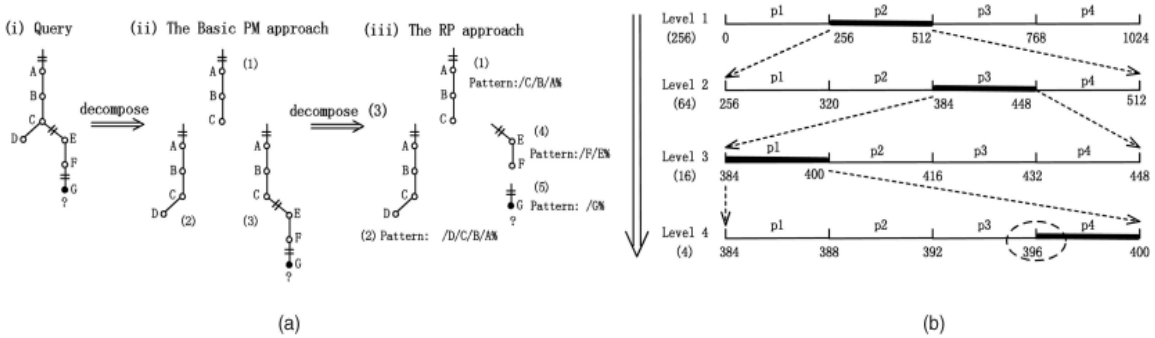
Hình 2.6 (a) bảng đường dẫn và (b) bảng đường dẫn ngược

Cách tiếp cận cụ thể hóa đường dẫn giảm số phép nối trong phần 2, nhưng với giá phải trả là tăng độ phức tạp của phép toán chọn đường dẫn trong phần 1.SQL có thể hỗ trợ hiệu quả so khớp mẫu chính xác ("=") nhờ *B+-tree indexes* làm sẵn trên các sâu ký tự. Tuy nhiên, các chỉ số *B+-tree* không hỗ trợ hiệu quả hàm "*LIKE*" (mẫu có nhiều "%", thì có thể phải quét nhiều xâu không liên quan).Cho nên, cách tiếp cận cụ thể hóa đường dẫn có thể không hỗ trợ hiệu quả các truy vấn với nhiều trục "//".

Một hạn chế khác của cách tiếp cận cụ thể hóa đường dẫn có thể dẫn tới trả lời truy vấn không chính xác khi có đệ quy trong dữ liệu XML. Xét truy vấn đường dẫn *"//A[B//C]"* trên đường dẫn dữ liệu đệ quy *"a1-b1-a2-c1"*. Mặc dù kết quả truy vấn chỉ là a1, cách tiếp cận cụ thể hóa đường dẫn trả về cả hai a1 và a2, vì phép chọn đường dẫn bằng *"%A"* trả về *{a1, a2}*, phép chọn đường dẫn bằng *"%A/B%C"* trả về *{c1}*, và phép nối đường dẫn *{a1, a2}* và *{c1}* thành *{a1, a2}*.

b) Cách tiếp cận được dẫn ngược

Năm 2004 *Pal* đề xuất cách tiếp cận đường dẫn ngược để khắc phục nhược điểm chính của cách tiếp cận hóa đường dẫn.Cách tiếp cận đường dẫn ngược sử dụng lược đồ quan hệ trong hình 2.2b.Ý tưởng then chốt là lưu trữ các đường dẫn ngược của các nút dữ liệu trong thuộc tính đường dẫn ngược.Hơn nữa, cách tiếp cận đường dẫn ngược sử dụng thuộc tính *ORDPATH* thay cho (*Start, End, Level*) của cách tiếp cận cụ thể hóa đường dẫn.*ORDPATH* coding là một biến thể của *Dewey coding*. Giống như *PrePost coding*, nó có thể dùng để xác định mối quan hệ giữa các nút.



Hình 2.7 (a) Cách tiếp cận đường dẫn ngược(b) Cách tiếp cận BLAS: Plabel ("/p2/p3/p1/p4")=396.

Hình 2.7a minh họa cách tiếp cận đường dẫn ngược trả lời các truy vấn twig chứa nhiều trục "//".

Bước 1 là lựa chọn đường dẫn: twig được phân chia thành 3 đường đi, giống như trong cách tiếp cận cụ thể hóa đường dẫn cơ sở. Đường dẫn 3 chứa 3 trục "//". Cách tiếp cận cụ thể hóa đường dẫn cơ sở sử dụng *"%A/B/C%E/F%G"* làm mẫu tìm kiếm trên thuộc tính đường dẫn để truy lục các nút dữ liệu tương ứng.Như đã nói trên, không dễ cài đặt trực tiếp kiểu so khớp mẫu này. Cách tiếp cận đường dẫn ngược sẽ phân rã tiếp theo đường dẫn 4 thành đường dẫn 4 và 5, mỗi cái chỉ chứ 1 trục "//" tại điểm đầu. Như vậy, ta có thể sử dụng *"/F/E%"* và *"/G%"* làm mẫu so khớp trên thuộc tính đường dẫn ngược.Mục đích ở đây là tìm kiếm xâu có tiền tố đã cho.Phép tìm này cài đặt hiệu quả hơn phép so khớp *"LIKE"* với nhiều "%".Ở bước cuối là phép nối đường dẫn, cách tiếp cận đường dẫn ngược sẽ nối các kết quả của các truy vấn đường dẫn sử dụng thuộc tính *ORDPATH*.

Cách tiếp cận đường dẫn ngược luôn cho kết quả truy vấn đúng, ngay cả khi có đệ quy trong dữ liệu XML, vì mỗi *path* sinh bởi thủ tục phân hủy đường dẫn chỉ gồm một trục "//" và chỉ tại điểm đầu.

Xét ví dụ đã sử dụng cách tiếp cận cụ thể hóa đường dẫn cơ bản, đường dẫn truy vấn *"//A[/B//C]"* áp dụng trên đệ quy đường dẫn dữ liệu *"a1-b1-a2-c1"*. Nó trả về kết quả truy vấn "a1" bởi vì nó sử dụng 3 phép chọn đường dẫn "/A%", "/B/A%" và *"/C%"* hơn là chỉ dùng 2 phép chọn đường dẫn *"%A"* và *"%A/B%C"* như trong cách tiếp cận cụ thể hóa đường dẫn cơ bản.

Cách tiếp cận đường dẫn ngược đã được tích hợp vào *IBM System RX*, *Microsoft SQL 2005* và *Oracle DB* và cũng được đề nghị độc lập bởi *Chen* năm 2005.

c) Cách tiếp cận BLAS

Cách tiếp cận đường dẫn ngược đưa việc so khớp *"LIKE"* tổng quát về một nhiệm vụ dễ hơn là so khớp tiền tố xâu ký tự - *string prefix matching* (SPM).Tuy nhiên không xét cách cài đặt hiệu quả SPM.Hình như dành việc này cho SQL. Năm 2004 *Chen* ngoài việc đề xuất cách tiếp cận đường dẫn ngược một cách độc lập còn đưa ra cách tiếp cận *BLAS* tinh vi hơn để cài đặt *SPM* hiệu quả.

Lược đồ của bảng *BLAS* như sau:

Ý tưởng then chốt của *BLAS* là mã hóa mỗi xâu đường dẫn ngược thành một giá trị *Plabel*.Cách mã hóa này được minh họa trong hình 2.6b. Trong ví dụ, ta giả thiết tài liệu XML có chỉ 4 thẻ *names* khác nhau, p1 đến p4. Ở mức 1, bốn thẻ chia đoạn [0, 1024] thành 4 đoạn bằng nhau, cùng độ dài 1024/4 =256. Tương tự, ở mức 2, bốn thẻ lại chia mỗi đoạn sau mức 1 thành 4 đoạn cùng độ dài 256/4 =64, cứ thế tiếp tục. Như vậy ta có *Plabel*(*"/p2/p3/p1/p4"*).

= 256*(2-1)+64*(3-1)+16*(1-1)+4*(4-1)

=396.

Một tính chất hay của *Plabel* là mọi xâu ký tự cùng tiền tố được gom tụ vào các vị trí kề nhau trong *Plabel*. Như vậy, có thể truy lục tất cả các xâu đường dẫn ngược với tiền tố đã cho thông qua một câu truy vấn SQL nếu xây dựng sẵn một chỉ số cụm *B+-tree* trên thuộc tính *Plabel* trong bảng *BLAS*. Ví dụ để truy lục mọi đường dẫn đảo ngược với tiền tố *"/p2/p3/"*, *BLAS* tính cận dưới *("/p2/p3") = Plabel("/p2/p3/")* = 384 và cận trên *("/p2/p3") = Plabel("/p2/p4/")* = 448 sau đó đưa ra câu truy vấn SQL để truy lục mọi đường dẫn đảo ngược có *Plabel* nằm trong [384, 448]. Hình 2.8 minh họa ý tưởng này.

```
Select G.Value
From   BLAS C, BLAS D, BLAS F, BLAS G
Where  lower_bound( '/C/B/A' )   <= C.PLabel < higher_bound( '/C/B/A' )   and ┐
       lower_bound( '/D/C/B/A' )<= D.PLabel < higher_bound( '/D/C/B/A' ) and  │ (1)
       lower_bound( '/F/E' )     <= F.PLabel < higher_bound( '/F/E' )     and  │
       lower_bound( '/G' )       <= G.PLabel < higher_bound( '/G' )       and ┘
       C.start < D.start   and    D.start < C.end                          and ┐
         C.level + 1 = D.level                                             and  │ (2)
       C.start < F.start   and    F.start < C.end                          and  │
       F.start < G.start   and    G.start < F.end                               ┘
```

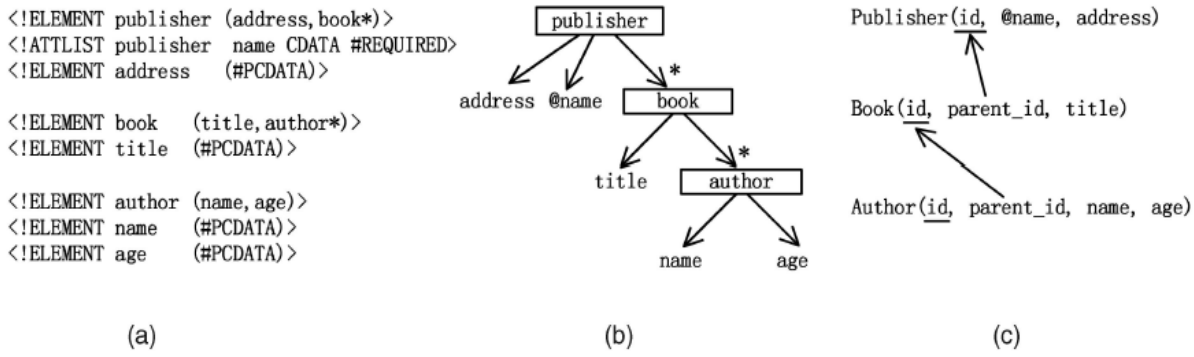
Hình 2.8 Cách tiếp cận BLAS: SQL cho truy vấn twig trong hình 2.7a.

Phép dọn đường dẫn trong phần 1 được cài đặt hiệu quả nhờ các truy vấn xếp loại trong SQL, còn trong phần 2 phép nối các nút dữ liệu nhận được từ phần 1 qua (*Start, End, Level*).

2.2.1.4. Cách tiếp cận DTD

Tất cả các phương pháp tiếp cận trên giải quyết việc lưu trữ và truy vấn dữ liệu XML (*schemaless*). Trong nhiều ứng dụng thực tế, dữ liệu XML cũng phù hợp với một "lược đồ" đến mức độ nào đó bởi vì một thỏa thuận chung trên các lược đồ của dữ liệu sẽ tạo điều kiện thuận lợi cho việc trao đổi dữ liệu giữa các ứng dụng khác nhau. Điều đó sẽ cung cấp thêm cơ hội cho lưu trữ nhỏ gọn hơn và hiệu quả hơn trong việc truy vấn dữ liệu XML. Cách tiếp cận DTD (*Document Descriptors Type*) mà chúng ta xem xét trong phần này sử dụng thông tin giản đồ quan trọng như vậy.

Lược đồ XML có thể được mô tả bằng cách sử dụng các DTD hay XML Schemas, mà chủ yếu là để mở rộng DTD. Bây giờ chúng ta giới thiệu qua các vấn đề cơ bản của các DTD. DTD là một tập hợp các câu lệnh chỉ định: 1) mối quan hệ giữa các yếu tố XML và phần tử con hoặc các thuộc tính của nó. 2) kiểu dữ liệu của các phần tử XML hoặc các thuộc tính. Hình 2.9acho một ví dụ của một tài liệu DTD, mà ngữ nghĩa có thểđược giải thích bằng cách sử dụng cây DTD trong hình2.9b.



Hình 2.9: Một DTD và các giản đồ quan hệ của nó. (a) Một tài liệu DTD. (b) Một cây DTD. (c) Giản đồ quan hệ.

Biểu tượng "*" Liên kết với một phần tử trong một DTD ngụ ý rằng nhiều bản sao của nguyên tố này có thể hiện diện dưới cha mẹ của nó phần tử.Ví dụ, một yếu tố của nhà xuất bản có thể có nhiều cuốn sách là phần tử con.

Không giống như các phương pháp tiếp cận khác (như cách tiếp cận nút, cạnh, và đường hóa dẫn), tạo ra cùng một lược đồ quan hệ (bảng cạnh, nút, đường

dẫn, …) cho tất cả các loại dữ liệu XML, bất kể cấu trúc của chúng, các phương pháp tiếp cận DTD tạo ralược đồ quan hệ khác nhau cho các DTD khác nhau. Hãy xem xét ví dụ trong hình 1.9c, nơi mối quan hệ được tạo racác yếu tố gốc (*publisher*) và cho tất cả các yếu tố (*book*và *author*).Mối quan hệ từng có một thuộc tính *id* như là chìa khóa của nó, vàmỗi quan hệ phần tử có một thuộc tính *parent_id*, mà là mộtkhóa ngoài tham chiếu đến bảng *parent-element* của nó. Lưu ý rằng: (*id, parent_id*) trong mỗi quan hệ phần tử đại diện cho một cạnh,tương tự như (*Source; Target*) trong bảng *Edge*.

Cách tiếp cận DTD biến đổi các truy vấn XML vào truy vấn SQL dựa trên các thông tin lược đồ trong cây DTD*:*

1) *Đối với một "/" lấy A/B, đầu tiên kiểm tra xem A có là phụ huynh của B trong cây DTD không. Nếu không, thì A/B là truy vấn không hợp lệ.Đúng thì mối quan hệ A và B được nối bằng cách sử dụngA.id = B:parent_id, tương tự như A.Target = B.Source trong phương pháp tiếp cận cạnh. Hình 2.10 đưa ra một ví dụ.*

2) Đối với trục "//" nhận A//B, đầu tiên nó kiểm tra xem A có là tổ tiên của B trong cây DTD không. Nếu không, thì A//B là một truy vấn không hợp lệ. Nếu đúng,quan hệ A và B và tất cả các mối quan hệ giữa chúng (mà có thểđược tìm thấy trong cây DTD) được tham gia bằng cách sử dụng trục "/". Ví dụ, truy vấn SQL trong hình.2.10 cũng là "=publisher[address =”Cambridge”]//author/name”bởi vìcây DTD trong hình2.10b ngụ ý rằng cuốn sách duy nhất có thểxuất hiện giữa các nhà xuất bản và tác giả. Lưu ý rằng đối với một trong hai A/B hoặc A//B, tham gia các hoạt động giữa các mối quan hệ A và Bcó thể tránh được nếu B đã được inlined vào mối quan hệ A như là mộtthuộc tính hơn là được lưu trữ độc lập như là mộtmối quan hệ.

```
Select Author.name
From   Publisher, Book, Author
Where  Publisher.id = Book.parent_id    and ]
       Book.id = Author.parent_id       and ] (2)
       Publisher.address =  'Cambridge'
```

Hình 2.10: Các phương pháp tiếp cận DTD, truy vấn SQL*:*
"/publisher[address"Cambridge"]/book/author/name" ("/publisher[address ="Cambridge"]//author/name").

So với các phương pháp tiếp cận khác, phương pháp tiếp cận DTD có thể làm giảm đáng kể số lượng tham gia.

Mặt khác, cách tiếp cận DTD có thể tác động nhiều hơn với các phương pháp tiếp cận nút hoặc cách tiếp cận đường hóa dẫn, vì nóbiến đổi mỗi trục "//" vào một loạt các trục "/".Tuy nhiên, theo ghi nhận của *Krishnamurthy* và các cộng sự, sự lãng phí có thể được giảm nhẹ bằng cách áp cụng phương pháp tiếp cận DTD với các chương trình số, như trong cách tiếp cận nút và cách tiếp cận đường hóa dẫn, có nghĩa là gán các thuộc tính (*Start; End; Level*) vào quan hệ. Ví dụ, khi *Publisher* và *Author*trong hình 2.9c được tăng cường theo cách này:

“//publisher[address = “Cambridge”]//author/name”

chỉ yêu cầu một tham gia (giữa *Publisher* và *Author*) chứ không phải hai tham gia, như trong hình 2.10.

Từ những điều trên, chúng ta có thể thấy rằng bằng cách sử dụng các lược đồ thông tin trong phương pháp tiếp cận DTD, các phương pháp tiếp cận DTD có có hiệu suất tốt hơn so với các phương pháp tiếp cận khác, đặc biệt khi số lượng trục “//” trong các truy vấn nhỏ được tăng lên rất nhiều.

2.2.1.5. Tổng kết

Cách tiếp cận quan hệ lưu trữ dữ liệu XML trong cơ sở dữ liệu quan hệ và biến đổi các truy vấn twig thành các truy vấn dữ liệu quan hệ trong SQL. Với cách tiếp cận này, tất cả các quá trình truy vấn và đánh giá có thể được đẩy vào trong đánh giá truy vấn. Chú ý rằng: khi dữ liệu XML không thể lược đồ hóa, cách tiếp cận cụ thể hóa đường dẫn có lợi thế so với cung và cách tiếp cận nút bởi vì:

- Nó hỗ trợ “//” nên các truy vấn có hiệu quả.
- Đòi hỏi ít phép nối hơn.

2.2.2. Tiếp cận theo hướng tự nhiên

Mặc dù cách tiếp cận quan hệ đơn giản và cài đặt trực tiếp, nhưng nó không có được hiệu năng cao khi xử lý các truy vấn. Để trả lời các truy vấn chứa trục “//” một cách hiệu quả, nút và cách tiếp cận cụ thể hóa đường dẫn sử dụng *θ-join* để cài đặt nút nối đường dẫn.

Hiện tại, hệ quản trị cơ sở dữ liệu quan hệ có các kỹ thuật xử lý *θ-join* hiệu quả, nhưng nói chung không hỗ trợ tốt cho *θ-joins*, nhất là các truy vấn chứa nhiều so sánh bất đẳng thức.

Nhiều kỹ thuật tự nhiên đã được phát triển để truy vấn dữ liệu XML hiệu quả.Chúng ta gọi các kỹ thuật là hướng tự nhiên vì các quá trình truy vấn được phát triển không dựa trên cơ sở dữ liệu quan hệ.

2.2.2.1. Cách tiếp cận phép nối

Cách tiếp cận phép nối là một cách tiếp cận tự nhiên quan trọng, cài đặt hiệu quả *θ-joins* có trong các truy vấn twig XML (*θ-joins* cũng gọi là "nối cấu trúc trong nhiều bài nghiên cứu"). Theo cách tiếp cận này, dữ liệu XML được lưu trữ trong các danh sách ngược.Khái niệm danh sách ngược bắt nguồn từ đánh chỉ số ngược, được áp dụng rộng rãi trong IR để cài đặt tìm kiếm văn bản hiệu quả.Mỗi danh sách ngược ứng với một thẻ khác nhau trong tài liệu XML, mỗi danh sách ghi lại vị trí của tất cả các phần tử có thẻ tên này, còn vị trí của phần tử được biểu diễn qua (*Start, End, Level*) hay (*vector Dewey*). Phầntử trong từng danh sách được xếp theo thứ tự tăng dần của số *start*.

Có thể thấy rằng danh sách ngược ở đây về bản chất cũng chính là bảng nút trong cách tiếp cận quan hệ, với điệu kiện mọi nút trong bảng nút được nhóm trước theo*label*, như trong cách tiếp cận nhị phân. Điều này có nghĩa là cách tiếp cận phép nối có thể sử dụng cách lưu trữ dữ liệu XML trong cơ sở dữ liệu quan hệ và cung cấp kỹ thuật xử lý truy vấn khác với vốn có trong hệ quản trị cơ sở dữ liệu quan hệ.

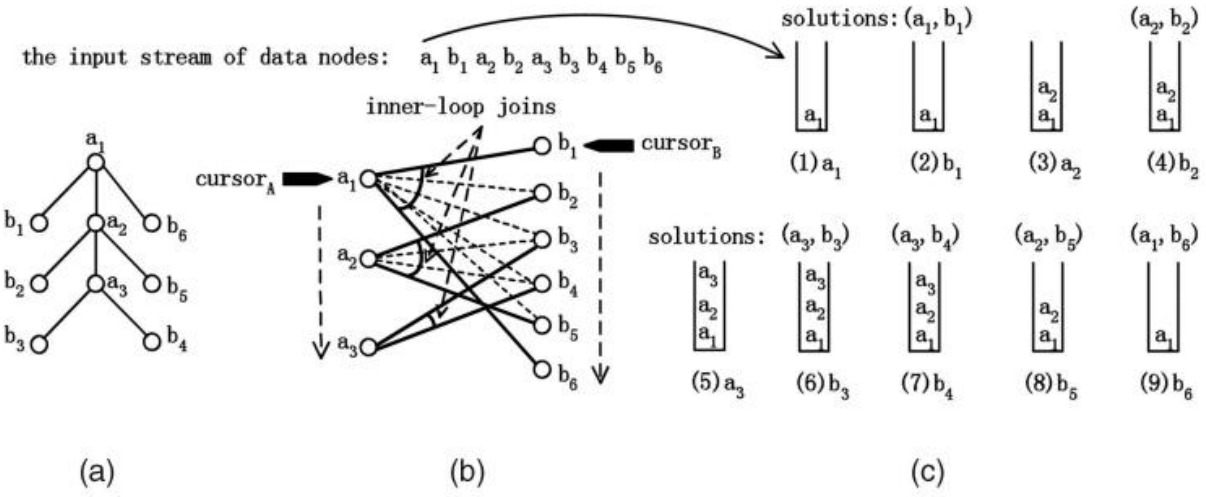
a) Cách tiếp cận phép nối dựa vào kết hợp nhiều thuộc tính (MPMGJN)

Năm 2001 *Zhang* đề nghị thuật toán tiếp cận phép nối dựa vào kết hợp nhiều thuộc tính đã được đề nghị, cách cào đặt này hơi giống với thuật toán "*nối kết hợp*" cổ điển dùng trong truy vấn quan hệ cho các *θ-join*. Để trả lời truy vấn "A//B" hay "A/B", trước hết tạo ra hai con trỏ vào đầu của listA và listB. Sau đó, hai con trỏ so sánh với nhau và tiến lên để triển khai nối kết hợp.

Trái với việc cài đặt nốt kết hợp chuẩn cho θ-join, cách tiếp cận này có cơ chế cải tiến con trỏ riêng của nó được chỉnh lại để hỗ trợ hiệu quả phép nối cấu trúc.Cụ thể, ở mỗi bước nó so sánh hai con trỏ như trong hình 2.11.

```
If  cursor_B.start < cursor_A.start  Then
   advance cursor_B;
Else
   temp_cursor_B = cursor_B;
   While( temp_cursor_B.start < cursor_A.end )    // the inner-loop join
      Output a tuple solution into join results. Specifically,
           Case 1 (For the `A/B' query):
               Output (cursor_A, temp_cursor_B) if cursor_A.level+1 = temp_cursor_B.level;
           Case 2 (For the `A//B' query):
               Output (cursor_A, temp_cursor_B);
      advance temp_cursor_B;
   Endwhile
   advance cursor_A;
```

Hình 2.11: Thuật toán tiếp cận phép nối dựa vào kết hợp nhiều thuộc tính.



Hình 2.12: Áp dụng MPMGJN và StackTree để truy vấn "A/B"(a) Cây dữ liệu. (b) Cách tiếp cận MPMGJN.(c) Cách tiếp cận StackTree.

b) Cách tiếp cận StackTree

Năm 2002 *Al-Khalifa* lưu ý rằng MPMGJN không xử lý hiệu quả các truy vấn chứa trục "/" trong một số trường hợp. Ví dụ hình 2.12b a1 chỉ có hai con B: b1 và b6. Tuy nhiên, trong MPMGJN, vòng lặp nối tại a1 cũng thăm nút b2 đến b5, là hậu duệ chứ không là con trực tiếp của a1 là không cần thiết.

Al-Khalifa đã đề nghị cách tiếp cận *StackTree* để tránh thăm các nút không cần thiết này. *StackTree* dùng một ngăn xếp để trữ những nút A được lồng trong cùng một đường dẫn trên cây dữ liệu. Hình 16 là lõi của thuật toán *StackTree*.Ở mỗi bước, nút dữ liệu có số start nhỏ nhất được bỏ ra khỏi danh sách của nó.Nếu nó là nút *A-tagged*, thì lại được đẩy vào ngăn xếp. Nếu nó là nút *B-tagged*,thì*StackTree*

thử dùng để lập một lời giải với các nút *A-tagged* hiện có trong ngăn xếp. Hình 2.12c minh họa tiến trình này.b3 chỉ so sánh với a3 (bước 6) chứ không với a1 đến a3 như trong hình 2.12b. Nói chung, *StackTree* cho hiệu năng xử lý tốt hơn MPMGJN.

Join order: Cả *StackTree* và MPMGJN đều là các thuật toán nối nhị phân, nghĩa là nối từng cặp trong danh sách ngược. Một truy vấn *twig* gồm một dãy nối nhị phân. Trật tự nối khác nhau dẫn đến kích thước tập kết quả trung gian khác nhau.Trật tự nối ảnh hưởng lớn đến hiệu quả xử lý truy vấn XML.

Các bộ tối ưu trong cơ sở dữ liệu quan hệ sử dụng phương pháp quy hoạch động để chọn thứ tự phép nối tối ưu chi phí. Wu đã đề nghị phương pháp quy hoạch động để chọn thứ tự nối nhị phân tối ưu hay gần tối ưu cho các truy vấn *twig* XML, trong đó ước lượng kích thước tập kết quả trung gian bằng kỹ thuật biểu đồ.

Output order: Một vấn đề quan trọng khác là thuật toán *StackTree* xuất ra tất cả các bộ lời giải theo thứ tự tăng dần của số start của các nút hậu duệ (tức là các nút *B-tagged*).

```
(1) min_start = Min(cursor_A.start, cursor_B.start);
    // suppose cursor_X.start = min_start
(2) Clear stack using min_start, i.e. all A nodes in stack with end number
    smaller than min_start are popped out of stack.
(3) If  X = A  Then    Push the node at cursor_A into stack;
    Else   Output tuple solutions into join results. Specifically,
              Case 1 (For the `A/B' query):
                 Output a tuple (top, cursor_B), where top is the A node on the
                 top of the current stack and top.level + 1 = cursor_B.level;
              Case 2 (For the `A//B' query):
                 Output all (a, cursor_B) tuples, where a is any A node in the
                 current stack;
(4) advance cursor_X;
```

Hình 2.13: Thuật toán StackTree

Ví dụ sáu bộ lời giải (hình 2.12c) được xuất ra theo trình tự b1 đến b6.*Al-Khalifa* đã đề xuất một biến thể của thuật toán *StackTree* mà xuất ra các bộ lời giải theo thứ tự tăng dần của số *start* của nút tổ tiên (tức là các nút *A-tagged*) nhờ tạm thời hoãn lại một số lời giải đã tìm thấy. Biến thể này rất quan trọng khi xử lý các truy vấn twig. Ví dụ xét truy vấn "A//B//C". Nếu ta chọn kế hoạch truy vấn A x (B x C), thì kết quả của B x C phải được sắp xếp theo nút B trước khi thực hiện nối nhị phân giữa A và B.

Skip: không cần phải đọc tuần tự toàn bộ danh sách ngược. Năm 2002 *Chen* phát triển *StackTree* với kỹ thuật "*skip*", chủ yếu dựa trên *B+-tree index* xây dựng sẵn trên số start của danh sách ngược, để tránh đọc tuần tự toàn bộ các nút dữ liệu trong danh sách ngược khi nối. Điều này giảm đáng kể chi phí đọc đĩa, vì nhiều nút dữ liệu không có trong các bộ lời giản cuối cùng.

c) Cách tiếp cận holistic

StackTree và *MPMGJN* phải phân rã các truy vấn twig thành nhiều phép nối nhị phân, làm phát sinh kết quả truy vấn trung gian kích thước lớn. Ví dụ thực hiện truy vấn (A x B) x C, thì kết quả trung gian A x B phải viết vào đĩa nếu kích thước lớn, gây ra chi phí đọc đĩa cao.

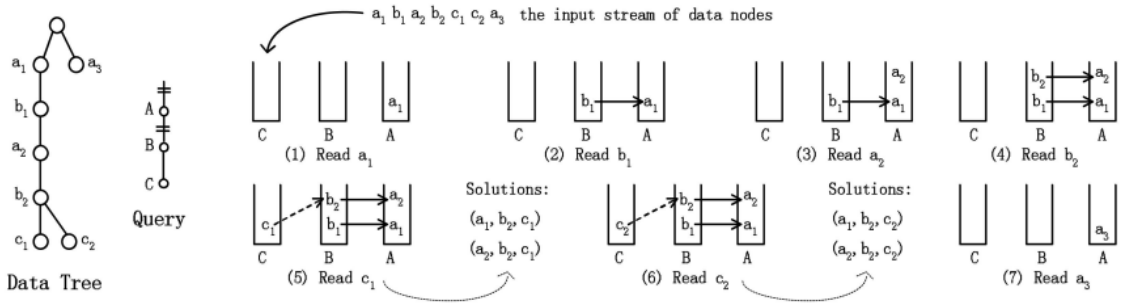
Năm 2002 *Bruno* đề xuất cách tiếp cận *holistic*, ý tưởng then chốt là đường ống, nối nhiều danh sách ngược đồng thời để tránh phải lưu lại kết quả trung gian.

Cách tiếp cận *PathStack*: Cách tiếp cận *holistic* để trả lời các truy vấn tuyến tính là thuật toán *PathStack*, minh hoạtrong hình 2.14.

```
(1) min_start = Min_i{ cursor_i.start };
    // suppose cursor_X.start = min_start
(2) Clear all stacks using min_start, i.e. all nodes in current stacks
    with end number smaller than min_start are popped out of stacks.
(3) If  Stack_parent(X) is not empty   Then
        Push the node at cursor_X into Stack_X with an associated pointer
        to the node on the top of Stack_parent(X);
        If  X is the leaf node of this path query   Then
            Output all tuple solutions implied by current stacks into
            join results through backtracking pointers between stacks;
(4) advance cursor_X;
```

Hình 2.14: Thuật toán PathStack

Khung của thuật toán hơi giống với *StackTree* trong hình 2.13. Điểm khác nhau là *StackTree* chỉ sử dụng một ngăn xếp để lưu trữ các nút A lồng nhau. Trái lại, *PathStack* dùng nhiều ngăn xếp, mỗi nút trong đường dẫn truy vấn một ngăn xếp.Hơn nữa, mỗi nút dữ liệu được lưu trong ngăn xếp có kèm con trỏ tới nút tương ứng trong ngăn xếp để theo vết các bộ lời giải. Một ví dụ minh họa trong hình 2.15.



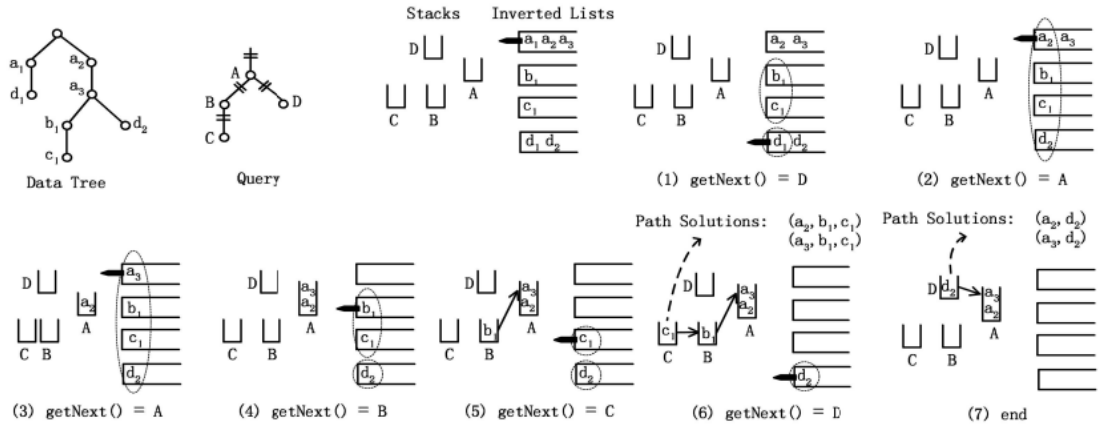
Hình 2.15: Cách tiếp cận PathStack

Cách tiếp cận *TwigStack*: Cách tiếp cận holistic trả lời các truy vấn twig tổng quát là một thuật toán *TwigStack*, gồm hai bước:

1) Tìm các đường dẫn đáp án (phân rã truy vấn twig thành nhiều truy vấn đường dẫn từ gốc tới lá và trả lời các truy vấn đường dẫn này trên cây dữ liệu).
2) Nối các đường dẫn đáp án (nối các *path solution* để nhận được lời giải cuối cùng cho truy vấn twig).

Có thể thực hiện bước 1 đơn giản theo cách xử lý từng truy vấn đường dẫn riêng biệt bằng PathStack. Tuy nhiên, phương pháp sơ đẳng này nói chung không hiệu quả, ví nó có thể trả về nhiều lời giải đường dẫn thừa ví dụ (a1, d1) (thỏa mãn truy vấn đường dẫn "//A//D" hình 2.16) nhưng không đóng góp cho bất cứ lời giải twig cuối cùng nào. Để giảm bớt số lời giải đường dẫn thừa thuật toán *TwigStack* đưa vào thêm một hàm phụ *q = getNext()*.

q = getNext() trả về nút truy vấn q sao cho q có một đáp án subtwig, nhưng các nút cha của nó không có. Ở mỗi bước, chỉ có con trỏ q được đẩy vào ngăn xếp.



Hình 2.16: Cách tiếp cận TwigStack

Đầu tiên, *getNext()* tiến *cursorA* từ a1 đến a2 vì *a1.end* < *b1.start* (a1 không thể góp phần vào bất cứ lời giải twig cuối cùng nào).

Sau đó, bước 1, B có đáp án *subtwig* tại b1, D có đáp án subtwig tại d1, nhưng cha của B và D là A không có đáp án *subtwig* tại a2. Do đó *getNext() = D* (getNext() khác B vì *d1.start* < *b1.start*), vì vậy d1 loại khỏi listD. Tuy nhiên, không đẩy d1 vào ngăn xếp D ở trước 2, vì ngăn xếp cha A là rỗng. Bằng cách này tránh được lời giải đường dẫn thừa (a1,d1).

Kết quả thực nghiệm chỉ ra rằng *TwigStack* nói chung có hiệu quả xử lý truy vấn cao hơn *StackTree*. Vì thế, gân đây *TwigStack* được nghiên cứu nhiều:

Optimality: Không có lời giải đường dẫn thừa. Nói chung, *TwigStack* vẫn còn có thể sinh ra lời giải đường dẫn thừa, mặc dù nó giảm số lượng so với phương pháp sơ đẳng xử lý từng truy vấn đường dẫn riêng biệt dùng *PathStack*. Năm 2002 *Bruno* đã chỉ ra rằng *TwigStack* là tối ưu cho các truy vấn twig chỉ chứa trục "//".

Ví dụ nếu trong hình 2.16 ta thấy "//" giữa A và D thành "/" thì ở bước 6 (*a2, b1, c1*) sẽ thành lời giải đường dẫn thừa. Tuy nhiên, *TwigStack* vẫn xuất ra lời giải đường dẫn này vì chỉ kiểm tra *cursorD, TwigStack* không thể xác định a2 có con D sau *cursorD = d2* hay không.

Năm 2003 *Choi* đã chỉ ra rằng bất cứ phiên bản nào của *TwigStack* mà đọc tuần tự danh sách ngược chỉ một lần thì không thể tối ưu đối với các truy vấn *twig* chứa hỗn hợp cả "//" và "/". Các nghiên cứu tập trung tối ưu hóa cho một lớp con các truy vấn *twig*.

Năm 2004 *Lu* đã đề xuất một biến thể *TwigStack* là *TwigStackList*, trong đó nhìn về phía trước một số nút dữ liệu trong danh sách ngược và lưu chúng vào bộ nhớ chính.Như thế *TwigStack* là cho các truy vấn mà mọi trục "/" là ở dưới các nút không phân nhánh.

Gần đây *Chen* mở rộng *TwigStack* thành *iTwigJoin*, để tối ưu cho các truy vấn twig chỉ chứa trục "/" hay chỉ có một nút phân nhánh bằng cách phân hoạch danh sách ngược thành nhiều danh sách con dựa trên mức hay đường dẫn gốc của nút dữ liệu trong danh sách ngược.

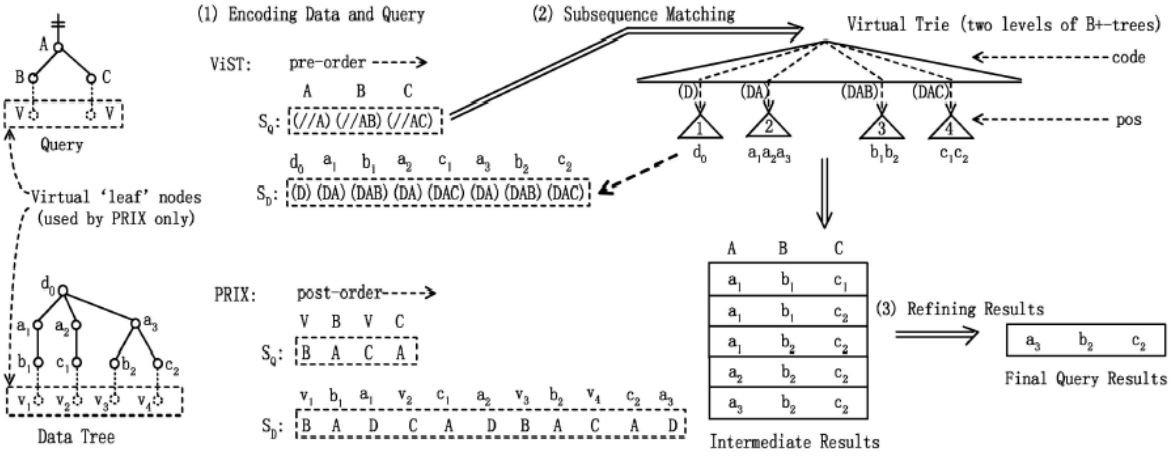
Skip: không phải đọc tuần tự toàn bộ danh sách ngược. Giống như trong nối nhị phân. Kỹ thuật skip sử dụng *XB-tree* hay *XR-tree* đánh chỉ số có thể giảm đáng kể chi phí đọc đĩa trong phép nối *holistic*.Sự khác nhau là với phép nối *holistic*, ở mỗi bước phải xác định truy vấn cung đứt quãng nào sẽ được nhảy qua trước tiên.

2.2.2.2. Cách tiếp cận tuần tự

Trong VIST, *Wang* và các cộng sự đề xuất một phương pháp, mà chúng ta gọi là phương pháp tiếp cận tuần tự, kết hợp với mô hình *twig*. Nhằm để tránh cácchi phí giải pháp kết nối theo đường dẫn được sử dụng trong pha thứ hai của *TwigStack* bằng cách sử dụng truy vấn đầu vào twig hơn là sử dụng các đường dẫn như là đơn vị cơ sở của truy vấn twig, phương pháp tiếp cận này làm việc như sau:

b1) tiềnxử lý: mã hóa dữ liệu và truy vấn, một dãy tất cả các nút ở trong cây dữ liệu được mã hóa theo một phương pháp cụ thể và sau đó sắp xếp trình tự theo dãy SD. tương tự như vậy các nút tất cả các nút của truy vấn twig được mã hóa và sắp xếp theo trình tự SQ. VIST mã hóa mỗi nút bằng cách sử dụng đường dẫn gốc của nó và sắp xếp tất cả các nút sử dụng preorder.

b2) nhiệm vụ trước tiên là so khớp mẫu twig được chuyển đổi sang tác vụ của việc so khớp dãy con trong quá trình tìm kiếm ở SD tất cả các dãy con mà phù hợp SQ. các nhiệm vụ này được thực hiện theo quá trình lặp thăm dò "virtual trie" các chỉ số được xây dựng trước SD thực thi vật lý như là hai mức của B+-tree với (code, pos) như keys (hình 2.17).



Hình 2.17:Cách tiếp cận tuần tự.

Mỗi *probe* là một cặp (*q.code, d.pos*). đầu tiên thăm dò mức đầu bằng cách sử dụng *q.code* tiếp cận được mức thứ hai và sau đó nó lấy thông tin của các nút ở trong mức thứ hai mà vị trí của nó lớn hơn *d.pos*. hình 2.17: đầu tiên chỉ số thăm dò được thông báo đã đến được cây thứ hai và tất cả các nút ở cây thứ hai đã được rút chọn. Do đó đối với một nút đã được rút chọn là nút a2. Một chỉ số thăm dò (*//AB), a2.pos*) cũng được thông báo đã được tiếp cận được cây 3 và b2 sẽ được rút chọn để đến cây 4.vậy (*a2, b2, c2*) là kết quả của dãy con được so khớp.

b3 hậu xử lý: làm min kết quả: không phải tất cả dãy con trả về kết quả phù hợp với kết quả trả về trong bước 2 tương tự với giải pháp twig cuối cùng. Ví dụ trong hình 2.17 chỉ có (a3, b2, c2) là giải pháp twig, bất cứ khi nào khác trong bộ bốn là đúng. Do đó sự so khớp trả về trong bước hai phải được làm mịn để nhận được một truy vấn chính xác. VIST sử dụng bốn pha phức tạp để tinh chỉnh.

Chú ý rằng b2 ở trên có thể sinh rất nhiều giải pháp twig dư thừa hơn là giải pháp dư thừa với đường dẫnđược sinh bằng *TwigStack*. Dư thừa bùng nổ bởi các dãy con và nó lặp đi lặp lại. *Wang* và *Meng* đã giải quyết vấn đề bằng các chọn b2 và b3 sử dụng trực quan minh họa này chỉ những nút trong *B+-tree* không vi phạm các thuật ngữ của họ sẽ được lấy. Hình 2.17 khi a2 được chọn (*(//AB),a2.pos*) để đến được cây 3,b2 không lấy đươc vì nó vi phạm. Cách tiếp cận tuần tự cũng có một số hạn chế như sau:

- *Nó chỉ được hỗ trợ truy vấn twig*
- *Nó có thể liên quanđến số lượng lớn các chỉ số thăm dò như khi chúng ta đề cập trong 2.3.1a*
- *Nó có thể lặp lại việc thăm các nút dữ liệu nhiều lần một cách không cần tiết*

2.2.2.3. Tổng kết

Cách tiếp cận phép nối cung cấp một sự bổ sung tự nhiên có hiệu quả cho sử dụng *θ-join* trong cách tiếp cận quan hệ.

Rõ ràng giữa các kỹ thuật nội thì:

- *Hiệu suất của Holistic tốt hơn MPMGJN hoặc StackTree.*
- *Khi một đồ thị chỉ số hoặc một chỉ số Plabeling có thể dùng được, nó có thế được dùng để giảm số phép nối hoặc thu ngắn lại dánh sách ngược trước khi nối.*

CHƯƠNG 3: ỨNG DỤNG XML TRONG CƠ SỞ DỮ LIỆU

3.1. GIỚI THIỆU VỀ EXIST

eXist là một hệ quản lý cơ sở dữ liệu mà nguồn mở dựa trên ngôn ngữ XML. Nó có thể dễ dàng được tích hợp vào các ứng dụng XML bằng một số phương pháp, từ ứng dụng dựa trên nền web đến các hệ thống văn bản được sử dụng trên máy tính.Cơ sở dữ liệu của nó được lưu trữ và có thể được phân phối bằng nhiều cách khác nhau.Nó có thể chạy như một tiến trình máy chủ riêng biệt hoặc có thể ở bên trong một *servlet-engine* hoặc được gắn trực tiếp vào một ứng dụng.

eXist hỗ trợ nhiều chuẩn công nghệ (web) khiến nó trở thành một nền tảng ứng dụng thông minh:

- *Xquery 1.0/Xpath2.0.*
- *XSLT 1.0 (sử dụng Apache Xalan) hay XSLT 2.0 (sử dụng Saxon).*
- *Giao diện HTTP: REST, WebDAV, SOAP, XMLRPC, AtomPublishing Protocol.*
- *Các cơ sở dữ liệu XML: XMLDB, XQJ/JSR-225 (đang phát triển), Xupdate, phần cập nhập mở rộng Xquery.*

eXist cung cấp lược đồ tài liệu tốn ít bộ nhớ hơn bằng các tập hợp có thứ bậc. *eXist* sử dụng cú pháp *Xpath*mở rộng, người dùng có thể truy vấn các phần riêng biệt của một tập hợp hoặc tất cả các tài liệu có trong cơ sở dữ liệu. Mặc dù khá nhẹ, bộ máy truy vấn của *eXist* vẫn thi hành có hiệu quả, nó xử lý truy vấn trên nền chỉ số. Một lược đồ chỉ số nâng cao hỗ trợ sự nhận dạng nhanh những mối quan hệ cấu trúc giữa các nút, như là *cha – con, con cháu – tổ tiên* hay *anh chị em ruột*. Dựa trên thuật toán nối đường dẫn, một phạm vi rộng những câu truy vấn biểu thức đường dẫn được xử lý bằng các thông tin chỉ số. Với *eXist* truy cập tới các nút hiện tại, nút mà được lưu ở trung tâm kho tài liệu XML không cần phải đánh các biểu thức.

eXist là cơ sở dữ liệu thích hợp nhất với các ứng dụng từ số lượng nhỏ tới số lượng lớn các tập hợp tài liệu XML, những tài liệu mà ít khi được cập nhật. *eXist* cung cấp một số mở rộng với tiêu chuẩn *Xpath*để tăng hiệu quả xử lý những câu truy vấn, bao gồm tìm kiếm theo từ khóa, nhưng câu truy vấn gần đúng với các thuật ngữ tìm kiếm hay biểu thức chính quy.

3.2. XÂY DỰNG TÀI LIỆU XML, CÀI ĐẶT EXIST VÀ TIẾN HÀNH TRUY VẤN

3.2.1. Xây dựng tài liệu XML

Cơ sở dữ liệu các sách.

```
<bangsach>
    <Sach>
        <Masach>MS3</Masach>
        <Tensach>Dế mèn phiêu lưu ký</Tensach>
        <Matacgia>MTG8</Matacgia>
        <NamXB>1988</NamXB>
        <Sotrang>10000</Sotrang>
        <Maloaisach>MLS1</Maloaisach>
        <MaNXB>MNXB3</MaNXB>
    </Sach>
    …
    <Sach>
        <Masach>MS8</Masach>
        <Tensach>Giáo dục công dân</Tensach>
        <Matacgia>MTG7</Matacgia>
        <NamXB>1978</NamXB>
        <Sotrang>10000</Sotrang>
        <Maloaisach>MLS2</Maloaisach>
        <MaNXB>MNXB3</MaNXB>
    </Sach>
    <Sach>
        <Masach>MS9</Masach>
        <Tensach>Tắt Đèn</Tensach>
        <Matacgia>MTG4</Matacgia>
        <NamXB>1978</NamXB>
        <Sotrang>10000</Sotrang>
        <Maloaisach>MLS2</Maloaisach>
        <MaNXB>MNXB1</MaNXB>
    </Sach>
</bangsach>
```

Cơ sở dữ liệu tác giả:

```
<bangtacgia>
    <Tacgia>
        <Matacgia>MTG3</Matacgia>
        <Tentacgia>Phạm Thanh Ba</Tentacgia>
        <Diachi>Hải Phòng</Diachi>
        <SDT>0979658256</SDT>
        <emailtg>bapt@gmail.com</emailtg>
    </Tacgia>
    <Tacgia>
        <Matacgia>MTG4</Matacgia>
        <Tentacgia>Vũ Đức Hậu</Tentacgia>
        <Diachi>Hải Phòng</Diachi>
        <SDT>0979658256</SDT>
        <emailtg>hauvd@gmail.com</emailtg>
    </Tacgia>
    …
    <Tacgia>
        <Matacgia>MTG5</Matacgia>
        <Tentacgia>Nguyễn Bá Tú</Tentacgia>
        <Diachi>Hải Phòng</Diachi>
        <SDT>0979658256</SDT>
        <emailtg>tunb@gmail.com</emailtg>
    </Tacgia>
    <Tacgia>
        <Matacgia>MTG6</Matacgia>
        <Tentacgia>Ngô Tất Tố</Tentacgia>
        <Diachi>Hải Phòng</Diachi>
        <SDT>0979658256</SDT>
        <emailtg>tont@gmail.com</emailtg>
    </Tacgia>
    <Tacgia>
        <Matacgia>MTG8</Matacgia>
        <Tentacgia>Tô Hoài</Tentacgia>
        <Diachi>Hải Phòng</Diachi>
        <SDT>0979658256</SDT>
        <emailtg>hoaito@gmail.com</emailtg>
    </Tacgia>
</bangtacgia>
```

Cơ sở dữ liệu nhà xuất bản:

```
<bangNXB>
     <NhaXB>
          <MaNXB>MNXB1</MaNXB>
          <TenNXB>Kim Đồng</TenNXB>
          <Dthoai>0979658256</Dthoai>
          <email>kimdong.com.vn</email>
     </NhaXB>
     <NhaXB>
          <MaNXB>MNXB2</MaNXB>
          <TenNXB>Đồng Nai</TenNXB>
          <Dthoai>0979658256</Dthoai>
          <email>dongnai.com.vn</email>
     </NhaXB>
     <NhaXB>
          <MaNXB>MNXB3</MaNXB>
          <TenNXB>Tiền Phong</TenNXB>
          <Dthoai>0979658256</Dthoai>
          <email>tienphong.com.vn</email>
     </NhaXB>
</bangNXB>
```

Cơ sở dữ liệu loại sách:

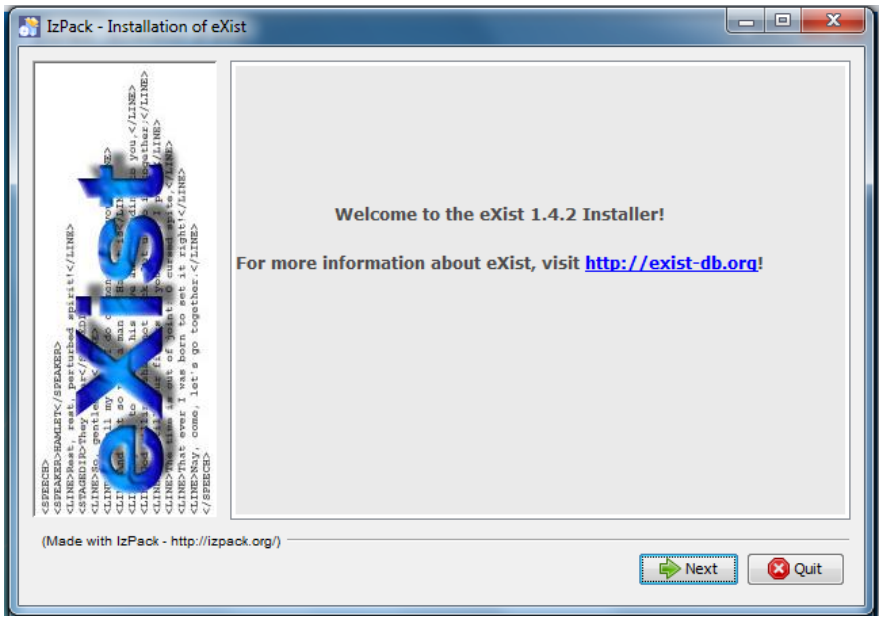
```
<bangloaisach>
     <Loaisach>
          <Maloaisach>MLS1</Maloaisach>
          <Tenloaisach>Truyện ngắn</Tenloaisach>
     </Loaisach>
     <Loaisach>
          <Maloaisach>MLS2</Maloaisach>
          <Tenloaisach>Tình yêu</Tenloaisach>
     </Loaisach>
     <Loaisach>
          <Maloaisach>MLS3</Maloaisach>
          <Tenloaisach>Giáo dục</Tenloaisach>
     </Loaisach>
     <Loaisach>
          <Maloaisach>MLS4</Maloaisach>
          <Tenloaisach>Linh tinh</Tenloaisach>
     </Loaisach>
</bangloaisach>
```

3.2.2. Cài đặt eXist và tiến hành truy vấn

Để có thể cài đặt *eXist* thì hệ thống phải có Java 1.4 hoặc mới hơn.

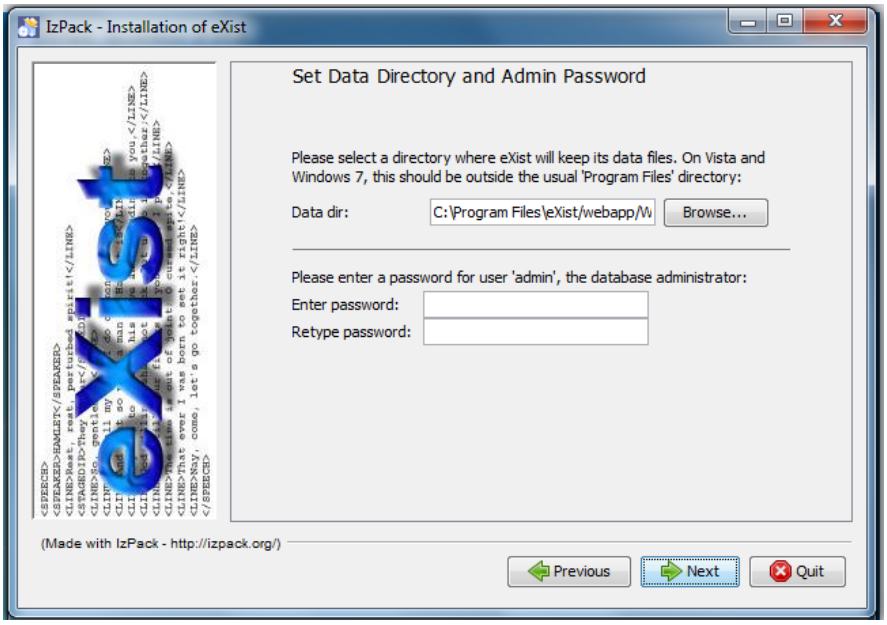
Phiên bản mới nhất *eXist – 1.4.2* có thể được tải về từ trang:

http://exist-db.org/exist/download.xml



Hình 3.1: Bắt đầu cài đặt eXist

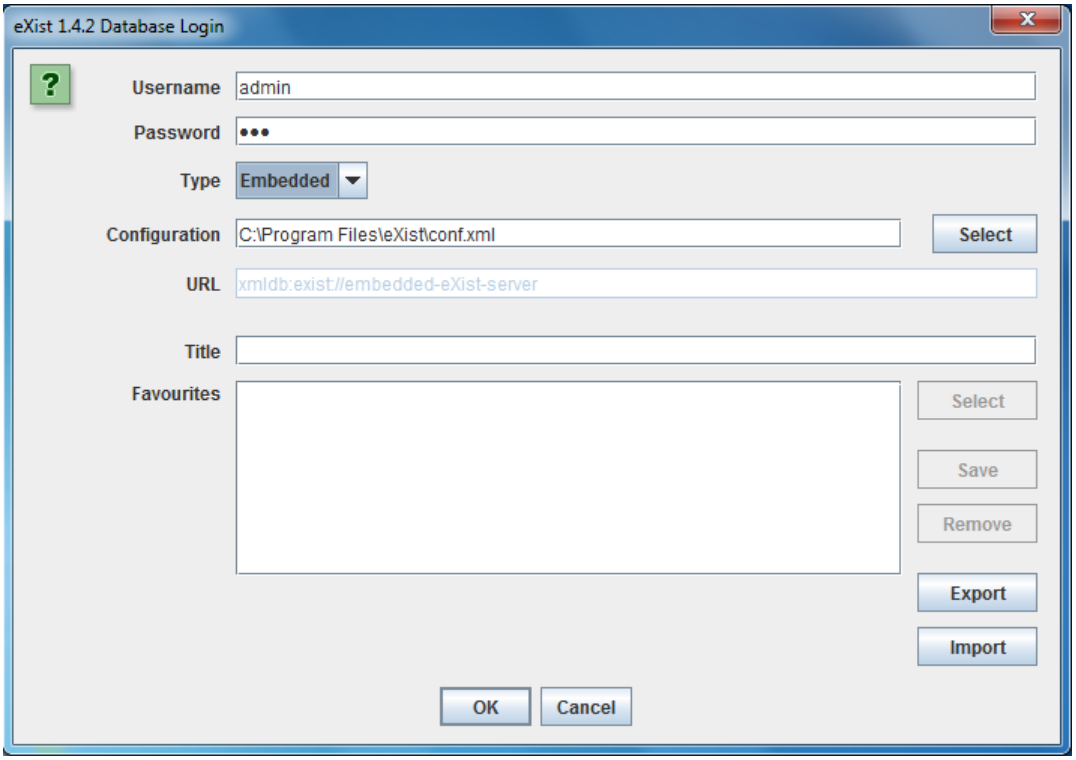
Trong quá trình cài đặt ta phải thiết lập mật khẩu cho tài khoản "*admin*". Tài khoản này sẽ được dùng để đăng nhập vào phần quản trị *eXist* và nhập tài liệu XML vào cơ sở dữ liệu:



Hình 3.2: Thiết lập mật khẩu cho tài khoản admin.

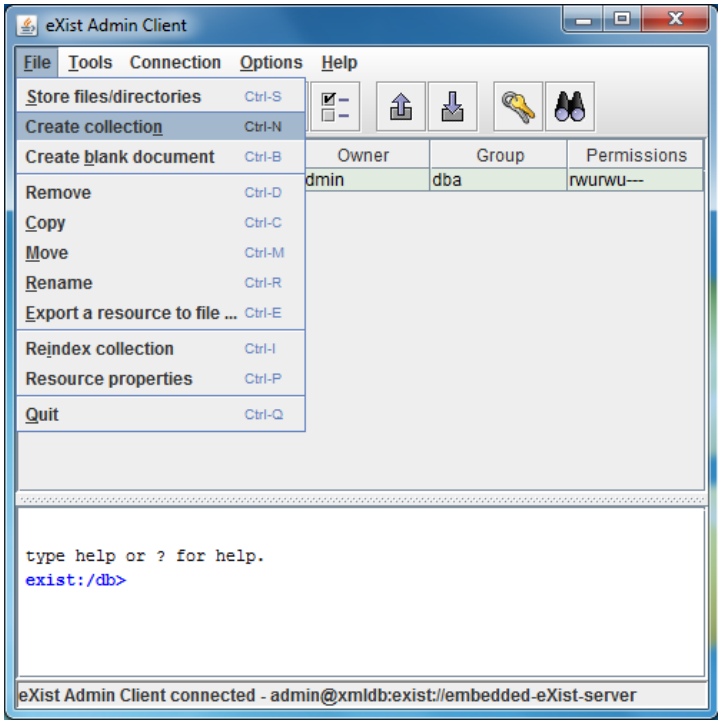
Nhập dữ liệu XML vào eXist

Sau khi chạy eXist ta có thể đăng nhập bằng cách thực thi *client.bat* trong thư mục *exist/bin*. Mặc định chương trình sẽ đặt địa chỉ của eXist là localhost và truy cập qua cổng 8080. Ta có thể thay đổi địa chỉ của cơ sở dữ liệu và cổng truy cập bằng cách nhập vào ô *textbox*:

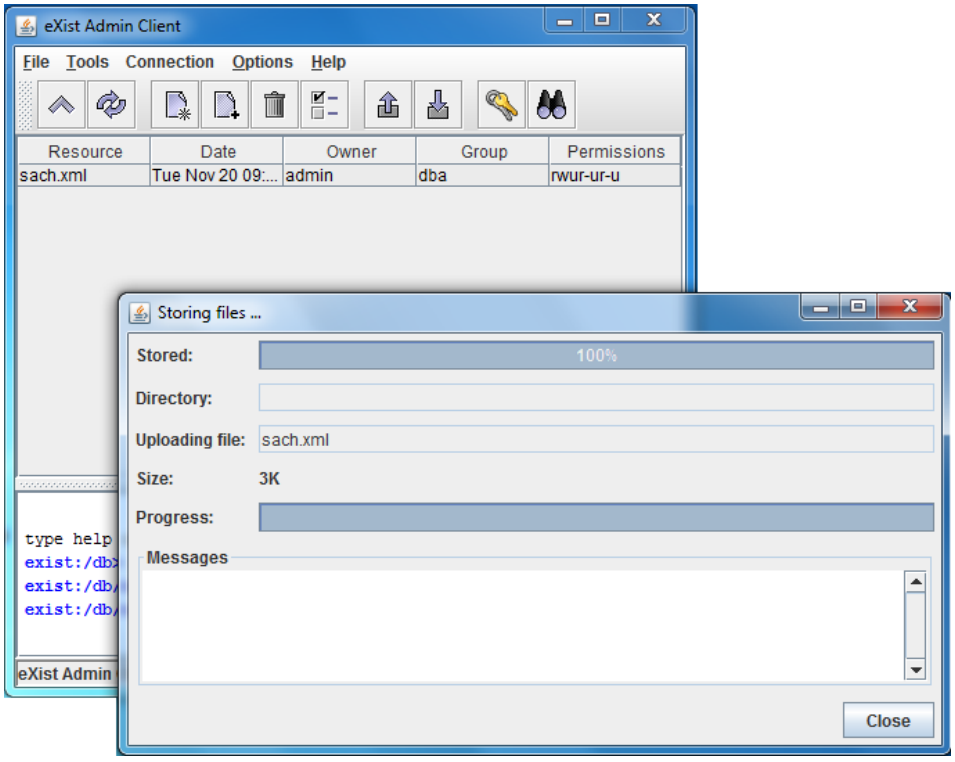


Hình 3.3: Đăng nhập vào eXist.

Sau khi đăng nhập ta tạo một *collection* tên là "*hoso*" rồi tải tài liệu XML đã xây dựng lên:



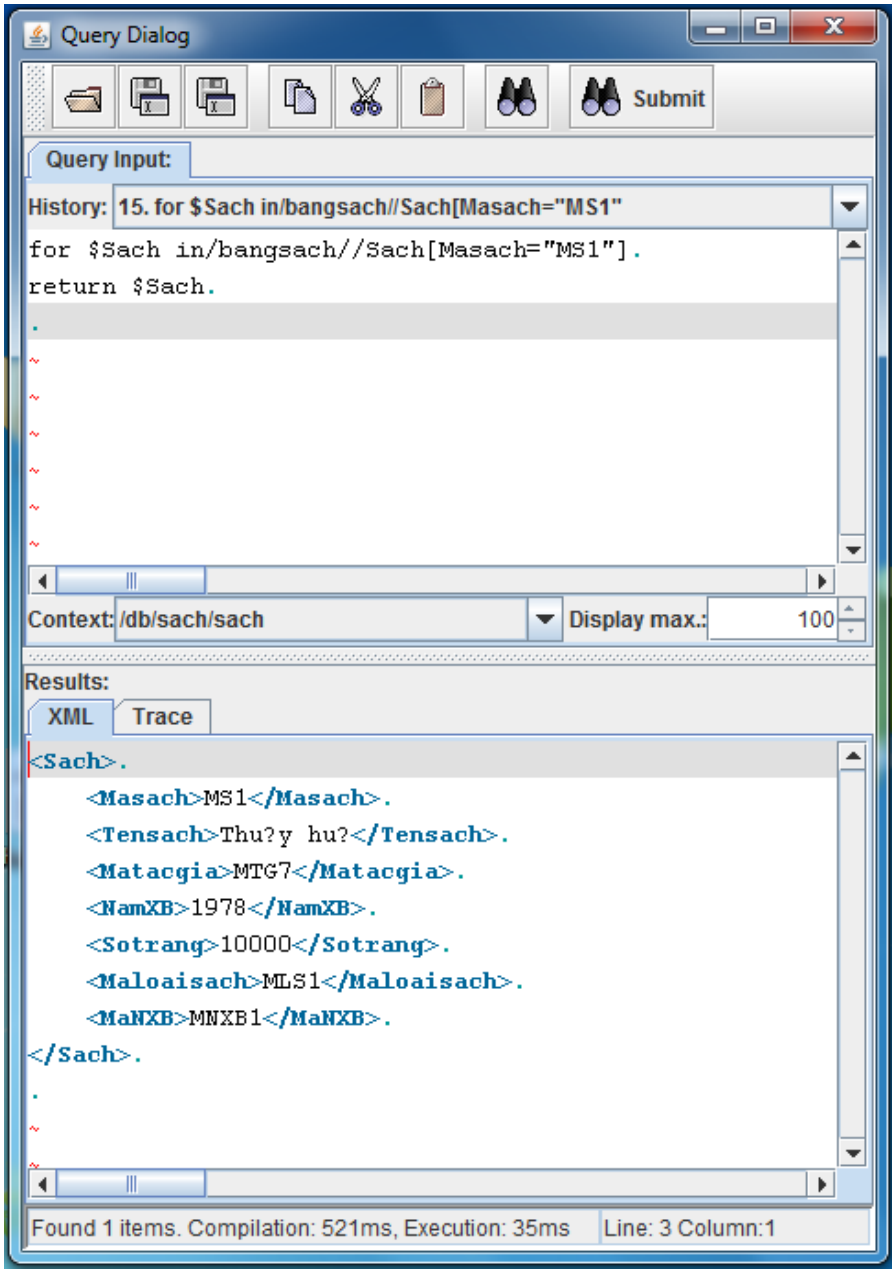
Hình 3.4: Tạo collection mới.



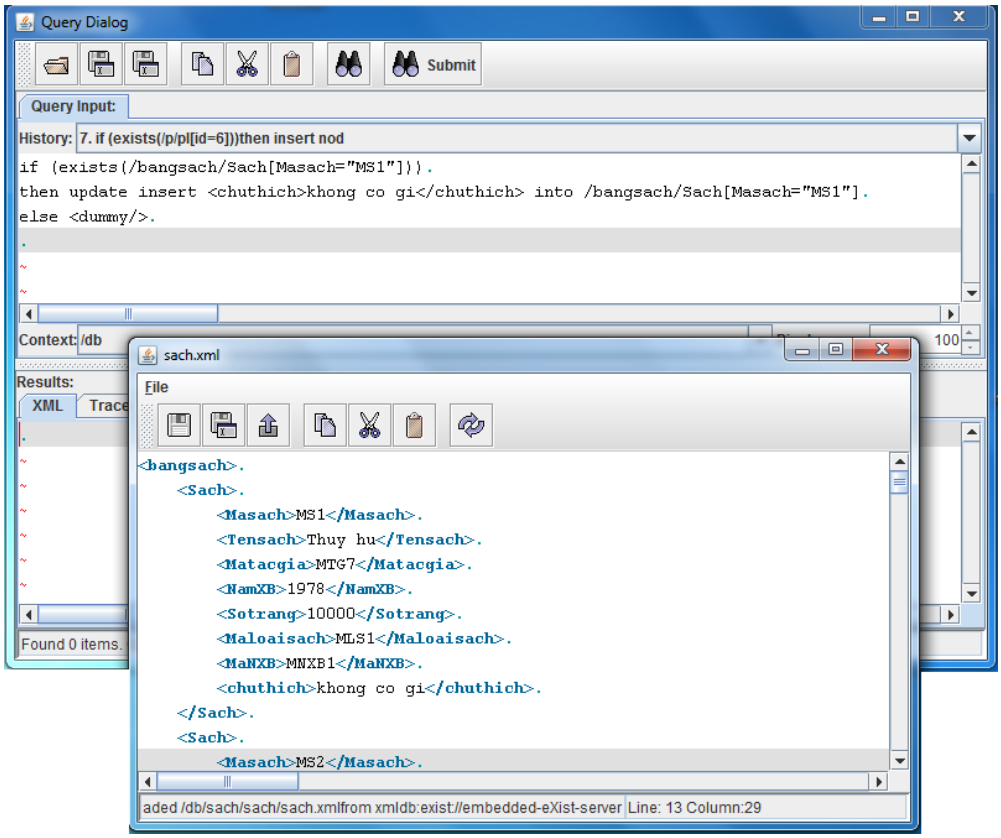
Hình 3.5: Tải tài liệu XML lên.

Tiến hành truy vấn

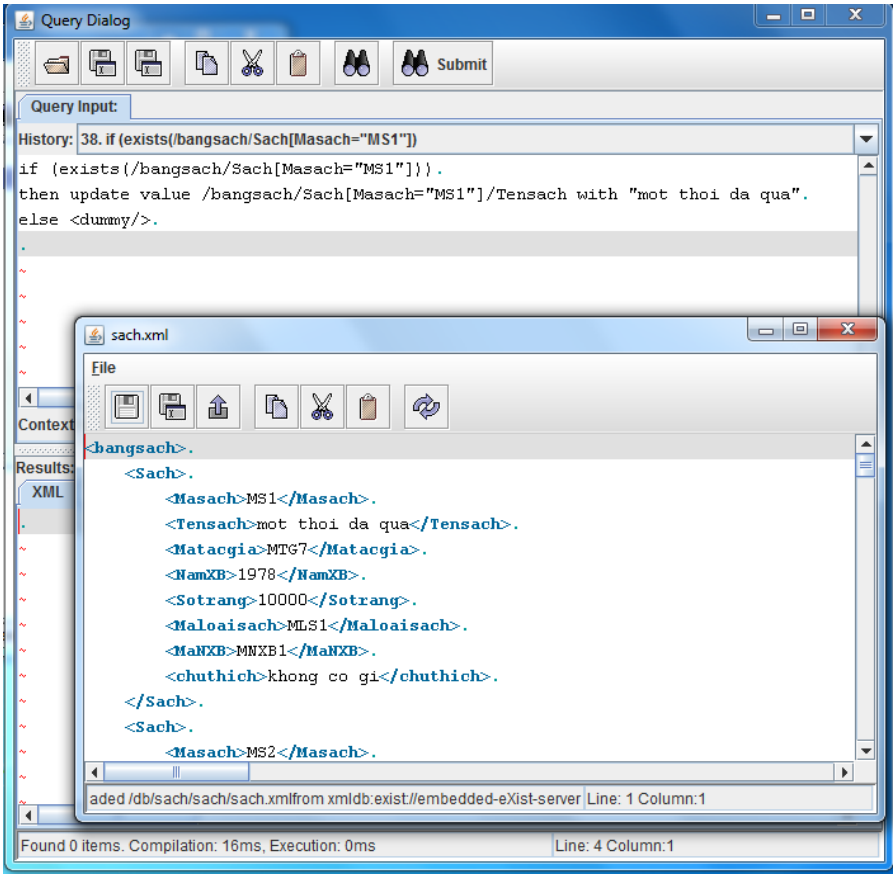
Sau khi tải tài liệu lên *eXist* ta bắt đầu truy vấn bằng cách nhấn vào biểu tượng "ống nhòm" của ứng dụng:



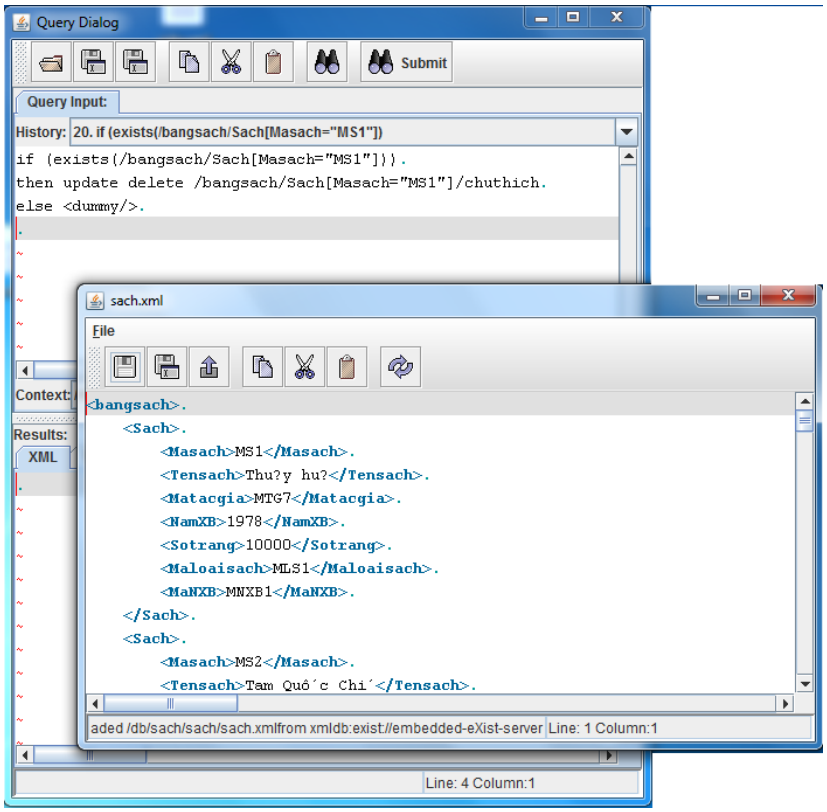
Hình 3.6: Tìm thông tin cuốn sách có mã sách là MS1.



Hình 3.7: Chèn tag chú thích vào cuốn sách có mã sách là MS1



Hình 3.8: Sửa tên cuốn sách có mã sách là MS1 thành “mot thoi da qua”.



Hình 3.9: Xóa tag chú thích trong cuốn sách có mã cuốn sách là MS1.

KẾT LUẬN

Đồ án tìm hiểu thuật toán truy xuất dữ liệu XML.

Trong đồ án này em đã giới thiệu và tìm hiểu bài toán lưu trữ và tối ưu hóa truy vấn với XML trên cả hai mặt: lý thuyết và thực nghiệm.

Về lý thuyết em đã trình bày một số thuật toán liên quan để có cái nhìn sâu hơn về vấn đề.Em đã tìm hiểu được hướng giải quyết là dùng lược đồ chỉ số cùng với các phép biến đổi các câu truy vấn thành câu truy vấn quan hệ trong SQL để giải quyết bài toán. Cách làm lược đồ cũng như biến đổi câu truy vấn đã được trình bày cụ thể.

Về thực nghiệm em đã cài đặt cơ sở dữ liệu XML eXist để truy vấn các tập tin ảnh.

Qua quá trình thực hiện đồ án, em đã tổng hợp lại được các kiến thức đã học trong thời gian học tập tại trường. Đồng thời, em đã tìm hiểu và nắm được các thuật toán truy xuất dữ liệu XML, nâng cao được các kỹ năng làm việc giúp em có thể chủ động tìm kiếm thông tin đẻ hoàn thành các công việc sau này.

TÀI LIỆU THAM KHẢO

1. Senthilkuma, R.,"Nested XPath *Query Optimization for XML Structured Document Database*" on Advanced Computing and Communications, 2008. ADCOM 2008. 16th International Conference on, 14-17 Dec. 2008.

2. Lijing Zhang,"The Query and *Application of XML Data Based on Xquery*" on Computational and Information Sciences (ICCIS), 2012 Fourth International Conference on, 17-19 Aug. 2012.

3. Gang Gou,Rada Chirkova, "*Efficiently Querying Large XML Data Repositories: A Survey*", IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 19, NO. 10, OCTOBER 2007.

4. Shichuan Li, "Highly efficient processing of XML path/twig queries using Index Caches" on Fuzzy Systems and Knowledge Discovery (FSKD), 2012 9th International Conference on, 29-31 May 2012.