UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

TRADUÇÃO DE CONSULTAS XQUERY PARA SQL

DISSERTAÇÃO DE MESTRADO

Marta Breunig Loose

Santa Maria, RS, Brasil

2013

TRADUÇÃO DE CONSULTAS XQUERY PARA SQL

Marta Breunig Loose

Dissertação apresentada ao Curso de Mestrado em Computação do Programa de Pós-Graduação em Informática da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação

Orientadora: Profª. Drª. Deise de Brum Saccol

Santa Maria, RS, Brasil

2013

Loose, Marta Breunig

Tradução de Consultas XQuery para SQL / por Marta Breunig Loose. – 2013.
138 p.; 30 cm.

Orientadora: Deise de Brum Saccol
Dissertação (Mestrado) - Universidade Federal de Santa Maria, Centro de Tecnologia, Mestrado em Computação, RS, 2013.

1. XQuery, SQL, Tradução. I. Brum Saccol, Deise de. II. Título.

E-mail: marta.breunig@gmail.com

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

TRADUÇÃO DE CONSULTAS XQUERY PARA SQL

elaborada por
Marta Breunig Loose

como requisito parcial para obtenção do grau de
Mestre em Ciência da Computação

COMISSÃO EXAMINADORA:

Deise de Brum Saccol, Drª.
(Presidente/Orientadora)

Juliana Kaizer Vizzotto, Drª. (UFSM)

Viviane Pereira Moreira, Drª. (UFRGS)

Santa Maria, 16 de Julho de 2013.

AGRADECIMENTOS

Em primeiro lugar agradeço a Deus, pois é Ele quem me sustenta dia-a-dia. Ele me acorda a cada manhã e me conduz em segurança até repousar. Dele vem o perfeito amor, a paz verdadeira e a vida em abundância. Poucas são as palavras para expressar o quanto sou grata a Deus, meu querido Pai.

Ao meu amado esposo, Luís, que sempre está ao meu lado e é um exemplo de dedicação. Obrigada por tantos momentos em que me incentivou a continuar e me fez sorrir em meio as dificuldades. Pelo amor, carinho e confiança.

À minha mãe, Clari, que é um exemplo de mulher batalhadora e forte. Obrigada por sempre fazer o melhor por mim e me ensinar a viver com dignidade.

Ao meu pai, Carlos, que sempre me incentivou ao estudo. Obrigada pelo apoio e carinho que sempre demonstrou por mim.

Aos meus pequenos irmãos, Gustavo e Heloísa, que sempre anseio em encontrar. Obrigada pelos momentos de alegria e pela esperança que trazem no olhar.

À Universidade Federal de Santa Maria e ao Programa de Pós-graduação em Informática, pela oportunidade de aperfeiçoamento e aprendizado durante o curso de mestrado.

À CAPES, pela bolsa de estudos que permitiu minha dedicação à realização deste trabalho.

À professora Deise, que me ensinou muito nesses dois anos e meio. Obrigada pela forma com que me orientou neste trabalho, sempre contribuindo positivamente a consolidá-lo e fornecendo sábios conselhos.

Ao professor Eduardo e à professora Juliana, que sempre quando solicitados me ajudaram e tiraram minhas dúvidas.

Aos demais colegas e professores do PPGI que de alguma forma trocaram experiências e apoio.

Aos meus familiares que, mesmo longe, sempre torcem por mim e me incentivam. Em especial, à minha prima (amiga e irmã) Daniele, que tanto tempo esteve ao meu lado e dividiu momentos bons e ruins comigo. Aos meus tios Glaci e Rudi, pelo carinho, incentivo e exemplo. Ao meu primo Charles e sua esposa Priscila, pela amizade e carinho.

Aos amigos de Panambi e de Santa Maria, pelos momentos de descontração, afeto e incentivo.

Aos demais amigos, colegas e familiares que de alguma forma colaboraram para a realização deste trabalho.

Muito obrigada!

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

TRADUÇÃO DE CONSULTAS XQUERY PARA SQL
AUTORA: MARTA BREUNIG LOOSE
ORIENTADORA: DEISE DE BRUM SACCOL
Local da Defesa e Data: Santa Maria, 16 de Julho de 2013.

Mesmo com a ampla utilização de XML como padrão para a troca de dados, a maior parte dos dados corporativos é armazenada em Bancos de Dados Relacionais (BDR). Além disso, muitas aplicações requerem o armazenamento eficiente de dados XML (*eXtensible Markup Language*), o que pode ser alcançado usando um BDR. Dessa forma, a capacidade de gerenciar ambos os tipos de dados (relacionais e XML) torna-se essencial para as organizações. Com isso, uma das alternativas para realizar esse armazenamento consiste no mapeamento da estrutura de documentos XML para um esquema relacional. Esse mapeamento torna-se complexo a medida em que os documentos XML possuem estruturas diferentes, mesmo representando dados de um domínio comum. Nesse caso, além da mapear a estrutura e os dados XML para tabelas em um BDR, é necessária a tradução de consultas XML, antes executadas sobre os documentos originais, para SQL, linguagem padrão utilizada nos BDRs. Com a finalidade de realizar essa tradução, este trabalho propõe um mecanismo que traduz consultas XQuery, linguagem de consulta XML, para consultas SQL equivalentes, chamado QMap (*Query Mapper*). A proposta está inserida no contexto do *framework* X2Rel (*XML to Relational*), um ambiente para armazenamento, gerenciamento e consulta a dados XML heterogêneos em BDRs.

Palavras-chave: XQuery, SQL, Tradução.

ABSTRACT

Master's Dissertation
Programa de Pós-Graduação em Informática
Federal University of Santa Maria

TRANSLATION OF XQUERY QUERIES TO SQL

AUTORA: MARTA BREUNIG LOOSE
ORIENTADORA: DEISE DE BRUM SACCOL
Defense Place and Date: Santa Maria, July 16st, 2013.

Even with the extensive use of XML as the standard for data exchange, most corporate data is stored in Relational Databases (RDB). Furthermore, many applications require efficient storage of XML data (eXtensible Markup Language), which can be achieved using a RDB. Thus, the ability to manage both kinds of data (relational and XML) becomes essential for organizations. Therefore, one of the alternatives to perform this storage consists of mapping the structure of XML documents to a relational schema. This mapping becomes complex since the XML documents have different structures even representing data from a common domain. In this case, in addition to mapping the structure and the XML data to relations in a RBD, it is also required to translate XML queries to SQL, the standard language used by RBD. In order to perform this translation, this work proposes a mechanism that translates XQuery queries to equivalent SQL queries, called QMap (Query Mapper). The proposal is part of a framework named X2Rel (XML to Relational), an environment for storing, managing and querying XML data in heterogeneous RBDs.

Keywords: XQuery, SQL, translation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BDR	*Banco de Dados Relacional*
BNF	*Backus-Naur Form*
CMap	*Concept Mapper*
DAG	*Directed Acyclic Graph*
DDL	*Data Definition Language*
DML	*Data Manipulation Language*
DTD	*Document Type Definition*
FLWR	*For, Let, Where e Return*
FLWOR	*For, Let, Where, Order by e Return*
OntoGen	*Ontology Generator*
OntoRel	*Ontology to Relational*
OWL	*Ontology Web Language*
PET	*Programa de Educação Tutorial*
QMap	*Query Mapper*
RASMap	*Relational Algebra to SQL Mapper*
RDB	*Relational Database*
SGBD	*Sistema Gerenciador de Banco de Dados*
SGBDR	*Sistema Gerenciador de Banco de Dados Relacional*
SQL	*Structured Query Language*
XALGen	*XQuery Algebra Generator*
XARMap	*XQuery Algebra to Relational Algebra Mapper*
XAT	*XML Algebra Tree*
X2Rel	*XML to Relational*
XMap	*XML Mapping*
XML	*Extensible Markup Language*
W3C	*World Wide Web Consortium*

LISTA DE SÍMBOLOS

$-$	Diferença
$\cap$	Intersecção
μ	Mu
$\bowtie$	Natural Join
φ	Phi
π	Pi
ρ	Rho
σ	Sigma
τ	Tau
θ	Theta
$\times$	Times
$\cup$	União

SUMÁRIO

1 INTRODUÇÃO

Sabe-se que XML (*eXtensible Markup Language*) é uma linguagem utilizada para especificação de dados estruturados ou semi-estruturados. De acordo com Moro et al. (2009), XML vem sendo explorada tanto pela comunidade de pesquisa quanto pela indústria. Com o passar do tempo, XML consolidou-se como o formato padrão para intercâmbio de dados, além de solucionar outros problemas. Dentre esses problemas estão a integração de dados, a interoperabilidade de dados e a publicação de dados na Web.

Mesmo com a ampla utilização de XML como padrão para troca de dados, a maior parte dos dados corporativos é armazenada em Bancos de Dados Relacionais (BDRs) (LI et al., 2003). Além disso, muitas aplicações requerem o armazenamento eficiente de dados XML, o que pode ser alcançado usando um BDR. Dessa forma, a capacidade de gerenciar ambos os tipos de dados (relacionais e XML) torna-se essencial para as organizações.

Segundo Krishnamurthy (2004), o conhecimento acumulado sobre o modelo relacional e ainda a disponibilidade de SGBDRs (*Sistema Gerenciador de Banco de Dados Relacional*) comerciais estáveis e maduros podem ser aproveitados para trabalhar com dados XML. Nesse sentido, surgem diferentes abordagens de armazenamento de dados XML em BDRs. Uma dessas abordagens é apresentada no *framework* X2Rel (*XML to Relational*) (BRUM SACCOL; CAMPOS ANDRADE; PIVETA, 2011).

Esse *framework* é uma alternativa às técnicas que, por exemplo, utilizam uma coluna *string* ou *XMLType* para armazenar documentos XML. Sua abordagem consiste em realizar o mapeamento da estrutura dos documentos XML para uma estrutura relacional, ou seja, para tabelas e colunas. Dessa forma, o *framework* X2Rel apresenta diferentes funcionalidades que permitem que o mapeamento de dados XML para um BDR seja completo, envolvendo a estrutura, os dados e as consultas.

Uma dessas funcionalidades é o componente QMap (*Query Mapper*), cujo objetivo é traduzir consultas XQuery - antes executadas sobre os documentos XML originais - em consultas SQL equivalentes. Essa tradução é necessária pois os dados XML foram mapeados para uma estrutura relacional. Entretanto, existem diferenças significativas entre BDRs e documentos XML, pois são baseados em esquemas de relações e em esquemas hierárquicos, respectivamente. Consequentemente, as suas linguagens de consulta também se diferenciam em vários aspectos (KRISHNAMURTHY, 2004), fato este que impacta na tradução de consultas.

Este trabalho apresenta o mecanismo de tradução de consultas XQuery para SQL (chamado QMap), descrito como um componente do *framework* X2Rel. O mecanismo de tradução de consultas proposto é baseado na especificação de regras de equivalência e possui três etapas principais: geração de uma expressão em álgebra XQuery a partir de uma consulta XQuery; tradução da álgebra XQuery para uma expressão em álgebra relacional; e por fim, a geração da consulta SQL equivalente.

1.1 Exemplo de motivação

Esta seção descreve um exemplo de motivação para o presente trabalho. Para isso, consideram-se os seguintes documentos XML:

Documento `Author.xml` representado na figura 1.1.

```
1 <author>
2   <address>Rua dos Andradas, 89, Santa Maria, RS</address>
3 </author>
```

Figura 1.1 – Documento `Author.xml`

Documento `Writer.xml` ilustrado na figura 1.2.

```
1 <writer>
2   <address>
3     <street>Rua Tuiuti, 45</street>
4     <city>Santa Maria</city>
5     <state>RS</state>
6   </address>
7 </writer>
```

Figura 1.2 – Documento `Writer.xml`

É possível notar que no documento `Author.xml` (Figura 1.1) o endereço é representado através de uma *string* única, conforme mostra a linha 2. Já no documento `Writer.xml` (Figura 1.2) o endereço é representado através de três *tags* diferentes: *street* (linha 3), *city* (linha 4) e *state* (linha 5).

Supondo que exista a seguinte consulta Q1 sobre os documentos XML anteriores: "*Mostrar o endereço dos autores que moram no RS*". Para responder esta consulta é necessário considerar as diferentes estruturas dos documentos XML.

Sendo assim, a partir da estrutura do documento `Author.xml`, uma possível consulta é a "Q1-A", ilustrada na figura 1.3.

```
1  for $a in /author
2  where $a/ends-with(address, "RS")
3  return $a/address
```

Figura 1.3 – Consulta XQuery “Q1-A”

Por outro lado, a consulta “Q1-B”, ilustrada na figura 1.4, pode ser elaborada levando em consideração a estrutura do documento `Writer.xml`.

```
1  for $a in /writer/address
2  where $a/state = "RS"
3  return
4    <address>
5      {concat($a/street, ",", $a/city, ",", $a/state)}
6    </address>
```

Figura 1.4 – Consulta XQuery “Q1-B”

Um possível esquema relacional para armazenar esses dados é a tabela `AUTHOR`, com uma coluna `address` armazenando o endereço completo (para o documento `Author.xml`) ou a concatenação dos sub-elementos de endereço (para o documento `Writer.xml`):

`AUTHOR` (codAuthor, . . . , `address`).

Nesse caso, somente uma instrução SQL (“Q1-SQL”) é suficiente para responder a consulta Q1:

```
select address from author where address like '%RS'.
```

Como o endereço é armazenado em uma *string* única na coluna *address*, o comando `like` foi utilizado para possibilitar a busca por *substrings*.

1.2 Objetivos e contribuições

O objetivo geral deste trabalho é definir um mecanismo de tradução automática de consultas XQuery para SQL, através de regras de equivalência entre a álgebra XML (álgebra XQuery) e a álgebra relacional. Como objetivos específicos, podem ser citados:

- Especificar a arquitetura geral do mecanismo de tradução de consultas;
- Definir regras de tradução entre álgebra XQuery e álgebra relacional;
- Especificar regras/mecanismos que garantam a equivalência entre as consultas XQuery de entrada e as consultas SQL de saída;
- Prototipar o mecanismo proposto para a tradução de consultas;

- Realizar testes de tradução e de verificação de equivalência, com análise dos resultados.

Neste sentido, as contribuições deste trabalho são:

- Definição da arquitetura de funcionamento para o componente QMap;
- Definição de regras de tradução e equivalência entre álgebra XQuery e álgebra Relacional;
- Especificação da gramática da álgebra XQuery.

A opção de realizar a tradução no nível algébrico, ou seja, entre a álgebra XQuery e a álgebra relacional, ocorre por alguns motivos, tais como: realizar a tradução em um nível semântico das linguagens; garantir a independência de linguagens de entrada e saída permitir a otimização dos resultados através de mecanismos já consolidados.

1.3 Organização do texto

Este trabalho está organizado da seguinte forma: o capítulo 2 aborda a fundamentação teórica sobre a linguagem XQuery e sua respectiva álgebra. Esse capítulo também descreve o problema da tradução de consultas XML para SQL e os trabalhos existentes que visam resolvê-lo. Além disso, o *framework* X2Rel é apresentado, destacando-se o componente CMap, cuja saída é utilizada nesse trabalho.

No capítulo 3 são definidas regras de tradução e equivalência entre a álgebra XQuery e a álgebra relacional. Também são apresentados os escopos da linguagem XQuery e de sua álgebra considerados pelas regras definidas. Além disso, as gramáticas da álgebra XQuery e da álgebra relacional são descritas a fim de serem a base para a definição das regras de tradução.

O capítulo 4 descreve o funcionamento do componente QMap. Além da arquitetura, a proposta de cada subcomponente é apresentada. Sendo assim, as regras de inferência utilizadas na geração de expressões algébricas a partir de consultas XQuery são descritas. Além disso, as regras de tradução entre a álgebra XQuery e a álgebra relacional são exemplificadas. Por fim, as regras de tradução entre operadores da álgebra relacional e da SQL são apresentadas.

No capítulo 5 um estudo de caso é apresentado, onde são realizados experimentos em cada um dos subcomponentes do componente QMap, com o objetivo de verificar a aplicação das regras definidas. Esse capítulo também descreve os detalhes relacionados à implementação do protótipo, as tecnologias utilizadas em cada etapa, bem como o seu funcionamento.

Por fim, no capítulo 6 são apresentadas as conclusões e as contribuições deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo descreve os conceitos básicos referentes à linguagem de consulta XQuery e uma de suas álgebras, importantes para o entendimento da proposta deste trabalho. Além disso, os trabalhos relacionados ao tema de tradução de consultas são apresentados e analisados.

2.1 Linguagem XQuery

O W3C (*World Wide Web Consortium*) desenvolveu XQuery como uma linguagem de consulta padrão para XML (W3C, 2010a). A XQuery permite, além da seleção dos dados XML de interesse, a reorganização e a transformação dos mesmos, com resultados estruturados de acordo com as necessidades do usuário. Ela incorpora praticamente todos os tipos básicos de expressões XPath (W3C, 2010b), inclusive as expressões de caminho.

2.1.1 Expressões FLWOR

XQuery (W3C, 2010a) fornece um recurso chamado de expressão FLWOR, que oferece suporte à iteração e à ligação de variáveis para resultados intermediários, sendo útil na computação de junções e na reestruturação de dados. Essas expressões são organizadas em cinco cláusulas: `for`, `let`, `where`, `order by` e `return`. A figura 2.1 ilustra uma consulta XQuery que utiliza uma expressão FLWOR com todas as cláusulas. Essa consulta retorna uma lista ordenada com os nomes de todos os produtos no catálogo do departamento “ACC”.

```
1  for $product in doc("catalog.xml")/catalog/product
2  let $name := $product/name
3  where $product/@dept = "ACC"
4  order by $name
5  return $name
```

Figura 2.1 – Exemplo de consulta XQuery (Walmsley (2007))

A cláusula `for` da XQuery é semelhante ao comando `from` da linguagem SQL, mas especifica variáveis que recebem os resultados das expressões XPath. No caso de serem especificadas mais de uma variável, os resultados incluem o produto cartesiano dos valores possíveis que as variáveis podem assumir, assim como na cláusula `from` da SQL.

Segundo Walmsley (2007), a cláusula `let` funciona como uma conveniência programática, evitando a repetição de uma determinada expressão. Pode-se considerar ainda que seu

uso melhora o desempenho das consultas, pois evita que uma expressão seja avaliada diversas vezes. A cláusula `let` pode conter uma ou mais variáveis, cada uma com sua expressão associada. Ao contrário da cláusula `for`, a cláusula `let` vincula cada variável ao resultado de sua expressão associada, sem iteração.

A cláusula `where` da XQuery, assim como a cláusula `where` da linguagem SQL, realiza testes adicionais sobre as tuplas unidas a partir das cláusulas `for` e `let`. A expressão na cláusula `where` é avaliada uma vez para cada uma dessas tuplas e define quais delas serão mantidas e usadas na cláusula de retorno.

A cláusula `order by` da XQuery, assim como a cláusula `order by` da linguagem SQL, permite a classificação da saída. Ela reordena as tuplas no fluxo de tuplas em uma nova ordem baseada em valor, mesmo que a chave de ordenação não seja incluída no resultado da expressão.

Por fim, a cláusula `return` permite a construção dos resultados em XML. Essa cláusula é avaliada uma vez para cada tupla no fluxo de tupla, e os resultados dessas avaliações são concatenados para formar o resultado da expressão FLWOR.

2.1.2 Operações de junção

De acordo com Walmsley (2007), um dos maiores benefícios das expressões FLWOR em XQuery é que elas permitem junções entre dados de diferentes fontes. Essas junções são semelhantes às junções em SQL, sendo que podem ser realizadas entre diferentes elementos de um ou mais documentos XML.

A consulta XQuery ilustrada na figura 2.2 considera a junção de informações de um catálogo de produtos (Figura 2.3) e seus pedidos (Figura 2.4). Para isso, supõe-se a necessidade de gerar uma lista de todos os itens do pedido, juntamente com seu número, nome e quantidade.

```
1  for $item in doc("order.xml")//item
2  let $product := doc("catalog.xml")//product
3  where $product/number = $item/@num
4  return <item  num = "{$item/@num}"
5                name = "{$product/name}"
6                quan = "{$item/@quantity}" />
```

Figura 2.2 – Exemplo de junção em XQuery (adaptado de Walmsley (2007))

Documento `catalog.xml`, ilustrado na figura 2.3.

```
1  <catalog>
2    <product dept="WMN">
3      <number>557</number>
4      <name language="en">Fleece Pullover</name>
5      <colorChoices>navy black</colorChoices>
6    </product>
7    <product dept="ACC">
8      <number>563</number>
9      <name language="en">Floppy Sun Hat</name>
10   </product>
11   <product dept="ACC">
12     <number>443</number>
13     <name language="en">Deluxe Travel Bag</name>
14   </product>
15   <product dept="MEN">
16     <number>784</number>
17     <name language="en">Cotton Dress Shirt</name>
18     <colorChoices>white gray</colorChoices>
19     <desc>Our <i>favorite</i> shirt!</desc>
20   </product>
21 </catalog>
```

Figura 2.3 – Documento `catalog.xml`

Documento `order.xml`, representado pela figura 2.4.

```
1 <order num="00299432" date="2006-09-15" cust="0221A">
2   <item dept="WMN" num="557" quantity="1" color="navy" />
3   <item dept="ACC" num="563" quantity="1" />
4   <item dept="ACC" num="443" quantity="2" />
5   <item dept="MEN" num="784" quantity="1" color="white" />
6   <item dept="MEN" num="784" quantity="1" color="gray" />
7   <item dept="WMN" num="557" quantity="1" color="black" />
8 </order>
```

Figura 2.4 – Documento `order.xml`

Nota-se na consulta (Figura 2.2) que o nome encontra-se no catálogo de produtos (linha 5) e a quantidade está cadastrada nos pedidos (linha 6). Como o número do produto está presente em ambos documentos de entrada, ele é usado para juntar as duas fontes, conforme mostrado na cláusula `where` (linha 3).

2.1.3 Outros recursos

Segundo Walmsley (2007), a XQuery permite a construção de consultas aninhadas. Sua principal utilidade é a geração de elementos com características distintas das encontradas no documento original. Geralmente as consultas aninhadas são expressões FLWOR definidas dentro de uma cláusula `return`, como mostrado nas linhas 7, 8 e 9 do exemplo da figura 2.5. Essa

consulta retorna uma lista com o nome do cliente (linha 5) e as informações sobre sua conta (linha 9) inseridos em um elemento `<cliente>` (linhas 4 a 11).

```
1  xquery version "1.0";
2  for $c in /banco/cliente
3  return
4     <cliente>
5        {$c/nome_cliente}
6        {
7           for    $d in /banco/depositante[nome_cliente = $c/nome_cliente],
8                  $a in /banco/conta[numero_conta = $d/numero_conta]
9           return $a
10       }
11    </cliente>
```

Figura 2.5 – Exemplo de consulta aninhada em XQuery (adaptado de Silberschatz (2006))

A XQuery também fornece diversas funções embutidas, que atendem a um amplo conjunto de funcionalidades. Walmsley (2007) afirma que as funções são usadas para manipular números, *strings* e datas, combinar sequências de elementos, dentre outros. Também é permitido aos usuários definir suas próprias funções, seja na consulta ou em uma biblioteca externa. A chamada *doc*, presente na linha 1 das consultas das figuras 2.1 e 2.2, é um exemplo de função que, nesse caso, retorna o nó do documento especificado como argumento.

Destacam-se outras funcionalidades oferecidas pela XQuery, tais como: construções `if- then-else`, geralmente usadas na cláusula `return`; quantificação existencial e universal (`some` | `every`, `satisfies`), que podem ser usadas na cláusula `where`; e ainda as expressões de comparação, classificadas como sendo de valores (“`eq`” | “`ne`” | “`lt`” | “`le`” | “`gt`” | “`ge`”), comparação geral (“=” | “`!=`” | “<” | “<=” | “>” | “>=”) ou entre nós XML (“`is`” | “«” | “»”).

A seção a seguir descreve as características de funcionamento e os operadores da álgebra XQuery, definida por Ré, Siméon e Fernández (2006).

2.2 Álgebra XQuery

O mecanismo de tradução de consultas proposto neste trabalho tem como base a definição de regras de equivalência. Essas regras são especificadas nos seguintes níveis: de XQuery para álgebra XQuery; de álgebra XQuery para a álgebra relacional; e de álgebra relacional para SQL.

A álgebra XQuery utilizada neste trabalho foi definida em (RE; SIMEON; FERNAN-

DEZ, 2006). Seu desenvolvimento tem como objetivo fornecer uma álgebra que seja a base para implementações completas, corretas e eficientes da linguagem XQuery. Essa álgebra mescla ideias de trabalhos anteriores, estendendo um grande fragmento de álgebra baseada em tupla com operadores XML específicos, baseados em árvore.

Os operadores dessa álgebra possuem a seguinte notação:

$$Op[p_1, ..., p_i]DOp_1, ..., DOp_h(Op_1, ..., Op_k)$$

Onde Op é o nome do operador; $p_1, ..., p_i$ são parâmetros estáticos do operador; $DOp_1, ...,$ DOp_h são operadores dependentes e $Op_1, ..., Op_k$ são operadores de entrada (ou independentes).

Para exemplificar a construção dos operadores da álgebra XQuery, a figura 2.6 apresenta a expressão em algébrica da consulta presente na figura 2.2.

```
1  MapToItem
2    {Element[item]
3      (Sequence
4          (Attribute[num]((IN#item)/@num),
5          Sequence
6              (Attribute[name]((IN#product)/name),
7              Attribute[quan]((IN#item)/@quantity))))
8    }
9    (Join {(IN#product)/number = (IN#item)/@num}
10     (MapConcat
11         {[product : doc("catalog.xml")//product]}
12             (MapConcat{MapFromItem{[item:IN]}(doc("order.xml")//item)}(IN))))
```

Figura 2.6 – Exemplo de álgebra XQuery (referente à consulta XQuery da Figura 2.2)

Os operadores da álgebra XQuery são classificados como: operadores XML, de tupla e XML/tupla. A seguir, cada grupo de operadores é descrito em detalhes.

2.2.1 Operadores XML

Representam operações sobre valores XML. Grande parte desses operadores equivale a exatamente uma expressão em XQuery, possuindo a mesma semântica. Esse grupo de operadores é subdivido em:

- Construtores XML. São operadores de construção de itens e sequências de itens XML. Sua semântica visa criar um nó (ou sequência de nós) correspondente. Alguns exemplos de operadores desse subgrupo são: *Sequence* (linhas 3 e 5 da figura 2.6), *Element* (linha

2 da figura 2.6), *Attribute* (linhas 4, 6 e 7 da figura 2.6), *Scalar*. Estes criam, respectivamente, sequências de itens XML, elementos, atributos e valores atômicos.

- Navegação e Projeção. Esse subgrupo contém operadores para navegação e projeção em árvores XML. Um exemplo é o operador de navegação *TreeJoin*, que retorna um conjunto de nós ordenados a partir da aplicação de um teste de nó sobre um conjunto de nós ordenados iniciais.

- Operadores de Tipo. Nesse subgrupo encontram-se os operadores que implementam operações de tipo em XQuery. São exemplos desse subgrupo os operadores *Cast*, *Validate* e *TypeMatches*, que realizam, respectivamente, operações de fusão, validação e correspondência de tipo.

- Operadores Funcionais. São operadores para funções e condições. A sua semântica é definida em um contexto de álgebra implícita, que contém parâmetros de funções e os planos de compilação de consulta para funções definidas pelo usuário. Os operadores *Cond*, *Var* e *Call* realizam, respectivamente, operações condicionais, de parametrização e chamada de funções.

- Operadores I/O. O último subgrupo de operadores XML contém os operadores *Parse* e *Serialize*, que realizam análise e serialização de documentos, respectivamente.

2.2.2 Operadores de tupla

Este grupo de operadores possui uma semântica que preserva a ordem de sequência. Os dois primeiros subgrupos contém os operadores padrão para construção de tupla, concatenação e acesso a campo, seleção, projeção e junções. Este grupo de operadores é subdividido em:

- Construtores de Tupla. Um exemplo de construtor de tupla é o operador ++, que realiza a concatenação de tuplas.

- Seleção, Projeção e Junção. Exemplos de operadores desse subgrupo são: *Select*, *Product* e *Join* (linha 9 da figura 2.6), que representam operações de seleção, produto cartesiano e junção, respectivamente.

- Mapeamento. Este subgrupo contém operadores de mapeamento. *Map* é um mapeamento funcional geral sobre sequências de tuplas. *MapConcat* (linhas 10 e 12 da figura 2.6) é

uma junção dependente. Os operadores *OMap*, *OMapConcat*, *MapIndex* e *MapIndexStep* são utilizados durante o desaninhamento de consulta. O operador *MapIndex* é usado para compilar expressões FLWOR com uma variável índice (*at* $ *a*, por exemplo) e também é usado para computar a operação *position()*.

- Agrupamento e Classificação. Este subgrupo contém os operadores *OrderBy* e *GroupBy*, para classificação e agrupamento de tuplas.

2.2.3 Operadores XML/tupla

São quatro operadores que ficam no limite entre a parte tupla e a parte XML da álgebra. Este grupo de operadores é subdividido em:

- Mapeamento. Este subgrupo contém os operadores *MapFromItem* (linha 12 da figura 2.6) e *MapToItem* (linha 1 da figura 2.6), que representam as cláusulas `for` e `return`, respectivamente.

- Quantificação. Os operadores *MapSome* e *MapEvery* são usados para compilar expressões de quantificação XQuery.

A seção a seguir descreve o *framework* X2Rel, no qual o presente trabalho está inserido.

2.3 Framework X2Rel

O X2Rel (*XML to Relational*) é um *framework* para armazenamento de dados XML em bancos de dados relacionais (BDR). O *framework* X2Rel segue uma abordagem baseada no uso de ontologias, sendo que este é seu diferencial como uma alternativa de mapeamento da estrutura XML para um esquema relacional. Dessa forma, documentos XML pertencentes a um mesmo domínio, mas com diferentes estruturas, podem ser adequadamente armazenados em um BDR.

Para permitir o mapeamento completo - de estrutura, dados e consultas - de dados XML para um BDR, o *framework* X2Rel apresenta alguns componentes, conforme ilustrado na figura 2.7:

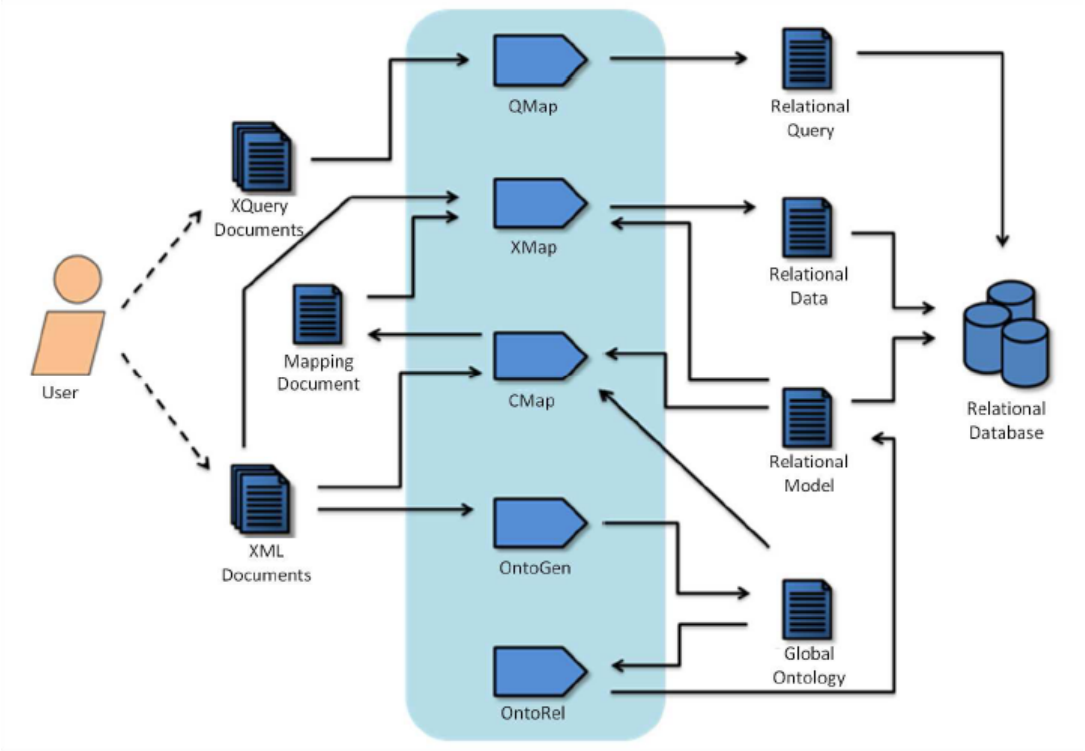

Figura 2.7 – Arquitetura do *framework* X2Rel (AVELAR; BRUM SACCOL; PIVETA, 2012)

Os componentes apresentados na figura 2.7 são responsáveis pelas funcionalidades descritas a seguir:

- Integração dos documentos XML em um esquema global, descrito por uma ontologia OWL (*Ontology Web Language*), fornecida pelo componente OntoGen (*Ontology Generator*). Este módulo recebe um conjunto de documentos XML e produz a ontologia (esquema integrado) (BRUM SACCOL et al., 2008);

- Tradução da ontologia OWL para um esquema relacional, fornecida pelo componente OntoRel (*Ontology to Relational*). Este módulo recebe a ontologia e produz um esquema relacional (um *script* SQL com os comandos *create table*) (BRUM SACCOL; CAMPOS ANDRADE; PIVETA, 2011);

- Detecção e representação das equivalências de conceitos entre os documentos XML, a ontologia OWL e o esquema relacional, fornecidas pelo componente CMap (*Concept Mapper*). Este módulo recebe os documentos XML, a ontologia e o esquema relacional, e produz o documento de mapeamento (um documento XML com as equivalências detectadas) (NEGRINI et al., 2012);

- Mapeamento e inserção dos dados XML originais no BDR, fornecidos pelo componente XMap (*XML Mapping*). Este módulo recebe os documentos XML, a ontologia e o esquema relacional e produz o *script* SQL com um conjunto de instruções *insert* (AVELAR; BRUM SACCOL; PIVETA, 2012);

- E finalmente, tradução das consultas XML originais em instruções SQL equivalentes, fornecida pelo componente QMap (*Query Mapper*). Este módulo recebe uma consulta XQuery e produz a consulta SQL correspondente.

Os componentes OntoGen, OntoRel, CMap e XMap são trabalhos finalizados e publicados. O foco deste trabalho é apresentar o mecanismo QMap, descrito no Capítulo 4. A seguir, o módulo CMap é brevemente detalhado, pois sua saída (documento de mapeamento) é utilizada pelo mecanismo QMap.

2.3.1 Concept Mapper (CMAP)

O módulo CMap (*Concept Mapper*) tem o objetivo de gerar um documento de mapeamento, que detecta e representa as equivalências entre os documentos XML, a ontologia global OWL e o esquema relacional. Este documento é descrito na linguagem XML, sendo que é possível encontrar representações tanto de elementos e atributos XML quanto de conceitos relacionais e da ontologia.

A seguir um fragmento de um documento de mapeamento é demonstrado na figura 2.8. É possível notar que no documento de mapeamento são representados os conceitos da ontologia, juntamente com as respectivas informações dos documentos XML de origem e do esquema relacional gerado. Dessa forma, o documento de mapeamento é utilizado pelo componente QMap para identificar a equivalência entre o elemento do documento XML, que é consultado pela expressão XQuery de entrada, e o elemento do esquema relacional, que deve ser utilizado na composição da consulta SQL de saída.

O fragmento descrito na figura 2.8 representa o mapeamento do conceito da ontologia `depositante` (linha 3). Nota-se que o conceito está presente apenas no *DocumentoA.xml* (linhas 4 a 6), já que este possui uma XPath, `/banco/depositante` (linha 5), que liga o conceito a um elemento XML do documento. O elemento <*relational*> (linhas 8 a 11) descreve o mapeamento do conceito `depositante` que, neste caso, foi mapeado para o esquema relacional como uma tabela, conforme demonstrado nas linhas 9 e 10.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <mapping>
3     <concept name="depositante">
4        <source id="DocumentoA.xml">
5           <xpath>/banco/depositante</xpath>
6        </source>
7        <source id="DocumentoB.xml"/>
8        <relational>
9           <type>table</type>
10          <name>depositante</name>
11       </relational>
12    </concept>
13    <concept name="nome_cliente">
14       <source id="DocumentoA.xml">
15          <xpath>/banco/depositante/nome_cliente</xpath>
16       </source>
17       <source id="DocumentoB.xml"/>
18       <relational>
19          <type>column</type>
20          <name>nome_cliente</name>
21          <table>depositante</table>
22          <domain>string</domain>
23       </relational>
24    </concept>
25       ...
26 </mapping>
```

Figura 2.8 – Fragmento de documento de mapeamento

O mapeamento do conceito da ontologia `nome_cliente` (linha 13) também é representado no fragmento anterior. Esse conceito está presente somente no *DocumentoA.xml* (linhas 14 a 16), sendo que é representado através da XPath `/banco/depositante/nome_cliente` (linha 15). As linhas 18 a 23 indicam o mapeamento do conceito `nome_cliente` para o esquema relacional como sendo uma coluna (linha 19) da tabela `depositante` (linha 21).

A especificação do documento de mapeamento faz parte do módulo XMap desenvolvido como uma dissertação de mestrado e publicado por (AVELAR; BRUM SACCOL; PIVETA, 2012). Já a especificação e implementação do módulo CMap, que gera o documento de mapeamento automaticamente, foi realizada como parte de um trabalho de conclusão de curso em Ciência da Computação, sendo que mais detalhes podem ser encontrados em (NEGRINI, 2011) e (NEGRINI et al., 2012).

2.4 Trabalhos relacionados

Existem várias abordagens que traduzem dialetos XQuery ou expressões XPath para consultas SQL, tais como: (DEHAAN et al., 2003); (KRISHNAMURTHY, 2004); (GRUST,

2005); (GRUST et al., 2007); (WANG; WANG; RUNDENSTEINER, 2007) e (FAN et al., 2009). Muitas destas abordagens definem regras de tradução ou algoritmos. A abordagem de tradução apresentada em (DEHAAN et al., 2003) usa uma codificação de intervalos dinâmicos de coleções de documentos XML e informações ligadas ao ambiente de avaliação da consulta. O trabalho considera a linguagem XQuery com expressões FLWR arbitrariamente aninhadas, construtores de elemento e funções embutidas.

Já o trabalho de (KRISHNAMURTHY, 2004) define dois cenários em que a tradução de consultas XML para SQL é necessária: publicação e armazenamento de dados XML em BDR. Para cada cenário, é definido um algoritmo de tradução de consultas XPath, que considera as informações do mapeamento entre os esquemas XML e relacional. Dois diferenciais apresentados pelo trabalho quando comparado com abordagens anteriores são o suporte a consultas XPath recursivas (com o operador //) e a geração de consultas SQL com maior qualidade.

A abordagem desenvolvida em (GRUST, 2005) apresenta um procedimento de compilação que deriva planos de álgebra relacional a partir de blocos FLWOR XQuery arbitrariamente aninhados. As técnicas desenvolvidas são consideradas puramente relacionais, pois utilizam uma álgebra clássica, que é apoiada pelos BDR e empregam conceitos relacionais como, por exemplo, dependência funcional, a fim de simplificar os planos emitidos.

Em (GRUST et al., 2007) é proposto um gerador de código SQL:1999 para o compilador XQuery Pathfinder (PATHFINDER, 2011). O projeto Pathfinder engloba um compilador *front-end* que traduz expressões XQuery de entrada em um plano de álgebra relacional no formato DAG (*Directed Acyclic Graph*). Assim, o gerador de código torna esses planos em sequências de declarações SQL:1999 ou definições de visões que implementam a semântica, as vezes complexa, da XQuery. Entretanto, o trabalho utiliza uma variação da álgebra relacional clássica, gerando planos algébricos fragmentados, fazendo com que o BDR avalie o plano geral em partes separadas.

A abordagem apresentada em (WANG; WANG; RUNDENSTEINER, 2007) propõe o processamento eficiente de XQuery sensível a ordem. Essa técnica envolve três etapas: inferência e isolamento da ordem semântica em expressões XQuery sobre visões XML; otimização lógica do plano de consulta, sem perda de semântica; e tradução desse plano lógico otimizado para SQL, garantindo a ordem semântica original. Esse trabalho utiliza a álgebra XAT (*XML Algebra Tree*), definida em (WANG; WANG; RUNDENSTEINER, 2007) para representar os planos de consulta e realiza várias reescritas a fim de otimizá-los, antes da geração dos co-

mandos SQL. As técnicas desenvolvidas são aplicadas ao mecanismo Rainbow (ZHANG et al., 2003).

O trabalho (FAN et al., 2009) define uma abordagem de tradução de consultas XPath, baseada em uma linguagem XPath estendida. Ao longo do trabalho, são apresentados dois algoritmos, sendo que um refere-se a tradução de expressões XPath para a linguagem XPath estendida, e o outro sobre tradução dessas consultas para SQL.

Particularmente, o trabalho de (KRISHNAMURTHY, 2004) contribui para que a definição do mecanismo QMap seja considerada pertencente a um cenário de armazenamento de dados XML em BDR. Já o trabalho de (WANG; WANG; RUNDENSTEINER, 2007) demonstra a existência e utilização de álgebras XML na tradução de consultas. Nesse sentido, os trabalhos de (GRUST, 2005) e (GRUST et al., 2007) contribuem como um exemplo de utilização da álgebra relacional no processo de tradução de consultas.

2.4.1 Comparativo

A abordagem proposta para o componente QMap usa um dialeto XQuery semelhante ao apresentado nos trabalhos (DEHAAN et al., 2003), (GRUST, 2005), (GRUST et al., 2007) e (WANG; WANG; RUNDENSTEINER, 2007)). Esta é uma vantagem sobre as abordagens de tradução XPath ((KRISHNAMURTHY, 2004) e (FAN et al., 2009)), pois a XPath é um subconjunto da XQuery. O presente trabalho também trata as expressões FLWOR XQuery, incluindo a cláusula `order by` inexistente em (DEHAAN et al., 2003).

Na seção 4.2 é definida a arquitetura do componente QMap, que recebe uma consulta XQuery e retorna uma instrução SQL equivalente. Ela é baseada em três etapas: geração da expressão em álgebra XQuery; tradução desta expressão para a álgebra relacional e geração da consulta SQL. Esta arquitetura proposta parece ser mais complexa comparada à arquitetura presente nos trabalhos (GRUST, 2005) e (GRUST et al., 2007), que realizam a tradução direta entre XQuery e álgebra relacional.

Porém, esses trabalhos estão inseridos no projeto do compilador XQuery Pathfinder (PATHFINDER, 2011), que utiliza uma variação da álgebra relacional clássica e usa a consulta SQL como uma etapa intermediária para obter resultados XML. Comparando a arquitetura do componente QMap à arquitetura definida em (WANG; WANG; RUNDENSTEINER, 2007), a primeira é menos fragmentada, não possuindo etapas de otimização e reescrita das expressões algébricas, como ocorre na segunda abordagem. Além disso, o componente QMap utiliza a

álgebra XQuery (RE; SIMEON; FERNANDEZ, 2006), que é uma evolução da álgebra XAT (ZHANG et al., 2002), utilizada em (WANG; WANG; RUNDENSTEINER, 2007).

Outro diferencial do mecanismo QMap é a definição de regras de tradução e equivalência. A maioria dos trabalhos relacionados, tais como (DEHAAN et al., 2003) e (FAN et al., 2009), apenas apresenta experimentos e resultados, sem especificar as regras de tradução utilizadas. Já outros trabalhos, como (KRISHNAMURTHY, 2004), apresentam algoritmos de tradução ou regras de compilação, sem estabelecer uma equivalência entre as duas linguagens envolvidas na tradução.

2.5 Considerações finais

Nesse capítulo foram abordados os conceitos básicos relacionados ao presente trabalho. A linguagem XQuery foi apresentada com suas expressões FLWOR, recurso que a torna semelhante a SQL. A álgebra XQuery, definida em (RE; SIMEON; FERNANDEZ, 2006), também foi descrita através dos conjuntos de operadores existentes.

O *framework* X2Rel também foi abordado nesse capítulo, a fim de contextualizar a proposta deste trabalho. Por fim, os trabalhos relacionados ao tema de tradução de consultas XML para SQL foram descritos e também comparados.

3 TRADUÇÃO DA ÁLGEBRA XQUERY PARA ÁLGEBRA RELACIONAL

Este capítulo tem o objetivo de descrever regras de tradução entre a álgebra XQuery e a álgebra relacional. As regras são definidas de acordo com o escopo da linguagem XQuery, da sua álgebra e da álgebra relacional. Também são definidas regras inversas de tradução, a fim de garantir a equivalência das expressões de entrada e saída.

3.1 Escopos e gramáticas

Esta seção descreve os escopos da linguagem XQuery, da álgebra XQuery e as gramáticas da álgebra XQuery e relacional usadas pelas regras de tradução.

3.1.1 Escopos da linguagem XQuery e álgebra XQuery

Esta seção apresenta os escopos da linguagem XQuery e sua álgebra. Basicamente, o dialeto XQuery suportado, mostrado na tabela 3.1, consiste em cláusulas FLWOR e seus respectivos sub-operadores que podem estar presentes em suas expressões internas. Similarmente, o escopo da álgebra XQuery, apresentado na tabela 3.2, contém os operadores algébricos que representam as cláusulas FLWOR e seus sub-operadores.

Tabela 3.1 – Dialeto suportado da linguagem XQuery

sequence (e_1, e_2)
variables ($\$v$)
atomic literals
XPath $(e_1/s[[e_2]])$
Comparison Operators (`=, eq, >=, ..`)
Boolean connectives (`and, or`)
`for` $\$v$ `in` e_1 `return` e_2
`let` $\$v$ `:=` e_1 `return` e_2
e_1 `where` φ `return` e_2
e_1 `order by` e_2`, .., ` e_n
`element {`e_1`}{`e_2`}`
`attribute {`e_1`}{`e_2`}`

A tabela 3.1 representa algumas características que constituem o dialeto considerado neste trabalho. Dentre essas características está uma sequência de expressões XQuery, representada por *sequence* (e_1, e_2). Essas expressões, e_1 e e_2, podem ser qualquer construção re-

presentada pelo dialeto. Também estão incluídas nesse dialeto as variáveis ($\$v$, por exemplo) e as literais atômicas (valores *string*, data ou numéricos, por exemplo). As expressões XPath também fazem parte do dialeto, sendo representadas no formato $e_1/s[[e_2]]$, onde s significa um passo XPath, e_1 e e_2 são expressões XQuery. Dentre os operadores que podem estar incluídos nessas expressões estão os de comparação (`=, eq, >=, ..`) e booleanos (`and, or`).

Da mesma forma, as cláusulas FLWOR e os construtores de elemento (`element {`e_1`} {`e_2`}`) e atributo (`attribute {`e_1`}{`e_2`}`) fazem parte do dialeto representado na tabela 3.1. As cláusulas `for` e `let` podem conter expressões (e_1) ligadas a variáveis ($\$v$), sempre seguidas pela cláusula `return` e sua expressão (e_2). Já a cláusula `where`, que é opcional, deve ser antecedida por uma expressão (e_1), definir uma expressão de comparação (φ) e ser seguida pela cláusula `return`. Semelhantemente, a cláusula `order by` é posterior à uma expressão (e_1) e é seguida por uma ou mais expressões (e_2, .., e_n) usadas na classificação.

Na tabela 3.2 os operadores algébricos $MapFromItem$ e $MapConcat$ representam, respectivamente, as cláusulas `for` e `let` da XQuery. A assinatura do operador $MapFromItem$ contém uma tupla (τ) sobre um item (valor atômico ou nó XML representado por i) e valores XML ($S(i)$). O operador $MapConcat$ é construído com uma tabela de valores XML ($S(\tau_2)$) sobre uma tupla do tipo τ_1 e outra tabela de valores XML (($S(\tau_1)$)).

Tabela 3.2 – Escopo suportado da álgebra XQuery

$MapFromItem\{i \rightarrow \tau\}(S(i))$
$MapConcat\{\tau_1 \rightarrow S(\tau_2)\}(S(\tau_1))$
$Select\{\tau_1 \rightarrow boolean\}(S(\tau_1))$
$OrderBy\{\tau, \tau \rightarrow boolean\}(S(\tau))$
$MapToItem\{\tau \rightarrow i\}(S(\tau))$
$Sequence(S(i_1), S(i_2))$
$Element[q](S(i))$
$Attribute[q](S(a))$

Os operadores $Select$ e $OrderBy$ equivalem-se as cláusulas `where` e `order by`, respectivamente. O operador $Select$ considera um operador booleano dependente ($boolean$) sobre uma tupla do tipo τ_1 e uma sequência de tuplas do tipo τ_1 ($S(\tau_1)$). Já o operador $OrderBy$ considera um operador booleano dependente ($boolean$) sobre duas tuplas (τ, τ) e uma sequência de tuplas ($S(\tau)$).

Já os operadores $MapToItem$, $Sequence$, $Element$ e $Attribute$ representam a cláusula `return` e seus possíveis construtores de elementos e atributos XML. A construção do operador $MapToItem$ considera um item (i) sobre uma tupla (τ) e uma sequência de tuplas ($S(\tau)$).

Já a assinatura do operador $Sequence$ consiste em duas sequências de itens ou tuplas XML ($S(i_1), S(i_2)$). Os operadores $Element$ e $Attribute$ consideram um nome de campo (atributo ou elemento XML representado por q) e, respectivamente, uma sequência de itens ($S(i)$) e uma sequência de valores atômicos (como inteiros, *strings* ou decimais, representada por $S(a)$).

A próxima seção descreve as gramáticas das álgebras envolvidas na tradução.

3.1.2 Gramáticas da álgebra XQuery e álgebra relacional

Esta seção apresenta as gramáticas utilizadas como base na definição das regras de tradução. A tabela 3.3 descreve a gramática da álgebra XQuery. É importante ressaltar que essa gramática segue o escopo definido na tabela 3.2, ou seja, somente operadores que representam as cláusulas FLWOR da XQuery são descritos. A especificação da gramática da álgebra XQuery é uma das contribuições deste trabalho, pois (RE; SIMEON; FERNANDEZ, 2006) define seus operadores e assinaturas, juntamente com as regras de inferência usadas na geração de expressões algébricas, mas não define sua sintaxe. A especificação das gramáticas, tanto da álgebra relacional quanto da álgebra XQuery, é importante pois elas são a base das regras de tradução apresentadas na próxima seção.

A notação BNF (*Backus-Naur Form*) foi utilizada para representar a gramática da álgebra XQuery na tabela 3.3. Nessa notação, palavras em minúsculas são não terminais, ou seja, podem ser substituídas. Já as palavras em maiúsculas e símbolos entre ‘ ’ são terminais, geralmente representando operadores, nomes ou valores. Primeiramente, define-se (linhas 1 a 8) que um operador da álgebra XQuery pode ser um operador que representa as cláusulas `for` (`OpFOR` linha 1), `let` (`OpLET` linha 2), `where` (`OpWHERE` linha 3), `order by` (`OpORDERBY` linha 4), `return` (`OpRETURN` linha 5), ou ainda algum construtor de sequência (`OpSEQUENCE` linha 6), de elemento (`OpELEMENT` linha 7) ou de atributo (`OpATTRIBUTE` linha 8).

A gramática também considera que as expressões em álgebra XQuery são construídas por uma abordagem *bottom-up*, em conformidade com as regras de inferência presentes em (RE; SIMEON; FERNANDEZ, 2006) e descritas na seção 4.3. Por esse motivo define-se, na linha 9, que uma expressão FLWOR (`FLWORExpr`), na álgebra XQuery, é um operador `return` (`OpRETURN`).

Tabela 3.3 – Gramática da álgebra XQuery

```
1   Op                ::= OpFOR
2                     | OpLET
3                     | OpWHERE
4                     | OpORDERBY
5                     | OpRETURN
6                     | OpSEQUENCE
7                     | OpELEMENT
8                     | OpATTRIBUTE
9   FLWORExpr         ::= OpRETURN
10  OpRETURN          ::= 'MapToItem {' PathExpr | OpELEMENT | OpSEQUENCE '}
                      ('OpORDERBY | OpWHERE | OpLET | OpFOR')'
11  OpORDERBY         ::= 'OrderBy {' PathExpr '} (' OpWHERE | OpLET |OpFOR
                      ')'
12  OpWHERE           ::= 'Select {' ComparisonExpr '} (' OpLET | OpFOR ')'
13                    | 'Join {' ComparisonExpr '} (' OpLET | OpFOR ')'
14  OpLET             ::= 'MapConcat { [' VarName ':'  PathExpr '] } (' OpFOR
                      ')'
15  OpFOR             ::= 'MapConcat { MapFromItem [' VarName ':  IN ] ('
                      PathExpr ') } (IN)'
16  OpSEQUENCE        ::= 'Sequence (' (PathExpr | OpATTRIBUTE| OpELEMENT |
                      OpSEQUENCE) ',' (PathExpr | OpATTRIBUTE | OpELEMENT |
                      OpSEQUENCE) ')'
17  OpELEMENT         ::= 'Element [' QName '] (' OpELEMENT ')'
18                    | 'Element [' QName '] (' OpSEQUENCE ')'
19                    | 'Element [' QName '] (' OpATTRIBUTE ')'
20                    | 'Element [' QName '] (' PathExpr ')'
21  OpATTRIBUTE       ::= 'Attribute [' QName '] (' PathExpr ')'
22  PathExpr          ::= '/' RelativePathExpr?
23                    | RelativePathExpr
24  RelativePathExpr  ::= StepExpr ('/' StepExpr)*
25  StepExpr          ::= FilterExpr
26                    | AxisStep
27  AxisStep          ::= '@'?  NodeName PredicadeList
28  NodeName          ::= QName
29  PredicadeList     ::= Predicate*
30  Predicate         ::= '[' ComparisonExpr ']'
31  FilterExpr        ::= PrimaryExpr PredicateList
32  PrimaryExpr       ::= Literal
33                    | VarRef
34                    | FunctionCall
35  Literal           ::= NumericLiteral
36                    | StringLiteral
37  NumericLiteral    ::= IntegerLiteral
38                    | DecimalLiteral
39                    | DoubleLiteral
```

```
40  VarRef            ::= '$' VarName
41  VarName           ::= QName
42  FunctionCall      ::= QName '(' (ExprSingle (',' ExprSingle)*)?  ')'
43  ComparisonExpr    ::= ComparisonExpr ('and' ComparisonExpr)*
44                    | ComparisonExpr ('or' ComparisonExpr)*
45                    | PathExpr (ValueComp | GeneralComp) PathExpr
46  ValueComp         ::= 'eq' | 'ne' | 'lt' | 'le' | 'gt' | 'ge'
47  GeneralComp       ::= '=' | '!=' | '<' | '<=' | '>' | '>='
48  QName             ::= QName
```

O operador `return` (linha 10), por sua vez, é definido sintaticamente como um operador *MapToItem*, que pode ser seguido por uma expressão XPath (`PathExpr`), um construtor de elemento (`OpELEMENT`) ou um construtor de sequência (`OpSEQUENCE`). Logo depois, são definidos alguns operadores de entrada, ou seja, operadores que são avaliados de forma independente: `order by`, `where`, `let` ou `for` (`OpORDERBY | OpWHERE | OpLET | OpFOR`). Desses, apenas um será avaliado na sequência de construção da expressão em álgebra XQuery. Caso o operador de entrada seja o `OpORDERBY`, por exemplo, o mesmo é avaliado conforme a definição de sua sintaxe presente na linha 11. Neste caso, gera-se um operador *OrderBy* seguido de uma expressão XPath (`PathExpr`). Semelhante ao operador `return`, na sequência do operador `order by` também são definidos alguns operadores de entrada: `OpWHERE | OpLET | OpFOR`. Desse modo, apenas um dos operadores será avaliado e seu resultado incorporado na expressão em álgebra XQuery.

Os demais operadores seguem o mesmo padrão de definição. Para o operador `where` (linhas 12 e 13) definem-se duas construções sintáticas possíveis: um operador *Select* (linha 12) ou um operador *Join* (linha 13), ambos seguidos por uma expressão de comparação (`ComparisonExpr`) e pelos operadores de entrada `let` e `for` (`OpLET | OpFOR `). Na linha 14 define-se a sintaxe do operador `let` (`OpLET`) como sendo um operador *MapConcat* seguido da ligação de uma variável (`VarName`) à uma expressão XPath (`PathExpr`). Nesse caso, apenas o operador `for` é a opção (não obrigatória) de operador de entrada. Por sua vez, o operador `for` é definido como a sequência dos operadores *MapConcat* e *MapFromItem* seguidos pela ligação de uma variável (`VarName`) à uma expressão XPath (`PathExpr`), conforme mostra a linha 15. Já a linha 16 representa a sintaxe do operador de contrução de sequência (`OpSEQUENCE`). O construtor de elemento é definido sintaticamente nas linhas 17 a 20. Na linha 21 a sintaxe do construtor de atributo é apresentada.

A definição dessa gramática baseia-se na gramática de XQuery (W3C, 2010c), prin-

cipalmente quanto à sintaxe das expressões XPath (linhas 22 a 48). Uma expressão XPath é definida como sendo uma expressão XPath relativa (`RelativePathExpr`, linhas 22 e 23) constituída por diferentes passos (`StepExpr`, linha 24). Conforme mostram as linhas 25 e 26, há duas possibilidades para cada passo (`StepExpr`): ser uma expressão de filtro (`FilterExpr`) ou um eixo (`AxisStep`). Segundo a linha 31, uma expressão de filtro consiste em uma expressão primária (`PrimaryExpr`), ou seja, valores literais, variável ou função (conforme linhas 32, 33 e 34 respectivamente), seguida de uma lista de predicados (`PredicateList`). Esses predicados (linhas 29 e 30) consistem em expressões de comparação, que são sintaticamente definidas nas linhas 43 a 45. Já um eixo (`AxisStep`), consiste em um nó XML que pode ser seguido por uma lista de predicados, conforme mostra a linha 27.

A tabela 3.4 descreve a gramática da álgebra relacional utilizada neste trabalho. Essa gramática está descrita em Molková (2009) e segue a definição presente em (SILBERSCHATZ; KORTH; SUNDARSHAN, 2006), sendo que foi adaptada, neste trabalho, utilizando a notação BNF.

Tabela 3.4 – Gramática da álgebra relacional (adaptado de Molková (2009))

```
1              | exp 'THETA_JOIN' '['condlist']' exp
2              | exp 'CARTESIAN_PRODUCT' exp
3              | 'SELECTION' '['condlist ']' '(' exp ')'
4              | 'PROJECTION' '['attrlist ']' '(' exp ')'
5              | 'RENAME' '[' name '(' attrlist ')' ']' '(' exp ')'
6              | 'RENAME' '[' name ']' '(' exp ')'
7              | 'ORDERBY' '[' attrlist ']' order '(' exp ')'
8              | relation
9   attrlist   ::= attribute
10             | attribute ',' attrlist
11  condlist   ::= condlist 'OR' condlist
12             | condlist 'AND'condlist
13             | 'NOT' condlist
14             | '(' condlist ')'
15             | compared comp compared
16  comp       ::= '<>' | '=' | '>=' | '<=' | '<' | '>'
17  compared   ::= attribute
18             | data
19  relation   ::= NAME
20  attribute  ::= NAME
21             | fullname
22  fullname   ::= NAME '.'  NAME
23  data       ::= NUMBER
24             | STRING_VALUE
25  order      ::= 'asc' | 'desc'
```

É possível notar que essa sintaxe não utiliza os símbolos clássicos (σ, π , ρ, τ, $\bowtie_\theta$, $\times$, ...) para representar as diferentes operações da álgebra relacional. No lugar dos símbolos usa-se a nomenclatura dessas operações, com o objetivo de facilitar sua compreensão.

Especificamente, o intervalo entre as linhas 1 e 8 da tabela 3.4 descreve as possíveis construções sintáticas das expressões de álgebra relacional. A expressão mais simples consiste em uma relação (linha 8), sendo que essa relação é sintaticamente definida na linha 19. A partir disso, é possível realizar combinações entre relações através de operadores binários, tais como `THETA_JOIN` e `CARTESIAN_PRODUCT`, conforme descrito nas linhas 1 e 2. Além disso, também é possível inserir essas expressões em outras expressões que iniciam com operações unárias, tais como `SELECTION`, `PROJECTION`, `RENAME` e `ORDERBY`, conforme demonstrado nas linhas 3 a 7.

O operador `ORDERBY` representado na linha 7 segue a definição presente em (GARCIA-MOLINA; ULLMAN; WIDOM, 2001). Consequentemente, a indicação do tipo de ordenação dado por `order` na linha 25 também procede da mesma fonte. Isso se deve ao fato de o operador `ORDERBY` não constar na gramática descrita por Molková (2009).

Nas expressões de projeção e renomeação, linhas 4 e 5 respectivamente, é necessário informar uma lista de atributos (`attrlist`). Essa lista de atributos é sintaticamente definida nas linhas 9 e 10, sendo que pode ser composta por um ou mais atributos (`attribute`). As expressões envolvendo junção-theta e seleção, apresentadas respectivamente nas linhas 1 e 3, necessitam de uma lista de condições (`condlist`). A sintaxe da lista de condições é definida nas linhas 11 a 15. A possibilidade mais simples encontra-se na linha 15, sendo que representa uma expressão de comparação composta por dois argumentos (`compared`) unidos por um operador de comparação (`comp`).

É importante notar que alguns operadores da álgebra relacional, tais como `UNION`, `INTERSECTION`, `SET_DIFFERENCE`, `NATURAL_JOIN`, `LEFT_JOIN` e `RIGHT_JOIN` não constam na tabela 3.4. Isso ocorre pelo fato de que esses operadores da álgebra relacional não possuem operadores equivalentes nos escopos da linguagem XQuery e sua álgebra.

A próxima seção descreve as regras de tradução entre a álgebra XQuery e a álgebra relacional.

3.2 Regras de tradução da álgebra XQuery para álgebra relacional

Nesta seção são definidas as regras de tradução entre operadores da álgebra XQuery e operadores da álgebra relacional de acordo com as gramáticas definidas na seção 3.1.

As regras de tradução apresentadas a seguir constituem a principal contribuição deste trabalho. Basicamente as regras foram definidas considerando algumas características dos operadores algébricos: na álgebra XQuery os operadores da cláusula `return` equivalem-se a operações de projeção na álgebra relacional; o operador da cláusula `order by` é representado como uma ordenação; operadores da cláusula `where` são equivalentes a seleção e operadores das cláusulas `for` e `let` são como a relação argumento na álgebra relacional.

É importante destacar que as regras são composicionais, ou seja, são definidas em função das subexpressões que compõem cada operador. Isso significa que algumas regras definem traduções que geram elementos terminais. Já as demais regras definem traduções que geram elementos não terminais, ou seja, que dependem da avaliação das regras anteriores.

Nas regras os símbolos $[\![\ ]\!]$ indicam tradução. As expressões algébricas que estão entre $[\![\ ]\!]$ devem ser traduzidas para outras expressões. O símbolo $=$ indica a equivalência de duas expressões no processo de tradução. Já o símbolo $|$ indica que há mais de uma possibilidade de construção de uma expressão.

3.2.1 Regras de tradução para expressões XPath

São 18 as regras definidas relacionadas à tradução de expressões XPath, de acordo com as possibilidades presentes na gramática da tabela 3.3. Elas subdividem-se da seguinte maneira: uma regra geral para expressões XPath, três regras gerais para expressões de comparação e quatorze regras específicas para operadores de comparação.

A primeira regra $(\mathbf{Tr_1})$ define que a tradução de uma expressão XPath ($PathExpr$) equivale ao retorno da função $f(PathExpr, Var, Op_{FLWOR})$, ou seja, as informações relativas ao esquema relacional.

$$(\mathbf{Tr_1})[\![PathExpr]\!] = f(PathExpr, Var, Op_{FLWOR})$$

Essa função realiza uma busca em um documento que contenha as informações do mapeamento dos dados XML para um BD relacional e é descrita mais adiante pelo algoritmo 1. Para um melhor entendimento, a estrutura de um exemplo de documento de mapeamento é apresentada a seguir na figura 3.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD generated by XMLSpy v2013 (x64) (http://www.altova.com)-->
<!ELEMENT xpath (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT table (#PCDATA)>
<!ELEMENT source ((xpath?))>
<!ATTLIST source
   id (book_p2.xml | book_p1.xml | bib.xml) #REQUIRED>
<!ELEMENT relational ((type, name, (table, domain)?))>
<!ELEMENT name (#PCDATA)>
<!ELEMENT mapping ((concept+))>
<!ATTLIST mapping
   xmlns:xsi CDATA #FIXED "http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation CDATA #FIXED "docMap.xsd">
<!ELEMENT domain (#PCDATA)>
<!ELEMENT concept ((source+, relational))>
<!ATTLIST concept
name (year | title | publisher | publish | price | noNamespaceSchemaLocation | last |
 id | genre | first | editor | description | catalog | book | bib | author | affiliation)
 #REQUIRED>
```

Figura 3.1 – Estrutura do documento de mapeamento

A consulta XQuery realizada sobre o documento de mapeamento é apresentada a seguir na figura 3.2 e utiliza como parâmetros a própria expressão XPath ($PathExpr$) e o nome do documento XML de origem:

```
1  for $c in //concept, $s in $c/source
2  where $s/@id="documento.xml" and $s/xpath="expressaoXPath"
3  return
4  <relational>
5    <name>{$c/relational/name/text()}</name>
6    <table>{$c/relational/table/text()}</table>
7    <domain>{$c/relational/domain/text()}</domain>
8  </relational>
```

Figura 3.2 – Consulta XQuery sobre o documento de mapeamento

Essa consulta XQuery retorna diferentes informações do modelo relacional representadas por elementos XML presentes no documento de mapeamento, conforme o trecho acima apresentado. Caso a expressão XPath corresponda a uma coluna no modelo relacional, então são retornados o nome (elemento `<name>` da linha 5), a tabela (elemento `<table>` da linha 6) e o domínio (elemento `<domain>` da linha 7) da coluna. Por outro lado, quando a expressão XPath refere-se a uma tabela no modelo relacional, retorna-se apenas o nome (elemento `<name>` da linha 5). Isso garante que os atributos e tabelas presentes nas expressões de saída em álgebra relacional estejam de acordo com o mapeamento estrutural realizado anteriormente.

Entretanto, antes de realizar a consulta sobre o documento de mapeamento, a função ($f(PathExpr, Var, Op_{FLWOR})$) precisa tratar uma série de questões específicas como, por exemplo, as partes que constituem a expressão XPath e as cláusulas em que a mesma está

inserida. Por esse motivo foi elaborado o algoritmo 1, apresentado a seguir, a fim de demonstrar o que a função de tradução de expressão XPath deve executar.

O algoritmo 1 é executado cada vez que a regra $\mathbf{Tr_1}$ é aplicada, ou seja, sempre que uma expressão XPath precisa ser traduzida. As informações de entrada para o algoritmo consistem na expressão XPath ($PathExpr$), a variável (Var) vinculada à expressão, se for o caso, e a cláusula (Op_{FLWOR}) em que a expressão está inserida. O documento com as informações de mapeamento, bem como a expressão em álgebra XQuery são carregados em memória.

Da mesma forma, uma matriz é mantida em memória contendo a cláusula (Op_{FLWOR}), a variável (Var), a expressão XPath ($PathExpr$) e as informações referentes ao esquema relacional (`tabela` e `coluna`). Essas duas últimas informações consistem no retorno da consulta presente na figura 3.2 e são utilizadas na construção da expressão final em álgebra relacional.

Basicamente o algoritmo considera as duas construções para expressões XPath: um operador '/' (linhas 2 a 4) seguido de uma expressão XPath relativa ($RelativePathExpr$) ou somente uma expressão XPath relativa ($RelativePathExpr$), conforme linhas 5 a 53.

Primeiramente o algoritmo trata a construção da expressão XPath. Caso a expressão inicie com um operador '/' o mesmo é concatenado à variável `expressão`, conforme códigos das linhas 2 e 3. Posteriormente, conforme mostra a linha 5, são testadas as possibilidades para todos os passos ($StepExpr$) da expressão XPath. Essa expressão passa a ser considerada como uma expressão XPath relativa ($RelativePathExpr$).

Um passo de expressão XPath pode ser um eixo ($AxisStep$) ou um filtro ($FilterExpr$). O primeiro caso é tratado pelo algoritmo nas linhas 6 a 24. Dentre as possibilidades, um eixo pode conter um operador '@', que deve ser concatenado à expressão, conforme as linhas 7 a 9. Da mesma forma, um nome de nó XML ($NodeName$) também pode fazer parte de um eixo, sendo que precisa ser concatenado à expressão, conforme as linhas 10 a 12. Um eixo também pode conter uma lista de predicados ($PredicadeList$). Neste caso, a expressão de comparação ($ComparisonExpr$) presente em cada predicado é extraída (linhas 14 e 15) e as regras de tradução ($\mathbf{Tr_4}$, $\mathbf{Tr_3}$ e $\mathbf{Tr_2}$) são aplicadas conforme seu tipo (linhas 16 a 22).

As linhas 25 a 49 tratam o caso em que um passo de expressão XPath é um filtro ($FilterExpr$). Este filtro pode ser constituído de uma expressão primária ($PrimaryExpr$) e um lista de predicados ($PredicadeList$). Para o caso de existir uma expressão primária (linha 26) a mesma pode ser avaliada como uma literal ($Literal$) ou uma referência à variável ($VarRef$). As linhas 27 e 28 demonstram que uma literal é apenas concatenada à expressão.

Algoritmo 1 Função de Tradução de Expressões XPath.

Entrada: $PathExpr$ Var Op_{FLWOR} {documento de mapeamento e álgebra XQuery ($FLWORExpr$) em memória}

Saída: `tabela`, `coluna`

```
1: expressão ← null
2: if PathExpr inicia com '/' then
3:    expressão ← concatena(expressão,'/' de PathExpr)
4: end if
5: for all StepExpr em RelativePathExpr do
6:    if StepExpr = AxisStep then
7:       if '@' está contido em AxisStep then
8:          expressão ← concatena(expressão,'@')
9:       end if
10:      if NodeName está contido em AxisStep then
11:         expressão ← concatena(expressão,NodeName)
12:      end if
13:      if PredicadeList está contido em AxisStep then
14:         for all Predicade em PredicadeList de AxisStep do
15:            comparação ← ComparisonExpr de Predicade
16:            if 'and' está contido em ComparisonExpr then
17:               Tr4(comparação)
18:            else if 'or' está contido em ComparisonExpr then
19:               Tr3(comparação)
20:            else
21:               Tr2(comparação)
22:            end if
23:         end for
24:      end if
25:   else if StepExpr = FilterExpr then
26:      if PrimaryExpr está contido em FilterExpr then
27:         if PrimaryExpr = Literal then
28:            expressão ← concatena(expressão,Literal)
29:         else if PrimaryExpr = VarRef then
30:            if VarRef não está contido em matriz then
31:               matriz ← carregaVar(FLWORExpr,VarRef)
32:            end if
33:            expressãoVar ← buscaVar(VarRef, matriz)
34:            expressão ← concatena(expressão,expressaoVar)
35:         end if
36:      end if
37:      if PredicadeList está contido em FilterExpr then
38:         for all Predicade em PredicadeList de FilterExpr do
39:            comparação ← ComparisonExpr de Predicade
40:            if 'and' está contido em ComparisonExpr then
41:               Tr4(comparação)
42:            else if 'or' está contido em ComparisonExpr then
43:               Tr3(comparação)
44:            else
```

45: $\mathbf{Tr_2}$(comparação)
46: **end if**
47: **end for**
48: **end if**
49: **end if**
50: **if** $StepExpr$ **é seguido por** $'/'$ **then**
51: expressão ← concatena(expressão,$'/'$ de $RelativePathExpr$)
52: **end if**
53: **end for**
54: tabela, coluna ← consulta(expressão)
55: **if** Op_{FLWOR} = Op_{FOR} **or** Op_{FLWOR} = Op_{LET} **then**
56: insere_matriz(matriz, Op_{FLWOR} , Var , expressão , tabela , coluna)
57: **else**
58: insere_matriz(matriz, Op_{FLWOR}, , expressão, tabela, coluna)
59: **end if**
60: **return** (tabela, coluna)

Já para o caso de referência à variável primeiramente é necessário verificar se a mesma ainda não está contida na matriz (linha 30). Neste caso, é preciso buscar na álgebra XQuery a expressão originalmente ligada à essa variável. Essa funcionalidade é realizada pela função '*carregaVar*' da linha 31 e descrita pelo algoritmo 2.

Algoritmo 2 Função que busca variável em uma expressão de álgebra XQuery.

Entrada: $FLWORExpr, Var$
Saída: matriz
1: **for all** $Op_{FOR}|Op_{LET}$ em $FLWORExpr$ **do**
2: **if** $VarRef = Var$ **then**
3: $\mathbf{Tr_1}(PathExpr, Var, Op_{FLWOR})$
4: **end if**
5: **end for**

No algoritmo 2 busca-se a referência da variável nos operadores das cláusulas `for` e `let` (linha 1). Ao localizar a variável (linha 2), aplica-se a regra $\mathbf{Tr_1}$ considerando a expressão XPath, a variável e o operador da cláusula, conforme mostrado na linha 3. Desse modo, o algoritmo 1 é executado recursivamente a fim de analisar a expressão XPath e incluí-la na matriz de informações relacionais que está carregada em memória.

Posteriormente, a execução do algoritmo 1 continua a partir da linha 33. Assim, através da função '*buscaVar*' (linha 33) realiza-se uma busca na matriz existente e concatena-se a expressão anteriormente ligada à variável com a expressão geral (linha 34). Caso o filtro contenha uma lista de predicados o tratamento (das linhas 37 a 48) é realizado da mesma forma demonstrada anteriormente nas linhas 13 a 24.

Os códigos posteriores (linhas 50 a 52) tratam o operador '/', que pode existir antes do próximo passo de expressão XPath e deve ser concatenado à expressão. A seguir, o algoritmo realiza a consulta ao documento de mapeamento (linha 54) e insere as informações na matriz de saída. As linhas 55 e 56 tratam o caso em que a cláusula original da expressão XPath é a `for` ou a `let`. Nesses casos, as informações inseridas incluem a própria cláusula (Op_{FLWOR}), a variável (Var), a expressão (`expressão`) e ainda as informações do esquema relacional (`tabela` e `coluna`). Para as demais cláusulas (linhas 58), somente a variável não é inserida da matriz, pois a mesma só é definida nas cláusulas `for` e `let`. Por fim, na linha 60 o algoritmo retorna a matriz com as informações de saída.

As próximas três regras, $\mathbf{Tr_2}$, $\mathbf{Tr_3}$ e $\mathbf{Tr_4}$, representam a tradução de expressões XPath específicas que realizam comparações. Esse tipo de expressão geralmente está presente em operadores da álgebra XQuery referentes a cláusula `where`.

A regra ($\mathbf{Tr_2}$) mostrada a seguir representa a tradução de uma expressão de comparação simples, $ComparisonExpr$, composta por duas expressões XPath ($PathExpr$) mediadas através de um operador de comparação, que pode ser de valor ($ValueComp$) ou geral ($GeneralComp$).

$$(\mathbf{Tr_2})[\![ComparisonExpr]\!] = [\![PathExpr]\!]([\![ValueComp]\!]|[\![GeneralComp]\!])[\![PathExpr]\!]$$

Essa regra define que a tradução de uma expressão de comparação simples consiste no retorno da tradução das expressões XPath ($PathExpr$) - obtida pela aplicação da regra ($\mathbf{Tr_1}$) - e do operador ($ValueComp$ ou $GeneralComp$).

A seguir, a regra ($\mathbf{Tr_3}$) descreve a tradução de expressões de comparação unidas pelo operador lógico `or`.

$$\begin{aligned}(\mathbf{Tr_3})[\![ComparisonExpr\ \text{`or'}\ ComparisonExpr]\!] = & [\![ComparisonExpr]\!] \\ & \text{`or'} \\ & [\![ComparisonExpr]\!]\end{aligned}$$

Essa regra define que a tradução dessas expressões depende da aplicação da regra ($\mathbf{Tr_2}$) para cada uma das expressões de comparação ($ComparisonExpr$) e que o operador lógico `or` é traduzido diretamente para seu equivalente na álgebra relacional.

Por sua vez, as regra ($\mathbf{Tr_4}$) representa a tradução de expressões de comparação unidas pelo operador lógico `and`.

$$(\mathbf{Tr_4})[\![ComparisonExpr \text{ `and'} ComparisonExpr]\!] = [\![ComparisonExpr]\!] \text{ `and' } [\![ComparisonExpr]\!]$$

Essa regra define que a tradução dessas expressões depende da aplicação da regra $(\mathbf{Tr_2})$ para cada uma das expressões de comparação ($ComparisonExpr$) e que o operador lógico `and` é traduzido diretamente para seu equivalente na álgebra relacional.

A regra a seguir, $(\mathbf{Tr_5})$, define a tradução de um operador de comparação de valores ($ValueComp$). Essa tradução é dependente da tradução de uma entre seis possibilidades. Cada uma dessas possibilidades possui sua regra específica, apresentada posterioremente.

$$(\mathbf{Tr_5})[\![ValueComp]\!] = [\![\text{`eq'} ]\!] | [\![ \text{`ne'}]\!] | [\![\text{`lt'} ]\!] | [\![ \text{`le'}]\!] | [\![\text{`gt'} ]\!] | [\![ \text{`ge'}]\!]$$

Já a regra $(\mathbf{Tr_6})$ descreve a tradução de um operador de comparação geral ($GeneralComp$). Da mesma forma, essa tradução depende da aplicação de regras específicas para seis possibilidades, que são apresentadas a seguir.

$$(\mathbf{Tr_6})[\![GeneralComp]\!] = [\![\text{`='} ]\!] | [\![ \text{`!='}]\!] | [\![\text{`<'} ]\!] | [\![ \text{`<='}]\!] | [\![\text{`>'} ]\!] | [\![ \text{`>='}]\!]$$

As seis regras a seguir definem a tradução dos operadores de comparação de valores (‘`eq`’, ‘`ne`’, ‘`lt`’, ‘`le`’, ‘`gt`’, ‘`ge`’) da álgebra XQuery para seus equivalentes na álgebra relacional (‘=’, ‘<>’, ‘<’, ‘<=’, ‘>’, ‘>=’).

A regra $(\mathbf{Tr_7})$ define que o operador de comparação de valor ‘`eq`’, da álgebra XQuery, é traduzido para o operador ‘=’ da álgebra relacional.

$$(\mathbf{Tr_7})[\![\text{`eq'} ]\!] = \text{`='}$$

A regra $(\mathbf{Tr_8})$ abaixo define que o operador de comparação de valor ‘`ne`’, da álgebra XQuery, é traduzido para o operador ‘<>’ da álgebra relacional.

$$(\mathbf{Tr_8})[\![\text{`ne'} ]\!] = \text{`<>'}$$

A seguir, a regra $(\mathbf{Tr_9})$ define que o operador de comparação de valor ‘`lt`’, da álgebra XQuery, é traduzido para o operador ‘<’ da álgebra relacional.

$$(\mathbf{Tr_9})[\![\text{`lt'} ]\!] = \text{`<'}$$

A regra $(\mathbf{Tr_{10}})$ define que o operador ‘<=’ da álgebra relacional é obtido a partir da tradução do operador de comparação de valor ‘`le`’, da álgebra XQuery.

$$(\mathbf{Tr_{10}})[\![\ \text{`le'}\ ]\!] = \text{`<='}$$

A regra $(\mathbf{Tr_{11}})$ a seguir define que o operador de comparação de valor ‘gt’, da álgebra XQuery é traduzido para o operador ‘>’ da álgebra relacional.

$$(\mathbf{Tr_{11}})[\![\ \text{`gt'}\ ]\!] = \text{`>'}$$

A seguir, a regra $(\mathbf{Tr_{12}})$ define que o operador ‘>=’ da álgebra relacional é obtido a partir da tradução do operador de comparação de valor ‘ge’, da álgebra XQuery.

$$(\mathbf{Tr_{12}})[\![\ \text{`ge'}\ ]\!] = \text{`>='}$$

De forma semelhante, as seis regras a seguir definem a tradução dos operadores de comparação gerais (‘=’, ‘!=’, ‘<’, ‘<=’, ‘>’, ‘>=’) da álgebra XQuery para seus respectivos operadores na álgebra relacional (‘=’, ‘<>’, ‘<’, ‘<=’, ‘>’, ‘>=’).

Nesse sentido, a regra $(\mathbf{Tr_{13}})$ define que o operador ‘=’ da álgebra XQuery é traduzido para o mesmo operador ‘=’ da álgebra relacional.

$$(\mathbf{Tr_{13}})[\![\ \text{`='}\ ]\!] = \text{`='}$$

A regra $(\mathbf{Tr_{14}})$ apresenta a tradução do operador ‘!=’ da álgebra XQuery para o operador ‘<>’ na álgebra relacional.

$$(\mathbf{Tr_{14}})[\![\ \text{`!='}\ ]\!] = \text{`<>'}$$

A regra $(\mathbf{Tr_{15}})$ define que o operador ‘<’ da álgebra XQuery é traduzido para o mesmo operador ‘<’ na álgebra relacional.

$$(\mathbf{Tr_{15}})[\![\ \text{`<'}\ ]\!] = \text{`<'}$$

A regra $(\mathbf{Tr_{16}})$ define a tradução do operador ‘<=’ da álgebra XQuery para o mesmo operador ‘<=’ na álgebra relacional.

$$(\mathbf{Tr_{16}})[\![\ \text{`<='}\ ]\!] = \text{`<='}$$

A seguir, a regra $(\mathbf{Tr_{17}})$ define que o operador ‘>’ da álgebra XQuery é traduzido para o mesmo operador ‘>’ na álgebra relacional.

$$(\mathbf{Tr_{17}})[\![\ \text{`>'}\ ]\!] = \text{`>'}$$

Por fim, a regra $(\mathbf{Tr_{18}})$ define que a tradução do operador ‘>=’ da álgebra XQuery para o mesmo operador ‘>=’ da álgebra relacional.

$$(\mathbf{Tr_{18}})[\![\ \text{'>='}\ ]\!] = \text{'>='}$$

A próxima seção apresenta as regras de tradução para os operadores referentes às cláusulas FLWOR da álgebra XQuery.

3.2.2 Regras de tradução para operadores de cláusulas FLWOR

Além das regras de tradução definidas para expressões XPath, também são necessárias regras que descrevam a tradução das expressões FLWOR. Sabe-se que as cláusulas presentes nas expressões FLWOR são representadas por diversos operadores na álgebra XQuery, conforme a gramática definida na tabela 3.3. Ao longo da seção as regras de tradução para esses operadores são apresentadas. A seguir, é descrita uma regra geral de tradução para expressões FLWOR:

$$(\mathbf{Tr_{19}})[\![FLWORExpr]\!] = [\![Op_{RETURN}]\!]$$

A regra ($\mathbf{Tr_{19}}$) é a primeira a ser executada no caso da expressão FLWOR ($FLWOR$-$Expr$) ser o ponto de partida da tradução. Essa regra define que a tradução da expressão FLWOR é dependente da tradução da cláusula `return`, seguindo a definição da gramática da álgebra XQuery.

Na seção a seguir são definidas regras de tradução para três diferentes construções da cláusula `return`.

3.2.2.1 Regras de tradução para operadores da cláusula `return`

Esta seção apresenta regras de tradução para o operador $MapToItem$, referente à cláusula `return`. Este operador pode ser seguido de uma expressão de caminho ($PathExpr$), um construtor de elemento ($Op_{ELEMENT}$) ou um operador de sequência ($Op_{SEQUENCE}$). Para cada uma dessas possibilidades são especificadas regras que indicam a tradução dos operadores da álgebra XQuery posteriores ao operador $MapToItem$ ($Op_{ORDERBY}$, Op_{WHERE}, Op_{LET} e Op_{FOR}). Esses operadores possuem regras específicas de tradução que serão abordadas adiante.

A regra ($\mathbf{Tr_{20}}$) define que o operador algébrico da cláusula `return` (Op_{RETURN}) possui doze possibilidades de tradução. Posteriormente define-se uma regra específica para cada possibilidade.

$$\begin{aligned}(\mathbf{Tr_{20}})[\![Op_{RETURN}]\!] = {} & [\![MapToItem\{PathExpr\}(Op_{ORDERBY})]\!] \\ & |[\![MapToItem\{PathExpr\}(Op_{WHERE})]\!]\end{aligned}$$

$$|[\![MapToItem\{PathExpr\}(Op_{LET})]\!]$$
$$|[\![MapToItem\{PathExpr\}(Op_{FOR})]\!]$$
$$|[\![MapToItem\{Op_{ELEMENT}\}(Op_{ORDERBY})]\!]$$
$$|[\![MapToItem\{Op_{ELEMENT}\}(Op_{WHERE})]\!]$$
$$|[\![MapToItem\{Op_{ELEMENT}\}(Op_{LET})]\!]$$
$$|[\![MapToItem\{Op_{ELEMENT}\}(Op_{FOR})]\!]$$
$$|[\![MapToItem\{Op_{SEQUENCE}\}(Op_{ORDERBY})]\!]$$
$$|[\![MapToItem\{Op_{SEQUENCE}\}(Op_{WHERE})]\!]$$
$$|[\![MapToItem\{Op_{SEQUENCE}\}(Op_{LET})]\!]$$
$$|[\![MapToItem\{Op_{SEQUENCE}\}(Op_{FOR})]\!]$$

A seguir é descrita a regra ($\mathbf{Tr_{21}}$), que especifica a tradução do operador de cláusula `return` ($MapToItem\{PathExpr\}$) que é seguido de um operador algébrico da cláusula `order by` ($Op_{ORDERBY}$). Neste caso, a tradução gera um operador de projeção (π) na álgebra relacional, seguido do resultado da tradução da expressão XPath ($PathExpr$) e do operador `order by` ($Op_{ORDERBY}$).

$$(\mathbf{Tr_{21}})[\![MapToItem\{PathExpr\}(Op_{ORDERBY})]\!] = \pi[\![PathExpr]\!][\![Op_{ORDERBY}]\!]$$

Já a regra ($\mathbf{Tr_{22}}$) apresenta a tradução para um operador $MapToItem\{PathExpr\}$ seguido de um operador da cláusula `where` (Op_{WHERE}) da álgebra XQuery. A tradução consiste em um operador de projeção da álgebra relacional (π) seguido do resultado da tradução da expressão XPath ($PathExpr$) e do operador `where` (Op_{WHERE}).

$$(\mathbf{Tr_{22}})[\![MapToItem\{PathExpr\}(Op_{WHERE})]\!] = \pi[\![PathExpr]\!][\![Op_{WHERE}]\!]$$

Permanecendo na mesma ideia, a regra ($\mathbf{Tr_{23}}$) descreve a tradução para um operador $MapToItem\{PathExpr\}$ seguido de um operador algébrico para a cláusula `let` (Op_{LET}). Sendo assim, a tradução gera uma projeção na álgebra relacional (π) seguida do retorno da aplicação das regras de tradução para a expressão XPath ($PathExpr$) e para o operador `let` (Op_{LET}).

$$(\mathbf{Tr_{23}})[\![MapToItem\{PathExpr\}(Op_{LET})]\!] = \pi[\![PathExpr]\!][\![Op_{LET}]\!]$$

A regra ($\mathbf{Tr_{24}}$) representa a tradução dos operadores algébricos para a cláusula `return` ($MapToItem\{PathExpr\}$) para o caso em que são seguidos pelo operador da cláusula `FOR` (Op_{FOR}). Essa tradução resulta no operador de projeção (π), da álgebra relacional, seguido pelo resultado da tradução da expressão XPath ($PathExpr$) e do operador `FOR` (Op_{FOR}).

$$(\mathbf{Tr_{24}})[\![MapToItem\{PathExpr\}(Op_{FOR})]\!] = \pi[\![PathExpr]\!][\![Op_{FOR}]\!]$$

As quatro regras a seguir ($(\mathbf{Tr_{25}})$, $(\mathbf{Tr_{26}})$, $(\mathbf{Tr_{27}})$ e $(\mathbf{Tr_{28}})$) tratam os casos específicos em que o operador $MapToItem$ juntamente com um construtor de elemento ($Op_{ELEMENT}$) são seguidos por operadores de cláusulas FLWOR (`order by`, `where`, `let` e `for`, respectivamente).

A regra de tradução $(\mathbf{Tr_{25}})$ resulta em uma projeção (π) na álgebra relacional, seguida pelo retorno da tradução do construtor de elemento ($Op_{ELEMENT}$) e do operador `order by` ($Op_{ORDERBY}$).

$$(\mathbf{Tr_{25}})[\![MapToItem\{Op_{ELEMENT}\}(Op_{ORDERBY})]\!] = \pi[\![Op_{ELEMENT}]\!][\![Op_{ORDERBY}]\!]$$

A regra $(\mathbf{Tr_{26}})$ representa a tradução para o caso em que o operador $MapToItem$ é seguido por um construtor de elemento ($Op_{ELEMENT}$) e um operador `where` (Op_{WHERE}). A tradução resulta em uma projeção (π) em álgebra relacional, acompanhada do retorno das traduções do construtor de elemento ($Op_{ELEMENT}$) e do operador `where` (Op_{WHERE}).

$$(\mathbf{Tr_{26}})[\![MapToItem\{Op_{ELEMENT}\}(Op_{WHERE})]\!] = \pi[\![Op_{ELEMENT}]\!][\![Op_{WHERE}]\!]$$

Já a regra $(\mathbf{Tr_{27}})$ descreve a tradução para a mesma combinação do operador $MapToItem$ com o construtor de elemento ($Op_{ELEMENT}$), seguida pelo operador `let` (Op_{LET}). Para este caso, a tradução para a álgebra relacional gera uma projeção (π) seguida do resultado da tradução do construtor de elemento ($Op_{ELEMENT}$) e do operador `let` (Op_{LET}).

$$(\mathbf{Tr_{27}})[\![MapToItem\{Op_{ELEMENT}\}(Op_{LET})]\!] = \pi[\![Op_{ELEMENT}]\!][\![Op_{LET}]\!]$$

A seguir, a regra $(\mathbf{Tr_{28}})$ apresenta a tradução para o caso em que o construtor de elemento ($Op_{ELEMENT}$), logo após o operador $MapToItem$, é seguido do operador `for` (Op_{FOR}). Neste caso, expressão em álgebra relacional é composta por uma projeção seguida do resultado da tradução do construtor de elemento ($Op_{ELEMENT}$) e do operador da cláusula `for` (Op_{FOR}).

$$(\mathbf{Tr_{28}})[\![MapToItem\{Op_{ELEMENT}\}(Op_{FOR})]\!] = \pi[\![Op_{ELEMENT}]\!][\![Op_{FOR}]\!]$$

As outras quatro regras ($(\mathbf{Tr_{29}})$, $(\mathbf{Tr_{30}})$, $(\mathbf{Tr_{31}})$ e $(\mathbf{Tr_{32}})$) definem as traduções para o caso específico em que o operador $MapToItem$ é seguido por um construtor de sequência ($Op_{SEQUENCE}$).

Nesse sentido, a regra ($\mathbf{Tr_{29}}$) traduz o caso em que a combinação de operadores ($MapToItem\{Op_{SEQUENCE}\}$) é seguida de um operador `order by` ($Op_{ORDERBY}$). A tradução gera um operador de projeção (π) seguido do resultado da tradução do construtor de sequência ($Op_{SEQUENCE}$) e do operador `order by` ($Op_{ORDERBY}$).

$$(\mathbf{Tr_{29}})[\![MapToItem\{Op_{SEQUENCE}\}(Op_{ORDERBY})]\!] = \pi[\![Op_{SEQUENCE}]\!][\![Op_{ORDERBY}]\!]$$

Já a regra ($\mathbf{Tr_{30}}$) representa a tradução do operador $MapToItem\{Op_{SEQUENCE}\}$ seguido pelo operador da cláusula `where` (Op_{WHERE}). Gera-se uma projeção (π), álgebra relacional, seguida do retorno da tradução dos dois últimos operadores ($Op_{SEQUENCE}$ e Op_{WHERE}).

$$(\mathbf{Tr_{30}})[\![MapToItem\{Op_{SEQUENCE}\}(Op_{WHERE})]\!] = \pi[\![Op_{SEQUENCE}]\!][\![Op_{WHERE}]\!]$$

A regra ($\mathbf{Tr_{31}}$) descreve a geração de uma expressão em álgebra relacional contendo uma projeção (π) seguida da tradução dos operadores de sequência ($Op_{SEQUENCE}$) e `let` (Op_{LET}).

$$(\mathbf{Tr_{31}})[\![MapToItem\{Op_{SEQUENCE}\}(Op_{LET})]\!] = \pi[\![Op_{SEQUENCE}]\!][\![Op_{LET}]\!]$$

Em seguida, a regra ($\mathbf{Tr_{32}}$) apresenta a tradução da combinação do operador $MapToItem\{Op_{SEQUENCE}\}$ com o operador da cláusula `for` (Op_{FOR}). Essa tradução consitui-se de uma projeção (π) acompanhada pelo retorno das regras de tradução para o construtor de sequência ($Op_{SEQUENCE}$) e do operador `for` (Op_{FOR}).

$$(\mathbf{Tr_{32}})[\![MapToItem\{Op_{SEQUENCE}\}(Op_{FOR})]\!] = \pi[\![Op_{SEQUENCE}]\!][\![Op_{FOR}]\!]$$

A próxima seção apresenta regras que definem a tradução dos operadores de construção relacionados ao operador da cláusula `return`.

3.2.2.2 Regras de tradução para operadores de construção

Esta seção define as regras de tradução para os seguintes operadores de construção: $Op_{ELEMENT}$, relativo ao construtor de elemento; $Op_{ATTRIBUTE}$, que é o construtor de atributo; e $Op_{SEQUENCE}$, construtor de sequências. Todos estes operadores podem ser incluídos em diferentes construções do operador da cláusula `return`.

A regra ($\mathbf{Tr_{33}}$) indica a tradução geral para um construtor de elemento ($Op_{ELEMENT}$). Cada possibilidade de tradução possui sua regra específica que é definida posteriormente.

$$(\mathbf{Tr_{33}})[\![Op_{ELEMENT}]\!] = [\![Element[QName](PathExpr)]\!] \\ |[\![Element[QName](Op_{ATTRIBUTE})]\!] \\ |[\![Element[QName](Op_{SEQUENCE})]\!] \\ |[\![Element[QName](Op_{ELEMENT})]\!]$$

As regras $(\mathbf{Tr_{34}})$, $(\mathbf{Tr_{35}})$, $(\mathbf{Tr_{36}})$ e $(\mathbf{Tr_{37}})$ definem as traduções para as quatro possibilidades de construção de elementos, respectivamente: uma expressão XPath ($PathExpr$), um construtor de atributo ($Op_{ATTRIBUTE}$), um operador de sequência ($Op_{SEQUENCE}$) e outro construtor de elemento ($Op_{ELEMENT}$). Essas possibilidades referem-se aos operadores internos, ou seja, posteriores ao operador $Element[QName]$ que indica construção de elemento. A seguir cada uma dessas possibilidades é descrita.

A regra $(\mathbf{Tr_{34}})$ apresenta a tradução do construtor de elemento $Element[QName]$ para o caso em que ele é seguido por uma expressão XPath ($PathExpr$). Pelo fato de estar contido em uma cláusula `return`, sua tradução consiste apenas no retorno da aplicação da regra para a expressão XPath ($PathExpr$).

$$(\mathbf{Tr_{34}})[\![Element[QName](PathExpr)]\!] = [\![PathExpr]\!]$$

Já a próxima regra, $(\mathbf{Tr_{35}})$, descreve a tradução do construtor de elemento ($Element[QName]$) quando este é seguido por um operador de construção de atributo ($Op_{ATTRIBUTE}$). O resultado da tradução é dependente do retorno da regra de tradução para o operador de construção de atributo ($Op_{ATTRIBUTE}$).

$$(\mathbf{Tr_{35}})[\![Element[QName](Op_{ATTRIBUTE})]\!] = [\![Op_{ATTRIBUTE}]\!]$$

A seguir, a regra $(\mathbf{Tr_{36}})$ apresenta a tradução para o caso em que um operador de sequência ($Op_{SEQUENCE}$) é posterior ao construtor de elemento ($Element[QName]$). Nesse caso, a tradução depende do retorno da regra específica para o operador de sequência ($Op_{SEQUENCE}$).

$$(\mathbf{Tr_{36}})[\![Element[QName](Op_{SEQUENCE})]\!] = [\![Op_{SEQUENCE}]\!]$$

A regra $(\mathbf{Tr_{37}})$, por sua vez, representa a tradução de um construtor de elemento ($Element[QName]$) seguido por um operador de construção de elemento ($Op_{ELEMENT}$). Para essa tradução, é necessário obter o retorno da tradução do operador de construção de elemento ($Op_{ELEMENT}$).

$$(\mathbf{Tr_{37}})[\![Element[QName](Op_{ELEMENT})]\!] = [\![Op_{ELEMENT}]\!]$$

A definição da tradução do construtor de atributos ($Op_{ATTRIBUTE}$) é dada por uma única regra, ($\mathbf{Tr_{38}}$).

$$(\mathbf{Tr_{38}})[\![Op_{ATTRIBUTE}]\!] = [\![Attribute[QName](PathExpr)]\!]$$

Um construtor de atributo ($Attribute[QName]$) indica uma expressão XPath, por esse motivo sua tradução é dependente da aplicação da regra tradução para essa expressão XPath ($PathExpr$), conforme mostra a regra ($\mathbf{Tr_{39}}$).

$$(\mathbf{Tr_{39}})[\![Attribute[QName](PathExpr)]\!] = [\![PathExpr]\!]$$

A seguir, serão descritas as regras de tradução para cada construção do operador de sequência ($Op_{SEQUENCE}$), cuja regra geral, com todas as possibilidades de tradução, é a ($\mathbf{Tr_{40}}$).

$$\begin{aligned}(\mathbf{Tr_{40}})[\![Op_{SEQUENCE}]\!] = {} & [\![Sequence(PathExpr_1, PathExpr_2)]\!] \\ & |[\![Sequence(PathExpr, Op_{ATTRIBUTE})]\!] \\ & |[\![Sequence(PathExpr, Op_{ELEMENT})]\!] \\ & |[\![Sequence(PathExpr, Op_{SEQUENCE})]\!] \\ & |[\![Sequence(Op_{ATTRIBUTE_1}, Op_{ATTRIBUTE_2})]\!] \\ & |[\![Sequence(Op_{ATTRIBUTE}, PathExpr)]\!] \\ & |[\![Sequence(Op_{ATTRIBUTE}, Op_{ELEMENT})]\!] \\ & |[\![Sequence(Op_{ATTRIBUTE}, Op_{SEQUENCE})]\!] \\ & |[\![Sequence(Op_{ELEMENT_1}, Op_{ELEMENT_2})]\!] \\ & |[\![Sequence(Op_{ELEMENT}, PathExpr)]\!] \\ & |[\![Sequence(Op_{ELEMENT}, Op_{ATTRIBUTE})]\!] \\ & |[\![Sequence(Op_{ELEMENT}, Op_{SEQUENCE})]\!] \\ & |[\![Sequence(Op_{SEQUENCE_1}, Op_{SEQUENCE_2})]\!] \\ & |[\![Sequence(Op_{SEQUENCE}, PathExpr)]\!] \\ & |[\![Sequence(Op_{SEQUENCE}, Op_{ATTRIBUTE})]\!] \\ & |[\![Sequence(Op_{SEQUENCE}, Op_{ELEMENT})]\!]\end{aligned}$$

O primeiro conjunto de regras para o operador de sequência (($\mathbf{Tr_{41}}$), ($\mathbf{Tr_{42}}$), ($\mathbf{Tr_{43}}$) e ($\mathbf{Tr_{44}}$)) define a tradução para as combinações de sequência a partir de uma expressão XPath ($PathExpr$). Neste caso, são quatro combinações de sequência possíveis: com outra expressão XPath ($PathExpr_2$); com um construtor de atributo ($Op_{ATTRIBUTE}$); com um construtor de elemento ($Op_{ELEMENT}$) e com outro operador de sequência ($Op_{SEQUENCE}$).

A regra $(\mathbf{Tr_{41}})$ descreve a tradução para o caso em que é construída uma sequência ($Sequence$) de duas expressões XPath ($PathExpr_1, PathExpr_2$). Nesse caso, a tradução depende do retorno da avaliação das duas expressões XPath ($PathExpr_1, PathExpr_2$).

$$(\mathbf{Tr_{41}})[\![Sequence(PathExpr_1, PathExpr_2)]\!] = [\![PathExpr_1]\!], [\![PathExpr_2]\!]$$

Já a regra $(\mathbf{Tr_{42}})$ apresenta a tradução de uma sequência ($Sequence$) contendo uma expressão XPath ($PathExpr$) e um operador de construção de atributo ($Op_{ATTRIBUTE}$). O resultado dessa tradução depende da aplicação das regras para a expressão XPath ($PathExpr$) e para o operador de construção de atributo ($Op_{ATTRIBUTE}$).

$$(\mathbf{Tr_{42}})[\![Sequence(PathExpr, Op_{ATTRIBUTE})]\!] = [\![PathExpr]\!], [\![Op_{ATTRIBUTE}]\!]$$

A seguir, a regra $(\mathbf{Tr_{43}})$ representa a tradução para uma sequência ($Sequence$) de expressão XPath ($PathExpr$) e operador de construção de elemento ($Op_{ELEMENT}$). A tradução é dependente do retorno das regras específicas para a expressão XPath ($PathExpr$) e para o operador de construção de elemento ($Op_{ELEMENT}$).

$$(\mathbf{Tr_{43}})[\![Sequence(PathExpr, Op_{ELEMENT})]\!] = [\![PathExpr]\!], [\![Op_{ELEMENT}]\!]$$

A regra $(\mathbf{Tr_{44}})$ descreve a tradução para o caso de uma sequência ($Sequence$) constituída por uma expressão XPath ($PathExpr$) e por um operador de construção de outra sequência ($Op_{SEQUENCE}$). Para obter a tradução aplicam-se as regras de tradução para a expressão XPath ($PathExpr$) e para o operador de construção de sequência ($Op_{SEQUENCE}$).

$$(\mathbf{Tr_{44}})[\![Sequence(PathExpr, Op_{SEQUENCE})]\!] = [\![PathExpr]\!], [\![Op_{SEQUENCE}]\!]$$

Já o segundo conjunto de regras para o operador de sequência ($(\mathbf{Tr_{45}})$, $(\mathbf{Tr_{46}})$, $(\mathbf{Tr_{47}})$ e $(\mathbf{Tr_{48}})$) define a tradução para as combinações de sequência a partir de um construtor de atributo ($Op_{ATTRIBUTE}$). Da mesma forma que o conjunto anterior, são quatro combinações de sequência possíveis: com outro construtor de atributo ($Op_{ATTRIBUTE_2}$); com uma expressão XPath ($PathExpr$); com um construtor de elemento ($Op_{ELEMENT}$) e com outro operador de sequência ($Op_{SEQUENCE}$). Mais uma vez, as regras definem que a tradução de cada sequência possível depende da aplicação das regras específicas para cada parte da sequência.

Nesse sentido, a regra $(\mathbf{Tr_{45}})$ descreve a tradução para uma sequência ($Sequence$) de dois operadores de construção de atributos ($Op_{ATTRIBUTE_1}, Op_{ATTRIBUTE_2}$). O retorno da tradução destes dois construtores é necessário para obter-se a tradução desta sequência.

$$(\mathbf{Tr_{45}})[\![Sequence(Op_{ATTRIBUTE_1}, Op_{ATTRIBUTE_2})]\!] = [\![Op_{ATTRIBUTE_1}]\!], [\![Op_{ATTRIBUTE_2}]\!]$$

Já a regra $(\mathbf{Tr_{46}})$ representa a tradução de uma sequência ($Sequence$) constituída por um construtor de atributo ($Op_{ATTRIBUTE}$) e uma expressão XPath ($PathExpr$).

$$(\mathbf{Tr_{46}})[\![Sequence(Op_{ATTRIBUTE}, PathExpr)]\!] = [\![Op_{ATTRIBUTE}]\!], [\![PathExpr]\!]$$

A seguir, a regra $(\mathbf{Tr_{47}})$ descreve a tradução para o caso em que uma sequência ($Sequence$) é composta por dois construtores, de atributo e de elemento ($Op_{ATTRIBUTE}, Op_{ELEMENT}$). A tradução, neste caso, consiste no retorno da tradução dos dois operadores de construção ($Op_{ATTRIBUTE}, Op_{ELEMENT}$).

$$(\mathbf{Tr_{47}})[\![Sequence(Op_{ATTRIBUTE}, Op_{ELEMENT})]\!] = [\![Op_{ATTRIBUTE}]\!], [\![Op_{ELEMENT}]\!]$$

A regra $(\mathbf{Tr_{48}})$ apresenta a tradução para uma sequência ($Sequence$) de construtor de atributo ($Op_{ATTRIBUTE}$) e de outro construtor de sequência ($Op_{SEQUENCE}$). O resultado da tradução dos dois construtores ($Op_{ATTRIBUTE}, Op_{SEQUENCE}$) compõe a tradução deste caso.

$$(\mathbf{Tr_{48}})[\![Sequence(Op_{ATTRIBUTE}, Op_{SEQUENCE})]\!] = [\![Op_{ATTRIBUTE}]\!], [\![Op_{ELEMENT}]\!]$$

O penúltimo conjunto de regras para o operador de sequência $(\mathbf{Tr_{49}})$, $(\mathbf{Tr_{50}})$, $(\mathbf{Tr_{51}})$ e $(\mathbf{Tr_{52}})$ define a tradução para as combinações de sequência a partir de um construtor de elemento ($Op_{ELEMENT}$). São quatro as combinações de sequência possíveis: com outro construtor de elemento ($Op_{ELEMENT_2}$); com uma expressão XPath ($PathExpr$); com um construtor de atributo ($Op_{ATTRIBUTE}$) e com outro operador de sequência ($Op_{SEQUENCE}$). Cada regra define que a tradução das sequências possíveis depende da aplicação das regras específicas para cada uma de suas partes.

A regra $(\mathbf{Tr_{49}})$ descreve a tradução para uma sequência ($Sequence$) de dois construtores de elemento ($Op_{ELEMENT_1}, Op_{ELEMENT_2}$). A tradução de cada um desses construtores constitui-se na tradução para este caso.

$$(\mathbf{Tr_{49}})[\![Sequence(Op_{ELEMENT_1}, Op_{ELEMENT_2})]\!] = [\![Op_{ELEMENT_1}]\!], [\![Op_{ELEMENT_2}]\!]$$

A tradução expressa na regra $(\mathbf{Tr_{50}})$ refere-se ao caso de uma sequência ($Sequence$) de construtor de elemento ($Op_{ELEMENT}$) e expressão XPath ($PathExpr$). Neste caso, a tradução depende da aplicação de regras específicas para os componentes da sequência ($Op_{ELEMENT}$, $PathExpr$).

$$(\mathbf{Tr_{50}})[\![Sequence(Op_{ELEMENT}, PathExpr)]\!] = [\![Op_{ELEMENT}]\!], [\![PathExpr]\!]$$

Já a regra $(\mathbf{Tr_{51}})$ apresenta a tradução de uma sequência ($Sequence$) composta por dois construtores, um de elemento ($Op_{ELEMENT}$) e outro de atributo ($Op_{ATTRIBUTE}$). O resultado da tradução dos dois construtores ($Op_{ELEMENT}, Op_{ATTRIBUTE}$) compõe a tradução para este caso.

$$(\mathbf{Tr_{51}})[\![Sequence(Op_{ELEMENT}, Op_{ATTRIBUTE})]\!] = [\![Op_{ELEMENT}]\!], [\![Op_{ATTRIBUTE}]\!]$$

A seguir, a regra $(\mathbf{Tr_{52}})$ representa a tradução para o caso de uma sequência ($Sequence$) contendo dois construtores, um de elemento ($Op_{ELEMENT}$) e outro de sequência ($Op_{SEQUENCE}$). Essa tradução é dependente do retorno da avaliação dos componentes da sequência ($Op_{ELEMENT}$, $Op_{SEQUENCE}$).

$$(\mathbf{Tr_{52}})[\![Sequence(Op_{ELEMENT}, Op_{SEQUENCE})]\!] = [\![Op_{ELEMENT}]\!], [\![Op_{SEQUENCE}]\!]$$

O quinto e último conjunto de regras para o operador de sequência ($(\mathbf{Tr_{53}})$, $(\mathbf{Tr_{54}})$, $(\mathbf{Tr_{55}})$ e $(\mathbf{Tr_{56}})$) define a tradução para as combinações de sequência a partir de outro operador de sequência ($Op_{SEQUENCE}$). Da mesma forma que os conjuntos anteriores as combinações de sequência possíveis são quatro: com outro operador de sequência ($Op_{SEQUENCE_2}$); com uma expressão XPath ($PathExpr$); com um construtor de atributo ($Op_{ATTRIBUTE}$) e com um construtor de elemento ($Op_{ELEMENT}$). As regras definem que a tradução das sequências possíveis depende da aplicação das regras específicas para cada uma de suas partes.

A seguir, a regra $(\mathbf{Tr_{53}})$ define a tradução para o caso em que uma sequência ($Sequence$) é composta por outros dois construtores de sequência ($Op_{SEQUENCE_1}, Op_{SEQUENCE_2}$). O retorno da aplicação de regras de tradução para os dois construtores de sequências ($Op_{SEQUENCE1}$, $Op_{SEQUENCE2}$) compõe a tradução para esse caso.

$$(\mathbf{Tr_{53}})[\![Sequence(Op_{SEQUENCE_1}, Op_{SEQUENCE_2})]\!] = [\![Op_{SEQUENCE_1}]\!][\![, Op_{SEQUENCE_2}]\!]$$

Já a regra $(\mathbf{Tr_{54}})$ descreve a tradução de uma sequência ($Sequence$) que contém outro construtor de sequência ($Op_{SEQUENCE}$) e uma expressão XPath ($PathExpr$). Essa tradução depende do retorno das regras específicas para os dois componentes dessa sequência ($Op_{SEQUEN\text{-}CE}, PathExpr$).

$$(\mathbf{Tr_{54}})[\![Sequence(Op_{SEQUENCE}, PathExpr)]\!] = [\![Op_{SEQUENCE}]\!], [\![PathExpr]\!]$$

A regra ($\mathbf{Tr_{55}}$) representa a tradução de uma sequência ($Sequence$) de dois construtores: de sequência ($Op_{SEQUENCE}$) e de atributo ($Op_{ATTRIBUTE}$). Para este caso, o retorno da tradução dos dois construtores ($Op_{SEQUENCE}, Op_{ATTRIBUTE}$) é necessária.

$$(\mathbf{Tr_{55}})[\![Sequence(Op_{SEQUENCE}, Op_{ATTRIBUTE})]\!] = [\![Op_{SEQUENCE}]\!], [\![Op_{ATTRIBUTE}]\!]$$

Por fim, a regra ($\mathbf{Tr_{56}}$) apresenta a tradução para o caso de uma sequência ($Sequence$) de dois construtores: outro construtor de sequência ($Op_{SEQUENCE}$) e um de elemento ($Op_{ELEMENT}$). O retorno da tradução dos dois construtores internos ($Op_{SEQUENCE}, Op_{ELEMENT}$) compõe a tradução para este caso.

$$(\mathbf{Tr_{56}})[\![Sequence(Op_{SEQUENCE}, Op_{ELEMENT})]\!] = [\![Op_{SEQUENCE}]\!], [\![Op_{ELEMENT}]\!]$$

A seção a seguir prossegue com a definição das regras de tradução referentes as cláusulas FLWOR.

3.2.2.3 Regras de tradução para operadores da cláusula `order by`

Esta seção apresenta regras referentes ao operador da álgebra XQuery que representa a cláusula `order by`. A regra geral de tradução do operador $Op_{ORDERBY}$ é a ($\mathbf{Tr_{57}}$). Essa regra indica que as possibilidades de construção do operador $Op_{ORDERBY}$ devem ser traduzidas por outras regras específicas.

$$\begin{aligned}(\mathbf{Tr_{57}})[\![Op_{ORDERBY}]\!] = {} & [\![OrderBy\{PathExpr\}(Op_{WHERE})]\!] \\ & |[\![OrderBy\{PathExpr\}(Op_{LET})]\!] \\ & |[\![OrderBy\{PathExpr\}(Op_{FOR})]\!]\end{aligned}$$

As três regras a seguir (($\mathbf{Tr_{58}}$), ($\mathbf{Tr_{59}}$) e ($\mathbf{Tr_{60}}$)) especificam que a tradução do operador $OrderBy$ gera um operador τ da álgebra relacional, dependente da tradução da expressão XPath ($PathExpr$) contida nele. Cada uma das regras também especifica a tradução para os operadores que podem ser posteriores ao operador $OrderBy$. Esses operadores, que são Op_{WHERE}, Op_{LET}, Op_{FOR}, possuem regras específicas que os traduzem e são apresentadas nas próximas seções.

A regra ($\mathbf{Tr_{58}}$) descreve a tradução do operador da cláusula `order by` ($OrderBy\{PathExpr\}$) seguido por um operador da cláusula `where` (Op_{WHERE}). Para este caso, gera-se um operador de ordenação (τ) da álgebra relacional, seguido do retorno da tradução da expressão XPath ($PathExpr$) e do operador `where` (Op_{WHERE}).

$$(\mathbf{Tr_{58}})[\![OrderBy\{PathExpr\}(Op_{WHERE})]\!] = \tau[\![PathExpr]\!][\![Op_{WHERE}]\!]$$

Já a regra $(\mathbf{Tr_{59}})$ apresenta a tradução para o operador `order by` ($OrderBy\{PathExpr\}$) que é seguido pelo operador de cláusula `let` (Op_{LET}). O operador (τ), juntamente com o retorno das traduções da expressão XPath ($PathExpr$) e do operador `let` (Op_{LET}) compõe a tradução para este caso.

$$(\mathbf{Tr_{59}})[\![OrderBy\{PathExpr\}(Op_{LET})]\!] = \tau[\![PathExpr]\!][\![Op_{LET}]\!]$$

A regra $(\mathbf{Tr_{60}})$ representa a tradução do operador `return` ($OrderBy\{PathExpr\}$) que é seguido pelo operador de cláusula `for` (Op_{FOR}). Neste caso, a tradução constitui-se de um operador de ordenação (τ) seguido pelo retorno da tradução da expressão XPath ($PathExpr$) e do operador `for` (Op_{FOR}).

$$(\mathbf{Tr_{60}})[\![OrderBy\{PathExpr\}(Op_{FOR})]\!] = \tau[\![PathExpr]\!][\![Op_{FOR}]\!]$$

A próxima seção descreve regras de tradução para operadores que representam filtros contidos em expressões FLWOR.

3.2.2.4 Regras de tradução para operadores de filtro

Nesta seção são descritas as regras de tradução para operadores algébricos que representam filtros inseridos em expressões FLWOR. Esses filtros são construídos como expressões de comparação ($ComparisonExpr$), sendo que podem estar inseridos em expressões XPath ou na cláusula `where` de expressões FLWOR. É importante notar que um filtro pode ser representado por dois operadores de álgebra XQuery: $Select$ e $Join$. O operador $Select$ representa as expressões de comparação entre elementos XML e valores literais (inteiros, caracteres, datas, ...). O operador $Join$, por sua vez, é gerado para os casos em que a expressão de comparação é entre dois elementos XML. Cada operador possui regras de tradução específicas, que são descritas adiante.

A regra geral $(\mathbf{Tr_{61}})$ define que o operador algébrico Op_{WHERE}, que representa a cláusula `where`, é traduzido para quatro construções diferentes. Cada construção possui um regra de tradução específica abordada a seguir.

$$\begin{aligned}(\mathbf{Tr_{61}})[\![Op_{WHERE}]\!] = {} & [\![Select\{ComparisonExpr\}(Op_{LET})]\!] \\ & |[\![Select\{ComparisonExpr\}(Op_{FOR})]\!] \\ & |[\![Join\{ComparisonExpr\}(Op_{LET})]\!]\end{aligned}$$

$$|[\![Join\{ComparisonExpr\}(Op_{FOR})]\!]$$

A regra ($\mathbf{Tr_{62}}$) descreve a tradução do operador de seleção ($Select\{ComparisonExpr\}$) seguido por um operador de cláusula `let` (Op_{LET}). Este caso é traduzido, em álgebra relacional, como um operador de seleção (σ) cujo predicado é obtido através do retorno da regra para a expressão de comparação ($ComparisonExpr$) e é seguido do resultado da tradução do operador `let` (Op_{LET}).

$$(\mathbf{Tr_{62}})[\![Select\{ComparisonExpr\}(Op_{LET})]\!] = \sigma[\![ComparisonExpr]\!][\![Op_{LET}]\!]$$

Já a regra ($\mathbf{Tr_{63}}$) define a tradução para um filtro ($Select\{ComparisonExpr\}$) que é seguido de um operador de cláusula `for` (Op_{FOR}). Neste caso, gera-se um operador de seleção (σ) da álgebra relacional, que tem o predicado composto pelo resultado da avaliação da expressão de comparação ($ComparisonExpr$) e é seguido pelo retorno da regra para o operador `for` (Op_{FOR}).

$$(\mathbf{Tr_{63}})[\![Select\{ComparisonExpr\}(Op_{FOR})]\!] = \sigma[\![ComparisonExpr]\!][\![Op_{FOR}]\!]$$

Já as regras ($\mathbf{Tr_{64}}$) e ($\mathbf{Tr_{65}}$) determinam que o operador $Join$ da álgebra XQuery seja traduzido para uma junção theta ($\bowtie_\theta$), ou seja, com critério de seleção. Cada uma delas é distinta quanto a definição da tradução para os dois operadores que podem ser posteriores ao operador $Join$: Op_{LET} e Op_{FOR}.

A regra seguinte, ($\mathbf{Tr_{64}}$), define que o operador $Join$ da álgebra XQuery é traduzido para uma junção ($\bowtie$), em álgebra relacional, cujo critério (θ) é obtido pela tradução da expressão de comparação ($ComparisonExpr$) e é seguido pelo retorno da tradução do operador `let` (Op_{LET}).

$$(\mathbf{Tr_{64}})[\![Join\{ComparisonExpr\}(Op_{LET})]\!] = \bowtie_\theta [\![ComparisonExpr]\!][\![Op_{LET}]\!]$$

Já a regra ($\mathbf{Tr_{65}}$) descreve a tradução para o operador $Join$, da álgebra XQuery, como sendo uma junção ($\bowtie$), em álgebra relacional, cujo critério (θ) é obtido pela tradução da expressão de comparação ($ComparisonExpr$) e é seguido pelo retorno da tradução do operador `for` (Op_{FOR}).

$$(\mathbf{Tr_{65}})[\![Join\{ComparisonExpr\}(Op_{FOR})]\!] = \bowtie_\theta [\![ComparisonExpr]\!][\![Op_{FOR}]\!]$$

A seção a seguir apresenta as regras referentes aos operadores algébricos de cláusulas `let` e `for`.

3.2.2.5 Regras de tradução para operadores das cláusulas `let` e `for`

Esta seção apresenta as regras $(\mathbf{Tr_{66}})$ e $(\mathbf{Tr_{67}})$ que definem a tradução de operadores referentes às cláusulas `let` e `for`.

A regra geral de tradução para o operador Op_{LET} é apresentada a seguir. A regra $(\mathbf{Tr_{66}})$ define que o operador Op_{LET} é traduzido para um operador algébrico $MapToItem$, que possui uma regra específica de tradução.

$$(\mathbf{Tr_{66}})[\![Op_{LET}]\!] = [\![MapConcat\{[VarName : PathExpr]\}(Op_{FOR})]\!]$$

A regra $(\mathbf{Tr_{67}})$ refere-se à tradução do operador $MapConcat$ da álgebra XQuery relativo a cláusula `let`, composto pela ligação de uma variável ($VarName$) à uma expressão XPath ($PathExpr$). Sua tradução consiste no retorno da tradução da expressão XPath ($PathExpr$) e do operador da cláusula `for` (Op_{FOR}), se for o caso.

$$(\mathbf{Tr_{67}})[\![MapConcat\{[VarName : PathExpr]\}(Op_{FOR})]\!] = [\![PathExpr]\!][\![Op_{FOR}]\!]$$

Já a regra $(\mathbf{Tr_{68}})$ define a tradução geral do operador Op_{FOR}. Essa tradução é dependente da aplicação de regra específica para o conjunto de operadores algébricos $MapConcat$ e $MapFromItem$.

$$(\mathbf{Tr_{68}})[\![Op_{FOR}]\!] = [\![MapConcat\{MapFromItem[Var : IN](PathExpr)\}]\!]$$

Por fim, a regra $(\mathbf{Tr_{69}})$ define a tradução para o operador de cláusula `for`, composto pelos operador $MapConcat$ e $MapFromItem$ da álgebra XQuery. Sua tradução é dependente da tradução da expressão de caminho $PathExpr$ que é ligada à variável Var.

$$(\mathbf{Tr_{69}})[\![MapConcat\{MapFromItem[Var : IN](PathExpr)\}]\!] = [\![PathExpr]\!]$$

A próxima seção apresenta os mecanismos utilizados a fim de garantir a equivalência entre as expressões algébricas.

3.3 Equivalência das expressões

Nesta seção são descritas as regras inversas de tradução, ou seja, de álgebra relacional para álgebra XQuery. O objetivo dessas regras é fornecer uma correspondência equacional entre as duas álgebras, ou seja, um conjunto de equações que as relacionem, conforme conceito presente em (SABRY; FELLEISEN, 1992). Dessa forma, é possível utilizar essa correspondência

equacional para elaborar provas formais. A base para essas regras é constituída pelos escopos e gramáticas apresentados na seção 3.1.

A seguir, a primeira regra ($\mathbf{Eq_1}$) define a tradução de uma expressão em álgebra relacional (exp), conforme as possibilidades presentes na gramática da tabela 3.4. Seis possibilidades são tratadas pela regra ($\mathbf{Eq_1}$), ou seja, não consideram-se as duas construções do operador de renomeação (ρ), pois o mesmo não é gerado nas regras de tradução definidas anteriormente. As possibilidades de expressões são: uma relação ($relation$), uma projeção ($\pi[attrlist](exp)$), uma seleção ($\sigma[condlist](exp)$), produto cartesiano ($exp \times exp$) , junção theta ($exp \bowtie_{\theta[condlist]} exp$) e ordenação ($\tau[attrlist]\ order(exp)$). Cada uma delas possui uma regra de tradução específica.

$$
\begin{aligned}
(\mathbf{Eq_1})[\![exp]\!]^{-1} = {} & [\![relation]\!]^{-1} \\
& |[\![\pi[attrlist](exp)]\!]^{-1} \\
& |[\![\sigma[condlist](exp)]\!]^{-1} \\
& |[\![exp \times exp]\!]^{-1} \\
& |[\![exp \bowtie_{\theta[condlist]} exp]\!]^{-1} \\
& |[\![\tau[attrlist] order(exp)]\!]^{-1}
\end{aligned}
$$

Nesse sentido, a regra ($\mathbf{Eq_2}$) define que uma relação ($relation$) na álgebra relacional é traduzida para uma função ($f(relation)$). Essa função tem o objetivo inverso ao da função que recebe uma XPath. Esta última realiza uma busca no documento de mapeamento e retorna as informações do esquema relacional (tabelas e atributos). Já a atual função recebe como entrada o nome de uma relação e o nome do documento XML original, realiza uma busca no documento de mapeamento retornando a expressão XPath correspondente à relação.

$$(\mathbf{Eq_2})[\![relation]\!]^{-1} = MapConcat\{MapFromItem[Var : IN](f(relation))\}$$

Já a regra a seguir, ($\mathbf{Eq_3}$), traduz uma projeção ($\pi[attrlist](exp)$). Essa tradução gera um operador $MapToItem$ da álgebra XQuery, que corresponde à cláusula `return`, seguido da tradução da lista de atributos ($attrlist$) e da expressão seguinte (exp).

$$(\mathbf{Eq_3})[\![\pi[attrlist](exp)]\!]^{-1} = MapToItem\{[\![attrlist]\!]^{-1}\}([\![exp]\!]^{-1})$$

A regra ($\mathbf{Eq_4}$) representa a tradução de uma seleção ($\sigma[condlist](exp)$). Neste caso, gera-se um operador $Select$ da álgebra XQuery, que representa um filtro, seguido da tradução da lista de condições ($condlist$) e da expressão seguinte (exp).

$$(\mathbf{Eq_4})[\![\sigma[condlist](exp)]\!]^{-1} = Select\{[\![condlist]\!]^{-1}\}([\![exp]\!]^{-1})$$

A regra $(\mathbf{Eq_5})$ apresenta a tradução para o caso de um produto cartesiano entre duas expressões ($exp \times exp$). Como não há um equivalente direto do operador $\times$ na álgebra XQuery e o mesmo é gerado em nível de aplicação, ele não é considerado na tradução. Assim, aplicam-se as regras pertinentes para cada expressão envolvida.

$$(\mathbf{Eq_5})[\![exp \times exp]\!]^{-1} = [\![exp]\!]^{-1}[\![exp]\!]^{-1}$$

A seguir, a regra $(\mathbf{Eq_6})$ define a tradução para o caso de uma junção theta entre duas expressões ($exp \quad \bowtie_{\theta[condlist]} \quad exp$). Neste caso, além de cada expressão ser traduzida, um operador $Join$ da álgebra XQuery é gerado e seguido pela tradução da lista de condições ($condlist$).

$$(\mathbf{Eq_6})[\![exp \quad \bowtie_{\theta[condlist]} \quad exp]\!]^{-1} = [\![exp]\!]^{-1} Join\{[\![condlist]\!]^{-1}\}[\![exp]\!]^{-1}$$

Já a regra $(\mathbf{Eq_7})$ apresenta a tradução de um operador de ordenação (τ) a partir de uma lista de atributos ($attrlist$) sobre uma expressão (exp). Nesse caso, gera-se um operador $OrderBy$ da álgebra XQuery, seguido da tradução da lista de atributos e da expressão. Na tradução, o tipo de ordenação ($order$) é ignorado pois na álgebra XQuery não há essa definição.

$$(\mathbf{Eq_7})[\![\tau[attrlist]order(exp)]\!]^{-1} = OrderBy\{[\![attrlist]\!]^{-1}\}([\![exp]\!]^{-1})$$

A seguir, a regra $(\mathbf{Eq_8})$ descreve a tradução de uma lista de atributos ($attrlist$). Conforme a gramática, essa lista de atributos pode ser apenas um atributo ($attribute$) ou um atributo seguido de outra lista de atributos ($attribute$ ‘,’ $attrlist$). Para cada um desses dois casos foram definidas regras especifícas que são apresentadas a seguir.

$$(\mathbf{Eq_8})[\![attrlist]\!]^{-1} = [\![attribute]\!]^{-1} | [\![attribute \text{ ‘,’ } attrlist]\!]^{-1}$$

A regra $(\mathbf{Eq_9})$ representa a tradução para um atributo seguido por uma lista de atributos ($attribute$ ‘,’ $attrlist$). Nesse caso, gera-se um operador de sequência da álgebra XQuery ($Sequence$), seguido pela aplicação das regras de tradução para o atributo e para a lista de atributos.

$$(\mathbf{Eq_9})[\![attribute \text{ ‘,’ } attrlist]\!]^{-1} = Sequence([\![attribute]\!]^{-1} \text{ ‘,’ } [\![attrlist]\!]^{-1})$$

A regra ($\mathbf{Eq_{10}}$) define que a tradução de um atributo ($attribute$) consiste no retorno de uma função ($f(attribute)$). Essa função é semelhante à descrita na regra ($\mathbf{Eq_2}$). A única diferença é que a consulta no documento de mapeamento é feita a partir de uma coluna e não de uma tabela do esquema relacional.

$$(\mathbf{Eq_{10}})[\![attribute]\!]^{-1} = f(attribute)$$

A regra ($\mathbf{Eq_{11}}$) apresenta as possibilidades de tradução para uma lista de condições ($condlist$). De acordo com a gramática, uma lista de condições pode conter: duas listas de condições ligadas pelo operador lógico '`OR`' ou '`AND`'; uma lista de condições antecedida pelo operador lógico '`NOT`'; uma lista de condições entre parênteses ou uma comparação simples ($compared\ \ comp\ \ compared$). Cada possibilidade é tratada por uma regra específica.

$$\begin{aligned}(\mathbf{Eq_{11}})[\![condlist]\!]^{-1} = {} & [\![condlist \text{ `OR' } condlist]\!]^{-1} \\ & [\![condlist \text{ `AND' } condlist]\!]^{-1} \\ & [\![\text{ `NOT' } condlist]\!]^{-1} \\ & [\![ \text{ `('} condlist \text{`)' }]\!]^{-1} \\ & [\![compared\ \ comp\ \ compared]\!]^{-1}\end{aligned}$$

A seguir, a regra ($\mathbf{Eq_{12}}$) trata o caso específico em que duas listas de condições são ligadas pelo operador lógico '`OR`'. Nesse caso, a tradução depende da aplicação das regras para as duas listas de condições e o operador lógico é traduzido para seu equivalente direto na álgebra XQuery.

$$(\mathbf{Eq_{12}})[\![condlist \text{ `OR' } condlist]\!]^{-1} = [\![condlist]\!]^{-1} \text{ `OR' } [\![condlist]\!]^{-1}$$

Já a regra ($\mathbf{Eq_{13}}$) define a tradução para o caso em que o operador lógico '`AND`' está entre duas listas de condições. Essa tradução consiste no retorno das regras de tradução para cada lista de condições e o operador '`AND`', que possui equivalente direto na álgebra XQuery.

$$(\mathbf{Eq_{13}})[\![condlist \text{ `AND' } condlist]\!]^{-1} = [\![condlist]\!]^{-1} \text{ `AND' } [\![condlist]\!]^{-1}$$

A seguir a regra ($\mathbf{Eq_{14}}$) define a tradução para o caso de uma lista de condições ser antecedida pelo operador lógico '`NOT`'. Nesse caso somente a lista de condições é traduzida, pois o operador lógico '`NOT`' não possui equivalente na álgebra XQuery de acordo com sua gramática.

$$(\mathbf{Eq_{14}})[\![\ \text{`NOT'}\ condlist]\!]^{-1} = [\![condlist]\!]^{-1}$$

A regra $(\mathbf{Eq_{15}})$ apresenta a tradução para uma lista de condições entre parênteses. Nesse caso, os parênteses são mantidos e a lista de condições é traduzida conforme a regra mais adequada.

$$(\mathbf{Eq_{15}})[\![\ \text{`('}condlist\text{`)'}\ ]\!]^{-1} = \text{`('}\ [\![condlist]\!]^{-1}\ \text{`)'}$$

A seguir, a regra $(\mathbf{Eq_{16}})$ traduz uma comparação simples ($compared\ \ comp\ \ compared$). A tradução depende da aplicação de regras para cada parte da comparação ($compared$) e para o operador de comparação ($comp$).

$$(\mathbf{Eq_{16}})[\![compared\ \ comp\ \ compared]\!]^{-1} = [\![compared]\!]^{-1}[\![comp]\!]^{-1}[\![compared]\!]^{-1}$$

Já a regra $(\mathbf{Eq_{17}})$ define que a parte de uma comparação pode ser a tradução de um atributo ou de um dado, conforme a álgebra relacional.

$$(\mathbf{Eq_{17}})[\![compared]\!]^{-1} = [\![attribute]\!]^{-1}|[\![data]\!]^{-1}$$

Por sua vez, a regra $(\mathbf{Eq_{18}})$ define que um operador de comparação ($comp$) pode ser traduzido para várias possibilidades, sendo que cada uma possui sua regra específica. Os operadores são: diferença (‘<>’), igualdade (‘=’), maior ou igual a (‘>=’), menor ou igual a (‘<=’), menor que (‘<’) e maior que (‘>’).

$$(\mathbf{Eq_{18}})[\![comp]\!]^{-1} = [\![\ \text{`<>'}\ ]\!]^{-1}|[\![\ \text{`='}\ ]\!]^{-1}|[\![\ \text{`>='}\ ]\!]^{-1}|[\![\ \text{`<='}\ ]\!]^{-1}|[\![\ \text{`<'}\ ]\!]^{-1}|[\![\ \text{`>'}\ ]\!]^{-1}$$

A regra $(\mathbf{Eq_{19}})$ define que um operador de comparação ‘<>’ na álgebra relacional é equivalente ao operador de comparação geral ‘!=’ na álgebra XQuery.

$$(\mathbf{Eq_{19}})[\![\ \text{`<>'}\ ]\!]^{-1} = \text{`!='}$$

Já a regra $(\mathbf{Eq_{20}})$ representa a tradução do operador de comparação ‘=’ na álgebra relacional para seu operador de comparação geral equivalente.

$$(\mathbf{Eq_{20}})[\![\ \text{`='}\ ]\!]^{-1} = \text{`='}$$

A seguir, a regra $(\mathbf{Eq_{21}})$ define que o operador ‘>=’ na álgebra relacional possui um operador equivalente direto na álgebra XQuery.

$$(\mathbf{Eq_{21}})[\![\ \text{`>='}\ ]\!]^{-1} = \text{`>='}$$

A regra $(\mathbf{Eq_{22}})$ apresenta a tradução do operador de comparação '<=' da álgebra relacional para o seu operador equivalente direto na álgebra XQuery.

$$(\mathbf{Eq_{22}})[\![\ \text{`<='}\ ]\!]^{-1} = \text{`<='}$$

Já a regra $(\mathbf{Eq_{23}})$ representa a tradução do operador '<', que é o mesmo tanto na álgebra relacional quanto na álgebra XQuery.

$$(\mathbf{Eq_{23}})[\![\ \text{`<'}\ ]\!]^{-1} = \text{`<'}$$

Da mesma forma, a regra $(\mathbf{Eq_{24}})$ define que o operador '>' possui um equivalente direto nas duas álgebras.

$$(\mathbf{Eq_{24}})[\![\ \text{`>'}\ ]\!]^{-1} = \text{`>'}$$

A regra $(\mathbf{Eq_{25}})$ define que um dado pode ser traduzido para um número ou um valor *string*.

$$(\mathbf{Eq_{25}})[\![data]\!]^{-1} = [\![number]\!]^{-1} | [\![string\_value]\!]^{-1}$$

Por sua vez, um valor numérico, na álgebra relacional, é traduzido para ele mesmo na álgebra XQuery, como mostra a regra $(\mathbf{Eq_{26}})$.

$$(\mathbf{Eq_{26}})[\![number]\!]^{-1} = number$$

Por fim, a regra $(\mathbf{Eq_{27}})$ demonstra que um valor *string*, na álgebra relacional, é equivalente a ele mesmo, na álgebra XQuery.

$$(\mathbf{Eq_{27}})[\![string\_value]\!]^{-1} = string\_value$$

3.4 Considerações finais

Esse capítulo teve como objetivo apresentar as regras de tradução e equivalência entre a álgebra XQuery e a álgebra relacional. Primeiramente foram descritos os escopos da linguagem XQuery e da álgebra XQuery considerados na definição das regras. A álgebra XQuery é usada juntamente com a álgebra relacional para realizar-se a tradução.

Nesse sentido, as gramáticas da álgebra relacional e da álgebra XQuery também foram descritas, pois as regras de tradução são definidas em nível de operadores algébricos. Para garantir a equivalência das expressões, a definição de regras inversas de tradução, de álgebra relacional para álgebra XQuery, são descritas na última seção do capítulo.

4 QMAP: TRADUÇÃO AUTOMÁTICA DE CONSULTAS XQUERY PARA SQL

Neste capítulo é descrita a proposta do mecanismo automático de tradução de consultas XQuery para SQL, chamado QMap (*Query Mapper*). Além da arquitetura do QMap, o funcionamento de seus componentes também é apresentado.

4.1 Visão geral

O objetivo do mecanismo QMap é traduzir consultas XQuery para consultas SQL equivalentes. Essa necessidade surge após o mapeamento de dados XML para um BDR, ou seja, quando as consultas sobre os dados XML originais devem ser executadas sobre os dados relacionais sem que o usuário tenha que traduzí-las manualmente. Por esse motivo, as informações relativas ao mapeamento dos conceitos (ontologia) e elementos XML para o esquema relacional, originadas pelos demais mecanismos do *framework* X2Rel, são consideradas por QMap.

Para solucionar o problema da tradução de consultas, o mecanismo proposto utiliza uma abordagem algébrica. Portanto, essa abordagem não realiza a tradução direta entre XQuery e SQL, mas utiliza regras de tradução entre a álgebra XQuery (RE; SIMEON; FERNANDEZ, 2006) e a álgebra relacional, garantindo uma solução mais genérica e passível de otimização. Essas regras de tradução foram definidas no capítulo 3 e são inseridas em um dos subcomponentes de QMap, constituindo-se como a base da tradução de consultas.

A próxima seção descreve a arquitetura do mecanismo QMap, proposto neste trabalho, com seus componentes.

4.2 Arquitetura proposta

O mecanismo QMap realiza a tradução de consultas XQuery para SQL em três etapas. Essas etapas são contempladas por três componentes que formam a arquitetura do mecanismo QMap:

- XALGen (*XQuery Algebra Generator*). Este subcomponente recebe uma expressão XQuery como entrada e, a partir de regras de inferência, gera uma expressão em álgebra XQuery.
- XARMap (*XQuery Algebra to Relational Algebra Mapper*). A partir da expressão em

álgebra XQuery este subcomponente aplica regras de tradução (definidas no Capítulo 3) gerando uma expressão em álgebra relacional.

- RASMap (*Relational Algebra to SQL Mapper*). Tem o objetivo de gerar uma consulta SQL com base em regras de equivalência entre operadores da álgebra relacional e da SQL.

Basicamente, a consulta XQuery de entrada passa pela geração da expressão algébrica (XALGen); posteriormente essa expressão é traduzida para a álgebra relacional (XARMap), com o auxílio de buscas ao documento de mapeamento e, por fim, a expressão em álgebra relacional é traduzida para uma consulta em SQL (RASMap). Essa arquitetura é ilustrada pela figura 4.1:

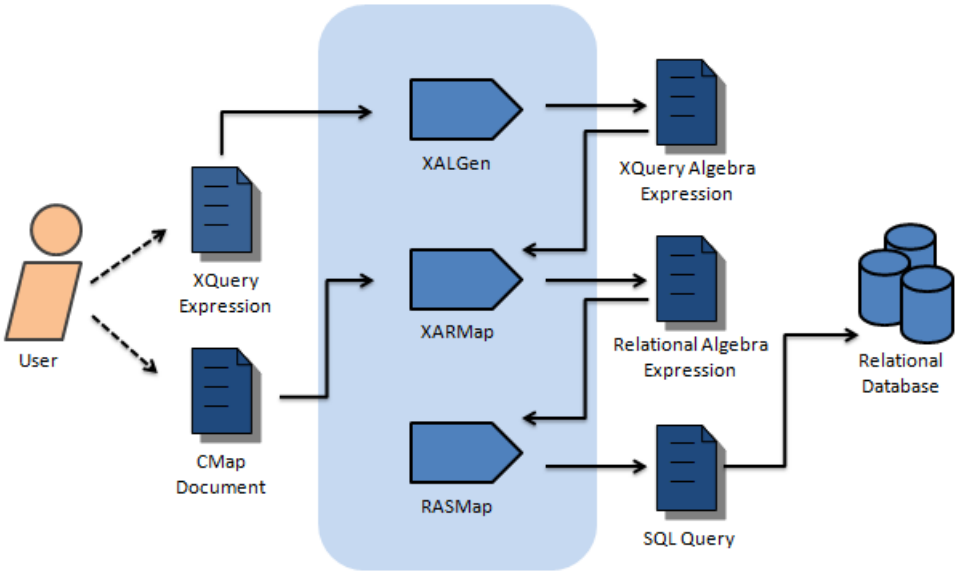

Figura 4.1 – Arquitetura QMAP

A fim de exemplificar o funcionamento do componente QMap considera-se a expressão XQuery ilustrada na figura 4.2.

```
1  for $c in /bank/client
2  return    <client account="{$c/account/account_number}">
3               {$c/client_name, $c/city}
4            </client>
```

Figura 4.2 – Expressão XQuery

Essa expressão XQuery retorna o elemento `<client>` contendo o número da conta (linha 2), o nome e a cidade do cliente (linha 3). A primeira etapa da tradução passa pelo

subcomponente XALGen, que produz a expressão em álgebra XQuery representada na figura 4.3.

```
MapToItem
  {Element[client]
    (Sequence
      (Attribute[account]((IN#c)/account/account_number),
      ((IN#c)/client_name),
      ((IN#c)/city)))}
(MapConcat{MapFromItem{ [c:IN]} (/bank/client)}(IN))
```

Figura 4.3 – Expressão em álgebra XQuery

A partir dessa expressão em álgebra XQuery o subcomponente XARMap gera a seguinte expressão em álgebra relacional: $\Pi_{ACCOUNT.account\_number,\ CLIENT.client\_name,\ CLIENT.city}(ACCOUNT \bowtie CLIENT)$.

Por fim, a consulta SQL ilustrada na figura 4.4 é gerada pelo subcomponente RASMap a partir da expressão em álgebra relacional anterior.

```
SELECT  ACCOUNT.account_number, CLIENT.client_name, CLIENT.city
FROM    CLIENT natural join ACCOUNT
```

Figura 4.4 – Consulta SQL

A seção seguinte descreve o funcionamento do componente XALGen, ou seja, a geração de expressões em álgebra XQuery a partir de consultas XQuery.

4.3 Tradução de XQuery para álgebra XQuery

O componente XALGen (*XQuery Algebra Generator*), Gerador de Álgebra XQuery, é o primeiro a ser executado no componente QMap. Ele recebe uma consulta XQuery como entrada e, a partir de regras de inferência, gera uma expressão de álgebra XQuery.

4.3.1 Regras de inferência e exemplo

Esta seção tem o objetivo de apresentar as regras de inferência utilizadas para gerar expressões algébricas com base em expressões XQuery. As regras de inferência, assim como os operadores da álgebra XQuery, foram definidas por (RE; SIMEON; FERNANDEZ, 2006) e implementadas no mecanismo XQuery Galax (GALAX, 2010). Para fins de simplificação, somente as regras de inferência para expressões FLWOR serão apresentadas neste trabalho.

É importante notar que as regras de inferência são compostas por duas partes principais: as premissas e a conclusão. As premissas estão acima da linha e a conclusão abaixo da linha. Dessa forma, a partir das premissas estabelecidas é possível obter a conclusão especificada.

A seguir, são apresentadas as regras de inferência para cada cláusula das expressões FLWOR.

$$\frac{Op_1 = \text{TypeAssert}\,[\texttt{T}]\,(Op_0)}{[\![\texttt{as T}]\!]_{Op_0} \Rightarrow Op_1} \tag{4.1}$$

$$\frac{\begin{array}{c} Expr_1 \Rightarrow Op_1 \\ [\![\texttt{as T}]\!]_{\texttt{IN}} \Rightarrow Op_2 \\ Op_3 = MapFromItem\{[\texttt{x :}Op_2]\}(Op_1) \\ Op_4 = MapConcat\{Op_3\}(Op_0) \\ Op_5 = MapIndex\,[\texttt{i}]\,(Op_4) \\ [\![Clauses|\$Var/\texttt{IN\#}Var, \$\texttt{i}/\texttt{IN\#i}]\!]_{Op_5} \Rightarrow Op_6 \end{array}}{[\![\texttt{for \$}Var\,[\texttt{as T}]\,[\texttt{as \$i}]\ \texttt{in}\ Expr_1\ \ Clauses]\!]_{Op_0} \Rightarrow Op_6} \tag{4.2}$$

$$\frac{\begin{array}{c} Expr_1 \Rightarrow Op_1 \\ [\![\texttt{as T}]\!]_{Op_1} \Rightarrow Op_2 \\ Op_3 = MapConcat\{[Var : Op_2]\}(Op_0) \\ [\![Clauses|\$Var/\texttt{IN\#}Var]\!]_{Op_3} \Rightarrow Op_4 \end{array}}{[\![\texttt{let \$}Var\,[\texttt{as T}]\ \texttt{in}\ Expr_1\ \ Clauses]\!]_{Op_0} \Rightarrow Op_4} \tag{4.3}$$

$$\frac{\begin{array}{c} Expr_1 \Rightarrow Op_1 \\ Op_2 = Select\{Op_1\}(Op_0) \\ [\![Clauses]\!]_{Op_2} \Rightarrow Op_4 \end{array}}{[\![\texttt{where}\ \ Expr_1\ \ Clauses]\!]_{Op_0} \Rightarrow Op_4} \tag{4.4}$$

$$\frac{\begin{array}{c} Expr_1 \Rightarrow Op_1 \\ Op_2 = OrderBy\{Op_1\}(Op_3) \\ [\![Clauses]\!]_{Op_2} \Rightarrow Op_4 \end{array}}{[\![\texttt{order by}\ \ Expr_1\ \ Clauses]\!]_{Op_3} \Rightarrow Op_4} \tag{4.5}$$

$$\frac{\begin{array}{c} Expr_1 \Rightarrow Op_1 \\ Op_2 = MapToItem\{Op_1\}(Op_3) \end{array}}{[\![\texttt{return}\ \ Expr_1]\!]_{Op_3} \Rightarrow Op_2} \tag{4.6}$$

O algoritmo de compilação implementado no mecanismo Galax (GALAX, 2010) usa a notação de regras de inferência. Dessa forma, a expressão $Expr_n \Rightarrow Op_n$ (presente nas regras 4.2 a 4.6) significa que a expressão $Expr$ é compilada para o plano algébrico Op. A

notação $Clauses|\$Var/\texttt{IN\#}Var$ (presente nas regras 4.2 e 4.3) representa a substituição de variável, onde toda a ocorrência da variável $\$Var$ é substituída pelo acesso ao campo de tupla correspondente ($\texttt{IN\#}Var$) nas cláusulas apresentadas.

Um aspecto delicado da compilação de cláusulas FLWOR é que a geração da propriedade fluxo de tupla requer a inversão do fluxo de dados *top-down*, da árvore sintática abstrata, para um fluxo de dados *bottom-up*, no plano de consulta algébrico. Para contemplar essa característica, é definida uma sentença auxiliar $[\![Expr]\!](Op_{n1}) \Rightarrow Op_{n2}$ (presente nas regras 4.2 a 4.6), que compila uma expressão $Expr$ em um plano algébrico Op_{n2} no contexto de um plano algébrico intermediário Op_{n1}. Esse plano intermediário é o resultado das cláusulas previamente compiladas.

Para exemplificar a aplicação das regras de inferência considera-se a consulta "Q1-B", já apresentada na seção 1.1, ilustrada na figura 4.5. Essa consulta retorna o endereço (linha 5) dos autores que moram no RS (linha 2).

De acordo com (RE; SIMEON; FERNANDEZ, 2006), a compilação aplica as regras de inferência sobre cada cláusula na expressão FLWOR, passando a expressão resultante como um parâmetro para a regra que compila a próxima cláusula. Assim, a expressão algébrica final, mostrada na figura 4.6, é construída e lida seguindo uma estratégia *bottom-up*. Cada operador tem um nome, alguns operadores independentes (ou de entrada) delimitados por (), e alguns sub-operadores dependentes delimitados por {}, cuja entrada é denotada por `IN`.

```
1  for $a in /writer/address
2  where $a/state = "RS"
3  return
4    <address>
5      {concat($a/street, ",", $a/city, ",", $a/state)}
6    </address>
```

Figura 4.5 – Consulta XQuery "Q1-B"

Primeiramente a cláusula `for` (linha 1 da figura 4.5) é compilada produzindo uma composição de um operador `MapFromItem` e um `MapConcat`, como mostra a linha 6 da figura 4.6. Esses operadores constroem uma sequência de tuplas com um campo único `a` e adicionam estas tuplas à sequência de tuplas de entrada. O operador `MapConcat` é útil quando a cláusula `for` ocorre no meio de uma expressão FLWOR, uma vez que ele recebe uma sequência de tuplas previamente avaliadas.

Depois disso, o plano intermediário é passado como um argumento para a regra de compilação da clausula `where` (linha 2 da figura 4.5). Essa regra introduz um operador de

seleção (`Select`) contendo a expressão de comparação, conforme mostrado na linha 5 da figura 4.6. Por fim, a regra para a cláusula `return` (linhas 3 a 6 da figura 4.5) introduz um mapeamento final (operador `MapToItem` na linha 1 da figura 4.6) que retorna os valores XML resultantes inseridos em um elemento `<address>` (linhas 2 a 4 da figura 4.6).

```
1  MapToItem
2    {Element[address]
3      {fn:concat((IN#a)/street, ", ", (IN#a)/city, ", ", (IN#a)/state)}
4    }
5      (Select {(IN#a)/state = "RS"}
6        (MapConcat{MapFromItem{[a:IN]}(/writer/address)}(IN)))
```

Figura 4.6 – Expressão algébrica da consulta XQuery “Q1-B”

Existem outras regras de inferência para expressões XPath e de tipo, por exemplo. Há também expressões XQuery, tais como construtores e chamadas de funções, que correspondem diretamente a um operador algébrico.

4.3.2 Representação em XML da álgebra XQuery

Esta seção descreve a representação em formato XML definida exclusivamente para a álgebra XQuery. A definição dessa estrutura é uma das contribuições deste trabalho, visto que ela facilita a utilização na implementação do mecanismo proposto.

A seguir, na figura 4.7 é apresentado um fragmento de um documento XML que representa uma expressão em álgebra XQuery.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xquery_algebra>
3    <operator name="Map">
4      <sub_operator name="element">
5        <static_param>book</static_param>
6        <sub_operator name="attribute">
7          <static_param>year</static_param>
8          <sub_operator name="fn:data">
9            <input_operator name="text">
10             <sub_operator name=""/>
```

Figura 4.7 – Fragmento de expressão em álgebra XQuery representada em XML

É possível notar que os elementos XML seguem a notação dos operadores da álgebra XQuery:

$$Op[p_1, ..., p_i]DOp_1, ..., DOp_h(Op_1, ..., Op_k)$$

Onde Op é o nome do operador e é representado pelo elemento `<operator>` (linha 3). Os parâmetros estáticos ($p_1, ..., p_i$) do operador são representados pelo elemento `<static_param>` (linhas 5 e 7). Já o elemento `<sub_operator>` (linha 4) representa os operadores dependentes ($DOp_1, ..., DOp_h$). E os operadores de entrada ($Op_1, ..., Op_k$) são representados por um elemento `input_operator`.

Essa estrutura XML é utilizada neste (XALGen) e também no próximo subcomponente (XARMap) para representar as expressões em álgebra XQuery. A DTD (*Document Type Definition*) que descreve esses documentos está disponível no apêndice A.

A seção a seguir descreve o subcomponente XARMap, que utiliza regras de tradução entre a álgebra XQuery e a álgebra relacional.

4.4 Tradução de álgebra XQuery para álgebra relacional

O núcleo central do mecanismo QMap é o componente XARMap (*XQuery Algebra to Relational Algebra Mapper*), ou seja, Mapeador de Álgebra XQuery para Álgebra Relacional. Nele são utilizadas as regras de tradução (definidas na seção 3.2) entre operadores da álgebra XQuery e operadores da álgebra relacional. O componente XARMap constitui-se da principal contribuição deste trabalho, visto que as regras dos demais módulos, XALGen e RASMap, são especificadas com base em outros trabalhos.

O componente XARMap retorna uma expressão em álgebra relacional ($AlgRel$) a partir de uma expressão em álgebra XQuery ($AlgX$) e das informações do documento de mapeamento (μ):

$$XARMap : AlgX \times \mu \rightarrow AlgRel$$

4.4.1 Exemplo de tradução de álgebra XQuery para álgebra relacional

Esta seção tem o objetivo de apresentar um exemplo de aplicação das regras de tradução entre álgebra XQuery e álgebra relacional. Para isso, considera-se a expressão em álgebra XQuery da figura 4.6. A seguir, as regras definidas na seção 3.2 serão aplicadas:

Primeiramente, a regra ($\mathbf{Tr_{19}}$) é aplicada sobre a expressão em álgebra XQuery ($FLWORExpr$), indicando a tradução do operador da cláusula `return`. Essa tradução é aplicada pela regra geral ($\mathbf{Tr_{20}}$) que, neste caso, conduz à regra específica ($\mathbf{Tr_{26}}$). Por sua vez, a regra ($\mathbf{Tr_{26}}$) gera um operador de projeção (π) na álgebra relacional. A projeção é seguida pelo re-

sultado da tradução do construtor de elemento ($Op_{ELEMENT}$) e do operador da cláusula `where` (Op_{WHERE}). A aplicação dessas três regras é mostrada a seguir:

$(\mathbf{Tr_{19}})[\![FLWORExpr]\!] = [\![Op_{RETURN}]\!]$
$(\mathbf{Tr_{20}})[\![Op_{RETURN}]\!] = [\![MapToItem\{Op_{ELEMENT}\}(Op_{WHERE})]\!]$
$(\mathbf{Tr_{26}})[\![MapToItem\{Op_{ELEMENT}\}(Op_{WHERE})]\!] = \pi \ \ [\![Op_{ELEMENT}]\!][\![Op_{WHERE}]\!]$

A próxima regra aplicada é a $(\mathbf{Tr_{33}})$ que indica a tradução de um construtor de elemento ($Element[QName]$) seguido por uma sequência ($Op_{SEQUENCE}$). Especificamente, a regra $(\mathbf{Tr_{36}})$ conduz à tradução do operador de construção de sequência. Neste caso, a regra $(\mathbf{Tr_{44}})$ é aplicada sob uma sequência contendo uma expressão XPath e outro construtor de sequência. As quatro regras são executadas conforme mostrado a seguir:

$(\mathbf{Tr_{33}})[\![Op_{ELEMENT}]\!] = [\![Element[QName](Op_{SEQUENCE})]\!]$
$(\mathbf{Tr_{36}})[\![Element[QName](Op_{SEQUENCE})]\!] = [\![Op_{SEQUENCE}]\!]$
$(\mathbf{Tr_{40}})[\![Op_{SEQUENCE}]\!] = [\![Sequence(PathExpr, Op_{SEQUENCE})]\!]$
$(\mathbf{Tr_{44}})[\![Sequence(PathExpr, Op_{SEQUENCE})]\!] = [\![PathExpr]\!], [\![Op_{SEQUENCE}]\!]$

Desse modo, a regra $(\mathbf{Tr_1})$ é aplicada a partir da expressão XPath e do operador da cláusula `return`. Essa regra indica o uso de uma função cujo funcionamento está definido no algoritmo 1. É possível notar que em $(\mathbf{Tr_1})'$ a expressão tratada é a mesma que está presente na álgebra XQuery, referenciando a variável $\$a$. Já na regra $(\mathbf{Tr_1})''$ essa variável é substituída pela expressão XPath ligada à ela. Esse processo ocorre durante a execução do algoritmo 1. A seguir, a aplicação da regra $(\mathbf{Tr_1})$ é demonstrada:

$(\mathbf{Tr_1})[\![PathExpr]\!] = f(PathExpr, Var, Op_{FLWOR})$
$(\mathbf{Tr_1})'[\![PathExpr]\!] = f(\texttt{(IN\#a)/street},, Op_{RETURN})$
$(\mathbf{Tr_1})''[\![PathExpr]\!] = f(\texttt{/writer/address/street},, Op_{RETURN})$

Após a execução da regra $(\mathbf{Tr_1})$ e, consequentemente, do algoritmo 1, tem-se a primeira versão da matriz de informações relacionais, mostrada na tabela 4.1.

Tabela 4.1 – Matriz de informações relacionais e de variáveis - 1ª versão

Cláusula FLWOR	**Variável**	**Expressão XPath**	**Tabela**	**Coluna**
Op_{RETURN}		`/writer/address/street`	`writer`	`street`

Prosseguindo com a tradução, a regra ($\mathbf{Tr_{44}}$) tem o retorno da regra ($\mathbf{Tr_1}$): coluna `street` da tabela `writer`. Dessa forma, o outro operador de sequência é traduzido através da regra geral ($\mathbf{Tr_{40}}$) e da regra específica ($\mathbf{Tr_{41}}$). Essa última trata o caso em que há uma sequência de duas expressões XPath. Assim, a regra ($\mathbf{Tr_1}$) é aplicada para as duas expressões XPath. Essas regras são aplicadas conforme a seguinte demonstração:

$(\mathbf{Tr_{44}})[\![Sequence(PathExpr, Op_{SEQUENCE})]\!] = \texttt{writer.street}, [\![Op_{SEQUENCE}]\!]$
$(\mathbf{Tr_{40}})[\![Op_{SEQUENCE}]\!] = [\![Sequence(PathExpr_1, PathExpr_2)]\!]$
$(\mathbf{Tr_{41}})[\![Sequence(PathExpr_1, PathExpr_2)]\!] = [\![PathExpr_1]\!], [\![PathExpr_2]\!]$
$(\mathbf{Tr_1})[\![PathExpr_1]\!] = f(\texttt{(IN\#a)/city},, Op_{RETURN})$
$(\mathbf{Tr_1})'[\![PathExpr_1]\!] = f(\texttt{/writer/address/city},, Op_{RETURN})$
$(\mathbf{Tr_1})[\![PathExpr_2]\!] = f(\texttt{(IN\#a)/state},, Op_{RETURN})$
$(\mathbf{Tr_1})'[\![PathExpr_2]\!] = f(\texttt{/writer/address/state},, Op_{RETURN})$

Depois das execuções do algoritmo 1 devido às regras ($\mathbf{Tr_1}$), a matriz de informações relacionais é atualizada, conforme mostrado na tabela 4.2.

Tabela 4.2 – Matriz de informações relacionais e de variáveis - 2ª versão

Cláusula FLWOR	Variável	Expressão XPath	Tabela	Coluna
Op_{RETURN}		`/writer/address/street`	`writer`	`street`
Op_{RETURN}		`/writer/address/city`	`writer`	`city`
Op_{RETURN}		`/writer/address/state`	`writer`	`state`

Com o retorno das últimas regras (colunas `street`, `city` e `state`) vão sendo contempladas recursivamente as regras anteriores até chegar na regra ($\mathbf{Tr_{20}}$). O próximo operador a ser traduzido é o da cláusula `where`, através da regra ($\mathbf{Tr_{61}}$). Essa regra indica, para este caso, a aplicação da regra ($\mathbf{Tr_{63}}$). Por sua vez, a regra ($\mathbf{Tr_{63}}$) gera um operador de seleção (σ) da álgebra relacional, seguido pelo retorno da tradução da expressão de comparação e do operador da cláusula `for`. A seguir, a aplicação dessas três regras é demonstrada:

$(\mathbf{Tr_{20}})[\![Op_{RETURN}]\!] = \pi\ \texttt{writer.street,writer.city,writer.state}[\![(Op_{WHERE})]\!]$
$(\mathbf{Tr_{61}})[\![Op_{WHERE}]\!] = [\![Select\{ComparisonExpr\}(Op_{FOR})]\!]$
$(\mathbf{Tr_{63}})[\![Select\{ComparisonExpr\}(Op_{FOR})]\!] = \sigma[\![ComparisonExpr]\!][\![Op_{FOR}]\!]$

A expressão de comparação, formada por duas expressões XPath, é traduzida pela regra ($\mathbf{Tr_2}$). Sendo assim, a primeira expressão XPath é traduzida pela regra ($\mathbf{Tr_1}$). Posteriormente,

as regras ($\mathbf{Tr_6}$) e ($\mathbf{Tr_{13}}$) são aplicadas a fim de traduzir o operador de comparação, neste caso o de igualdade (‘=’). Por fim, a outra expressão XPath é traduzida pela regra ($\mathbf{Tr_1}$) e o retorno dessas traduções vai sendo aplicado até a regra ($\mathbf{Tr_{63}}$). A aplicação dessas regras é descrita a seguir:

($\mathbf{Tr_2}$)$[\![ComparisonExpr]\!] = [\![PathExpr]\!]([\![ValueComp]\!]|[\![GeneralComp]\!])[\![PathExpr]\!]$
($\mathbf{Tr_1}$)$[\![PathExpr]\!] = f(\texttt{(IN\#a)/state},,Op_{WHERE})$
($\mathbf{Tr_1}$)$'[\![PathExpr]\!] = f(\texttt{/writer/address/state},,Op_{WHERE})$
($\mathbf{Tr_2}$)$[\![ComparisonExpr]\!] = \texttt{writer.state}([\![ValueComp]\!]|[\![GeneralComp]\!])[\![PathExpr]\!]$
($\mathbf{Tr_6}$)$[\![GeneralComp]\!] = [\![$ ‘=’ $]\!]$
($\mathbf{Tr_{13}}$)$[\![$ ‘=’ $]\!] =$ ‘=’
($\mathbf{Tr_2}$)$[\![ComparisonExpr]\!] = \texttt{writer.state} = [\![PathExpr]\!]$
($\mathbf{Tr_1}$)$[\![PathExpr]\!] = f(\text{“}RS\text{”},,Op_{WHERE})$
($\mathbf{Tr_2}$)$[\![ComparisonExpr]\!] = \texttt{writer.state} =\text{“}RS\text{”}$
($\mathbf{Tr_{63}}$)$[\![Select\{ComparisonExpr\}(Op_{FOR})]\!] = \sigma\ \texttt{writer.state} =\text{“}RS\text{”}\ [\![Op_{FOR}]\!]$

A regra ($\mathbf{Tr_{63}}$) ainda indica a tradução do operador da cláusula `for`. As regras ($\mathbf{Tr_{68}}$) e ($\mathbf{Tr_{69}}$) traduzem esse operador. Esta última indica a regra ($\mathbf{Tr_1}$) para traduzir a expressão XPath ligada à variável $\$a$. Essas três regras são aplicadas conforme mostrado a seguir:

($\mathbf{Tr_{68}}$)$[\![Op_{FOR}]\!] = [\![MapConcat\{MapFromItem[a:IN](/writer/address)\}]\!]$
($\mathbf{Tr_{69}}$)$[\![MapConcat\{MapFromItem[a:IN](/writer/address)\}]\!] = [\![PathExpr]\!]$
($\mathbf{Tr_1}$)$[\![PathExpr]\!] = f(\texttt{/writer/address}, a, Op_{FOR})$

Depois da execução dessa regra, as informações do esquema relacional presentes na matriz são atualizadas, conforme pode ser visto na tabela 4.3.

Tabela 4.3 – Matriz de informações relacionais e de variáveis - 3ª versão

Cláusula FLWOR	Variável	Expressão XPath	Tabela	Coluna
Op_{RETURN}		`/writer/address/street`	`writer`	`street`
Op_{RETURN}		`/writer/address/city`	`writer`	`city`
Op_{RETURN}		`/writer/address/state`	`writer`	`state`
Op_{FOR}	a	`/writer/address`	`writer`	

Com o retorno dessas últimas regras sendo aplicado recursivamente, as regras ($\mathbf{Tr_{69}}$), ($\mathbf{Tr_{68}}$) e ($\mathbf{Tr_{63}}$) vão sendo atualizadas. Da mesma forma a regra ($\mathbf{Tr_{61}}$) recebe o retorno

e, consequentemente atualiza-se o retorno para a regra ($\mathbf{Tr_{20}}$). Por fim, a expressão final é atualizada para a primeira regra ($\mathbf{Tr_{19}}$). A seguir, é demonstrada a aplicação dessas regras:

$(\mathbf{Tr_{69}})[\![MapConcat\{MapFromItem[a:IN](/writer/address)\}]\!] = \texttt{writer}$

$(\mathbf{Tr_{68}})[\![Op_{FOR}]\!] = \texttt{writer}$

$(\mathbf{Tr_{63}})[\![Select\{ComparisonExpr\}(Op_{FOR})]\!] = \sigma\ \texttt{writer.state} =\text{“}RS\text{”}\ (\texttt{writer})$

$(\mathbf{Tr_{61}})[\![Op_{WHERE}]\!] = \sigma\ \texttt{writer.state} =\text{“}RS\text{”}\ \texttt{writer}$

$(\mathbf{Tr_{20}})[\![Op_{RETURN}]\!] = \pi\ \texttt{writer.street,writer.city,writer.state}$
$(\sigma\ \texttt{writer.state} =\text{“}RS\text{”}\ \texttt{writer})$

$(\mathbf{Tr_{19}})[\![FLWORExpr]\!] = \pi\ \texttt{writer.street,writer.city,writer.state}$
$(\sigma\ \texttt{writer.state} =\text{“}RS\text{”}\ \texttt{writer})$

A próxima seção descreve como as expressões em álgebra relacional são representadas através do uso de XML.

4.4.2 Representação em XML da álgebra relacional

Esta seção tem o objetivo de descrever a representação XML de expressões em álgebra relacional. Da mesma forma que ocorre para a álgebra XQuery, as expressões em álgebra relacional também são descritas como documentos XML. Essa representação visa facilitar a implementação das regras de tradução e também consiste em uma das contribuições deste trabalho. Os subcomponentes XARMap e RASMap consideram essa representação ao utilizar ou construir expressões em álgebra relacional.

A seguir, na figura 4.8 um exemplo de documento XML que representa uma expressão em álgebra relacional:

Os elementos XML foram definidos de acordo com os operadores da álgebra relacional presentes em sua gramática. É possível notar a existência de um elemento `<relational_algebra>` (linha 2), no qual todos os outros elementos são incluídos. O elemento `<expression>` (linhas 3 a 20) visa representar uma expressão em álgebra relacional.

No exemplo acima, a expressão algébrica envolve a tabela `cliente`, representada no elemento `<relations>` das linhas 4 a 6. Um operador de seleção também é representado (elemento `<selection>` nas linhas 7 a 13), sendo que o predicado de seleção é mostrado pelo elemento `<condlist>` nas linhas 8 a 12. As linhas 14 a 19 representam um operador de projeção (elemento `<projection>`), juntamente com uma lista de colunas (elemento `<columnlist>` da linha 15 a 18).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<relational_algebra>
  <expression>
    <relations>
      <table>CLIENTE</table>
    </relations>
    <selection>
      <condlist>
        <column table="CLIENTE">nome_cliente</column>
        <comp_op> = </comp_op>
        <data>"Smith"</data>
      </condlist>
    </selection>
    <projection>
      <columnlist>
        <column table="CLIENT">rua_cliente</column>
        <column table="CLIENT">cidade_cliente</column>
      </columnlist>
    </projection>
  </expression>
</relational_algebra>
```

Figura 4.8 – Exemplo de expressão em álgebra relacional representada em XML

Outros operadores da álgebra relacional também podem ser representados em XML, tais como os de ordenação, agrupamento e renomeação. Estes operadores e outros detalhes podem ser verificados na DTD do apêndice B que define a estrutura desses documentos XML.

A próxima seção descreve as regras de tradução entre a álgebra relacional e a linguagem SQL.

4.5 Tradução de álgebra relacional para SQL

O módulo RASMap, Mapeador de Álgebra Relacional para SQL, tem como finalidade a geração de uma consulta SQL a partir da expressão em álgebra relacional determinada pelo módulo XARMap.

Nesta etapa são utilizadas as regras de equivalência entre os operadores da álgebra relacional e da SQL definidas em (SILBERSCHATZ; KORTH; SUNDARSHAN, 2006). Essas regras são implementadas por diversos mecanismos, sendo que neste trabalho utiliza-se a implementação presente em Lautert (2010). Esse trabalho implementa um simulador de consultas em álgebra relacional chamado SimAlg.

A tabela 4.4 mostra as regras de mapeamento implementadas pelo SimAlg (LAUTERT, 2010), onde `P` é um predicado de seleção; `Rn` é uma relação e `S` é um conjunto de atributos de `Rn`. Entretanto é preciso ressaltar que serão necessárias mais regras do que as implementadas

em (LAUTERT, 2010) (para operações de renomeação e agregação, por exemplo).

Tabela 4.4 – Regras de equivalência entre álgebra relacional e SQL

Operador AR	**Comando SQL**	**Expressão AR**	**Instrução SQL**
Selection (σ)	`WHERE`	$\sigma\ P(R)$	`select * from` R `where` P
Projection (π)	`SELECT`	$\pi\ S(R)$	`select` S `from` R
Order by (τ)	`ORDER BY`	$\tau\ S$ `asc|desc` (R)	`select * from` R `order by` S `asc|desc`
Cartesian Product ($\times$)	`CROSS JOIN`	$R_1 \times R_2$	`select * from` R_1 `CROSS JOIN` R_2
Theta Join ($\bowtie_\theta$)	`JOIN`	$R_1 \bowtie_{\theta P} R_2$	`select * from` R_1 `JOIN` R_2 `ON` (P)

A fim de exemplificar as regras de equivalência apresentadas na tabela 4.4 considera-se a expressão em álgebra relacional gerada na seção anterior:

π `street, city, state`$(\sigma$ `state` $=$"RS" `(writer)`).

A instrução SQL correspondente a essa expressão algébrica é representada na figura 4.9.

```
1 SELECT  street, city, state
2 FROM    writer
3 WHERE   state = 'RS'
```

Figura 4.9 – Instrução SQL relativa à expressão em álgebra relacional

O operador de projeção (π) é equivalente ao comando `SELECT` da linguagem SQL. As colunas listadas na projeção (`street, city, state`) são diretamente colocadas no comando `SELECT` (linha 1 da figura 4.9).

Já o operador de seleção (σ) da álgebra relacional é traduzido para o comando `WHERE` da SQL. O predicado de seleção (`state` $=$"RS") é traduzido diretamente para a expressão de comparação do comando `WHERE` (linha 3).

Por fim, a relação `writer` envolvida na expressão da álgebra relacional é traduzida diretamente para o comando `FROM` (linha 2) da linguagem SQL.

4.6 Considerações finais

Nesse capítulo um mecanismo de tradução de consultas XQuery para SQL, nomeado QMap, foi apresentado. Após uma visão geral do funcionamento de QMap, sua arquitetura é apresentada com três componentes que realizam as etapas de tradução. As funcionalidades de cada componente são descritas em seções distintas.

O primeiro componente apresentado foi o XALGen, que realiza a geração da álgebra XQuery a partir de um expressão XQuery original. Logo depois, a funcionalidade que realiza a tradução entre as álgebras XQuery e relacional descreve o componente XARMap. Por fim, o componente RASMap é apresentado através de regras de equivalência entre os operadores da álgebra relacional e da SQL.

5 EXPERIMENTOS E AVALIAÇÃO DOS RESULTADOS

Neste capítulo são descritos os experimentos realizados a fim de validar a proposta do componente QMap. Antes de efetuar a tradução de consultas, o processo de mapeamento de documentos XML para BDR é realizado, através do *framework* X2Rel. Os artefatos gerados pelos componentes OntoRel e CMap do *framework* X2Rel são demonstrados para melhor entendimento dos experimentos deste trabalho.

5.1 Arquivos XML

Com o objetivo de simular uma situação real de uso do *framework* X2Rel, buscou-se a utilização de documentos XML pertencentes a um domínio comum, mas com estruturas distintas. Esses documentos descrevem informações de livros, tais como título, autores, ano de publicação, editora, gênero, entre outras, sendo que cada um estrutura esses dados de uma forma diferente.

Como fonte de documentos XML, considerou-se a página web *XML Query Use Cases* (W3C, 2007), do W3C. Essa página descreve alguns casos de uso criados pelo W3C a fim de demonstrar aplicações importantes da linguagem de consulta sobre dados XML. Dentre as opções disponíveis utilizou-se o caso de uso 1.1, que segundo (W3C, 2007) contém exemplos de consultas que atendem requisitos das comunidades de banco de dados e de documentos. Antes de descrever as consultas do caso de uso, são apresentados os documentos XML utilizados. O principal documento XML do caso de uso em questão é `bib.xml`, apresentado no apêndice C. Este documento é utilizado nos experimentos deste trabalho, pois a maioria das consultas do caso de uso são executadas sobre ele.

Entretanto, os outros documentos do caso de uso descrevem informações complementares, tais como capítulos, preços e revisões, não pertencendo ao mesmo domínio (livros) do documento `bib.xml`. Por esse motivo, foi necessário buscar outra fonte de dados sobre livros. Desse modo, a página web *Sample XML File*, da Microsoft (MICROSOFT, 2012), surgiu como outra alternativa de fonte de dados XML. Essa página web fornece o documento `books.xml` (apresentado no Anexo A) que contém informações sobre livros e uma estrutura diferente do documento `bib.xml`.

A fim de aumentar a variabilidade de estruturas dos documentos utilizados neste trabalho, foram criados dois outros documentos a partir de `books.xml`: o `book_p1.xml` e o

book_p2.xml, apresentados respectivamente nos apêndices D e E. Metade dos livros representados no documento original books.xml foi representada em book_p1.xml alterando-se o elemento publish_date para publish, contendo somente o ano de publicação e não mais a data. A outra metade dos livros foi representada em book_p2.xml, sendo que o nome do autor possui primeiro e último nome (elementos first e last), o ano de publicação é representado pelo elemento year e a descrição (elemento description presente em book_p1.xml e books.xml) foi excluída. Com essas alterações, são três os documentos XML utilizados neste capítulo para a demonstração dos experimentos: bib.xml (apêndice C), book_p1.xml (apêndice D) e book_p2.xml (apêndice E).

5.2 Esquema lógico relacional

Considerando os documentos XML descritos anteriormente, efetua-se o processo de mapeamento dos dados XML para um BDR utilizando o *framework* X2Rel. A primeira etapa desse processo consiste na geração da ontologia através do mecanismo OntoGen (BRUM SACCOL et al., 2008). Essa ontologia descreve os conceitos presentes no domínio de conhecimento do qual fazem parte os documentos XML de entrada, que neste caso são informações referentes a livros. Apesar de sua importância, a ontologia gerada pelo mecanismo da OntoGen não será descrita neste trabalho, por motivos de simplificação e também pelo fato de a tradução de consultas envolver diretamente o conjunto de tabelas do BDR.

Desse modo, a próxima etapa do processo de mapeamento consiste na geração do esquema lógico relacional, ou seja, o conjunto de tabelas a serem criadas no BDR. Nessa etapa utiliza-se o mecanismo OntoRel (BRUM SACCOL; CAMPOS ANDRADE; PIVETA, 2011) que, a partir da ontologia gerada anteriormente, retorna um arquivo XML que descreve o esquema lógico relacional e um *script* com comandos DDL (*Data Definition Language*) de criação das tabelas, colunas e restrições. A seguir, é mostrada a representação do esquema lógico relacional gerado a partir da ontologia descritiva sobre os conceitos presentes nos documentos XML de entrada.

Book (Cod_book, Price, Publisher, Cod_author, Cod_editor, Year, Title, Description, Publish, Id, Genre) Cod_author referencia Author, Cod_editor referencia Editor

Editor (Cod_editor, Affiliation, Last, First)

Author (Cod_author, First, Last)

Bibbook (Cod_bib, Cod_book) Cod_bib referencia Bib, Cod_book referencia Book

Catalogbook (Cod_catalog, Cod_book) Cod_catalog referencia Catalog, Cod_book referencia Book

Nota-se que a tabela Book indica apenas um autor (Cod_author) e um editor (Cod_editor). Isso ocorre pois os dados XML originais contém livros que possuem apenas um autor ou um editor.

A próxima seção apresenta o documento de mapeamento, gerado a partir do mecanismo CMap.

5.3 Documento de mapeamento

A próxima etapa do processo de mapeamento dos dados XML para um BDR consiste na geração do documento de mapeamento. Essa etapa é executada pelo mecanismo CMap, que recebe como entrada os documentos XML originais, a ontologia OWL - originada pela OntoGen - e o esquema lógico relacional - originado pela OntoRel. O documento de mapeamento resultante associa os elementos dos documentos XML originais com os conceitos presentes na ontologia e também indica seu equivalente no esquema lógico relacional. Devido a essas características, o documento de mapeamento é utilizado pelo mecanismo QMap, sendo consultado durante a tradução entre a álgebra XQuery e a álgebra relacional, a fim de garantir que as colunas e tabelas utilizadas na expressão de saída em álgebra relacional sejam corretas.

A figura 5.1 ilustra dois fragmentos do documento de mapeamento gerado a partir da ontologia OWL, dos arquivos XML originais (bib.xml, book_p1.xml e book_p2.xml) e do esquema lógico relacional citado anteriormente. Nota-se pela estrutura dos fragmentos da figura 5.1 que cada conceito da ontologia é representado pelo atributo name em um elemento <concept> (linhas 100 e 159, por exemplo). A associação desse conceito com os documentos XML de origem é representada pelo elemento <source> (linhas 101 e 160, por exemplo), que identifica o arquivo XML pelo seu atributo id e a localização do elemento correspondente ao conceito através do sub-elemento xpath (linhas 102 e 161, por exemplo) que contém uma expressão de caminho. Por sua vez, as informações referentes ao modelo relacional são representadas pelo elemento relational (linhas 110 e 169). Caso o conceito tenha sido mapeado para uma coluna, essas informações consitem no tipo (elemento <type> da linha 111), no nome (elemento <name> da linha 112), na sua tabela (elemento <table> da linha 113) e no

seu domínio (elemento `<domain>` da linha 114). Já no caso em que o conceito tenha sido mapeado para uma tabela apenas o tipo (elemento `<type>` da linha 170) e o nome (elemento `<name>` da linha 171) são representados.

```
100 <concept name="title">
101   <source id="bib.xml">
102     <xpath>/bib/book/title</xpath>
103   </source>
104   <source id="book_p1.xml">
105     <xpath>/catalog/book/title</xpath>
106   </source>
107   <source id="book_p2.xml">
108     <xpath>/catalog/book/title</xpath>
109   </source>
110   <relational>
111     <type>column</type>
112     <name>title</name>
113     <table>book</table>
114     <domain>string</domain>
115   </relational>
116 </concept>
```

```
159 <concept name="book">
160   <source id="bib.xml">
161     <xpath>/bib/book</xpath>
162   </source>
163   <source id="book_p1.xml">
164     <xpath>/catalog/book</xpath>
165   </source>
166   <source id="book_p2.xml">
167     <xpath>/catalog/book</xpath>
168   </source>
169   <relational>
170     <type>table</type>
171     <name>book</name>
172   </relational>
173 </concept>
```

Figura 5.1 – Fragmentos do documento de mapeamento

O primeiro fragmento presente na figura 5.1 ilustra as associações ao conceito `title` (linha 100 a 116). Este conceito está presente nos três documentos XML de origem, sendo que em `bib.xml` a expressão de caminho para localizar o elemento é `/bib/book/title` (linha 102) e tanto em `book_p1.xml` quanto em `book_p2.xml` a expressão de caminho é a mesma, `/catalog/book/title` (linhas 105 e 108). Esse fragmento ainda indica que o conceito `title` foi mapeado para uma coluna *string* (linhas 111, 112 e 114) da tabela `book` (linha 113).

Já o segundo fragmento ilustra o conceito *book* (linha 159 a 173). Nota-se que este conceito também está presente nos três documentos XML originais. No documento `bib.xml` (linha 160) ele é representado pela XPath `bib/book` (linha 161). Já nos outros dois documentos, `book_p1.xml` (linha 163) e `book_p2.xml` (linha 166), o conceito é representado por outra expressão XPath, `catalog/book` (linhas 164 e 167).

Com o documento de mapeamento gerado é possível continuar o processo de mapeamento dos dados XML para um BDR, através do mecanismo XMap (AVELAR; BRUM SACCOL; PIVETA, 2012). Esse mecanismo visa armazenar os dados XML originais em um BDR.

5.4 Consultas XQuery originais

Assim como a estrutura e os dados de documentos XML são mapeados para um BDR, as consultas XQuery anteriores a esse processo também são mapeadas para o ambiente relacional. A fim de realizar a tradução de consultas XQuery para SQL, linguagem padrão de BDR, definiu-se o mecanismo QMap no capítulo 4.

Assim, esta seção descreve algumas consultas XQuery existentes e executadas sobre os documentos XML originais (apresentados na seção 5.1) para exemplificar o processo de tradução proposto neste trabalho. Essas consultas foram elaboradas com base no caso de uso 1.1 da página web *XML Query Use Cases* (W3C, 2007).

A próxima seção apresenta uma consulta XQuery existente sobre o documento `bib.xml`.

5.4.1 Consulta XQuery sobre o documento `bib.xml`

Esta seção descreve uma das consultas existentes sobre o documento `bib.xml`. A figura 5.2 ilustra a consulta `Q1` (W3C, 2007), que lista os livros publicados pela editora *Addison-Wesley* após 1991, mostrando o ano e o título de cada livro. É possível notar que nesta consulta estão presentes as cláusulas `for`, `where` e `return`, respectivamente nas linhas 4, 5 e 6.

```
1  xquery version "1.0";
2  <bib>
3   {
4    for $b in doc("bib.xml")/bib/book
5    where $b/publisher = "Addison-Wesley" and $b/@year > 1991
6    return
7      <book year="{ $b/@year }">
8       { $b/title }
9      </book>
10  }
11 </bib>
```

Figura 5.2 – Consulta Q1 sobre o documento `bib.xml`

Na cláusula `for` são selecionados os livros (elemento `book` na linha 4), enquanto que na cláusula `where` é aplicado um filtro no elemento `publisher` e no atributo `year` (linha 5). Por fim, a cláusula `return` monta um elemento `book` de saída contendo o atributo `year` (linha 7) e o elemento `title` (linha 8). A figura 5.3 ilustra o resultado obtido pela execução da consulta Q1 sobre o documento `bib.xml`.

```
1  <bib>
2     <book year="1994">
3        <title>TCP/IP Illustrated</title>
4     </book>
5     <book year="1992">
6        <title>Advanced Programming in the Unix environment</title>
7     </book>
8  </bib>
```

Figura 5.3 – Resultado da consulta Q1 sobre o documento `bib.xml`

A seção a seguir descreve uma consulta XQuery executada sobre o documento `book_p1`.

5.4.2 Consulta XQuery sobre o documento `book_p1`

Nesta seção é descrita uma consulta XQuery efetuada sobre o documento `book_p1`. A figura 5.4 demonstra a consulta `Q2`, elaborada sobre o documento `book_p1.xml`. Essa consulta possui todas as cláusulas FLWOR: na cláusula `for` são selecionados os livros cujo autor é *Corets, Eva* (linha 4); a `let` define uma variável que armazena a data de publicação (linha 5); na cláusula `where` aplica-se o filtro para o ano (linha 6); na `order by` classifica-se os resultados pelo título dos livros (linha 7) e, por fim, a cláusula `return` (linha 8) define o elemento `book` contendo o atributo `year` (linha 9) e o sub-elemento `title` (linha 10).

```
1  xquery version "1.0";
2  <results>
3  {
4   for $b in doc("book_p1.xml")/catalog/book[author = "Corets, Eva"]
5   let $a := $b/publish
6   where $a > 2000
7   order by $b/title descending
8   return
9     <book year="{ $a }">
10     { $b/title }
11    </book>
12 }
13 </results>
```

Figura 5.4 – Consulta Q2 sobre o documento `book_p1.xml`

Já na figura 5.5 é possível notar o retorno da consulta Q2 ao ser executada sobre o documento `book_p1.xml`. O resultado consiste em uma lista dos livros (com ano e título) publicados por *Eva Corets* depois de 2000 em ordem decrescente pelo título.

```
1  <results>
2     <book year="2001">
3        <title>The Sundered Grail</title>
4     </book>
5     <book year="2001">
6        <title>Oberon's Legacy</title>
7     </book>
8  </results>
```

Figura 5.5 – Resultado da consulta Q2 sobre o documento `book_p1.xml`

A seção seguinte descreve uma consulta efetuada sobre o documento `book_p2.xml`.

5.4.3 Consulta XQuery sobre o documento `book_p2.xml`

Nesta seção é descrita uma das consultas XQuery elaboradas e executadas sobre o documento `book_p2.xml`. Na figura 5.6 é ilustrada a consulta `Q3` sobre os documentos `book_p1.xml` e `book_p2.xml`. Na cláusula `for` são definidas duas variáveis (linhas 4 e 5), uma para cada elemento `book` de cada fonte, já na cláusula `where` é feita uma junção, atavés do elemento `genre` (linha 6), entre as duas fontes. Por fim, a cláusula `return` (linha 7) lista o preço e o título do livro em cada uma das fontes (linhas 9 e 10).

```
1  xquery version "1.0";
2  <results>
3  {
4     for $b1 in doc("book_p1.xml")/catalog/book,
5        $b2 in doc("book_p2.xml")/catalog/book
6     where $b2/genre = $b1/genre
7     return
8        <book-compared>
9           <title_p1 price="{$b1/price}">{$b1/title/text()}</title_p1>
10          <title_p2 price="{$b2/price}">{$b2/title/text()}</title_p2>
11       </book-compared>
12 }
13 </results>
```

Figura 5.6 – Consulta Q3 sobre o documento `book_p2.xml`

A figura 5.7 apresenta o resultado da execução da consulta Q3 sobre os dois documentos. Ela retorna uma lista contendo o título e o preço dos livros de mesmo gênero. É importante notar que o único gênero presente em ambos os documentos é o *Computer*. Pelo fato de que no documento `book_p1.xml` há apenas um livro (*XML Developer's Guide*) desse gênero, o resultado da consulta traz a comparação desse livro com os três livros de mesmo gênero presentes no documento `book_p2.xml`.

```
1  <results>
2     <book-compared>
3        <title_p1 price="44.95">XML Developer's Guide</title_p1>
4        <title_p2 price="36.95">Microsoft .NET: The Programming Bible</title_p2>
5     </book-compared>
6     <book-compared>
7        <title_p1 price="44.95">XML Developer's Guide</title_p1>
8        <title_p2 price="36.95">MSXML3: A Comprehensive Guide</title_p2>
9     </book-compared>
10    <book-compared>
11       <title_p1 price="44.95">XML Developer's Guide</title_p1>
12       <title_p2 price="49.95">Visual Studio 7: A Comprehensive Guide</title_p2>
13    </book-compared>
14 </results>
```

Figura 5.7 – Resultado da consulta Q3 sobre o documento `book_p2.xml`

A partir das consultas apresentadas e do documento de mapeamento gerado (apresentado na seção 5.3) é possível iniciar o processo de tradução utilizando o mecanismo QMap definido. A próxima seção descreve a realização da primeira parte da tradução: a geração da álgebra XQuery.

5.5 Expressões em álgebra XQuery

Esta seção tem o objetivo de demonstrar os resultados da primeira etapa da tradução, ou seja, a execução dos experimentos no componente XALGen do mecanismo QMap. Conforme descrito na seção 4.3, o componente XALGen utiliza as regras de compilação definidas em (RE; SIMEON; FERNANDEZ, 2006). Essas regras foram implementadas pelo mecanismo *open source* Galax (GALAX, 2010). Em sua arquitetura de funcionamento existe um passo de compilação que produz uma expressão em álgebra XQuery. Atualmente, a expressão em álgebra XQuery do componente XALGen é gerada através do mecanismo Galax. Isso se deve ao fato de não existir um *parser* específico para essa álgebra e o desenvolvimento de um novo *parser* não estar incluído nos objetivos deste trabalho.

A partir disso, as álgebras XQuery geradas pelo mecanismo Galax são representadas em documentos XML, a fim de facilitar sua utilização na implementação. A seguir são descritas as expressões em álgebra XQuery geradas para as consultas. Para cada documento de origem são descritos alguns fragmentos da expressão em álgebra XQuery. Isso é necessário para fins de simplicidade e também para priorizar operadores importantes da álgebra XQuery sem muitas repetições.

5.5.1 Álgebra XQuery da consulta Q1 sobre documento `bib.xml`

Nesta seção são apresentados fragmentos do documento XML que representa uma expressão em álgebra XQuery. Essa expressão foi gerada a partir da consulta Q1 existente sobre o documento `bib.xml`.

A seguir, na figura 5.8, encontra-se o elemento `<operator name= "Map">` (linha 3), que representa o operador `Map`. Este operador refere-se à cláusula `return`, que neste caso possui um construtor de elemento e outro de atributo, representados pelos elementos `<sub_operator name="element">` (linha 4) e `<sub_operator name= "attribute">` (linha 6) respectivamente. Cada um deles possui um parâmetro estático (elementos `<static_param>` das linhas 5 e 7) que indica o nome do elemento e do atributo, neste caso *book* e *year*, respectivamente.

```
3   <operator name="Map">
4    <sub_operator name="element">
5     <static_param>book</static_param>
6     <sub_operator name="attribute">
7      <static_param>year</static_param>
...    ...
14          <input_operator name="Map">
15           <sub_operator name="TreeJoin">
16            <static_param>@year</static_param>
17            <input_operator name="#glx:dot_9"/>
18           </sub_operator>
19           <input_operator name="MapConcat">
20            <sub_operator>
21             <static_param>glx:dot_9 : #glx:b_6</static_param>
...    ...
31      <input_operator name="Map">
32       <sub_operator name="TreeJoin">
33        <static_param>/title</static_param>
34        <input_operator name="#glx:dot_10"/>
35       </sub_operator>
36       <input_operator name="MapConcat">
37        <sub_operator>
38         <static_param>glx:dot_10 : #glx:b_6</static_param>
```

Figura 5.8 – Fragmento da consulta Q1 referente à cláusula `return`

Os fragmentos das linhas 14 a 21 e 31 a 38, ainda na figura 5.8, estão inseridos no elemento referente ao operador `Map` da cláusula `return` da XQuery. A combinação de elementos `<input_operator name="Map">` (linhas 14 e 31) e <sub_operator name="TreeJoin"> (linhas 15 e 32) indica um passo de uma expressão XPath. Esses dois elementos podem ser seguidos de `<input_operator name="MapConcat">` (linhas 19 e 36) que indica a variável que está sendo referenciada pela expressão XPath. A referência de variáveis funciona

como um controle de versões, pois cada vez que determinada variável é usada em um novo passo de expressão XPath, guarda-se uma nova versão da mesma. Isso ocorre nas linhas 21 e 38 (Figura 5.8), onde um elemento `<static_param>` indica uma nova versão de variável (`glx:dot_9` e `glx:dot_10`) e em seguida, indica qual a variável anterior utilizada nessa nova versão, nesse caso `#glx:b_6`.

A figura 5.9 ilustra o fragmento que contém os elementos XML pertencentes ao operador `Select`, que representa a cláusula `where` da XQuery. A linha 45 contém o elemento `<input_operator name="Select">`, que indica o início do operador.

```
45  <input_operator name="Select">
46    <sub_operator op="where">
47      <input_operator name="some $fs:v_2 in fn:data">
48        <input_operator name="fs:distinct-docorder">
49          <input_operator name="Map">
50            <sub_operator name="TreeJoin">
51              <static_param>/publisher</static_param>
52              <input_operator name="#glx:dot_7"/>
53            </sub_operator>
...     ...
60        <sub_operator name="satisfies some $fs:v_3 in 'Addison-Wesley'">
61          <sub_operator name="satisfies op:string-equal">
62            <input_operator name="$fs:v_2 cast as xs:string"/>
63            <input_operator name="$fs:v_3"/>
64          </sub_operator>
...     ...
67      <input_operator name="and">
68        <input_operator name="some $fs:v_4 in fn:data">
69          <input_operator name="fs:distinct-docorder">
70            <input_operator name="Map">
71              <sub_operator name="TreeJoin">
72                <static_param>@year</static_param>
73                <input_operator name="#glx:dot_8"/>
74              </sub_operator>
...     ...
81          <sub_operator name="satisfies some $fs:v_5 in 1991">
82            <sub_operator name="satisfies op:gt">
83              <input_operator name="fs:untyped-to-any">
84                <input_operator name="$fs:v_4"/>
85                <input_operator name="$fs:v_5"/>
```

Figura 5.9 – Fragmento da consulta Q1 referente à cláusula `where`

Após esse elemento, são representadas as expressões de comparação, contidas na cláusula `where`, através de passos de expressões XPath (com referência a variáveis) e valores fixos (*strings*, inteiros, ...). A primeira parte da expressão de comparação consiste num passo de expressão XPath (linhas 48 a 53) e é identificada por `$fs:v_2` (linha 47). Já a segunda parte é um valor fixo (*string* '*Addison-Wesley*') idendificada por `$fs:v_3` (linha 60). O fragmento das linhas 61 a 64 representa a indicação de que o conteúdo referenciado por `$fs:v_2` (linha

62) deve ser comparado com o de `$fs:v_3` (linha 63), através de uma operação de igualdade (`op:string-equal` da linha 61).

A linha 67 contém o elemento `<input_operator name="and">` que é um exemplo de operador lógico utilizado para vincular duas ou mais expressões de comparação. Nesse caso, ele vincula as expressões representadas nas linhas 46 a 64 e 68 a 85. Os fragmentos das linhas 82, 84 e 85 têm o mesmo objetivo das linhas 61, 62 e 63, respectivamente, alterando as referências para `$fs:v_4` (linha 84) e `$fs:v_5` (linha 85) e o operador de comparação para '>' (linha 82).

Por fim, a figura 5.10 ilustra fragmentos que representam partes do operador `Map` referente a cláusula `for` da XQuery. As linhas 93 a 96 indicam o início do operador, definindo que a expressão XPath adiante é representada por uma variável, neste caso `glx:b_6` (linha 95).

```
93   <input_operator name="Map" op="for">
94      <sub_operator>
95         <static_param>glx:b_6:ID</static_param>
96      </sub_operator>
...     ...
99            <sub_operator name="TreeJoin">
100              <static_param>/book</static_param>
101              <input_operator name="#glx:dot_4"/>
102           </sub_operator>
...     ...
109                  <sub_operator name="TreeJoin">
110                     <static_param>/bib</static_param>
111                     <input_operator name="#glx:dot_2"/>
112                  </sub_operator>
...     ...
117                     <input_operator name="Parse">
118                        <input_operator name="bib.xml"/>
119                     </input_operator>
```

Figura 5.10 – Fragmento da consulta Q1 referente à cláusula `for`

As demais linhas representam os passos contidos na expressão XPath. O mais inferior, nas linhas 117 a 119, indica o documento utilizado (neste caso `bib.xml`, linha 118). Nas linhas 109 a 112, o segundo passo da expressão XPath, ou seja, a seleção do elemento `bib` (linha 110) é representado. Já as linhas 99 a 102 indicam outra etapa da expressão XPath, que é a seleção do elemento `book` (100). Os três últimos fragmentos (linhas 99 a 119) na prática representam a seguinte expressão XPath: `doc("bib.xml")/bib/book`.

A próxima seção descreve o documento referente à álgebra XQuery de uma das consultas existentes sobre o documento `book_p1.xml`.

5.5.2 Álgebra XQuery da consulta Q2 sobre documento `book_p1.xml`

Nesta seção serão descritos fragmentos da expressão em álgebra XQuery da consulta Q2 existente sobre documento `book_p1.xml`. Como mostrado anteriormente, essa consulta possui todas as cláusulas `FLWOR`. A seguir, são destacados fragmentos que contém os principais elementos e também elementos que não estão presentes nos documentos anteriores.

O primeiro fragmento destacado pela figura 5.11 representa o operador `Map` (linha 3), referente a cláusula `return` da XQuery. Nesse caso, ele possui dois construtores, um de elemento (linha 4) e outro de atributo (linha 6). Cada um desses construtores indica o nome do elemento (`book`) ou atributo (`year`) através de elementos `<static_param>`, presentes nas linhas 5 e 7, respectivamente. Assim como nos documentos anteriores, os elementos seguintes representam os passos das expressões XPath necessárias para indicar quais são as informações resultantes.

```
3   <operator name="Map">
4     <sub_operator name="element">
5       <static_param>book</static_param>
6       <sub_operator name="attribute">
7         <static_param>year</static_param>
```

Figura 5.11 – Fragmento da consulta Q2 referente à cláusula `return`

A figura 5.12 contém o fragmento que representa o operador `OrderBy` (linha 32). Basicamente ele pode apresentar elementos internos que representam passos de expressão XPath e referências a variáveis. Na linha 37 o elemento `title` é selecionado a partir da variável `b_8` (linha 41).

```
32  <input_operator name="OrderBy">
...   ...
35        <input_operator name="Map">
36          <sub_operator name="TreeJoin">
37            <static_param>/title</static_param>
...   ...
41            <static_param>glx:dot_11 : #glx:b_8</static_param>
```

Figura 5.12 – Fragmento da consulta Q2 referente à cláusula `order by`

O fragmento representado pela figura 5.13 constitui-se de elementos XML que representam o operador `Select`, referente a cláusula `where` da XQuery. O início do operador `Select` é indicado na linha 47. Já as linhas 52, 53 e 55 indicam que o conteúdo referenciado por `$fs:v_6` deve ser comparado com o de `$fs:v_7` através de um operador de comparação (‘>’).

```
47  <sub_operator name="Select">
48    <sub_operator op="where">
49      <input_operator name="some $fs:v_6 in fn:data">
50        <input_operator name="#glx:a_10"/>
51        <sub_operator name="satisfies some $fs:v_7 in 2000">
52          <sub_operator name="satisfies op:double-gt"> --
53            <input_operator name="$fs:v_6 cast as xs:double"/>
54            <input_operator name="fs:promote-to-numeric">
55              <input_operator name="$fs:v_7"/>
```

Figura 5.13 – Fragmento da consulta Q2 referente à cláusula `where`

A figura 5.14 contém os elementos que representam a cláusula `let` da XQuery. Na linha 62, o início do operador é indicado. Na linha 63, através do elemento `<static_param>`, é indicada a variável que armazena a expressão XPath, ou seja, `a_10`. A expressão XPath é representada por diferentes elementos XML que descrevem os passos da expressão. Nesse caso, o elemento `publish` é selecionado (linha 67), como um passo da expressão XPath, a partir da variável `b_8` (linha 72).

```
62  <input_operator op="let">
63    <static_param>glx:a_10</static_param>
64    <input_operator name="fs:distinct-docorder">
65      <input_operator name="Map">
66        <sub_operator name="TreeJoin">
67          <static_param>/publish</static_param>
68          <input_operator name="#glx:dot_9"/>
69        </sub_operator>
70        <input_operator name="MapConcat">
71          <sub_operator>
72            <static_param>glx:dot_9 : #glx:b_8</static_param>
```

Figura 5.14 – Fragmento da consulta Q2 referente à cláusula `let`

Já na figura 5.15, a linha 80 indica o início do operador `Map` referente à cláusula `for`. Nessa consulta a cláusula `for` vincula uma variável à uma expressão XPath com filtro. As próximas linhas da figura 5.15 referem-se a esse filtro. A linha 90 contém o elemento que representa o início do operador `Select`, assim como na cláusula `where`, porém nesse caso ele está contido no operador da cláusula `for`.

As linhas 98 a 101 contém os elementos que representam a expressão de comparação incluída no operador `Select`. Esses fragmentos indicam que o conteúdo referenciado por `$fs:v_2` (linha 99) deve ser comparado com o de `$fs:v_3` (linha 100) através de um operador de comparação de igualdade (‘=’ na linha 98).

O restante dos elementos referem-se aos demais passos contidos na expressão XPath da cláusula `for`. Os fragmentos desses elementos são semelhantes aos presentes na consulta Q1,

por esse motivo não serão demonstrados aqui. Esses elementos, juntamente com a representação do filtro representam a expressão XPath: `doc("bib.xml")/catalog/book[author = "Corets, Eva"]`.

```
80   <input_operator name="Map" op="for">
...  ...
90                  <sub_operator name="Select">
91                    <sub_operator op="where">
...  ...
98                            <sub_operator name="satisfies op:string-equal" op="where">
99                              <input_operator name="$fs:v_2 cast as xs:string"/>
100                             <input_operator name="$fs:v_3"/>
101                           </sub_operator>
```

Figura 5.15 – Fragmento da consulta Q2 referente à cláusula `for`

Na próxima seção será descrita a álgebra referente à uma das consultas sobre o documento `book_p2.xml`.

5.5.3 Álgebra XQuery da consulta Q3 sobre documento `book_p2.xml`

Esta seção descreve fragmentos da álgebra XQuery da consulta Q3 existente sobre o documento `book_p2.xml`. Essa consulta possui uma cláusula `where` que gera um operador de junção (`Join`) na álgebra XQuery. Além disso, também seleciona elementos dos documentos `book_p2.xml` e `book_p1.xml`.

Na figura 5.16 a seguir, o operador `Map` (linha 3) referente à cláusula `return` é representado. Inseridos nesse operador estão os seguintes construtores: do elemento `book-compared` (linha 5); do sub-elemento `title_p1` (linha 7) e seu atributo `price` (linha 9); do sub-elemento `title_p2` (linha 66) e seu atributo `price` (linha 68).

```
3    <operator name="Map">
4      <sub_operator name="element">
5        <static_param>book-compared</static_param>
6        <sub_operator name="element">
7          <static_param>title_p1</static_param>
8          <sub_operator name="attribute">
9            <static_param>price</static_param>
...        ...
65         <input_operator name="element">
66           <static_param>title_p2</static_param>
67           <sub_operator name="attribute">
68             <static_param>price</static_param>
```

Figura 5.16 – Fragmento da consulta Q3 referente à cláusula `return`

Já na figura 5.17 a linha 124 contém o início do operador `Join`, gerado a partir da cláu-

sula `where`. A expressão de comparação presente na cláusula `where` envolve dois elementos XML. Por esse motivo, o operador `Join` é gerado ao invés do típico operador `Select`.

```
124   <input_operator name="Join">
125     <sub_operator op="where">
126       <input_operator name="some $fs:v_3 in fn:data">
127         <input_operator name="fs:distinct-docorder">
128           <input_operator name="Map">
129             <sub_operator name="TreeJoin">
130               <static_param>/genre</static_param>
131               <input_operator name="#glx:dot_13"/>
132             </sub_operator>
133             <input_operator name="ID ++">
134               <static_param>glx:dot_13 : #glx:b2_12</static_param>
135             </input_operator>
136           </input_operator>
137         </input_operator>
138         <sub_operator name="satisfies some $fs:v_4 in fn:data">
139           <input_operator name="fs:distinct-docorder">
140             <sub_operator name="TreeJoin">
141               <static_param>/genre</static_param>
142               <input_operator name="#glx:dot_14"/>
143             </sub_operator>
144             <input_operator name="ID ++">
145               <static_param>glx:dot_14 : #glx:b1_6</static_param>
146             </input_operator>
147           </input_operator>
148           <sub_operator name="satisfies op:string-equal" op="join">
149             <input_operator name="$fs:v_3 cast as xs:string"/>
150             <input_operator name="$fs:v_4 cast as xs:string"/>
151           </sub_operator>
```

Figura 5.17 – Fragmento da consulta Q3 referente à cláusula `where`

As linhas 126 a 137 representam a primeira parte da expressão de comparação do operador `Join`. A linha 126 identifica essa parte como `v_3`. Na linha 130 o elemento `genre` é selecionado a partir do conteúdo da variável `b2_12` (conforme mostra a linha 134).

Já as linhas 138 a 147 representam a segunda parte da comparação, identificando-a como `v_4` (linha 138). Na linha 141 o elemento `genre` é selecionado. Para isso, considera-se a versão da variável `b1_6`, conforme representado na linha 145.

O fragmento que possui a comparação das duas partes da expressão do operador `Join` está nas linhas 148 a 151. Na linha 148 o operador de igualdade é indicado (`string-equal`) para comparar `v_3` (linha 149) com `v_4` (linha 150).

A figura 5.18 ilustra o operador `Map` (linha 155) referente a cláusula `for`. Logo a seguir, as linhas 178 a 180 representam a seleção do documento `book_p1.xml`. Os demais passos da expressão XPath foram ocultados para fins de simplicidade. Essa consulta, além de

selecionar o documento `book_p1.xml`, também seleciona o documento `book_p2.xml`. Por esse motivo existe outro operador `Map` referente a cláusula `for` na linha 186. As linhas 214 a 216 representam a seleção do documento `book_p2.xml`.

```
155  <input_operator name="Map" op="for">
...  ...
178                  <input_operator name="Parse">
179                     <input_operator name="book_p1.xml"/>
180                  </input_operator> ...
...  ...
186     <input_operator name="Map" op="for">
...  ...
214                              <input_operator name="Parse">
215                                 <input_operator name="book_p2.xml"/>
216                              </input_operator>
```

Figura 5.18 – Fragmento da consulta Q3 referente à cláusula `for`

A seção a seguir descreve as expressões em álgebra relacional, geradas a partir da aplicação das regras apresentadas na seção 4.4.

5.6 Expressões em álgebra relacional

Nesta seção são descritos os resultados da execução do componente XARMap, ou seja, a segunda etapa do mecanismo QMap. As regras definidas na seção 3.2 são a base para a geração desses resultados. As expressões em álgebra relacional são representadas através de documentos XML, a fim de facilitar sua compreensão e implementação. A DTD que descreve a estrutura desses documentos é apresentada no apêndice B.

A seção a seguir demostra a construção da álgebra relacional para a consulta Q1.

5.6.1 Álgebra relacional da consulta Q1 sobre documento `bib.xml`

Esta seção descreve a expressão em álgebra relacional da consulta Q1 executada sobre o documento `bib.xml`. A figura 5.19 ilustra o documento XML que representa a expressão em álgebra relacional correspondente à consulta Q1 sobre o documento `bib.xml`.

As duas primeiras regras de tradução executadas são a $(\mathbf{Tr_{19}})$ e $(\mathbf{Tr_{20}})$, que traduzem uma expressão FLWOR e o operador da cláusula `return`, respectivamente. Nesse caso, a construção específica do operador `return` conduz à execução da regra $(\mathbf{Tr_{26}})$. Esta regra gera um operador de projeção (Π na álgebra relacional (linhas 20 a 25 da figura 5.19), que é seguido pela tradução do construtor de elemento e conduz à tradução do construtor de elemento e do operador da cláusula `where`.

Posteriormente, a regra geral para o construtor de elemento, ($\mathbf{Tr_{33}}$), indica a execução da regra específica ($\mathbf{Tr_{36}}$). Por sua vez, essa regra conduz à tradução do construtor de sequência presente no construtor de elemento. O operador de sequência é traduzido de forma geral pela regra ($\mathbf{Tr_{40}}$) e, especificamente, pela regra ($\mathbf{Tr_{46}}$). Essa regra define a tradução para uma sequência composta por um operador de atributo e uma expressão XPath.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <relational_algebra>
3    <expression>
4      <relations>
5        <table>book</table>
6      </relations>
7      <selection>
8        <condlist>
9          <column table="book">publisher</column>
10         <comp_op> "=" </comp_op>
11         <data>'addison-wesley'</data>
12       </condlist>
13       <logical_op>and</logical_op>
14       <condlist>
15         <column table="book">year</column>
16         <comp_op> ">" </comp_op>
17         <data>1991</data>
18       </condlist>
19     </selection>
20     <projection>
21       <columnlist>
22         <column table="book">year</column>
23         <column table="book">title</column>
24       </columnlist>
25     </projection>
26   </expression>
27 </relational_algebra>
```

Figura 5.19 – Documento XML da expressão em álgebra relacional referente à consulta Q1

Nesse sentido, as regras executadas a seguir são a ($\mathbf{Tr_{38}}$) e a ($\mathbf{Tr_{39}}$). A primeira é a regra geral para o operador de construção de atributo e a segunda é específica. Esta última define que a expressão XPath presente na construção do atributo deve ser traduzida. Sendo assim, a regra ($\mathbf{Tr_1}$) é executada duas vezes. Uma para a expressão XPath presente no operador de construção de atributo que origina a coluna `year` (linha 22). Também é executada para a expressão XPath da sequência embutida na construção de elemento, gerando a coluna `title` (linha 23). A expressão em álgebra relacional gerada até o presente momento é: $\Pi_{year,\ title}$

Depois disso, a regra ($\mathbf{Tr_{61}}$) é executada para traduzir o operador da cláusula `where`. Nesse caso, indica-se a execução da regra específica ($\mathbf{Tr_{63}}$), que gera um operador de seleção (σ) na álgebra relacional. Posteriormente, a expressão de comparação presente no filtro da

consulta original é traduzida pelas regras ($\mathbf{Tr_4}$) e ($\mathbf{Tr_2}$), sendo que a última é executada para as duas comparações unidas pelo operador lógico 'and'. A regra ($\mathbf{Tr_2}$) indica que cada parte da comparação deve ser traduzida pela regra ($\mathbf{Tr_1}$) e o operador de comparação pela regra geral ($\mathbf{Tr_6}$), nesse caso.

Sendo assim, as colunas publisher (linha 9) e year (linha 15), bem como os valores 'addison-wesley' (linha 11) e 1991 (linha 17) são obtidos através da execução da regra ($\mathbf{Tr_1}$). Os operadores de comparação '=' (linha 10) e '>' (linha 16) são gerados a partir das regras específicas ($\mathbf{Tr_{13}}$) e ($\mathbf{Tr_{17}}$), respectivamente. Essas traduções resultam na seguite parte da expressão em álgebra relacional: $\sigma_{publisher = \text{“addison-wesley”} \wedge year > 1991}$.

Por fim, o operador da cláusula for é traduzido pela regra geral ($\mathbf{Tr_{68}}$) e pela regra específica ($\mathbf{Tr_{69}}$). Esta última, indica novamente a execução da regra ($\mathbf{Tr_1}$) para a expressão XPath ligada à variável da cláusula for. Nessa consulta o retorno da regra ($\mathbf{Tr_1}$) sempre indica a tabela book. Por esse motivo as linhas 4 a 6 representam a tabela *book* que é a única envolvida na expressão.

A expressão em álgebra relacional representada pelo documento XML da figura 5.19 também pode ser representada no formato clássico:

$\Pi_{year,\, title}(\sigma_{publisher = \text{“addison-wesley”} \wedge year > 1991}(book))$.

A próxima seção demonstra a construção da álgebra relacional para a consulta Q2.

5.6.2 Álgebra relacional da consulta Q2 sobre documento book_p1.xml

Nesta seção é descrita a geração da expressão em álgebra relacional para a consulta Q2 executada sobre o documento book_p1.xml. A seguir, a figura 5.20 ilustra o documento XML que representa a expressão em álgebra relacional da consulta Q2 sobre o documento book_p1.xml.

Primeiramente, as regras ($\mathbf{Tr_{19}}$) e ($\mathbf{Tr_{20}}$) são executadas, traduzindo a expressão em álgebra XQuery e o operador da cláusula return de modo geral. Logo depois, a regra ($\mathbf{Tr_{25}}$) indica a tradução do operador $MapToItem$ como sendo um operador de projeção (Π) da álgebra relacional (linhas 24 a 29 da figura 5.20), seguido pelo retorno da tradução do operador de construção de elemento e do operador da cláusula order by.

A seguir, a regra geral ($\mathbf{Tr_{33}}$) e a específica ($\mathbf{Tr_{36}}$), encarregam-se de traduzir o operador de construção de elemento. Nesse caso, o elemento contém um operador de construção de sequência. Por sua vez, a sequência é traduzida de forma geral pela regra ($\mathbf{Tr_{40}}$). Por ser

uma sequência de construtor de atributo e expressão XPath a regra ($\mathbf{Tr_{46}}$) é aplicada. A mesma indica a tradução do construtor de atributo pelas regras ($\mathbf{Tr_{38}}$) e ($\mathbf{Tr_{39}}$), geral e específica respectivamente.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <relational_algebra>
3   <expression>
4    <relations>
5     <table>book</table>
6     <relation_op>
7      <name>CARTESIAN_PRODUCT</name>
8     </relation_op>
9     <table>author</table>
10   </relations>
11   <selection>
12    <condlist>
13     <column table="book">publish</column>
14     <comp_op>">"</comp_op>
15     <data>2000</data>
16    </condlist>
17    <logical_op>AND</logical_op>
18    <condlist>
19     <column table="author">first</column>
20     <comp_op> "=" </comp_op>
21     <data>"Corets, Eva"</data>
22    </condlist>
23   </selection>
24   <projection>
25    <columnlist>
26     <column table="book">publish</column>
27     <column table="book">title</column>
28    </columnlist>
29   </projection>
30   <order_by>
31    <orderlist>
32     <column table="book">title</column>
33     <order>desc</order>
34    </orderlist>
35   </order_by>
36  </expression>
37 </relational_algebra>
```

Figura 5.20 – Documento XML da expressão em álgebra relacional referente à consulta Q2

Nesse sentido, a regra ($\mathbf{Tr_1}$) é executada para a expressão XPath do construtor de atributo, retornando a coluna `publish` (linha 26). Já a coluna `title` (linha 27) é originada pela execução da regra ($\mathbf{Tr_1}$) para a expressão XPath da sequência presente no construtor de elemento. A expressão em álgebra relacional gerada até o momento é: $\Pi_{publish,\ title}$.

Em seguida, a fim de traduzir o operador da cláusula `order by`, as regras ($\mathbf{Tr_{57}}$) e ($\mathbf{Tr_{58}}$) são aplicadas. A regra ($\mathbf{Tr_{58}}$) é específica, sendo que gera um operador de classificação (τ) na álgebra relacional (linhas 30 a 35), indica a execução da regra ($\mathbf{Tr_1}$), para traduzir a expressão XPath, e da regra geral ($\mathbf{Tr_{61}}$) para traduzir o operador da cláusula `where`. Da execução da regra ($\mathbf{Tr_1}$) é retornada a coluna `title` (linha 32). Já o tipo de ordenação (linha 33), ou seja, se ascendente ou descendente, é obtido através da consulta XQuery original, pois não é possível identificá-lo na expressão algébrica. A partir dessas regras, gera-se a seguinte parte da expressão em álgebra relacional (linhas 30 a 35): $\tau_{title\ \texttt{desc}}$.

Posteriormente, a regra específica ($\mathbf{Tr_{62}}$) é executada a partir da regra geral ($\mathbf{Tr_{61}}$) para traduzir o operador $Select$. Essa execução gera um operador de seleção (σ) na álgebra relacional (linhas 11 a 23). Este operador é seguido pela tradução da expressão de comparação, que é obtida pela regra ($\mathbf{Tr_2}$), e pela tradução do operador da cláusula `let`. A regra ($\mathbf{Tr_2}$) indica a execução da regra ($\mathbf{Tr_1}$) para as duas expressões XPath que compõe a comparação, retornando a coluna `publish` (lina 13) e o valor `2000` (linha 15). Já o operador de comparação '>'

(linha 14) é obtido através da regra geral ($\mathbf{Tr_6}$) e da regra específica ($\mathbf{Tr_{17}}$). O fragmento de álgebra relacional gerado por essas regras encontra-se nas linhas 11 a 16 da figura 5.20 e pode ser representado como: $\sigma_{publish > 2000}$.

O operador da cláusula `let` é traduzido pela regra geral ($\mathbf{Tr_{66}}$) e, posteriormente, pela regra específica ($\mathbf{Tr_{67}}$). Essa última regra indica a tradução da expressão XPath ligada a variável do operador $MapConcat$ pela regra ($\mathbf{Tr_1}$) e do operador da cláusula `for` pela regra geral ($\mathbf{Tr_{68}}$), se for o caso. As informações retornadas pela regra ($\mathbf{Tr_1}$) são carregadas em memória e consistem na coluna `publish` da tabela `book`.

Por fim, a regra específica ($\mathbf{Tr_{69}}$), para os operadores $MapConcat$ e $MapFromItem$, indica a tradução da expressão XPath ligada a variável presente na cláusula `for`. Essa expressão XPath é traduzida pela regra ($\mathbf{Tr_1}$). Durante a execução da função (dada pelo algoritmo 1) a regra ($\mathbf{Tr_2}$) é indicada para traduzir o filtro existente na expressão XPath. Sendo assim, um operador '`and`' (linha 17) é adicionado a seleção da álgebra relacional, juntamente com o retorno da regra ($\mathbf{Tr_2}$). Nesse caso, a coluna `first` (linha 19) da tabela `author` e o valor "`Corets, Eva`" (linha 21) são retornados pela regra ($\mathbf{Tr_1}$) e o operador '=' pelas regras ($\mathbf{Tr_6}$) e ($\mathbf{Tr_{13}}$).

O fragmento das linhas 4 a 10 contém as tabelas *book* (linha 5) e *author* (linha 9) envolvidas na expressão obtidas através das execuções da regra ($\mathbf{Tr_1}$) para as expressões XPath. Essas tabelas são relacionadas através de um operador de produto cartesiano (linha 7). A operação de produto cartesiano é gerada em nível de implementação para os casos em que não há operador de junção na álgebra XQuery original.

A expressão em álgebra relacional representada na figura 5.20 também pode ser escrita no formato clássico: $\tau_{title\ \texttt{desc}}(\Pi_{publish,\ title}(\sigma_{publish > 2000\ \wedge\ first = \text{"Corets, Eva"}}(author \times book)))$

A seguir, a próxima seção descreve a expressão em álgebra relacional referente à consulta Q3.

5.6.3 Álgebra relacional da consulta Q3 sobre documento `book_p2.xml`

Esta seção demonstra a construção da expressão em álgebra relacional gerada a partir da consulta Q3 sobre o documento `book_p2.xml`. O documento XML ilustrado na figura 5.21 representa essa expressão em álgebra relacional.

As primeiras regras a serem aplicadas nesse caso também são a ($\mathbf{Tr_{19}}$) e ($\mathbf{Tr_{20}}$), traduzindo a expressão FLWOR e o operador da cláusula `return` de maneira geral. Em seguida a

regra ($\mathbf{Tr_{27}}$) é executada a fim de traduzir o caso específico do operador $MapToItem$, gerando um operador de projeção (Π) na álgebra relacional (linhas 16 a 23). A regra ($\mathbf{Tr_{33}}$) é então indicada para traduzir o operador de construção de elemento presente na cláusula `return`.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <relational_algebra>
3    <expression>
4      <relations>
5        <table rename="book1">book</table>
6        <relation_op>
7          <name>THETA JOIN</name>
8          <condlist>
9            <column table="book1">genre</column>
10           <comp_op> "=" </comp_op>
11           <column table="book2">genre</column>
12         </condlist>
13       </relation_op>
14       <table rename="book2">book</table>
15     </relations>
16     <projection>
17       <columnlist>
18         <column table="book1">price</column>
19         <column table="book1">title</column>
20         <column table="book2">price</column>
21         <column table="book2">title</column>
22       </columnlist>
23     </projection>
24   </expression>
25 </relational_algebra>
```

Figura 5.21 – Documento XML da expressão em álgebra relacional referente à consulta Q3

Especificamente, a regra ($\mathbf{Tr_{36}}$) é executada pois o construtor de elemento possui um operador de sequência. A tradução geral desse operador é dada pela regra ($\mathbf{Tr_{40}}$) e o caso específico, composto por dois construtores de elemento, é tratado pela regra ($\mathbf{Tr_{49}}$). Cada um dos construtores de elemento é tratado novamente pela regra geral ($\mathbf{Tr_{33}}$) e, posteriormente, pela regra específica ($\mathbf{Tr_{36}}$), pois são compostos por uma sequência de um construtor de atributo e uma expressão XPath. Sendo assim, a regra geral de sequência ($\mathbf{Tr_{40}}$) é executada e em seguida a regra específica ($\mathbf{Tr_{46}}$).

Para traduzir o operador de construção de atributo, a regra geral ($\mathbf{Tr_{38}}$) indica a regra específica ($\mathbf{Tr_{39}}$). Por sua vez, a regra ($\mathbf{Tr_1}$) é aplicada a fim de traduzir a expressão XPath presente no operador de construção de atributo, retornando as colunas `price` das linhas 18 e 20. A expressão XPath presente em cada construtor de elemento também é traduzida através da regra ($\mathbf{Tr_1}$), retornando as colunas `title` das linhas 19 e 21. A expressão em álgebra relacional construída até o momento é: $\Pi_{book1.price,\ book1.title,\ book2.price,\ book2.title}$.

Em seguida, o operador da cláusula `where` é traduzido pela regra geral ($\mathbf{Tr_{61}}$) que, posteriormente, aplica a regra específica ($\mathbf{Tr_{65}}$). Essa regra trata o operador algébrico $Join$ gerando uma junção theta ($\bowtie_\theta$) na álgebra relacional. Essa junção é aplicada às tabelas envolvidas e possui um critério de junção (linhas 8 a 12) obtido através da tradução da expressão de comparação do operador $Join$. Para isso, a regra ($\mathbf{Tr_2}$) é aplicada à expressão de comparação. Por sua vez, a regra ($\mathbf{Tr_1}$) é aplicada para cada parte da comparação, resultando nas colunas `genre` (linhas 9 e 11). O operador '=' (linha 10) é obtido através da regra geral ($\mathbf{Tr_6}$) e da específica ($\mathbf{Tr_{13}}$).

Por fim, o operador da cláusula `for` é traduzido pela regra geral ($\mathbf{Tr_{68}}$). Essa regra indica a tradução específica através da regra ($\mathbf{Tr_{69}}$), que aplica a regra ($\mathbf{Tr_1}$) para a expressão XPath ligada à cada variável presente no operador da cláusula `for`. O retorno dessa regra consiste na tabela `book` que, nesse caso, é referenciada duas vezes (linhas 5 e 14). Isso acontece pelo fato de existir uma junção theta (linha 7) com critério de junção pela coluna `genre` (linhas 9 a 12). Por esse motivo, é necessário renomear, no nível de implementação, cada tabela através do atributo `rename` presente nas linhas 5 e 14. Sendo assim, as tabelas passam a ser referenciadas por estes novos nomes, conforme pode ser visto nas linhas 9, 11 e 18 a 21.

A expressão final em álgebra relacional pode ser representada em seu formato clássico: $\Pi_{\textit{book1.price, book1.title, book2.price, book2.title}}(\rho_{book2}(book) \bowtie_{\theta \textit{book2.genre = book1.genre}} \rho_{book1}(book))$.

A próxima seção descreve as consultas SQL geradas a partir das expressões em álgebra relacional.

5.7 Consultas SQL

Esta seção apresenta a geração de consultas SQL, papel do último componente de QMap, o RASMap. As consultas SQL são obtidas através das regras de equivalência entre os operadores da álgebra relacional e operadores da SQL, conforme descrito na tabela 4.4 da seção 4.5.

A próxima seção apresenta a consulta SQL referente à consulta XQuery Q1.

5.7.1 Instrução SQL equivalente à consulta Q1 sobre o documento `bib.xml`

Nesta seção é monstrada a consulta SQL resultante do processo de tradução realizado sobre a consulta Q1. A figura 5.22 ilustra a consulta SQL gerada a partir da consulta Q1, executada sobre o documento `bib.xml`.

```
1  SELECT  year, title
2  FROM    book
3  WHERE   publisher = 'addison-wesley' AND
4          year > 1991
```

Figura 5.22 – Consulta SQL Q1

A cláusula `SELECT` da SQL origina-se a partir da regra de equivalência com o operador de projeção (Π) da álgebra relacional. Nesse caso em específico, a projeção das colunas `year` e `title`, presente na expressão em álgebra relacional, equivale à cláusula `SELECT` mostrada na linha 1 da figura 5.22.

O operador de seleção (σ) da álgebra relacional equivale à cláusula `WHERE` da SQL. Na expressão algébrica da consulta Q1 o filtro envolvendo as colunas `publisher` e `year` origina a cláusula `WHERE` presente nas linhas 3 e 4.

Consequentemente, a tabela `book` utilizada na expressão em álgebra relacional é traduzida para a cláusula `FROM` da SQL mostrada na linha 2 da figura 5.22.

A seção seguinte apresenta a consulta SQL resultante da tradução da consulta XQuery Q2.

5.7.2 Instrução SQL equivalente à consulta Q2 sobre o documento `book_p1.xml`

Esta seção descreve a geração da consulta SQL referente à tradução da consulta XQuery Q2, originalmente executada sobre o documento `book_p1.xml`. A figura 5.23 ilustra a instrução SQL resultante do processo de tradução.

```
1  SELECT  title, publish
2  FROM    book CROSS JOIN author
3  WHERE   publish > 2000 AND
4          first = 'Corets, Eva'
5  ORDER BY title desc
```

Figura 5.23 – Consulta SQL Q2

Nota-se que a cláusula `SELECT` presente na linha 1 da figura 5.23 origina-se do operador de projeção (Π) da álgebra relacional. Essa projeção envolve as colunas `publish` e `title`. Já a cláusula `FROM` (linha 2) origina-se da operação de produto cartesiano ($\times$) entre as tabelas `book` e `author` presente na expressão em álgebra relacional.

A cláusula `WHERE`, representada nas linhas 3 e 4, origina-se do operador de seleção (σ), que neste caso contém uma comparação envolvendo as colunas `publish` e `first`. Já a cláusula `ORDER BY` (linha 5) é equivalente ao operador de classificação (τ) da álgebra relacional,

que nesta expressão utiliza a coluna `title` decrescente como referência para a ordenação do resultado.

A próxima seção contém a instrução SQL referente à consulta XQuery Q3.

5.7.3 Instrução SQL equivalente à consulta Q3 sobre o documento `book_p2.xml`

Nesta seção é apresentada a instrução SQL resultante do processo de tradução da consulta Q3. A figura 5.24 representa a instrução SQL relativa à consulta Q3 do documento `book_p2.xml`.

```
1  SELECT  book1.genre, book1.title, book2.genre, book2.title
2  FROM    book as book1
3          join
4          book as book2
5                  on (book1.genre = book2.genre)
```

Figura 5.24 – Consulta SQL Q3

Nessa consulta a cláusula `SELECT` (linha 1) é gerada a partir da projeção (Π) das colunas `price` e `title`. A junção theta ($\bowtie_\theta$) origina a junção presente na cláusula `FROM` (linhas 2 a 5) da instrução SQL. A operação de renomeação (ρ) é representada nas linhas 2 e 4 através do uso do operador `as` da SQL.

A próxima seção demonstra a tradução inversa para uma das três consultas utilizadas nos experimentos.

5.8 Equivalência das consultas

Esta seção tem o objetivo de exemplificar a aplicação das regras inversas de tradução definidas na seção 3.3. Essas regras dizem respeito ao subcomponente XARMap, ou seja, pretendem comprovar a equivalência da tradução entre a expressão em álgebra XQuery e a expressão em álgebra relacional.

Para isso, utiliza-se como exemplo a consulta Q1, mostrada na figura 5.2. Após a aplicação das regras de tradução definidas no subcomponente XARMap, gera-se a seguinte expressão em álgebra relacional (também apresentada na figura 5.19):

$$\Pi_{year,\ title}(\sigma_{publisher = \text{“addison-wesley”} \wedge year > 1991}(book)).$$

A partir dessa expressão em álgebra relacional aplicam-se as regras de tradução inversa, definidas na seção 3.3, a fim de gerar uma expressão em álgebra XQuery de acordo com a

apresentada na seção 5.5.1. A figura 5.25 ilustra a expressão simplificada em álgebra XQuery que foi obtida através da aplicação das regras de tradução inversa.

```
1  MapToItem
2      {Sequence(
3          /bib/book/year,
4          /bib/book/title
5          )
6      }(Select {
7          /bib/book/publisher = 'addison-wesley'
8          AND
9          /bib/book/year > 1991}
10             (MapConcat{MapFromItem[Var : IN]/bib/book})
```

Figura 5.25 – Álgebra XQuery da consulta Q1 gerada pela tradução inversa

Primeiramente, a regra geral ($\mathbf{Eq_1}$) é aplicada sobre toda a expressão em álgebra relacional referente à consulta Q1. Logo depois, a regra ($\mathbf{Eq_3}$) é aplicada a fim de traduzir a projeção (Π) de atributos e o restante da expressão. Sendo assim, um operador $MapToItem$ (linha 1) da álgebra XQuery é gerado, seguido da tradução da lista de atributos.

Desse modo, a regra geral ($\mathbf{Eq_8}$) indica a tradução da lista de atributos através da regra ($\mathbf{Eq_9}$). Esta, por sua vez, gera um construtor de sequência ($Sequence$) na linha 2 da álgebra XQuery, seguido da tradução dos atributos. A regra ($\mathbf{Eq_{10}}$) é executada e, a partir da busca no documento de mapeamento, gera-se a expressão XPath relativa à coluna `year` (linha 3). Em seguida, a regra geral ($\mathbf{Eq_8}$) é executada a fim de traduzir o restante dos atributos. Ela indica novamente a regra ($\mathbf{Eq_{10}}$) que deve retornar a expressão XPath referente à coluna `title` (linha 4).

Em seguida, a regra geral ($\mathbf{Eq_1}$) é novamente aplicada a fim de indicar a próxima tradução. Nesse caso, executa-se a regra ($\mathbf{Eq_4}$) a fim de traduzir o operador de seleção (σ) da álgebra relacional. Essa regra origina o operador $Select$ (linha 6) na álgebra XQuery, que é seguido pela tradução da lista de condições e do restante da expressão.

Em geral, a lista de condições é traduzida pela regra ($\mathbf{Eq_{11}}$). Nesse caso, a regra ($\mathbf{Eq_{13}}$) deve ser aplicada pois a lista de condições é composta por duas outras condições ligadas pelo operador lógico '`AND`', que permanece o mesmo na álgebra XQuery (linha 8). Dessa forma, a regra geral ($\mathbf{Eq_{11}}$) é aplicada à cada uma das condições, indicando a regra ($\mathbf{Eq_{16}}$) nas duas vezes. Por sua vez, a regra ($\mathbf{Eq_{16}}$) aplica a regra ($\mathbf{Eq_{17}}$) à cada parte das comparações e a regra ($\mathbf{Eq_{18}}$) aos operadores de comparação. Sendo assim, a regra ($\mathbf{Eq_{10}}$) é aplicada às colunas `publisher` e `year` retornando suas respectivas expressões XPath (linhas 7 e 9). Os valores

‘addison-wesley’ e 1991 (linhas 7 e 9) permanecem os mesmos, sendo traduzidos pela regra geral ($\mathbf{Eq_{25}}$) e, respectivamente, pelas regras ($\mathbf{Eq_{27}}$) e ($\mathbf{Eq_{26}}$).

Por fim, a regra ($\mathbf{Eq_1}$) é novamente aplicada sobre o restante da expressão, que nesse caso é a tabela book. Nesse sentido, a regra ($\mathbf{Eq_2}$) é executada e, a partir da uma busca no documento de mapeamento, a expressão XPath relativa à tabela book é retornada. Essa expressão é ligada à uma variável presente nos operadores $MapConcat$ e $MapFromItem$ da álgebra XQuery (linha 10).

5.9 Implementação

Nesta seção é descrita uma proposta de implementação baseada na especificação do componente QMap apresentada no capítulo anterior. Essa implementação tem o objetivo de automatizar as regras de mapeamento entre a álgebra XQuery e a álgebra relacional, função do subcomponente XARMap. O desenvolvimento deste protótipo foi conduzido pelo acadêmico e bolsista PET (Programa de Educação Tutorial), do curso de Sistemas de Informação da UFSM, Leonardo de Oliveira Nicorena.

O aplicativo desenvolvido recebe como entrada a representação XML da expressão em álgebra XQuery. Essa representação XML foi elaborada ao longo deste trabalho e consiste em uma definição própria a fim de facilitar sua implementação. Ela tem como base a notação de operadores da álgebra XQuery definida em (RE; SIMEON; FERNANDEZ, 2006): $Op[p1, ..., pi]DOp1, ..., DOph(Op1, ..., Opk)$. Para representar esses operadores foram definidas algumas tags XML: <operator> representa Op; <static_param> representa os parâmetros estáticos $[p1, ..., pi]$ do operador Op; os operadores dependentes $DOp1, ..., DOph$ são representados por <sub_operator> e, por fim, <input_operator> representa os operadores de entrada $(Op1, ..., Opk)$.

Os documentos XML que representam as expressões em álgebra XQuery são validados pela DTD (*Document Type Definition*) presente no apêndice A. Outra característica importante desses documentos XML é que em alguns elementos foi necessário adicionar um atributo op, que indica o operador que está sendo representado. Isso foi necessário para os elementos que representam operadores das cláusulas for, let e where, pois estas apresentam-se em combinações muito variadas, dificultando sua identificação. Esse atributo também facilita a implementação do mecanismo, pois indica diretamente o operador que deve ser tratado.

As expressões em álgebra relacional também são representadas através de documentos

XML. Essa representação é baseada na definição formal da álgebra, presente em (SILBERSCHATZ; KORTH; SUNDARSHAN, 2006). A DTD que valida esses documentos é apresentada no apêndice B.

5.9.1 Tecnologias utilizadas

Para a implementação do componente XARMap do *framework* X2Rel utilizou-se a linguagem de programação Java (plataforma J2SE versão 1.7.0_10) e o ambiente de programação NetBeans (IDE 7.2.1). As principais bibliotecas usadas são a JDOM, na manipulação de arquivos XML de entrada, e a SAXON, na execução de consulta XQuery sobre o documento de mapeamento.

Para a geração da álgebra XQuery a partir das consultas originais utiliza-se o mecanismo *open source* Galax (GALAX, 2010). Nele são implementadas as regras de inferência definidas em (RE; SIMEON; FERNANDEZ, 2006). A versão do Galax utilizada é a 1.0, sendo executada no sistema operacional Ubuntu.

A seguir, é apresentado e descrito o diagrama de classes da implementação do XARMap.

5.9.2 Diagrama de classes

Esta seção descreve o diagrama de classes da implementação de QMap. A figura 5.26 apresenta o diagrama com os pacotes e classes utilizadas pelo aplicativo desenvolvido.

Intuitivamente, o pacote `logger` e sua respectiva classe *Logging* têm o objetivo de armazenar os *logs* da execução do aplicativo. Esse armazenamento vai além de erros e mensagens, sendo muito importante por guardar versões das variáveis ligadas à expressões XPath que estão presentes nas consultas XQuery originais. Essas informações armazenadas em memória são utilizadas por outras classes, como por exemplo a classe *Equivalencias*, que efetivamente aplicam as regras de tradução.

Basicamente, o pacote `core` engloba as principais classes da implementação, ou seja, as que manipulam os documentos de entrada, realizam a tradução e geram as informações de saída. A classe *Core* centraliza e determina a ordem de execução das outras classes. Já as classes *AlgebraXqHandler*, *EquivalenciaHandler* e *AlgebraRelHandler* manipulam, respectivamente, os arquivos XML de álgebra XQuery, de mapeamento e de álgebra relacional. No caso dos arquivos XML de entrada - álgebra XQuery e documento de mapeamento - suas classes também extraem as informações necessárias para a tradução. Já a classe *AlgebraRelHandler* constrói o

documento XML que representa a álgebra relacional gerada a partir da aplicação das regras definidas na seção 3.2.

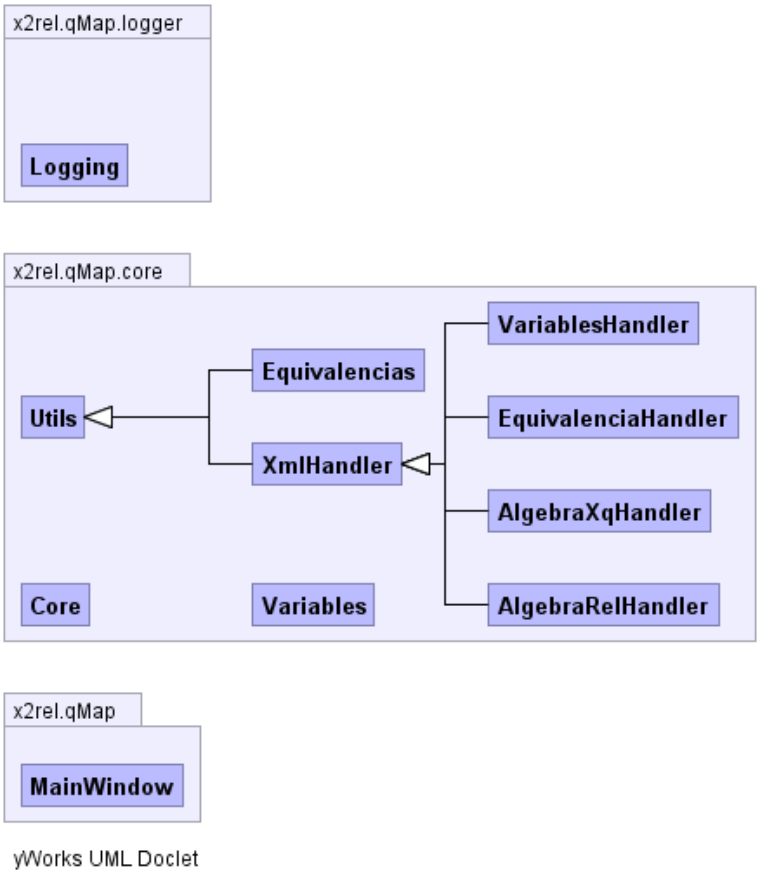

Figura 5.26 – Diagrama de classes

As classes *Variables* e *VarablesHandler* armazenam e manipulam, respectivamente, as variáveis presentes na expressão em álgebra XQuery. Já a classe *XmlHandler* possui métodos específicos para manipular e percorrer os documentos XML. A classe *Equivalencias* possui métodos que obtém informações da tradução a fim de construir o documento XML que representa a álgebra relacional. Por fim, a classe *Utils* contém alguns métodos para fins específicos, tais como execução de consulta XQuery, leitura de arquivo, busca de variáveis, entre outros. Já a classe *MainWindow* é a interface do aplicativo.

O apêndice F contém os diagramas que detalham cada classe em específico. A seção a seguir apresenta o protótipo desenvolvido, descrevendo seu funcionamento.

5.9.3 Protótipo de XARMap

Nesta seção o protótipo desenvolvido para XARMap é apresentado. Basicamente o aplicativo recebe como entrada a álgebra XQuery e o documento de mapeamento, ambos representados em XML. A saída é um documento XML que representa a expressão em álgebra relacional.

A figura 5.27 ilustra a tela inicial do protótipo. O mesmo possui espaços e botões utilizados para selecionar as informações de entrada, neste caso os documentos XML da álgebra XQuery (*XQuery Álgebra*) e o de mapeamento (*Equivalências*). Já a funcionalidade *Gerar álgebra relacional* aplica as regras de tradução definidas neste trabalho (seção 3.2) gerando como saída a representação XML de uma expressão em álgebra relacional.

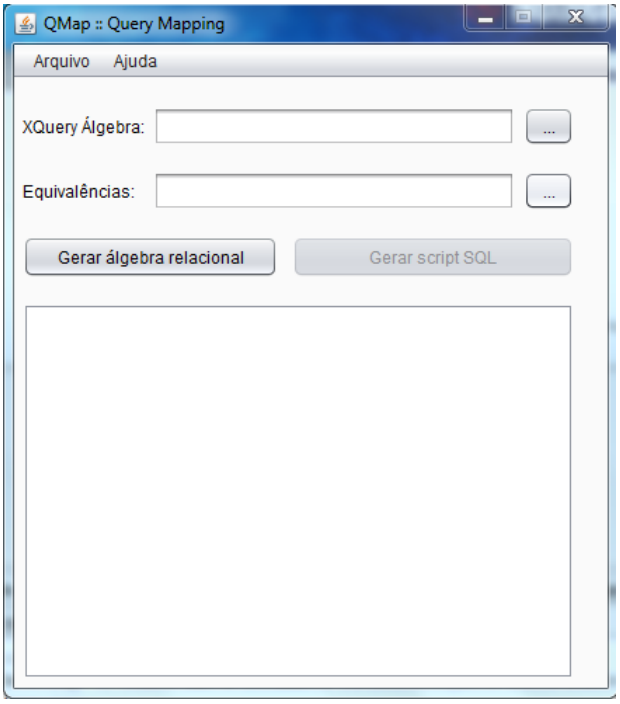

Figura 5.27 – Tela inicial de XARMap

Um exemplo de utilização da ferramenta é apresentado na figura 5.28. Nele são selecionados os documentos XML referentes à expressão em álgebra XQuery de uma consulta Q1 e o documento de mapeamento, respectivamente.

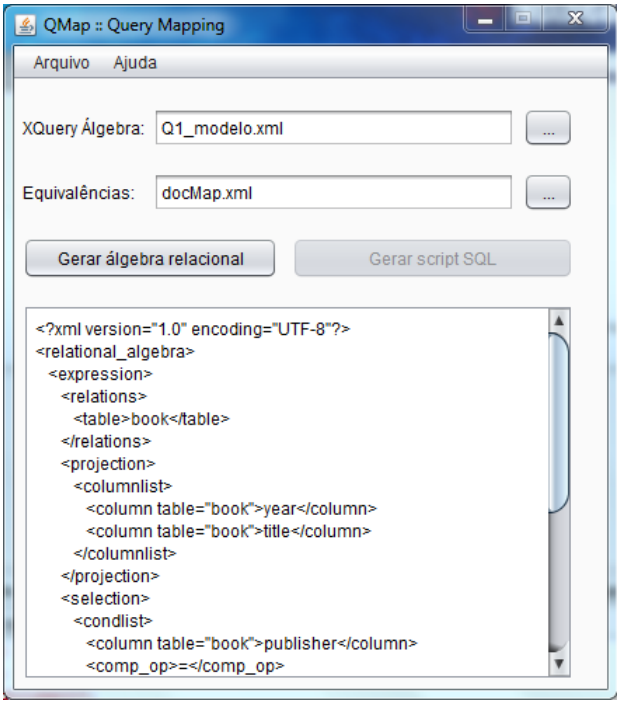

Figura 5.28 – Utilização de XARMap

Após a funcionalidade *Gerar álgebra relacional* ser acionada, a expressão resultante da aplicação das regras de tradução é mostrada em formato XML.

5.10 Considerações finais

Este capítulo teve o objetivo de demonstrar experimentos práticos a partir da proposta definida nesse trabalho. Para isso, foi necessária a utilização dos componentes pertencentes ao *framework* X2Rel anteriores ao QMap. Somente após a geração da ontologia, do esquema relacional e do documento de mapeamento, o processo de tradução iniciou-se. O objetivo da utilização de consultas sobre diferentes documentos foi de garantir a diversadade de regras necessárias para traduzí-las.

Neste capítulo também foi apresentada uma proposta de implementação de XARMap. Essa implementação tem o objetivo de gerar uma expressão em álgebra relacional a partir de uma expressão em álgebra XQuery. Essa expressão em álgebra XQuery é obtida através da utilização do mecanismo Galax (GALAX, 2010) que implementa as regras de geração definidas em (RE; SIMEON; FERNANDEZ, 2006). A fim de facilitar a implementação, optou-se por representar as expressões algébricas de entrada e saída através de documentos XML. A estrutura desses documentos foi definida ao longo deste trabalho.

A consulta SQL ainda precisa ser definida manualmente, a partir da expressão em álge-

bra relacional gerada como saída pela ferramenta. As classes e o protótipo da implementação também foram apresentados ao longo do capítulo. A descrição do funcionamento da ferramenta é realizado a partir do diagrama de classes e de figuras ilustrativas do protótipo.

6 CONCLUSÕES

Problemas como a integração, intercâmbio, publicação e interoperabilidade de dados podem ser solucionados através do uso de XML. Apesar dessa ampla utilização de XML, a maior parte dos dados corporativos continua sendo armazenada em Bancos de Dados Relacionais (LI et al., 2003). Sendo assim, cada vez mais são necessárias soluções que gerenciem ambos tipos de dados (XML e relacionais).

Nesse sentido, o *framework* X2Rel apresenta diferentes funcionalidades que permitem o mapeamento de dados XML para um Banco de Dados Relacional. Sabe-se que é muito comum que esses dados XML sejam consultados através de expressões XQuery. Para que o mapeamento realizado pelo *framework* X2Rel seja completo, as consultas existentes no ambiente XML precisam ser traduzidas para consultas equivalentes no ambiente relacional. Sendo assim, o componente QMap, apresentado neste trabalho, tem o papel de traduzir as consultas XQuery para consultas SQL através de uma abordagem algébrica.

Essa abordagem propõe a definição de regras de tradução entre a álgebra XQuery e a álgebra relacional, garantindo uma solução mais genérica e passível de otimização. Em comparação com os trabalhos relacionados, a utilização da álgebra XQuery, definida em (RE; SIMEON; FERNANDEZ, 2006), juntamente com as regras de tradução para álgebra relacional, é um dos diferenciais deste trabalho.

Outras contribuições desta proposta são relacionadas à definição da arquitetura de QMap, especificação da gramática da álgebra XQuery, definição das regras de tradução, especificação de representações XML de expressões algébricas e implementação de um protótipo que aplica as regras definidas.

A arquitetura proposta para QMap é independente de implementação. O princípio básico consiste na tradução de consultas em nível algébrico. Nesse sentido, os componentes de QMap são estruturados a fim de contemplar as fases necessárias para a tradução entre as álgebras. Por esse motivo, o primeiro componente, XALGen, tem a função de gerar a álgebra XQuery a partir de uma consulta XQuery de entrada. O segundo componente, XARMap, realiza a tradução em nível algébrico, entre a álgebra XQuery e relacional. Por fim, o componente RASMap gera a consulta SQL de saída a partir da álgebra relacional gerada anteriormente. Dessa forma, é possível afirmar que a arquitetura é flexível no sentido de implementação da mesma, pois permite que existam etapas intermediárias, se necessário, e não restringe o uso de ferramentas

já existentes para a implementação de seus componentes.

A especificação da gramática da álgebra XQuery também é uma contribuição deste trabalho. Em (RE; SIMEON; FERNANDEZ, 2006) a álgebra XQuery é definida em relação à assinatura de seus operadores e regras de inferência, que são utilizadas em sua geração a partir de consultas XQuery. Como em qualquer tipo de linguagem, a especificação da gramática da álgebra XQuery é importante a fim de estruturar as possibilidades de construções da suas expressões. Neste trabalho em específico, a gramática foi utilizada como base para o desenvolvimento das regras de tradução e equivalência.

Tanto as regras de tradução (seção 3.2) quanto as de equivalência (seção 3.3) são contribuições do presente trabalho. As primeiras tem o objetivo de traduzir os operadores da álgebra XQuery para operadores da álgebra relacional, de acordo com os escopos definidos na seção 3.1. Já as regras de equivalência constituem-se de um mecanismo que garante a equivalência entre as expressões em álgebra XQuery e relacional.

A implementação apresentada na seção 5.9 tem o objetivo de demonstrar uma possibilidade de aplicação prática da proposta presente neste trabalho. A fim de facilitar a implementação, foram definidas representações XML para expressões em álgebra XQuery e relacional. Essas representações também são consideradas contribuições deste trabalho. O principal motivo dessas definições em XML é a indisponibilidade de um *parser* para a álgebra XQuery. Dessa forma, é possível utilizar técnicas amplamente conhecidas de manipulação de arquivos XML à curto prazo. O desenvolvimento de um *parser* pode ser considerado como uma solução alternativa à médio prazo.

A proposta presente neste trabalho possui uma publicação referente à fase intermediária do desenvolvimento do QMap. No mês de setembro de 2012 o artigo "*Translating XML Queries into Equivalent SQL Statements"* - que descreve o princípio do mecanismo QMap - foi aceito para publicação na IADIS *International Conference on WWW/Internet*, que possui classificação B2 no Qualis da Capes, e ocorreu entre os dias 18 e 21 de Outubro de 2012 em Madrid, Espanha.

Com a finalização da dissertação, espera-se submeter à um periódico outro artigo que descreva a proposta consolidada, os experimentos realizados e os resultados obtidos.

6.1 Trabalhos futuros

Existem alguns aspectos que podem ser indicados como sendo futuros à este trabalho. Entre eles está a implementação das regras de inferência presentes em (RE; SIMEON; FER-

NANDEZ, 2006) para a geração de expressões em álgebra XQuery específicas para o QMap. Outra implementação futura é em relação às regras de tradução entre álgebra relacional e SQL.

A abrangência do escopo da linguagem XQuery e, consequentemente, das álgebras XQuery e relacional pode ser aumentado. Também podem ser definidas regras de tradução e equivalência para o operador de renomeação (ρ) da álgebra relacional. Além disso, um *parser* para a álgebra XQuery poderia ser desenvolvido.

REFERÊNCIAS

AVELAR, F. T. M. de; BRUM SACCOL, D. de; PIVETA, E. K. An Ontology-based Approach for Storing XML Data Into Relational Databases. In: SEKE. **Anais...** [S.l.: s.n.], 2012. p.438–443.

BRUM SACCOL, D. de; CAMPOS ANDRADE, T. de; PIVETA, E. K. Mapping OWL ontologies to relational schemas. In: IRI. **Anais...** [S.l.: s.n.], 2011. p.71–76.

BRUM SACCOL, D. de et al. Managing application domains in P2P systems. In: IRI'08. **Anais...** [S.l.: s.n.], 2008. p.451–456.

DEHAAN, D. et al. **A Comprehensive XQuery to SQL Translation Using Dynamic Interval Encoding**. 2003.

FAN, W. et al. Query translation from XPath to SQL in the presence of recursive DTDs. **The VLDB Journal**, Secaucus, NJ, USA, v.18, n.4, p.857–883, Aug. 2009.

GALAX. **Galax**: an implementation of xquery. Acessado em Abril/2012, http://www.galaxquery.org/.

CAMPUS (Ed.). **Implementação de sistemas de bancos de dados**. Rio de Janeiro, RJ, Brasil: Campus, 2001.

GRUST, T. Purely Relational FLWORs. In: IN <XIME-P. **Anais...** [S.l.: s.n.], 2005.

GRUST, T. et al. A SQL:1999 code generator for the pathfinder xquery compiler. In: IN PROC. SIGMOD. **Anais...** [S.l.: s.n.], 2007.

KRISHNAMURTHY, R. **XML-to-SQL Query Translation**. 2004. Tese (Doutorado em Ciência da Computação) — University of Wisconsin, Madison.

LAUTERT, L. R. **Implementação de um Simulador de Consultas em Álgebra Relacional**. 2010.

LI, C. et al. Composing XSL Transformations with XML Publishing Views. In: IN SIGMOD. **Anais...** [S.l.: s.n.], 2003.

MICROSOFT. **Sample XML File**. Acessado em Novembro/2012, http://msdn.microsoft.com/pt-br/library/windows/desktop/ms762271(v=vs.85).aspx.

MOLKOVA, L. **Relational Algebra Expression Evaluation**. Acessado em Abril/2012, http://is.muni.cz/th/208197/fi_b/bc_thesis.pdf.

MORO, M. M. et al. XML: some papers in a haystack. **SIGMOD Rec.**, New York, NY, USA, v.38, n.2, p.29–34, Oct. 2009.

NEGRINI, D. **CMAP**: geração e representação de equivalências entre documentos xml, ontologia e modelo relacional. Santa Maria: Curso de Ciência da Computação. Universidade Federal de Santa Maria., 2011.

NEGRINI, D. et al. CMAP: tracking equivalences from xml documents to relational schemas. In: IRI. **Anais...** [S.l.: s.n.], 2012. p.332–339.

PATHFINDER. **Pathfinder**: a purely relational xquery processor. Acessado em Setembro/2012, http://db.inf.uni-tuebingen.de/research/pathfinder/.

RE, C.; SIMEON, J.; FERNANDEZ, M. A Complete and Efficient Algebraic Compiler for XQuery. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 22., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2006. p.14–. (ICDE '06).

SABRY, A.; FELLEISEN, M. Reasoning about programs in continuation-passing style. In: ACM CONFERENCE ON LISP AND FUNCTIONAL PROGRAMMING, 1992., New York, NY, USA. **Proceedings...** ACM, 1992. p.288–298. (LFP '92).

ELSEVIER (Ed.). **Sistema de Banco de Dados**. 5th.ed. Boston, MA, USA: Elsevier, 2006.

W3C. **XML Query Use Cases**. Acessado em Novembro/2012, http://www.w3.org/TR/xquery-use-cases/.

W3C. **XQuery 1.0**: an xml query language (second edition). Acessado em Abril/2011, http://www.w3.org/TR/xquery/.

W3C. **XML Path Language (XPath) 2.0 (Second Edition)**. Acessado em Abril/2011, http://www.w3.org/TR/xpath20/.

W3C. **XQuery 1.0**: an xml query language (second edition). Acessado em Julho/2012, http://www.w3.org/TR/xquery/.

WALMSLEY, P. **XQuery**. [S.l.]: O'Reilly Media, Inc., 2007.

WANG, S.; WANG, L.; RUNDENSTEINER, E. A. Isolating order semantics in order-sensitive xquery-to-SQL translation. In: BRITISH NATIONAL CONFERENCE ON DATABASES, 24., Berlin, Heidelberg. **Proceedings...** Springer-Verlag, 2007. p.147–159. (BNCOD'07).

ZHANG, X. et al. **XAT**: xml algebra for the rainbow system. [S.l.: s.n.], 2002.

ZHANG, X. et al. Rainbow: multi-xquery optimization using materialized xml views. In: IN SIGMOD DEMO. **Anais...** ACM Press, 2003. p.671.

APÊNDICES

APÊNDICE A – Estrutura dos documentos XML que representam álgebra XQuery

Este é o documento que define a estrutura dos documentos XML utilizados para representar as expressões em álgebra XQuery.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD generated by XMLSpy v2013 (x64) (http://www.altova.com)-->
<!ELEMENT xquery_algebra ((operator))>
<!ELEMENT sub_operator ((sub_operator | input_operator | static_param)*)>
<!ATTLIST sub_operator
        name CDATA #IMPLIED
        op (where | join) #IMPLIED
>
<!ELEMENT static_param (#PCDATA | input_operator)*>
<!ATTLIST static_param
        name CDATA #IMPLIED
>
<!ELEMENT operator ((sub_operator, input_operator))>
<!ATTLIST operator
        name CDATA #FIXED "Map"
>
<!ELEMENT input_operator ((sub_operator | input_operator | static_param)*)>
<!ATTLIST input_operator
        name CDATA #IMPLIED
        op (let | for) #IMPLIED
>
```

APÊNDICE B – Estrutura dos documentos XML que representam álgebra relacional

Este é o documento que define a estrutura dos documentos XML utilizados para representar as expressões em álgebra relacional.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--DTD generated by XMLSpy v2013 (x64) (http://www.altova.com)-->
<!ELEMENT table_name (#PCDATA)>
<!ELEMENT table (#PCDATA)>
<!ELEMENT selection ((condlist, logical_op, condlist))>
<!ELEMENT rename ((table_name, columnlist))>
<!ELEMENT relations ((table, (relation_op, table)*))>
<!ELEMENT relational_algebra ((expression, (relation_op, expression)*))>
<!ELEMENT relation_op ((name, condlist?))>
<!ELEMENT projection ((columnlist))>
<!ELEMENT orderlist ((column+, order))>
<!ELEMENT order_by ((orderlist))>
<!ELEMENT order (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT logical_op (#PCDATA)>
<!ELEMENT group ((columnlist, functionlist))>
<!ELEMENT functionlist ((function, column, as?)+)>
<!ELEMENT function (#PCDATA)>
<!ELEMENT expression (#PCDATA | relations | selection | projection | group | order_by | rename)*>
<!ELEMENT data (#PCDATA)>
<!ELEMENT condlist ((column, comp_op, (column | data)))>
<!ELEMENT comp_op (#PCDATA)>
<!ELEMENT columnlist ((column+))>
<!ELEMENT column (#PCDATA)>
<!ATTLIST column
        table CDATA #IMPLIED
>
<!ELEMENT as (#PCDATA)>
```

APÊNDICE C – Documento `bib.xml`

A figura C.1 ilustra o documento `bib.xml`.

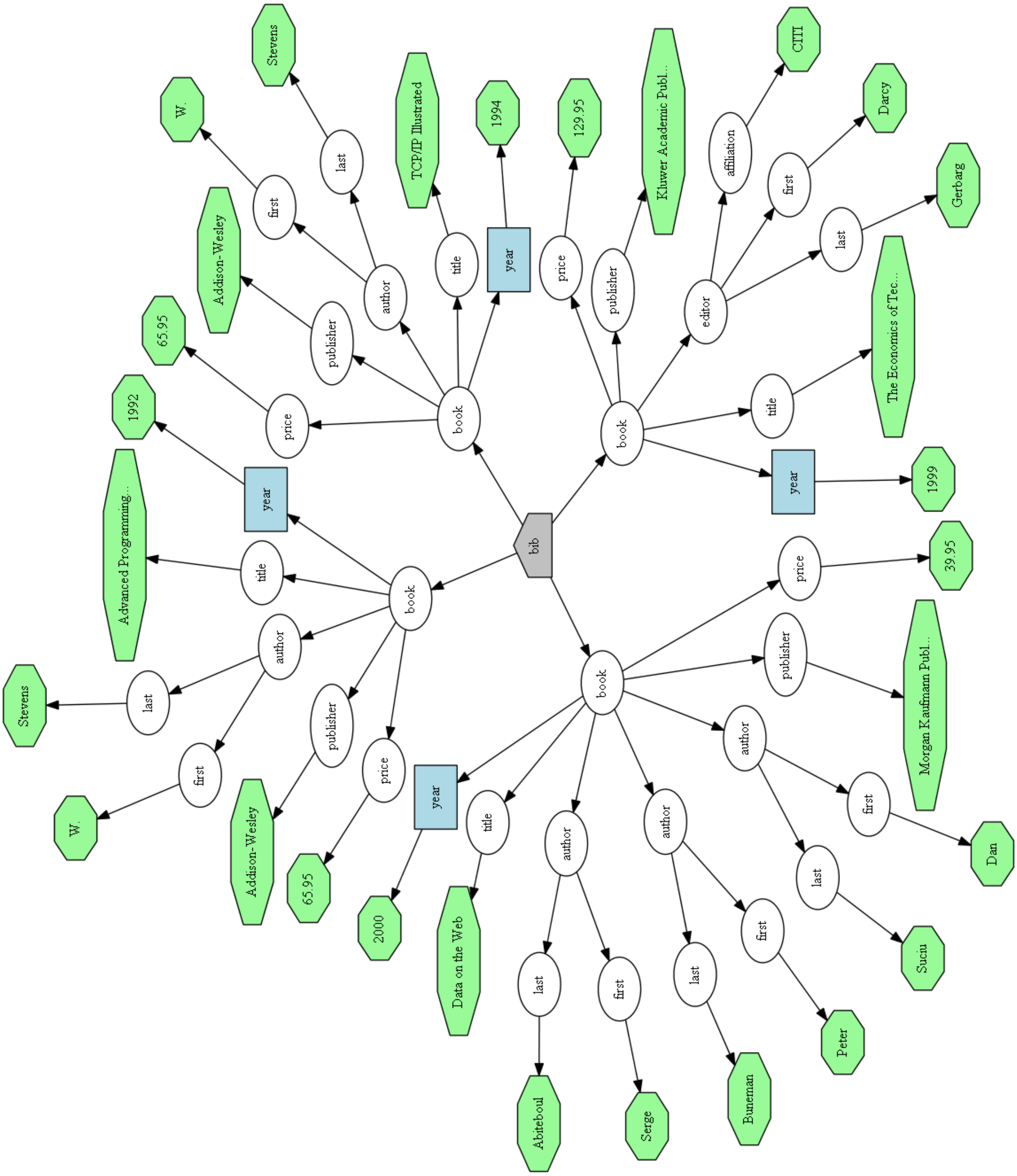

Figura C.1 – Documento `bib.xml`

APÊNDICE D – Documento `book_p1.xml`

A figura D.1 ilustra o documento `book_p1.xml`.

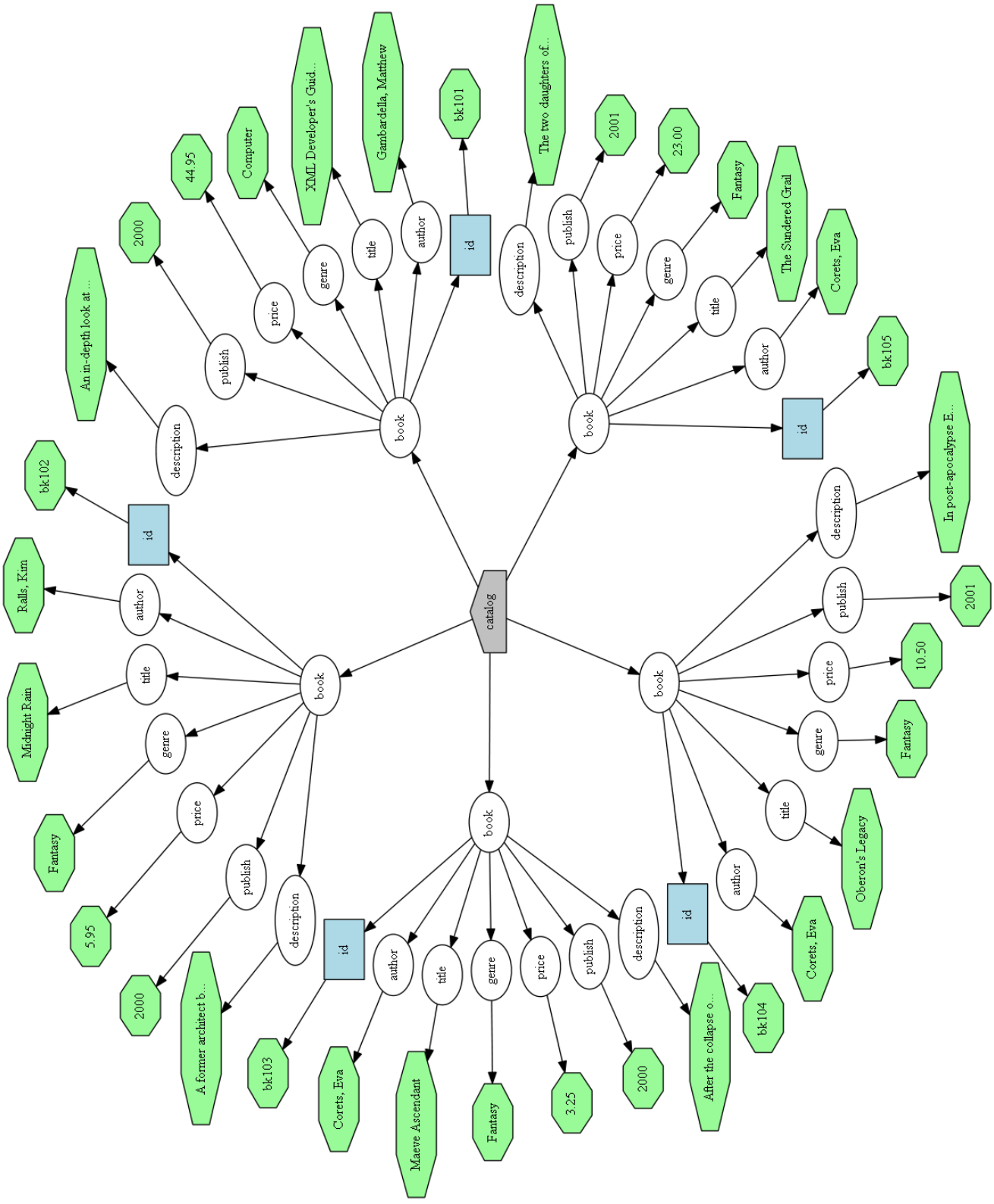

Figura D.1 – Documento `book_p1.xml`

APÊNDICE E – Documento `book_p2.xml`

A figura E.1 ilustra parte do documento `book_p2.xml`.

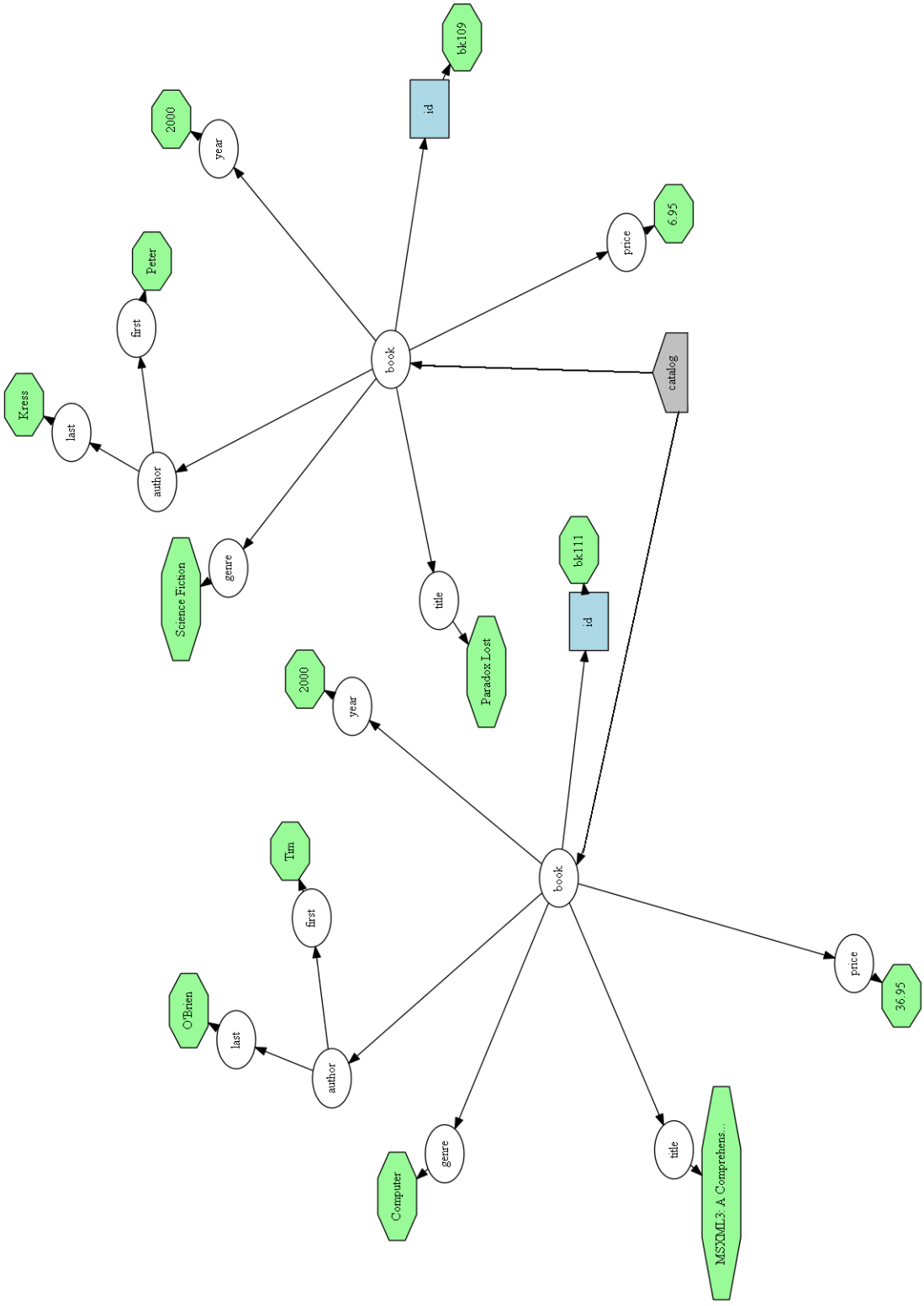

Figura E.1 – Documento `book_p2.xml`

APÊNDICE F – Classes implementadas

A figura F.1 ilustra a classe *Variables*.

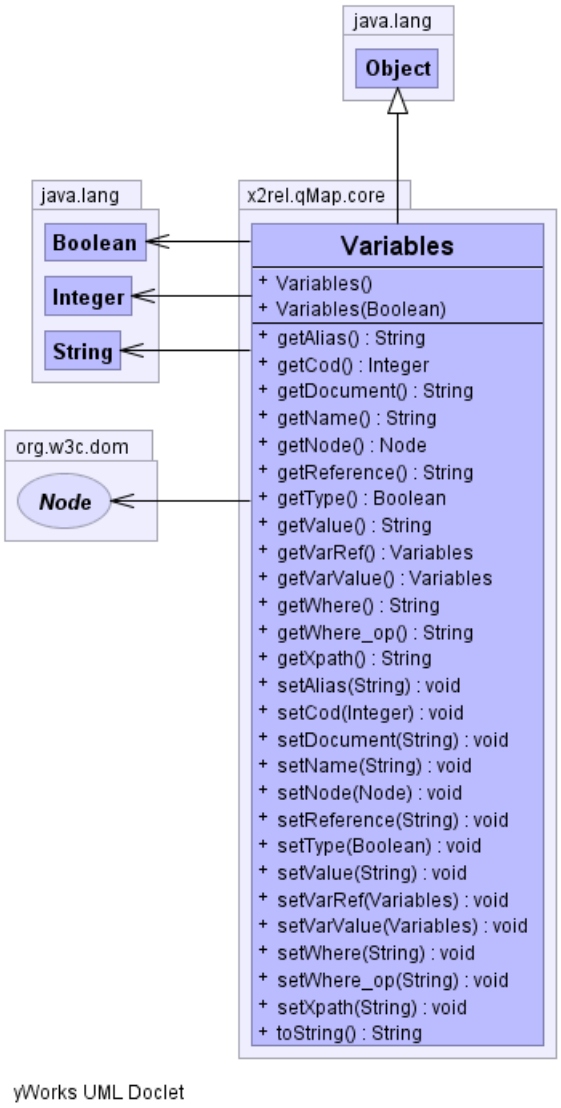

Figura F.1 – Classe *Variables*

A figura F.2 ilustra a classe *Core*.

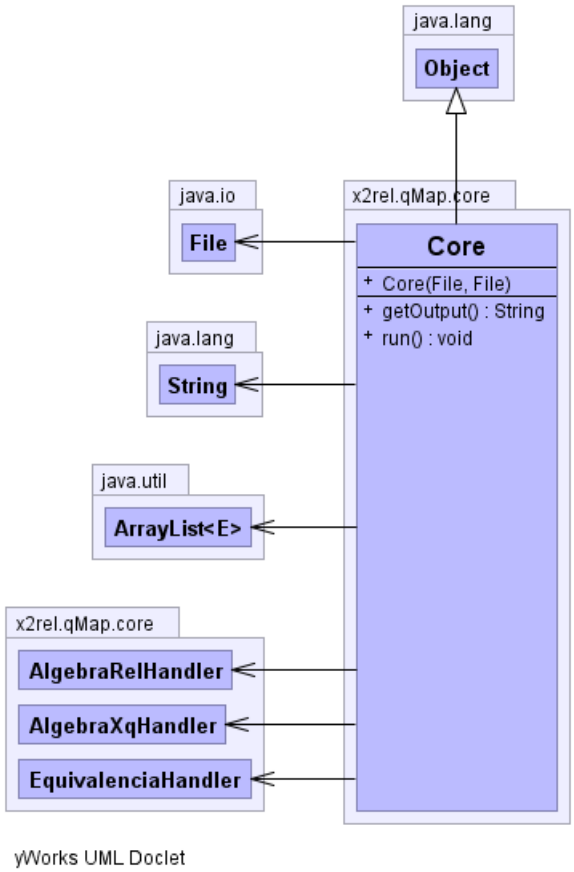

Figura F.2 – Classe *Core*

A figura F.3 ilustra a classe *XmlHandler*.

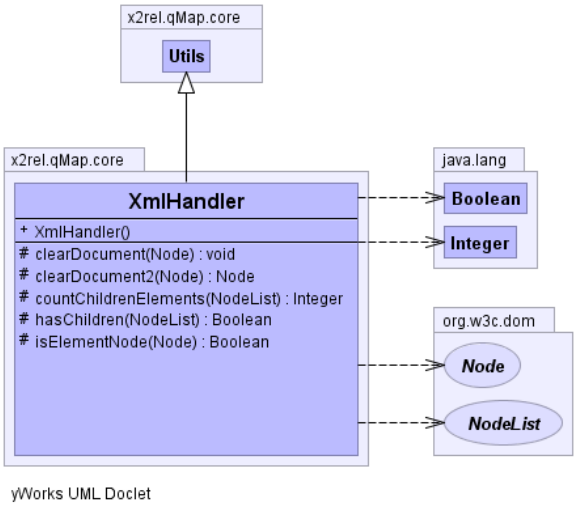

Figura F.3 – Classe *XmlHandler*

A figura F.4 ilustra a classe *Utils*.

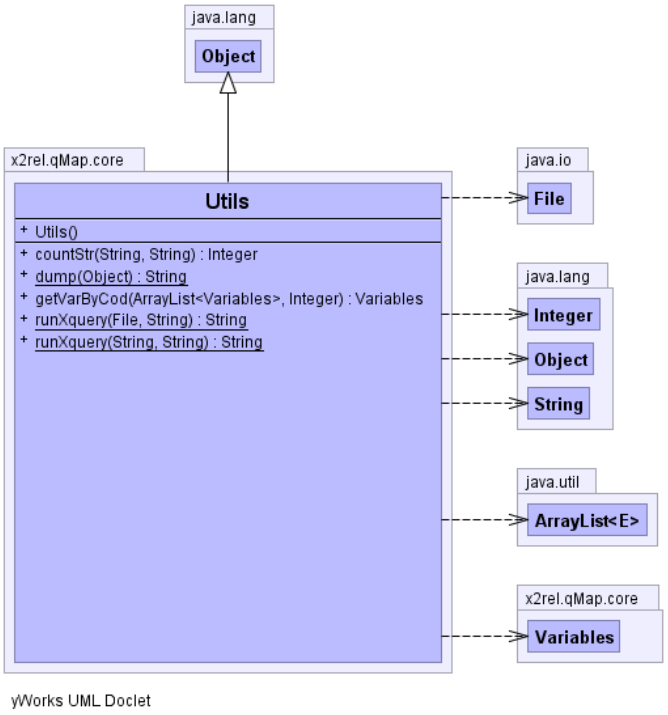

Figura F.4 – Classe *Utils*

A figura F.5 ilustra a classe *Equivalencias*.

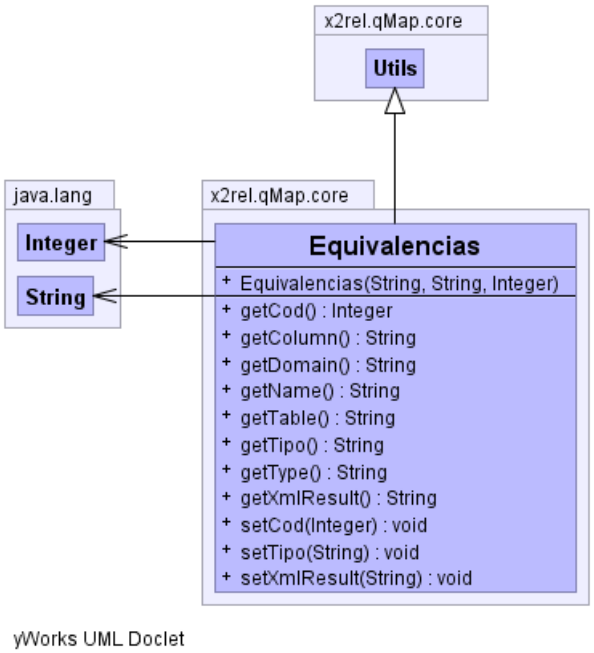

Figura F.5 – Classe *Equivalencias*

A figura F.6 ilustra a classe *AlgebraRelHandler*.

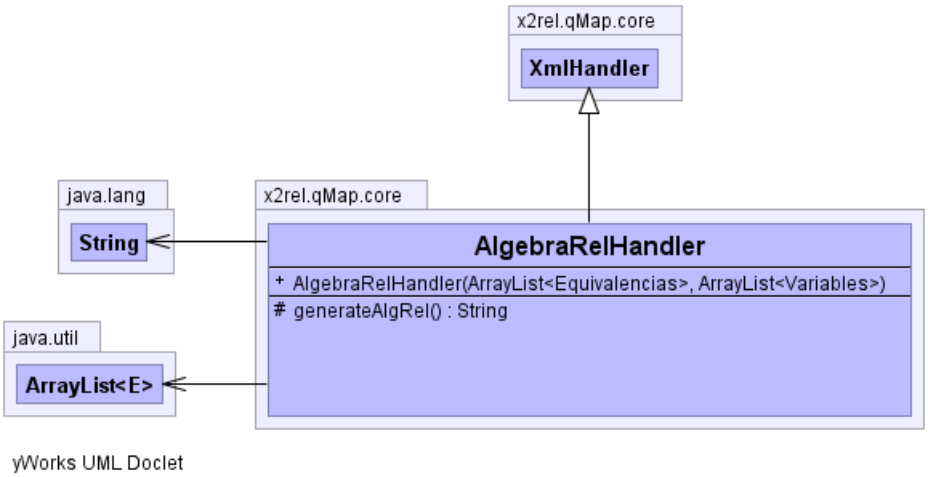

Figura F.6 – Classe *AlgebraRelHandler*

A figura F.7 ilustra a classe *AlgebraXqHandler*.

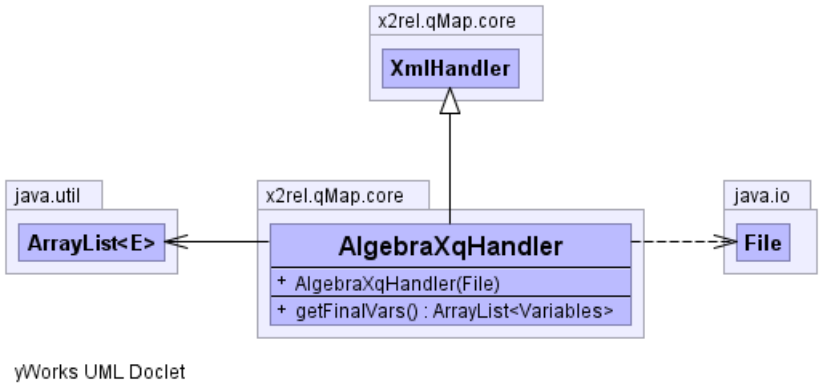

Figura F.7 – Classe *AlgebraXqHandler*

A figura F.8 ilustra a classe *VariablesHandler*.

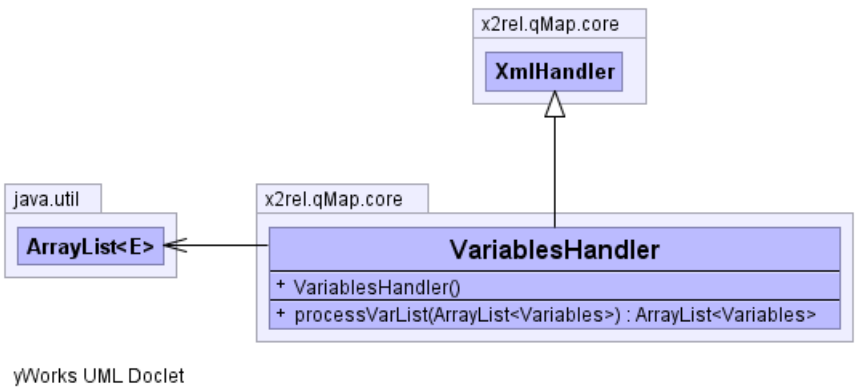

Figura F.8 – Classe *VariablesHandler*

A figura F.9 ilustra a classe *EquivalenciaHandler*.

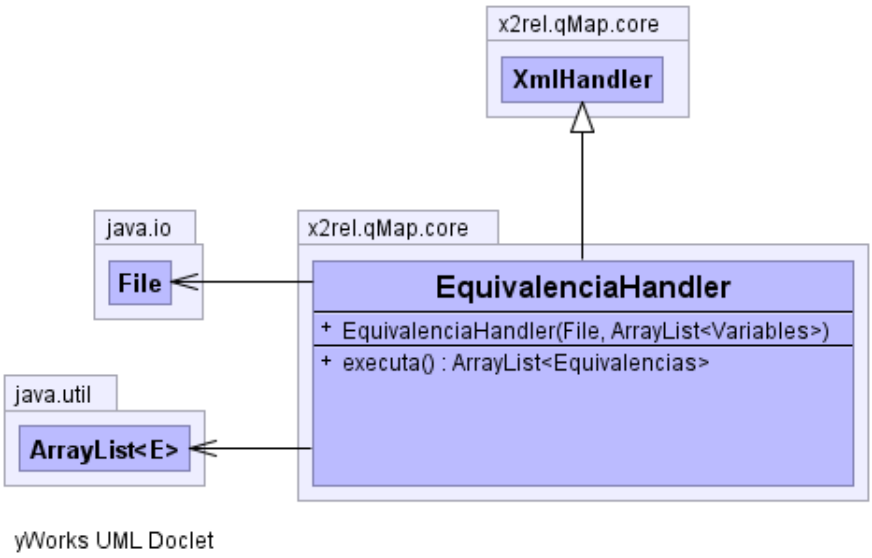

Figura F.9 – Classe *EquivalenciaHandler*

A figura F.10 ilustra a classe *Logging*.

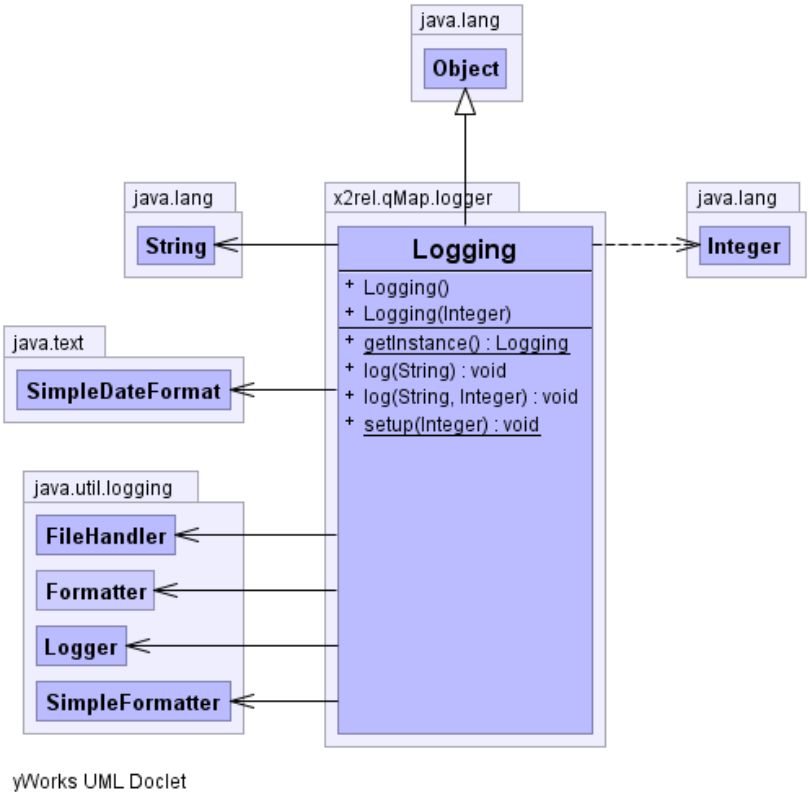

Figura F.10 – Classe *Logging*

A figura F.11 ilustra a classe *MainWindow*.

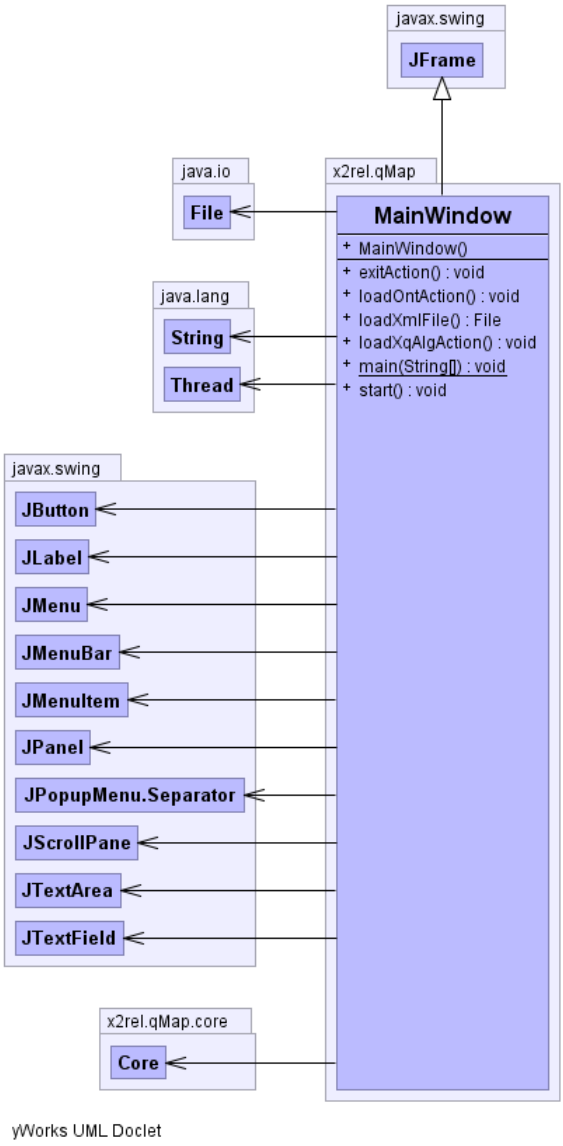

Figura F.11 – Classe *MainWindow*

ANEXOS

ANEXO A – Documento book.xml

Este é o documento `book.xml`, que descreve informações de livros, tais como autor, título, gênero, preço, data e descrição.

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications
    with XML.</description>
  </book>
  <book id="bk102">
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies,
    an evil sorceress, and her own childhood to become queen
    of the world.</description>
  </book>
  <book id="bk103">
    <author>Corets, Eva</author>
    <title>Maeve Ascendant</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-11-17</publish_date>
    <description>After the collapse of a nanotechnology
    society in England, the young survivors lay the
    foundation for a new society.</description>
  </book>
  <book id="bk104">
    <author>Corets, Eva</author>
    <title>Oberon's Legacy</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2001-03-10</publish_date>
    <description>In post-apocalypse England, the mysterious
    agent known only as Oberon helps to create a new life
    for the inhabitants of London. Sequel to Maeve
    Ascendant.</description>
  </book>
  <book id="bk105">
    <author>Corets, Eva</author>
    <title>The Sundered Grail</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2001-09-10</publish_date>
    <description>The two daughters of Maeve, half-sisters,
    battle one another for control of England. Sequel to
    Oberon's Legacy.</description>
  </book>
  <book id="bk106">
    <author>Randall, Cynthia</author>
    <title>Lover Birds</title>
    <genre>Romance</genre>
    <price>4.95</price>
    <publish_date>2000-09-02</publish_date>
    <description>When Carla meets Paul at an ornithology
    conference, tempers fly as feathers get ruffled.</description>
  </book>
  <book id="bk107">
```

```
    <author>Thurman, Paula</author>
    <title>Splish Splash</title>
    <genre>Romance</genre>
    <price>4.95</price>
    <publish_date>2000-11-02</publish_date>
    <description>A deep sea diver finds true love twenty
    thousand leagues beneath the sea.</description>
  </book>
  <book id="bk108">
    <author>Knorr, Stefan</author>
    <title>Creepy Crawlies</title>
    <genre>Horror</genre>
    <price>4.95</price>
    <publish_date>2000-12-06</publish_date>
    <description>An anthology of horror stories about roaches,
    centipedes, scorpions  and other insects.</description>
  </book>
  <book id="bk109">
    <author>Kress, Peter</author>
    <title>Paradox Lost</title>
    <genre>Science Fiction</genre>
    <price>6.95</price>
    <publish_date>2000-11-02</publish_date>
    <description>After an inadvertant trip through a Heisenberg
    Uncertainty Device, James Salway discovers the problems
    of being quantum.</description>
  </book>
  <book id="bk110">
    <author>O'Brien, Tim</author>
    <title>Microsoft .NET: The Programming Bible</title>
    <genre>Computer</genre>
    <price>36.95</price>
    <publish_date>2000-12-09</publish_date>
    <description>Microsoft's .NET initiative is explored in
    detail in this deep programmer's reference.</description>
  </book>
  <book id="bk111">
    <author>O'Brien, Tim</author>
    <title>MSXML3: A Comprehensive Guide</title>
    <genre>Computer</genre>
    <price>36.95</price>
    <publish_date>2000-12-01</publish_date>
    <description>The Microsoft MSXML3 parser is covered in
    detail, with attention to XML DOM interfaces, XSLT processing,
    SAX and more.</description>
  </book>
  <book id="bk112">
    <author>Galos, Mike</author>
    <title>Visual Studio 7: A Comprehensive Guide</title>
    <genre>Computer</genre>
    <price>49.95</price>
    <publish_date>2001-04-16</publish_date>
    <description>Microsoft Visual Studio 7 is explored in depth,
    looking at how Visual Basic, Visual C++, C#, and ASP+ are
    integrated into a comprehensive development
    environment.</description>
  </book>
</catalog>
```