XML NATIVO COMO ALTERNATIVA AO MODELO RELACIONAL CLÁSSICO

Caio de Lima Mello[1], **Daniel Mendes Barbosa**[1]

[1]Instituto de Ciências Exatas e Tecnológicas – Universidade Federal de Viçosa(UFV)
Campus UFV-Florestal – Florestal – MG – Brasil

caio.mello@ufv.br, danielmb@gmail.com

Resumo. *Com a crescente evolução e adoção de tecnologias computacionais no cotidiano, é possível perceber um visível aumento na relação entre usuários de dados. Criou-se então uma necessidade de resolver a volatilidade de dados quaisquer tornando-os persistentes e ao mesmo tempo recuperados de forma integra e trivial. Neste contexto este trabalho pretende explorar a viabilidade de Bancos de Dados XML (Extended Markup Language) Nativos (NXMLDB) como alternativa a Bancos de Dados Relacionais(RMDB) através de uma abordagem empírica e comparativa. A abordagem proposta consiste numa série de experimentos dado dois bancos (XML e SQL) e um roteiro de operações cujas métricas são em função do número de tuplas processadas versus o tempo necessário para processá-las.*

1. Introdução

Entre as muitas alternativas propostas ao modelo relacional clássico tem-se o modelo por armazenamento XML nativo. Dito *Extensible Markup Language*, o formato descreve uma classe documentos que em sua estrutura descrevem como se comportam e a própria estrutura de dados que eles descrevem [Bray et al. 2006]. O uso de documentos XML é altamente consolidado durante exportação e importação de dados para aplicações externas, então é interessante pensar nas implicações em armazenar dados em um banco de dados nativamente XML.

Atualmente parte substancial dos bancos de dados utilizam o modelo relacional [Phillip J. Pratt 2014]. Este modelo de armazenamento de dados tem cumprido as exigências de usuários e empresas desde sua criação, mas diante da crescente evolução dos meios de comunicação virtual, popularização de computadores pessoais e dispositivos móveis, identificou-se uma crescente demanda por novos meios de se armazenar informação. Os problemas anteriormente citados estão relacionados à quantidade massiva de dados gerados, podendo-se identificar um aumento substancial na complexidade e relação entre eles, o que torna a tarefa de buscar por alternativas de armazenamento algo não trivial, já que não se trata somente de espaço físico.

O objetivo deste trabalho é descrever e contextualizar o uso de *RMDB* e *NXMLDB* para em seguida realizar experimentos comparativos, determinando as vantagens e desvantagens do uso destes em cada cenário. As diferenças entre estes modelos de banco de dados e as métricas e infraestrutura dos experimentos serão devidamente estendidos nas seções 2 e 3 respectivamente. A seção 4 apresenta os resultados obtidos e a seção 5 as conclusões do trabalho.

2. Contextualizando os Modelos de Banco

Antes que se possa fazer uma comparação direta entre os modelos de banco e suas consequentes formas de armazenamentos deve-se em primeiro lugar explicitar em detalhes do que se trata cada um destes modelos. As subseções 2.1 e 2.2 fazem uma breve descrição dos mesmos e de suas principais características.

2.1. Bancos de Dados Relacionais

Como descrito e formalizado primeiro por [Codd 1985], existe uma série de características típicas que um banco de dados deve possuir para que seja considerado relacional. Estas características apontam para uma topologia estruturada por uma série de tabelas indexadas. Estas tabelas são chamadas relações, e representam entidades do mundo real e relacionamentos entre estas entidades.

Entidades são em termos gerais abstrações de objetos do mundo real ou de um determinado domínio de negócio [Beynon-Davies 2004] enquanto relacionamentos definem a natureza com a qual as entidades se relacionam. Fatores como grau numérico de relação (quantos para quantos), participação e dependência entre tabelas são observados nestes relacionamentos.

A Figura 1 demonstra explicitamente como funciona esta propriedade relacional entre entidades por meio de uma consulta em SQL (*Structured Query Language*) que explora o relacionamento entre as tabelas serviço e rota. No caso a consulta relaciona serviço e rota e explicita que deve ser recuperada uma relação de dados onde a identificação é 24.

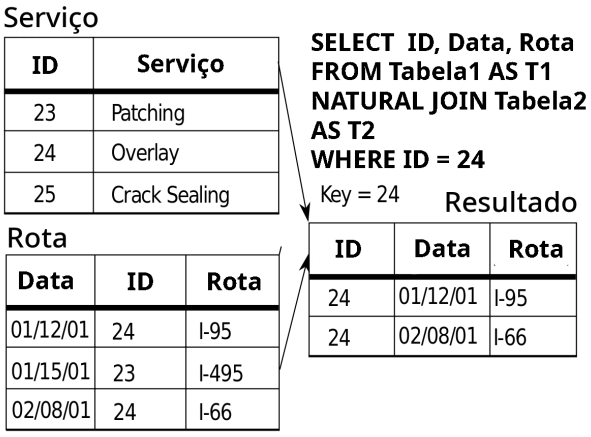

Figura 1. Consulta básica demonstrando a propriedade relacional entre entidades

2.2. XML e Bancos XML Nativos

XML é uma linguagem construída a partir de um conceito híbrido, onde as unidades básicas da linguagem são ditas *entidades* que contém tanto dados quanto metadados [Bray et al. 2006], isto é, dados que descrevem a estrutura e natureza dos dados no documento.

Uma característica interessante desta linguagem é que ela e é extensível e permite ao usuário a definição de suas próprias *tags*. *Tags* são estruturas típicas de linguagens de marcação (de conteudo híbrido) e consistem em delimitadores que descrevem a natureza daquilo que eles contém. Uma *tag* $< passo >$ num manual de instruções poderia muito bem representar uma ação necessária para cumprir determinada função.

Tipicamente, as *tags* são organizadas de forma hierárquica num arquivo XML tal qual uma arvore n-ária. Essa propriedade hierárquica é vista até na forma em que consultas XML são feitas, partindo de um caminho e buscando no sub-conjunto contido neste [News 1999]. No exemplo da Figura 2 uma consulta típica partiria da raiz, daí em diante há a opção de explorar qualquer uma das sub-arvores ou então restringir a consulta a uma em específico.

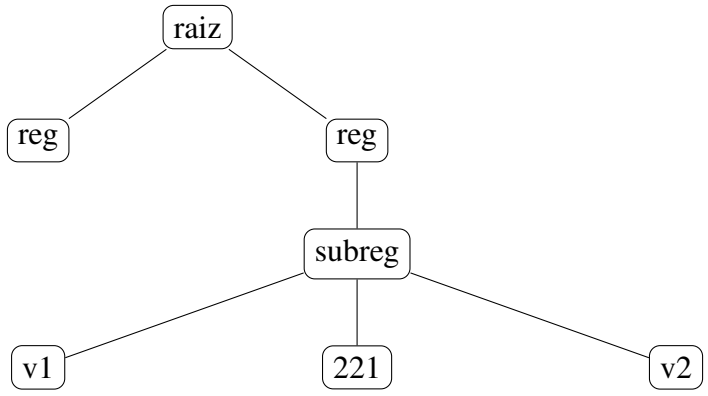

Figura 2. Uma topologia básica de arquivo XML estruturado

Uma transcrição do modelo acima em XML formatado pode ser feito tal qual o trecho de código 1.

Trecho de código 1. Exemplo de esquema XML Básico

```
<raiz>
   <reg>
      </subreg><subreg>
   </reg>
   <reg>
      <subreg>
         <c1>v1</v1><c2>221</c2><c3>v2</c3>
      </subreg>
   </reg>
</raiz>
```

2.2.1. XPath e XQuery

Assim como em qualquer esquema de banco de dados, a linguagem XML possui duas especificações principais para navegar em seus registros. Primeiramente tem-se o *XML Path Language*, conhecido por XPath, que foi idealizado considerando a estrutura de árvore em documentos XML[Bergeron 2000], portanto foi baseado em navegação nodo-a-nodo considerando uma certa condição. A título de exemplo, pode-se utilizar a Figura 2 e buscar especificamente por todo registro nulo com $/raiz[reg =''] / reg$.

Apesar de idealizado posteriormente, a *XML Query Language* tem o *XPath* como um subconjunto de suas operações, sendo que ambas foram desenvolvidas pelo mesmo conjunto de autores: O XSL Working Group. A linguagem XQuery, como é chamada frequentemente, é uma linguagem funcional, orientada a expressões e possui um sistema de tipagens relativamente simples [Kilpeläinen 2012]. Todas as expressões de consulta são construidas de forma a avaliar conjuntos de itens ou valores atômicos.

Um exemplo de como uma consulta por registros vazios no exemplo da Figura 2 funcionaria em XQuery seria tal qual o Código 2. Pode-se inclusive notar claros trechos de XPath neste exemplo.

Trecho de código 2. Consulta XML por registros vazios

```
for $x in doc("esquema.xml")/raiz/reg
where $x=''
return $x;
```

2.2.2. Modelos de Banco XML Nativo

De acordo com [XMLDBInitiative 2013] pode-se definir um banco de dados XML Nativo a partir de três categorias: *Native XML Database*, *XML Enabled Database* e *Hybrid XML Database*

O primeiro define um modelo lógico para um documento XML da modelagem dos dados presentes nas estruturas. Para que um DB seja considerado NXMLDB ele deve obedecer a uma série de exigências mínimas: Em primeira instância deve conter elementos, atributos, metadados e ordem do documento, sendo que este documento deverá conter uma unidade fundamental de armazenamento lógico tal qual um RMDB possui uma tupla em suas tabelas. A principal característica deste tipo de Banco XML é justamente não estipular um modelo de armazenamento físico, sendo relacional, hierárquico, orientado a objetos e até indexado.

O segundo tipo, conhecido por *XML Enabled Database*, trata-se de um banco de dados normal porém munido de uma camada de mapeamento XML. Esta camada trata de todas as informações armazenadas e sua recuperação. Dados mapeados neste BD normalmente são feitos em função de formatos específicos e os metadados XML originais podem ou não se perder no meio disto tudo, isto é, não há garantia de recuperação dos dados em seu formato XML original.

O terceiro e último tipo é uma abordagem híbrida que normalmente pende mais para as características de um dos modelos supracitados.

2.3. Trabalhos Relacionados

Existe uma série de trabalhos recentes relativos a interação entre as tecnologias relacionais e documentos XML. Alguns deles são:

- **SISTEMA GERENCIADOR DE BANCO DE DADOS: SGBD eXist XML** - Um exemplo de sistema XML Nativo em desenvolvimento criado a partir da necessidade de se exportar dados para diversos outros tipos, em especial se tratando de sistemas web.
- **UMA FERRAMENTA PARA CARGA DE BANCOS DE DADOS RELACIONAIS A PARTIR DE FONTES DE DADOS XML** - Uma aplicação prática que tira proveito da flexibilidade e rapidez de BDXMLs, construindo bancos relacionais (os mais frequentemente usados) a partir de um BDXML nativo.
- **XMAP: MAPEAMENTO E ARMAZENAMENTO DE DADOS XML EM BANCOS DE DADOS RELACIONAIS** - Outra aplicação que visa mapear dados de XML para relacional, o que é especialmente interessante considerando a complexidade da hierarquia de alguns documentos indexados por meio de XML.

3. Metodologia

A metodologia utilizada neste trabalho é puramente empírica, derivando suas conclusões a partir de uma série de experimentos e das métricas levantadas a partir destes. Os experimentos contemplam dois cenários: SQL e XML, e evidentemente estes cenários fazem uso das bases de dados das quais derivam seus nomes.

Em especial, o cenário XML possui duas variantes: Indentado e Compacto; tipicamente arquivos XML são formatados de forma que fique evidente a hierarquia de suas *tags* através de indentação, no entanto podem ser representados por um único *stream* de caracteres não formatados desde que fique claro onde começa e termina cada registro e que a raiz da relação esteja uma linha acima da *stream.*

Quanto às especificações destes experimentos, além dos cenários é preciso que se especifique o contexto em qual se encaixam; explicitando restrições, universo e infraestrutura, estas sendo expandidas na subseções 3.1 e 3.2.

3.1. Ferramentas

Faz se necessária a utilização de alguns *front-ends* para cada base de dados a fim de facilitar a coleta de informação das operações a serem realizadas. Infraestrutura mínima e disponibilidade foram os dois requisitos principais na escolha das ferramentas que resultaram na escolha do *HeidiSQL* para o *front-end* SQL e do *BaseX* para o *front-end* XML Nativo.

No caso específico do *BaseX* ele fornece tanto a infraestrutura XML Nativa quanto o *front-end* para as consultas. Todos os experimentos foram executados considerando a arquitetura e especificação mostradas na tabela 1.

Sistema Operacional	Arch GNU/Linux
Processador	Intel Core i3-4030U @ 1.90GHz
RAM	8 GB

Tabela 1. Especificações do sistema

3.2. Decisões Preliminares e Infraestrutura Experimental

Observado o que foi dito nas seções 2.1 e 2.2, é possível extrair que traduções de SQL para XML não são uma tarefa trivial, já que a forma como suas entidades são organizadas é fundamentalmente diferente. Optou-se então por utilizar somente uma tabela/estrutura única juntamente a uma quantidade de tuplas suficientemente densas.

Para determinar a densidade ideal de tuplas, utilizou-se uma abordagem empírica que consiste na geração de dados por meio de *scripts* verificando quais quantidades gerariam resultados visíveis e que ainda fossem gerados em tempo razoável. Os dados de teste, ditos *dummy data*, foram gerados a partir de *scripts* triviais de manipulação de strings na linguagem Python.

Como resultado direto destes *scripts* geradores tem-se os trechos 3, 4 e 5. É relevante notar que os códigos gerados tanto constroem as estruturas como inserem as tuplas.

Primeiramente temos o resultado do *script* para o cenário *SQL*, em MySQL explicitando todas as palavras chaves nas operações de inserção (INSERT), já que existe a possibilidade de omitir algumas delas.

Trecho de código 3. Seção de código SQL com construção da tabela inclusa

```
CREATE TABLE IF NOT EXISTS tbl (
   `col1` INT,`col2` VARCHAR (16),`col3` INT,`col4` VARCHAR (16)
);
INSERT INTO tbl (col1, col2, col3, col4)VALUES (1882,
   'JyrNmlHrVHrbBrSu', 592, 'dVxYEQuujxpUDtDW');
INSERT INTO tbl (col1, col2, col3, col4)VALUES (1166,
   'yLJAgNaIiEYIPTGK', 1002, 'FEGkrsZLZdTHkWHx');
INSERT INTO tbl (col1, col2, col3, col4)VALUES (8, 'SCvQWBtifRvyfTYb',
   1641, 'PdRfKplRZOKTcKjr');
INSERT INTO tbl (col1, col2, col3, col4)VALUES (1755,
   'duiGKbrDhgaEOrpq', 1073, 'LYnGTmDZPgiuqsUW');
```

Os trechos 4 e 5 abaixo foram extraídos do mesmo código em duas variantes porém se referem a seções diferentes do código fonte. É claramente visível a diferença entre as variantes compacta e indentada do cenário XML como pode-se observar.

Trecho de código 4. Seção inicial de código XML (puro)

```
<?xml version='1.0' encoding='UTF-8'?>
<raiz>
<record><c1>1848</c1><c2>YRPwJmKNYIFeQfpK</c2>
<c3>294</c3><c4>zuaBbJzjABaRwPvu</c4>
<record><record><c1>793</c1><c2>BCtNXrzgAEfHpPOU</c2>
<c3>389</c3><c4>yQDPoFmWhacCcMx</c4>
</record><record><c1>62</c1><c2>eBTammrbJBlkPnFM</c2>
<c3>1305</c3><c4>pSxDJMSjwoIYLmr</c4>
</record><record><c1>882</c1><c2>LtWgCRxWcDoYLquV</c2>
<c3>1612</c3><c4>ffZIbuchTylHYYdo</c4>
```

Trecho de código 5. Seção de código XML (Formatado)

```
<record>
     <c1>1856</c1>
     <c2>zVFTvpuURNZzLYVk</c2>
     <c3>1711</c3>
     <c4>wZpzEqkVjxOXzepn</c4>
</record>
<record>
     <c1>1368</c1>
     <c2>GJrksbipTejDSLwP</c2>
     <c3>1422</c3>
     <c4>TazjkJCjNdwwkTOs</c4>
</record>
```

Após uma série de testes de geração aleatória e inserção nas bases de dados, concluiu-se que tuplas acima da ordem de 10^5 costumam apresentar elevado custo temporal, visto que o o tempo de inserção da maior ordem no cenário *SQL* demonstrou um custo temporal superior a 15 horas. Estipulou se então que 10^5 tuplas é um teto razoável. Sendo o teto factível da ordem de 10^5 a escala de progressão é dada: 1000,5000,10000,50000 e 100000 tuplas.

Quanto às características das estruturas, definiu-se um modelo constituído de quatro campos, sendo estes numéricos e de caracteres posicionados intercaladamente. Quanto à natureza das tuplas inseridas neste modelo, um intervalo arbitrário $[0, 2000]$ foi estipulado para as colunas numéricas. Já as colunas de caractere possuem 16 caracteres não *case-sensitive*.

Como resultado destas especificações a topologia mostrada na Figura 3 foi obtida. A topologia especificada servirá de infraestrutura geral para as bases de dados sendo então a forma de todas as tuplas inseridas.

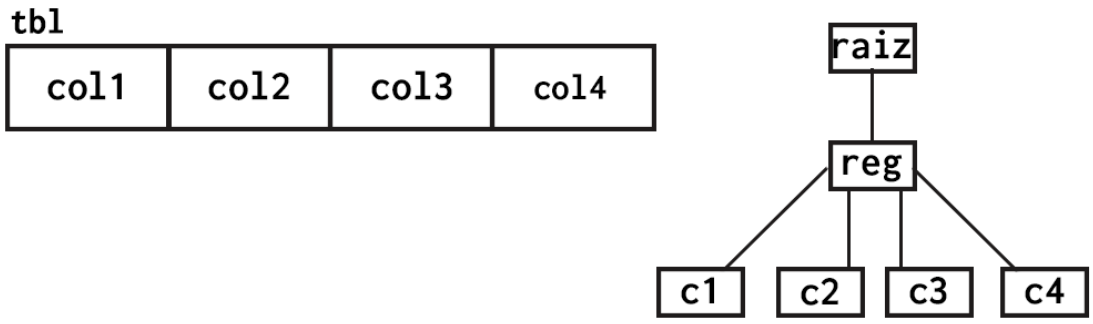

Figura 3. Infraestrutura geral das tabelas de cada base

4. Resultados

Considerando a infraestrutura previamente explicada, para cada cenário foram extraídas métricas a partir das seguintes operações: **Geração de *Dummy Data***, **Inserção e**

Construção das Bases, **Consulta** e *Reconstrução para XML.*

A operação de **Geração de *Dummy Data*** é justamente a geração dos *scripts* que irão construir as bases para logo em seguida inserir os dados. A métrica extraída a partir desta operação é justamente o tamanho médio dos arquivos gerados.

Para as operações seguintes, as métricas extraídas são em função do tempo médio que cada uma leva em seu processamento dado um cenário e escala de tuplas. Quanto às suas funções, a operação de *Inserção e Construção* consiste na execução dos dos *scripts* gerados, construindo a infraestrutura da base para logo em seguida inserir os dados, enquanto a **Consulta** é a recuperação dos dados de um BD dado um filtro ou especificação. Por fim, a *Reconstrução para XML* é a extração dos resultados de uma consulta para um arquivo XML.

4.1. Análise dos Resultados

Dadas as decisões e infraestrutura explicitadas nas seções anteriores pode-se então prosseguir com a análise dos resultados que foram gerados a partir das operações. Em se tratando do tamanho dos arquivos gerados pelos *scripts* de *dummy data* é possível extrair as seguintes métricas:

	1000	5000	10000	50000	100000
SQL	102kB	509kB	1000kB	5100kB	10200kB
XML Compacto	91kB	459kB	919kB	4600kB	9200kB
XML Identado	123kB	619kB	1200kB	6200kB	12400kB

Tabela 2. Tamanho médio dos arquivos gerados

Considerando que XML Compacto é somente uma *stream* de caracteres, é evidente que ele apresenta menor média de tamanho, no entanto não foi possível prever com exatidão que os arquivos XML da variante Indentada iriam se apresentar mais verbosos que SQL padrão para um cenário tão reduzido.

No que se refere às operações de construção e inserção dos dados na base notam-se os seguintes resultados na Tabela 3 e Figura 4:

	1000	5000	10000	50000	100000
SQL	51832.0	252003.0	526123.0	2584652.0	5520032.0
XML Compacto	45.8	98.4	155.0	772.5	1432.2
XML Identado	28.8	81.6	151.1	743.1	1523.1

Tabela 3. Tempo médio (em milissegundos) para construção e inserção de dados

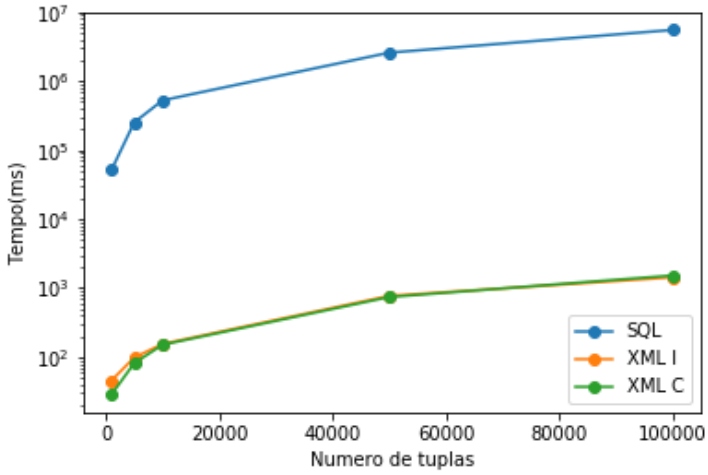

Figura 4. Relação dos tempos médio para construção e inserção

Por estes resultados, o carregamento de arquivos *XML* parece ser muito mais rápido. No entanto a variante Indentada demonstra ser custosa para números mais elevados de tuplas. Apesar dos resultados relativos à Inserção demonstrarem uma distância de ordens um do outro em se tratando dos cenários *SQL* vs *XML*, o mesmo não parece acontecer em relação à operação de consulta. É de especial valor esta primeira conclusão já que a operação de Consulta é uma das mais senão a mais relevante quando se fala BDs.

Trecho de código 6. Exemplo de Consulta Usada

```
SELECT * FROM tbl WHERE col1 > 200
```

Trecho de código 7. Consulta equivalente em Xquery para ambas as variantes XML

```
for $x in doc("arq_xml_raw.xml")/raiz/reg
where $x/c1>200

for $x in doc("arq_xml_for.xml")/raiz/reg
where $x/c1>200
```

Como resultado das consultas executadas nos trechos 6 e 7 é possível observar os seguintes resultados:

	1000	5000	10000	50000	100000
SQL	1.0	6.3	8.2	21.2	29.0
XML Compacto	26.5	32.1	41.5	50.7	99.1
XML Identado	11.3	23.4	29.8	47.7	89.5

Tabela 4. Tempos médios (em milissegundos) de consultas

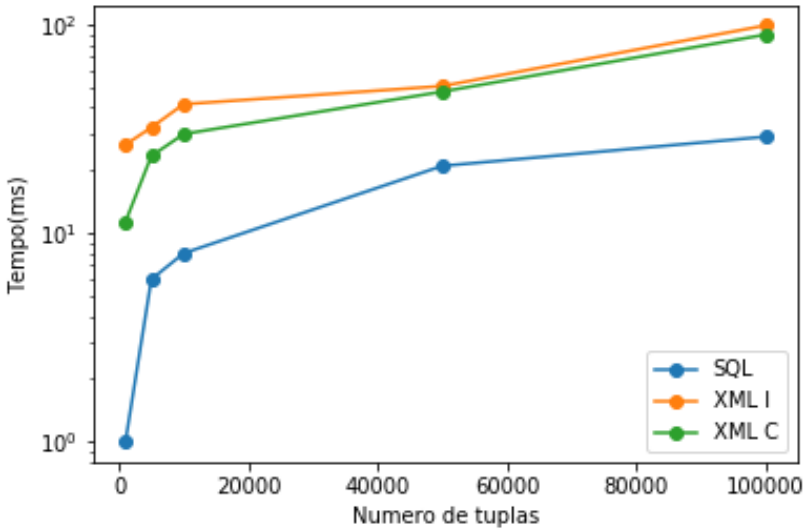

Figura 5. Relação dos tempos médios de consulta

No gráfico da Figura 5 é possível verificar que a variante XML Indentado apresenta um comportamento que pode ser facilmente linearizado. Um comportamento similar porém mais temporalmente custoso é visto na variante XML Compacta. Apesar disto tanto os cenários XML e SQL no geral apresentam uma diferença razoavelmente pequena entre si.

4.2. Reconstrução de consultas em XML

Uma das aplicações mais frequentes do formato *XML* em bancos de dados é justamente seu uso na exportação de dados. Então é especialmente interessante verificar qual o impacto de reconstruir dados quaisquer neste formato.

Nativamente, isto é uma tarefa relativamente simples dado que a própria linguagem fornece meios de fazê-lo por meio do método *file:append*. No trecho de código 8 percebe-se um exemplo dessa função sendo usada.

Trecho de código 8. Reconstrução da Saída em XML

```
file:append('/home/cdlmello/Bases/100000/resres.xml',
    doc('/home/cdlmello/Bases/100000/arq_xml_for.xml')/raiz/record)
```

A linguagem SQL e suas implementações não possuem uma forma padrão de reconstruir consultas para outros formatos além de *csv* que por si só é restrito a algumas implementações. O *front-end* para o BD que foi utilizado durante os experimentos, o *HeidiSQL*, fornece um mecanismo de exportação de consultas para XML padrão, mas como este é função do sistema em si, não há como medir o tempo necessário para reconstrução de consultas SQL para XML.

Fez-se necessária a elaboração de um mecanismo que possa tratar isso ao passo que fossem geradas estatísticas de tempo. Portanto a solução direta foi a construção de dois métodos em SQL nativo, como podem ser verificados nos trechos 9 e 10.

No trecho 9 tem-se o método responsável pelo processamento inicial do resultado de uma consulta. A motivação é a existência de palavras e caracteres reservados, em especial, os delimitadores. Delimitadores em linguagens como *XML* são especialmente importantes, pois ditam a estrutura do corpo principal.

Trecho de código 9. Método que processa palavras reservadas no XML

```
DELIMITER $$
CREATE FUNCTION `del_palavras_reserv`(valortag VARCHAR(256))
RETURNS VARCHAR(256)
DETERMINISTIC
BEGIN
  IF (valortag IS NULL) THEN
    RETURN null;
  END IF;
  RETURN REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
  valortag,'&','&amp;'),
  '<','&lt;'),
  '>','&gt;'),
  '"','&quot;'),
  '\'','&apos;');
END $$
DELIMITER ;
```

Já no código 10 tem-se o método responsável pela construção de *tags* e *subtags* simples. O resultado pode-se verificar logo em seguida no código 12.

Trecho de código 10. Método que constroi tags e subtags em XML

```
DELIMITER $$
DROP FUNCTION IF EXISTS `cria_tag` $$
CREATE FUNCTION `cria_tag`(nometag VARCHAR(256),valortag VARCHAR(256),
    subtags VARCHAR(256))
RETURNS VARCHAR(256)
DETERMINISTIC
BEGIN
  DECLARE result VARCHAR(256);
  SET result = CONCAT('<' , nometag);
  IF (valortag IS NULL AND subtags IS NULL) THEN
  RETURN CONCAT(result,' />');
  END IF;
  RETURN CONCAT(result ,'>',ifnull(del_palavras_reserv(valortag),''),
  ifnull(subtags,''),'</',nometag, '>');
END $$
DELIMITER ;
```

No exemplo do trecho 11 é possivel o uso dos métodos explicitados acima, quanto a estrutura do argumentos, para cada *subtag* (que é o terceiro argumento) deve se chamar a função para criar uma *tag*.

Trecho de código 11. Exemplo de uso dos métodos

```
select cria_tag('record',null,concat(cria_tag('col1',col1,null,null),
  cria_tag('col2',col2,null),
  cria_tag('col3',col3,null),
```

```
   cria_tag('col4',col4,null)
)
)
from tbl
where col1 > 1998
INTO OUTFILE '/tmp/res.xml'
```

Por fim podemos verificar o resultado final do uso destes métodos juntamente a consultas ao banco.

Trecho de código 12. Resultado da reconstrução

```
<record><col1>1999</col1><col2>KfoTLrqfXkogeKnf</col2><col3>1148</col3>
<col4>zYJKydFNBDkJOLKd</col4></record>
<record><col1>1999</col1><col2>ibYJAVYknAhNvuTI</col2><col3>129</col3>
<col4>uIalF
vLlewfIhgrt</col4></record>
<record><col1>1999</col1><col2>oRNdKFcqMUqnIBxT</col2><col3>1669</col3>
<col4>vDusDh
gONjAveKen</col4></record>
```

Considerando a escala de tamanho para cada base, os métodos elaborados e a topologia do experimento pode-se extrair os resultados quanto à reconstrução das consultas em arquivo XML mostrados na Tabela 5 e na Figura 6.

	1000	5000	10000	50000	100000
XML	7.1	31.0	139.2	331.2	834.1
SQL	12.4	103.1	129.2	526.1	1120.1

Tabela 5. Tempos médios de recuperação e reconstrução para arquivos XML

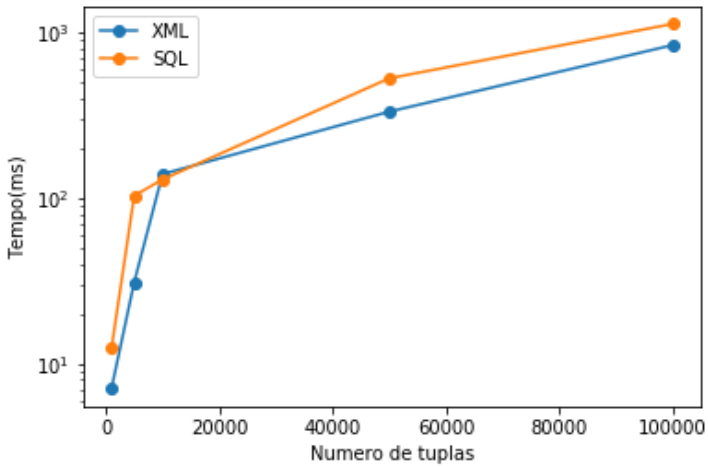

Figura 6. Relação dos tempos médios de recuperação e reconstrução

Considerando os resultados é perceptível que o comportamento de ambos os cenários é relativamente similar, com o cenário XML se sobressaindo por pouco. A partir da ordem de 10^4 o comportamento de ambos parece cair abruptamente e retomam seu

crescimento rápido na próxima ordem de tamanho. Percebe-se ainda que ambas as categorias se aproximam a partir de determinados valores, em especial no intervalo $[10^4, 10^5]$ o que pode indicar uma tendência onde o SQL se sobressai. Então é possível possível concluir nesta instância que para valores suficientemente grandes a reconstrução de *queries* SQL/XML é interessante e indica a viabilidade de aplicar a reconstrução para XML em alguns contextos relacionais.

5. Conclusões

Considerando os resultados é possível concluir que para o cenário apresentado, NXMLDBs possuem desempenho similar a RMDBs, no entanto estes resultados foram extraídos de experimentos controlados e que não correspondem a natureza dos dados aplicados em mundo real. Apesar disto é possível extrair diversas conclusões que tangem o contexto: O cenário de reconstrução XML mostrou que há grande viabilidade em utilizar arquivos XML como formato de exportação, o que significa que existe aplicabilidade em se tratando do uso destas duas tecnologias em uma abordagem híbrida SQL/XML.

Outras possibilidades envolvem o uso do XML Nativo quando se tem uma situação crítica onde a recuperação dos dados de forma mais rápida é necessária, isto torna esta tecnologia de especial importância em se tratando da recuperação de um estado inesperado, na recuperação de estados anteriores (backups) e ainda como forma de indexar dados de forma a reduzir a carga das mais custosas no modelo relacional clássico, mesmo considerando o custo posterior de normalizar estes dados para formato relacional.

Referências

Bergeron, R. (2000). Xpath - retrieving nodes from an xml document.

Beynon-Davies, P. (2004). *Database Systems*. Palgrava Macmillan, third edition.

Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., Yergeau, F., and Cowan, J. (2006). Extensible markup language (xml) 1.1 (second edition). W3c recommendation, W3C - World Wide Web Consortium.

Chaudhri, A., Zicari, R., and Rashid, A. (2003). *XML Data Management: Native XML and XML Enabled DataBase Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Codd, E. (1985). Is your dbms really relational? *Computer World*.

de Avelar, F. T. M. (2012). Xmap: Mapeamento e armazenamento de dados xml em bancos de dados relacionais. Master's thesis, Universidade Federal de Santa Maria.

Haerder, T. and Reuter, A. (1983). Principles of transaction-oriented database recovery. *ACM Comput. Surv.*, 15(4):287–317.

Kilpeläinen, P. (2012). Using xquery for problem solving. *Software: Practice and Experience*, 42(12):1433–1465.

Luis Carlos Tanaka, Felipe Melo Camargo, R. G. (2012). Sistema gerenciador de banco de dados: Sgbd exist xml. In *Revista Eletronica de Sistemas de Informação e Gestão Tecnológica*, volume 2 of *1*, pages 71–86. Uni-FACEF.

Monteiro, E. (2011). Uma ferramenta para carga de bancos de dados relacionais a partir de fontes de dados xml.

News, X. (1999). Xml basics.

Nicola, M. and van der Linden, B. (2005). Native xml support in db2 universal database. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 1164–1174. VLDB Endowment.

Phillip J. Pratt, M. Z. L. (2014). *Concepts of Database Management*. Course Technology, eighth edition.

XMLDBInitiative (2013). What is a xml database?