Limpeza de Dados XML

Diogo Fernandes Simões

Dissertação para a obtenção de Grau de Mestre em

Engenharia Informática e de Computadores

Júri

Presidente: Prof. Mário Jorge Costa Gaspar da Silva
Orientador: Prof. Bruno Emanuel da Graça Martins
Co-orientadora: Prof.ª Helena Isabel de Jesus Galhardas
Vogais: Prof. José Luís Brinquete Borbinha

Junho de 2013

Abstract

The eXtended Markup Language (XML) is currently the language of choice for storing and transmitting data across diverse application domains. Moreover, the XQuery language has emerged as a powerful and standardized way for querying XML documents. However, and despite its expressiveness, XQuery still lacks appropriate mechanisms for effectively handling XML data cleaning problems. Data cleaning refers to the process of correcting anomalies in a dataset, that may for instance be due to typographical errors or duplicate representations of real world objects. With the growing amount of XML data, approaches to effectively and efficiently clean XML are clearly needed, although this issue is not addressed by existing data cleaning systems, which mostly specialize on relational data. This thesis advocates the usage of XQuery together with extension functions for XML data cleaning. We discuss the advantages of such an approach, introduce some example application scenarios, detail the implementation on top of an existing XQuery engine and compare this approach with other ways to perform XML data cleaning.

Keywords: Data Cleaning , XML , XQuery

Resumo

A *eXtended Markup Language* (XML) é atualmente a linguagem de eleição para armazenar e transmitir dados através de diversos domínios aplicacionais. Para além disto, o XQuery tem vindo a surgir como uma abordagem poderosa e padronizada para realizar consultas em documentos XML. Contudo, e apesar da sua expressividade, o XQuery ainda apresenta alguns problemas relativos à eficiência com que lida com problemas de limpeza de dados. A limpeza de dados refere-se ao processo de correção de anomalias existentes em bases de dados, que podem dever-se, por exemplo, a erros ortográficos ou representações duplicadas de objetos do mundo real. Com o crescimento da quantidade de dados XML, abordagens para realizar limpeza de dados XML de forma eficiente e eficaz são claramente necessárias. Apesar desta necessidade, a grande maioria dos sistemas de limpeza de dados não permite a limpeza de dados XML, especializando-se na limpeza de dados relacionais. Esta tese defende a utilização do XQuery com uma extensão de funções para permitir a limpeza de dados XML. Neste documento são discutidas as vantagens desta abordagens, são introduzidos alguns cenários de utilização desta extensão do XQuery, é detalhado o conjunto de funções existentes na extensão criada e é realizada uma comparação entre esta abordagem e outras formas de desempenhar limpeza de dados XML.

Keywords: Data Cleaning , XML , XQuery

Agradecimentos

Ao longo do último ano e meio dediquei a maior parte do meu tempo à procura de novas soluções para resolver problemas relacionados com a limpeza de dados XML. Foi claramente uma das tarefas mais duras que tive até hoje. Contudo, tenho a felicidade de estar rodeado de grandes pessoas que me ajudaram a minimizar as adversidades que foram surgindo no decorrer deste trabalho. Assim, gostaria de dedicar este capítulo a todos aqueles que me ajudaram a superar esta última etapa do meu percurso académico.

Em primeiro lugar agradeço aos meus orientadores, Professor Bruno Martins e Professora Helena Galhardas pelo acompanhamento que me foram dando ao longo do tempo em que elaborei esta dissertação e por todo o conhecimento referente ao tema da dissertação que me foram transmitindo. Ajudaram-me a perceber como funciona o mundo da investigação cientifica, o que de certa forma me ajudou bastante a decidir o caminho profissional que quero seguir.

Gostaria ainda de agradecer à *FLWOR Foundation* por ter apoiado financeiramente este projeto e por ter disponibilizado o software utilizado para a construção do módulo de funções para limpeza de dados em XQuery.

Agradeço também aos meus colegas de projeto, da Accenture que sempre me apoiaram na concretização desta dissertação.

Agradeço ainda aos amigos de sempre, os que mesmo percebendo pouco ou quase nada do assunto desta dissertação fizeram sempre questão de saber acerca da sua evolução: a Catarina Diogo, a Cláudia, a Inês, o Diogo, o Duarte, o João Duarte, o João Santos e o Luís.

Quero deixar um agradecimento enorme aos maiores responsáveis pelo equilíbrio do meu estado de espírito: à Catarina por não deixar que o meu Ser adolescente esmoreça, ao Gil pela forma como contagia com a sua persistência, à Bárbara por ser parte da minha consciência em momentos de devaneio, ao Paulo por ser o amigo sempre presente, mesmo a 4570 milhas de distância e ao Gonçalo por conseguir inspirar qualquer pessoa, fazendo parecer que nada é impossível.

Mas o agradecimento mais especial vai para os meus pais. Um agradecimento que tem de ir muito para além do período em que tive ocupado com esta tese de mestrado. É o agradecimento de uma vida que dificilmente poderia ser melhor, muito por culpa da forma como os meus pais me incutiram os valores em que acreditam e que me têm, sem qualquer dúvida, encaminhado da melhor forma.

Sumário

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

A limpeza de dados consiste na correção de problemas de qualidade de dados que podem traduzir-se em erros ortográficos, dados aproximadamente duplicados ou dados não normalizados. Bases de dados com estes tipos de problemas afetam a assertividade de sistemas que dependam destas. Por exemplo, um sistema de gestão de relacionamento com o cliente (*CRM - Customer Relationship Management*), i.e., sistema que providencia informação acerca dos clientes de uma organização e das suas preferências de acordo com análises estatísticas dos seus pedidos) seria afetado se a qualidade dos dados fosse diminuída pela existência de dados duplicados.

Existem vários fatores que causam problemas na qualidade de dados. Um deles é a inserção de dados errados nas bases de dados. Por vezes, os dados podem ser inseridos através de sistemas que não possuem qualquer tipo de validação da informação recolhida (por exemplo, ao se introduzir o registo de uma morada numa base de dados permitir a inserção de um código postal que não corresponde ao valor da freguesia). Contudo, o problema que mais potencia problemas na qualidade de dados é a integração de dados provenientes de várias fontes de dados que são gerados de forma independente, o que pode potenciar a criação de bases de dados com dados duplicados ou com dados não normalizados.

Por forma a colmatar estes problemas, na última década foram publicados alguns trabalhos científicos que abordam a limpeza de dados relacionais [2; 13; 15; 16; 18]. Porém, podem existir outras fontes de dados que não as relacionais (i.e., fontes de dados não estruturadas ou fontes de dados semi-estruturadas), com problemas de qualidade de dados. Relativamente a estas existe ainda pouco trabalho cientifico acerca de mecanismos que assegurem a sua qualidade.

É na limpeza de dados XML (eXtensible Markup Language) que se foca esta dissertação. O XML

é um esquema de codificação de dados semi-estruturados, organizados hierarquicamente, que nos últimos anos tem sido cada vez mais utilizado, uma vez que este é uma das principais formas de transporte e armazenamento de dados na web. Assim, o desenvolvimento de mecanismos que executem a limpeza de dados XML (i.e, identificação e métodos de correção automática de problemas de qualidade de dados e métodos de correção automáticos dos problemas) tem vindo a ganhar uma maior importância.

1.1 Hipótese e Metodologia

No contexto desta tese de mestrado foi proposta a conceção e o desenvolvimento de uma biblioteca de funções em XQuery, portável para qualquer processador desta linguagem, que pudesse ser utilizada para escrever programas de limpeza de dados XML, em XQuery [8], a linguagem de interrogação de dados XML. Assim, esta tese de mestrado tem como principal objetivo a validação das seguintes hipóteses que advêm do processo de limpeza de dados em XML:

- Será possível desenvolver uma biblioteca para limpeza de dados XML, portável para qualquer processador de XQuery, que possa ser usada para implementar processos de limpeza de dados de forma eficaz e eficiente?
- Qual é o custo de se desenvolver uma biblioteca para limpeza de dados XML em XQuery, portável para qualquer processador ao invés de se desenvolver uma biblioteca de limpeza de dados XML noutra linguagem à partida mais eficiente, como por exemplo o Java?
- É possível criar um mecanismo que identifique todos os problemas que podem existir em fontes de dados XML?
- Existe uma metodologia que assegura a limpeza de dados XML?

Por forma a validar estas hipóteses foram realizadas duas experiências. Na primeira experiência é realizada uma comparação entre os tempos de execução de funções de semelhança de cadeias de caracteres escritas em XQuery (que estão incluídos na biblioteca para limpeza de dados em XML) e as mesmas escritas em Java. Assim, é validada a hipótese cientifica de que o ganho da portabilidade da biblioteca tem um custo no seu desempenho.

Na segunda experiência são comparadas diferentes abordagens para limpeza de dados XML - a linguagem XQuery com recurso a funções incluídas na biblioteca de funções para limpeza de dados XML criada no âmbito desta tese de mestrado, a ferramenta de integração e transformação de dados *SQL Server Integration Services* (SSIS), da Microsoft e o XClean, um protótipo

de uma ferramenta para limpeza de dados XML. Nesta experiência são comparados tempos de execução entre as diferentes abordagens e a sua assertividade, verificada através do número de falsos duplicados identificados. Com esta experiência, é possível verificar a utilidade na biblioteca de funções e demonstrar a sua viabilidade enquanto alternativa a outras abordagens para limpeza de dados XML. O processo de levantamento de problemas na qualidade dos dados utilizados nesta experiência foi realizado através da taxonomia proposta. Assim, com esta experiência, é possível demonstrar que pode existir um mecanismo capaz de identificar corretamente problemas de qualidade de dados que existam em fontes de dados XML. A metodologia adotada no processo de limpeza dos dados é a que proponho nesta tese e que está descrita na Secção 4.2., provando assim que existe uma metodologia que assegura a limpeza de dados XML.

1.2 Principais Contribuições

As principais contribuições desta tese são as seguintes:

- *Uma taxonomia para a identificação de problemas de qualidade de dados XML*. Esta é uma taxonomia baseada nos problemas de qualidade de dados que ocorrem ao nível dos dados relacionais mas que tem em conta a natureza hierárquica e semi-estruturada dos documentos XML. Atualmente, que seja do nosso conhecimento, não existe outra taxonomia para a deteção de problemas de qualidade de dados XML publicada.

- *Uma metodologia de limpeza de dados XML* que segue uma abordagem *bottom-up*, considerando que a limpeza dos nós pais é assegurada pela limpeza dos nós filhos de um elemento XML. Assim, de forma recursiva dos nós filhos para os nós pais são realizadas tarefas de correção, normalização, deteção e eliminação de dados duplicados e enriquecimento dos dados que asseguram a limpeza dos nós superiores.

- *Uma biblioteca de funções para limpeza de dados XML em XQuery*, portável para qualquer processador desta linguagem. Esta biblioteca inclui módulos de semelhança entre cadeias de caracteres, de semelhança entre elementos do documento, de conversão e normalização de valores, e de consolidação de dados.

- Um estudo experimental que verifica qual é o custo em termos de velocidade de processamento que implica assegurar a portabilidade da biblioteca de funções para limpeza de dados XML em XQuery, ao invés de ser utilizada a linguagem Java.

1.3 Organização do Documento

Este documento começa por descrever, no Capítulo 2, os conceitos fundamentais relacionados com a tecnologia XML e com os problemas de qualidade de dados. No Capítulo 3, é apresentado o trabalho científico relacionado com a limpeza de dados em XML. O Capítulo 4 apresenta as principais contribuições desta dissertação (descrevendo a taxonomia de problemas na qualidade de dados na Secção 4.1, a metodologia de limpeza de dados XML na Secção 4.2, e a implementação do processo de limpeza de dados XML usando a linguagem XQuery na Secção 4.3). A validação destas contribuições é realizada no Capítulo 5 onde são apresentadas as experiências realizadas, o conjunto de dados utilizado e os resultados obtidos. Por fim, no Capítulo 6, são descritas as conclusões desta dissertação.

Capítulo 2

Conceitos

Neste capítulo, são descritos os conceitos fundamentais relacionados com a limpeza de dados em XML. Na Secção 2.1, são descritos os conceitos básicos da linguagem XML e das suas tecnologias. De seguida, na Secção 2.2, são apresentadas restrições de integridade que podem evitar problemas de qualidade de dados e é descrito o processo de qualidade de dados.

2.1 Tecnologias XML

A eXtensible Markup Language (*XML*), [4] é uma recomendação do World Wide Web Consortium (W3C) criada para facilitar a troca de informação na Internet. Resumidamente, a linguagem XML é um esquema de codificação para dados semi-estruturados, organizados hierarquicamente. A Listagem 2.1. exemplifica um documento XML que codifica informação acerca de CDs de música. Cada elemento que representa um CD de música é constituído por um elemento identificador, *did*, um elemento correspondente ao artista, *artist*, um elemento referente ao título do CD, *dtitle*, outro elemento que providencia informação acerca do género do CD, *genre*, e outro que corresponde ao ano de lançamento do CD, *year*. Pode ainda existir o elemento *cdextra* que contém informações suplementares acerca do CD.

A recomendação XML [4] providencia uma lista de regras de sintaxe que devem ser satisfeitas para que um documento XML seja considerado bem formado. Para tal, um documento XML deve obedecer a regras, tais como:

- Um documento XML possui pelo menos um elemento (definido como o elemento raíz do documento XML).

- Só pode existir um único elemento raíz.
- Cada elemento deve possuir uma etiqueta de abertura e outra de fecho.
- Os valores dos atributos devem ser referidos dentro de aspas (por exemplo, <student id = "12345" age="22"> é a forma correta de referir os atributos, enquanto que <student id = 12345 age=22> está incorreto).
- As etiquetas não podem conter certos caracteres, como: !"#$%&'()*+,/;<=>?@[]‘{|} .
- As etiquetas não podem iniciar-se por *"-, ."*, ou por um digito.

```
<cddb>
  <disc>
    <did>950ad90c</did>
    <artist>Pearl Jam</artist>
    <dtitle>Pearl Jam</dtitle>
    <genre>Grunge</genre>
    <year>1993</year>
    <cdextra> YEAR: 1993</cdextra>
  </disc>
  <disc>
    <did>b45jh3j</did>
    <artist>Coldplay</artist>
    <dtitle>Viva la Vida</dtitle>
    <genre>Baroque pop</genre>
    <year>2008</year>
  </disc>
</cddb>
```

Listagem 2.1: Exemplo de um documento XML.

Por si só, o XML não impõe restrições ao nível do conteúdo dos documentos. Contudo, através da linguagem *XML Schema* [20], é possível definir regras para a estrutura, o conteúdo e a semântica de um documento XML. Na Listagem 2.2, é apresentado um exemplo de XML Schema que valida o documento XML presente na Listagem 2.1. Como se verifica no exemplo de XML Schema dado, o documento XML presente na Listagem 2.1 é constituído por um elemento raíz *cddb* que contém vários elementos *disc*. Cada elemento *disc* é por sua vez constiuído pelo conjunto de elementos *did*, *artist*, *dtitle*, *genre*, *year* e *cdextra*, todos eles do tipo *xsd:string*.

Para aceder a partes de um documento XML, existe uma linguagem de interrogação, denominada *XPath* [11]. O resultado de uma expressão XPath é constituído por um conjunto de nós ou um valor atómico provenientes do(s) documento(s) dado(s) como entrada na expressão. Este

resultado deriva de uma expressão de caminho que providencia uma forma de endereçar hierarquicamente os nós de uma árvore XML.

É possível aceder a nós que contenham um valor específico. Para isso a expressão XPath deve ter um predicado responsável por filtrar os nós com base em expressões booleanas ou numéricas, envolvendo os valores dos nós. A Listagem 2.3 apresenta um exemplo de uma expressão XPath sobre o documento XML representado na Listagem 2.1, e do seu resultado. Este exemplo devolve a listagem de artistas, definida pela expressão de caminho (cddb/disk/artist). Como predicado, é estipulado que apenas serão listados os artistas cujo disco a que pertencem é do género “Grunge” (disk[genre = ’Grunge’]).

```
<?xml version="1.0" encoding="utf-16"?>
<xsd:schema attributeFormDefault="unqualified"
elementFormDefault="qualified" version="1.0"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="cddb">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="disc">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="did" type="xsd:string" />
              <xsd:element name="artist" type="xsd:string" />
              <xsd:element name="dtitle" type="xsd:string" />
              <xsd:element name="genre" type="xsd:string" />
              <xsd:element name="year" type="xsd:int" />
              <xsd:element name="cdextra" type="xsd:string" />
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listagem 2.2: Exemplo de um XSD para o documento da Listagem 2.1.

```
cddb/disk[genre = 'Grunge']/artist

<artist>Pearl Jam</artist>
```

Listagem 2.3: Expressão XPath que produz como resultado o conjunto de artistas cujo género do CD é "Grunge".

A linguagem XPath não tem expressividade suficiente para criar interrogações complexas. Para colmatar esta limitação, foi criada uma linguagem de interrogação para conjuntos de dados em

XML, denominada por *XQuery* [8]. A linguagem XQuery providencia funcionalidades, tais como: *(i)* expressões XPath que dão suporte à navegação, selecção e extracção de valores; *(ii)* construtores para criar novos valores; *(iii)* funções para manipulação dos dados; e *(iv)* expressões *FLWOR* para um processamento de dados mais complexo, permitindo iterar sobre sequências (através da cláusula *for*), definir e atribuir valores a variáveis (utilizando a cláusula *let*), ordenar resultados (através da cláusula *order by*), aplicar filtros a resultados (utilizando a cláusula *where*) e construir um resultado (através da cláusula *return*). Na Listagem 2.4, é apresentado um exemplo de uma interrogação escrita em XQuery e os resultados obtidos da sua execução sobre o documento XML apresentado na Listagem 2.1. Neste exemplo, são percorridos todos os elementos *disk* do documento XML apresentado na Listagem 2.1, e para cada um, será guardado, na variável *$name*, o nome do artista se se verificar que o género do disco corresponde a "Grunge". A lista de nomes dos artistas é depois retornada, ordenada pelo valor do atributo *id* do CD correspondente.

Para além destas funcionalidades, a linguagem XQuery possui também extensões como o XQuery Full-Text [1] que permite realizar interrogações sobre informação textual, e o XQuery Update [9] que permite a actualização de dados em documentos XML.

Com a linguagem XQuery é ainda possível implementar novas funções para além das pré-existentes, capazes de manipular dados XML [8]. A possibilidade de implementação de novas funções em XQuery permite a criação de bibliotecas de funções que estendam as funcionalidades básicas do XQuery. Por exemplo, um utilizador pode criar um conjunto de funções que possibilitem a limpeza de dados em XML.

```
for \$disk in \$listing21/disk
let \$name := \$disk/artist/text()
where \$disk/genre/text() = 'Grunge'
order by \$disk/@did
return \$name

Pearl Jam
```

Listagem 2.4: Exemplo de uma expressão FLWOR em XQuery e o seu resultado.

2.2 Problemas de Qualidade de Dados

Existem vários tipos de problemas de qualidade de dados. Podem, por exemplo, existir valores em falta, erros de sintaxe, dados duplicados, dados contraditórios ou dados desatualizados. Alguns destes problemas podem ser evitados através da definição do esquema da base de

dados, definindo regras que os impeçam (ou seja, restrições de integridade).

Esta secção começa por descrever algumas propriedades que devem ser tidas em conta para que se proceda ao desenho de um bom esquema de dados XML, introduzindo o conceito de restrições de integridade. De seguida, é apresentado o processo de qualidade de dados.

2.2.1 Restrições de Integridade em XML

As restrições de integridade são condições definidas durante o processo de desenho de uma base de dados que devem ser satisfeitas pelos dados para que esta seja considerada consistente. Ao se especificarem estas restrições, alguns problemas de qualidade de dados são evitados. Em bases de dados relacionais podemos definir as seguintes restrições de integridade [19]:

- **Restrição de Integridade da Entidade:** Especifica que, em todas as relações, existe uma chave primária e que os atributos escolhidos para formar esta chave primária têm de ser únicos e não nulos.
- **Restrição de Dependência Funcional:** Especifica que em dois conjuntos de atributos X e Y, X determina univocamente Y se cada valor de X está associado a um e um só valor de Y.
- **Restrição de Unicidade:** Especifica que o valor de um atributo tem de ser único.
- **Restrição de Domínio:** Especifica que os valores de um atributo devem respeitar o seu tipo e as restrições impostas sobre o seu atributo (por exemplo, um atributo referente a uma idade é sempre um valor inteiro positivo) [6].
- **Restrição de Integridade Referencial:** Especifica atributos de uma relação cujos valores referenciam valores da chave primária de outra tabela.

As restrições de integridade existentes em conjuntos de dados de XML são ligeiramente diferentes das restrições de integridade que existem nas bases de dados relacionais. De acordo com os trabalhos de Buneman e Fan [5], e posteriormente de Fan [12], é possível identificar restrições de chave e dependências funcionais. Para além destas restrições, são estabelecidas outras três restrições de integridade: as restrições de integridade referencial, de unicidade e de domínio:

- **Restrição de Chave:** Esta restrição é similar à restrição de integridade da entidade existente nas bases de dados relacionais. Assim, cada elemento pertencente a uma árvore

XML deve ter um atributo chave, único e não nulo. Na definição do esquema de um documento XML, esta restrição pode ser assegurada pelo construtor *xs:key*.

- **Restrição das Dependências Funcionais:** Existem várias definições possíveis para as dependências funcionais em XML. Algumas destas definições são baseadas nos caminhos dos elementos, enquanto que outras são baseadas nas sub-árvores que estes elementos constituem. Uma definição possível para as dependências funcionais em XML, baseada em caminhos, refere que um caminho *p1* determina um caminho *p2* se os valores de *p1* determinam univocamente os valores de *p2* (i.e., se para cada conjunto de valores de *p1* está associado um e um só conjunto de valores de *p2*) [10].

- **Restrição de Integridade Referencial:** Esta restrição especifica que o valor de um atributo ou elemento corresponde aos valores da chave ou de um atributo ou elemento com valores únicos. É possível definir esta restrição no esquema do documento XML utilizando o construtor *xs:keyref*.

- **Restrição de Unicidade:** Esta restrição define que o valor de um dado elemento ou atributo tem de ser único. É possível assegurar esta restrição utilizando o construtor *xs:unique* no esquema do documento XML.

- **Restrição de Domínio:** Esta restrição tem uma definição similar à restrição de integridade com o mesmo nome em bases de dados relacionais, especificando o tipo e o conjunto de valores possíveis de um dado elemento ou atributo.

2.2.2 Processo de Qualidade dos Dados

Como já foi referido nesta tese, a qualidade dos dados tem consequências sobre a sua manipulação. Desta forma, é importante que os dados sejam periodicamente auditados e limpos para melhorar ou manter a sua qualidade e facilitar a sua manipulação. Assim, um processo de qualidade de dados tem por objetivo maximizar a qualidade dos dados e minimizar o impacto negativo que a sua fraca qualidade pode ter. A Figura 2.2 ilustra os passos que um processo de qualidade de dados deve respeitar.

Num primeiro passo, é executada a *avaliação da qualidade dos dados*. Aqui, são identificados os problemas de qualidade de dados existentes e é medido o impacto que estes problemas podem ter aquando da utilização destes dados num dado processo ou aplicação. Os problemas de qualidade de dados podem ser identificados através de taxonomias como as referidas na Secção 4.1.

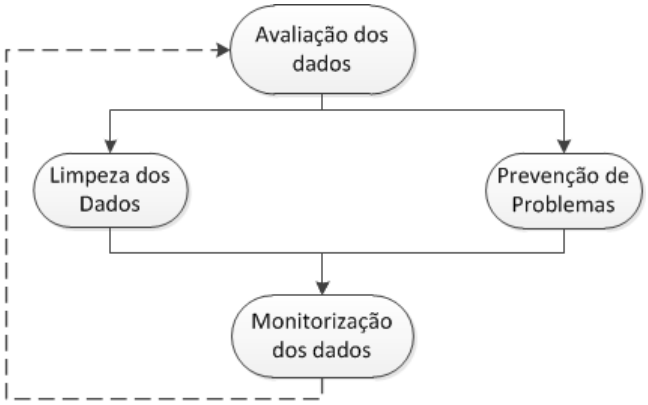

Figura 2.1: Processo de Qualidade de Dados.

Idealmente, todos os problemas identificados seriam posteriormente corrigidos. Contudo, o custo da correção desses dados pode ser muito elevado, sendo importante tomar uma decisão quanto à escolha dos dados que devem ser limpos, pesando o seu custo e o seu benefício.

Após a avaliação da qualidade dos dados, estes são alterados de acordo com as necessidades dos seus utilizadores. Este passo de melhoria da qualidade dos dados denomina-se por *limpeza dos dados* (que é o tema desta tese de mestrado). Esta tarefa foca-se na normalização dos dados, na eliminação dos seus erros e na remoção de duplicados.

A limpeza de dados é uma tarefa que pode ser realizada manualmente e/ou automaticamente. Realizá-la de forma inteiramente manual tem a grande desvantagem de ser um processo moroso, caro, subjetivo e extremamente susceptível a erros, pois necessita de recursos humanos, que podem interpretar erros de diversas formas e que podem facilmente originar erros durante o processo de correção dos dados. Por outro lado, realizar a tarefa de limpeza de dados de forma exclusivamente automática tem as suas limitações, principalmente no que toca ao processo de desambiguações de informação, que em grande parte das vezes não podem ser realizadas computacionalmente (por exemplo, descobrir o verdadeiro significado do nome D. F. Simões, é uma tarefa que, manualmente, pode ser mais eficaz). Desta forma, os processos de limpeza de dados, são tipicamente realizados de forma automática, e os resultados obtidos são posteriormente verificados e melhorados através de processos manuais.

As etapas de um processo de limpeza de dados são, tipicamente as seguintes: a *(i) normalização dos dados*, em que estes são transformados de forma a estarem em conformidade com um formato pretendido; a *(ii) remoção de dados errados e duplicados*, em que através de técnicas, como a utilização de dicionários por forma a corrigir erros ortográficos, aprendizagem automática, a consolidação de duplicados aproximados ou a utilização de bibliotecas com funções de

limpeza pré-definidas, como é proposto neste trabalho, é feita a melhoria dos dados; e o *(iii) enriquecimento dos dados*, em que é adicionada informação complementar, proveniente de outras fontes de dados, aos dados existentes.

Melhorar os dados através de processos de limpeza não é, por si só, suficiente para assegurar a qualidade dos dados: é também importante, *prevenir a origem de novos erros*. Grande parte dos dados armazenados digitalmente são introduzidos através de interfaces gráficas. Desta forma, é possível prevenir uma boa parte dos erros se estas interfaces estiverem bem programadas com métodos para validação dos dados e prevenção de inserção de dados duplicados. Outra origem dos problemas na qualidade de dados é o processo de integração de dados provenientes de múltiplas fontes. Desta forma, é possível minimizar problemas da qualidade de dados redobrando os cuidados em processos de integração de dados (i.e., fazer uma análise das fontes de dados, antes do processo de integração, verificando os seus pontos comuns e os seus pontos distintos), tendo em conta a possibilidade da existência de dados duplicados.

Após ser assegurada a qualidade dos dados através dos processos definidos anteriormente, é importante *monitorizar a qualidade dos dados* regularmente, de forma a evitar que os dados atinjam um estado de sujidade que possa originar problemas na sua utilização e manipulação. Para monitorizar a qualidade dos dados, estes devem ser avaliados recorrentemente e corrigidos sempre que necessário, criando um novo ciclo no processo de qualidade de dados.

Capítulo 3

Trabalho Relacionado

Devido à importância que a qualidade dos dados tem para as organizações que os possuem, na última década, foram propostas várias técnicas e ferramentas para limpeza de dados. Grande parte destes trabalhos de limpeza de dados propõem métodos e ferramentas para dados armazenados em bases de dados relacionais. Estes trabalhos serviram como base para a criação dos métodos e ferramentas existentes para bases de dados em XML. Neste capítulo, são apresentados alguns dos trabalhos existentes relacionados com a limpeza de dados em XML. Na Secção 3.1 é apresentado o protótipo para limpeza de dados em XML, denominado XClean. Na Secção 3.2, é apresentado o *framework* para a deteção de duplicados em XML, DogmatiX. De seguida, na Secção 3.3, são apresentadas estratégias para a fusão de dados XML duplicados. O capítulo é encerrado, com uma discussão acerca do trabalho relacionado (Secção 3.4) e uma comparação entre algumas das ferramentas existentes para a limpeza de dados XML (Secção 3.5).

3.1 XClean

O XClean é um sistema desenhado para limpar dados armazenados em documentos XML [21]. A Figura 3.2 representa a arquitetura do XClean. O utilizador escreve um programa de limpeza de dados utilizando uma linguagem declarativa denominada XClean/PL. Este programa é implementado com recurso a funções implementadas em Java e em XQuery que estão disponíveis numa biblioteca de funções pré-definidas. Esta biblioteca inclui funções recorrentemente utilizadas em processos de limpeza de dados, como por exemplo funções de formatação e normalização de dados e funções de similaridade entre cadeias de caracteres. Para além das

Operador	Objetivo
Candidate Selection	Seleciona os elementos a serem limpos.
Scrubbing	Remove erros do texto.
Enrichment	Especifica os dados relevantes.
Duplicate filtering	Filtra elementos não duplicados.
Pairwise Duplicate Classification	Classifica pares de elementos como duplicados, não duplicados,...
Duplicate Clustering	Determina os agrupamentos de duplicados.
Fusion	Cria uma representação única de uma entidade.
XML view	Cria uma vista XML dos dados limpos.

Tabela 3.1: Operadores do XClean/PL.

funções pré-definidas, esta biblioteca pode ser estendida com outras funções, mais específicas para a limpeza dos dados que estão a ser manipulados, escritas em XQuery ou em Java. Os programas desenvolvidos pelo utilizador são posteriormente compilados e traduzidos em XQuery para serem executados. Esta particularidade permite uma execução mais eficiente do programa, comparativamente à utilização de linguagens procedimentais como o Java, uma vez que o XQuery possui planos de otimização de interrogações.

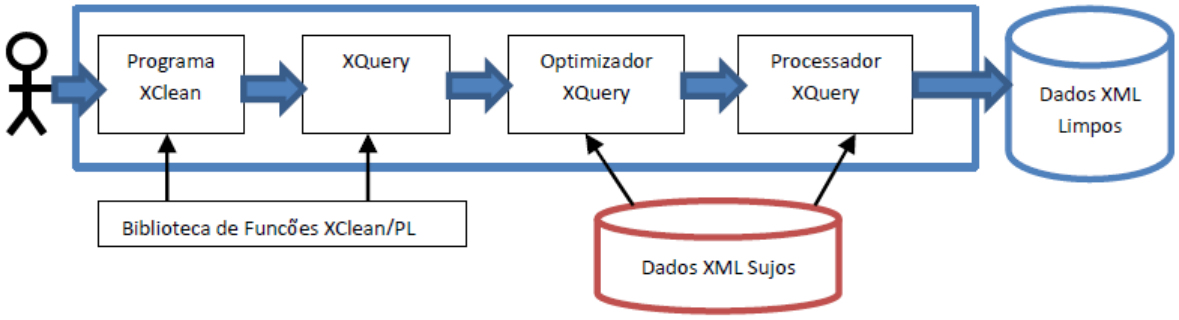

Figura 3.2: Arquitetura do XClean [21].

A linguagem XClean/PL disponibiliza os operadores apresentados na Tabela 3.1. O operador *candidate selection* define quais os elementos numa base de dados em XML que são relevantes para a limpeza dos dados, aceitando como entrada o caminho destes elementos no documento. Após a escolha dos dados que serão limpos, os valores devem ser normalizados. O operador *Scrubbing* é o responsável por essas normalizações, utilizando, entre outras, funções de normalização de datas ou de separação de cadeias de caracteres.

Um problema de qualidade de dados bastante recorrente é a existência de elementos duplicados. O primeiro passo para resolver este problema é a escolha dos dados que devem ser usados para realizar a comparação entre dois candidatos a duplicados. Com a linguagem XClean/PL é possível especificar que dados são relevantes para realizar essas comparações entre elementos através de um operador denominado *Enrichment*.

A deteção de elementos duplicados é um passo computacionalmente dispendioso num processo de limpeza de dados. Para acelerar o processo de deteção de duplicados e, assim, evitar a comparação de cada registo com os restantes (i.e., produto cartesiano), existe o operador *Duplicate Filtering* que elimina todas as comparações desnecessárias. Por exemplo, se estivermos a verificar se dois elementos representam a mesma pessoa no mundo real, comparando os nomes registados, é possível filtrar o número de comparações assumindo que não é necessário comparar dois nomes que tenham o primeiro caracter diferente.

Após o passo de filtragem é necessário emparelhar os duplicados. Na linguagem XClean/PL, existe um operador denominado *Pairwise Duplicate Detection*, que aceita como dados de entrada pares de elementos possivelmente duplicados. Através de funções de comparação de cadeias de caracteres, os candidatos a pares são classificados como elementos duplicados, possivelmente duplicados ou não duplicados. Posteriormente, estes pares são agrupados com outros elementos que lhes são duplicados, através do operador *Duplicate Clustering*. De seguida, para cada grupo de elementos duplicados é necessário escolher uma representação única e limpa. O operador *Fusion* executa essa tarefa, aplicando uma função da biblioteca, que escolhe a melhor representação de cada elemento.

Por fim, é possível construir o resultado final do processo de limpeza dos dados através do operador *XML View*. Este operador pode também ser utilizado em fases intermédias do processo de limpeza de dados, sendo possível verificar os dados de saída dos diversos passos do processo.

Weis e Monolescu apresentam uma comparação entre a implementação de programas de limpeza de dados com recurso ao XClean/PL e a implementação utilizando XQuery [21]. Uma das métricas utilizadas para esta comparação foi o número de palavras de cada programa de limpeza de dados. Verificou-se que existe uma poupança do número de palavras utilizadas para escrever um programa de limpeza de dados com XClean/PL comparativamente com a escrita desse mesmo programa utilizando XQuery.

3.2 DogmatiX

Como já foi referido, a existência de dados duplicados é um dos problemas de qualidade de dados que mais ocorre, consequentemente é importante ter em consideração processos de deteção e remoção de duplicadas em programas de limpeza de dados. Uma forma de lidar com este problema é descrita em [22], onde os autores apresentam uma framework para detetar dados duplicados em qualquer modelo de dados e especificam, também, um algoritmo para lidar com a deteção de dados duplicados em XML.

3.2.1 Framework para Deteção de Dados Duplicados

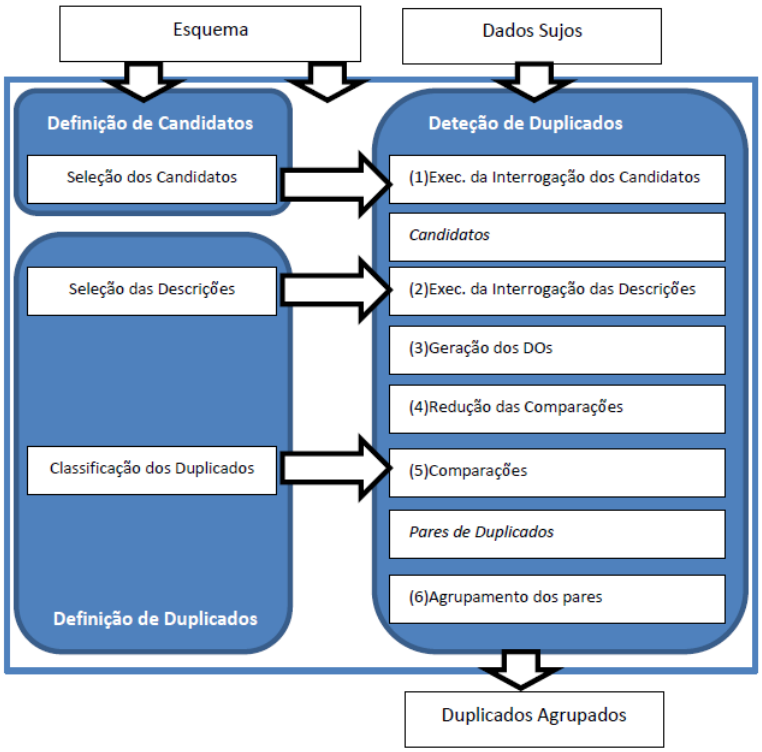

Figura 3.3: Uma framework para identificar registos duplicados [22].

A *framework* proposta (e representada na Figura 3.3), é constituída por três componentes principais: *(i) definição de candidatos*, *(ii) definição de duplicados*, e *(iii) deteção de duplicados*.

Na componente de *definição de candidatos* seleciona-se a informação relevante para a identificação de um objeto do mundo real. Assim é definida a informação necessária para verificar se dois elementos são representações do mesmo objeto. Uma fonte de dados pode armazenar informação acerca de vários tipos de objetos do mundo real mas nem toda ela necessita de ser considerada para detetar se dois objetos são na realidade o mesmo. Por exemplo, se quisermos comparar registos de uma base de dados, referentes a CDs de música, a fim de verificar a existência de duplicados, pode bastar fazer a comparação entre os valores dos artistas, dos títulos dos CDs e das suas faixas. Outra observação que deve ser tida em conta é que, de entre os elementos relevantes para a caraterização de um objeto, alguns podem estar representados de forma diferente, apesar de representarem o mesmo tipo de informação acerca do objeto. Estes elementos têm de ser comparados. Por exemplo, se tivermos a comparar referências discográficas de duas bases de dados distintas, em que numa os nomes dos seus produtores são dados por um atributo único *producer* e na outra os nomes dos produtores estão representados por

id	title	artist	track-title	track-composer
1	With The Beatles	Beatles	Hey Jude	P. McCartney
2	With Beatles	The Beatles	Hey Jude	P. McCartney
3	Live on Ten Legs	Pearl Jam	Yellow	E. Vedder

Tabela 3.2: Exemplo de dados XML, em conformidade com o esquema representado na Figura 3.4.

dois atributos distintos (*producer-fname*, *producer-lname*), é necessário que se faça a comparação entre os valores tendo em conta que as representações são diferentes. Da mesma forma, objetos que não representam o mesmo no mundo real não podem ser comparados, porque não podem ser duplicados. Por exemplo, para comparar referências discográficas, é irrelevante comparar títulos dos CDs com artistas.

Após a definição da informação que será utilizada para comparar os registos é realizado o processo de *Definição de Duplicados*. Nesta componente, a informação relevante é transformada num formato que facilita a comparação entre os elementos que caraterizam os objetos que serão comparados. Esta informação denomina-se por *Descrição de Objeto* (DO) e é definida no passo de *Seleção das Descrições*. A Tabela 3.2 apresenta um exemplo destas DOs, tendo em conta o esquema descrito pela árvore representada na Figura 3.4 e os dados contidos na Tabela 3.2, em que são escolhidos como informação relevante para a descrição dos CDs, o seu título e o seu artista.

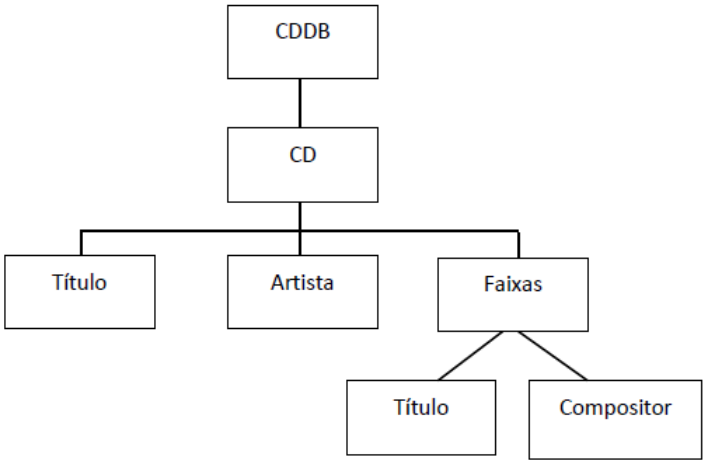

Figura 3.4: Ilustração de um esquema XML.

Ainda na componente de *Definição de Duplicados* é realizado o processo de *Classificação de Duplicados*. Neste passo, o utilizador especifica de que forma vai realizar as comparações entre os objetos candidatos a duplicados. Para isso é necessário definir classificadores que, através de funções de similaridade, definam se os objetos comparados são ou não duplicados.

Com base na informação gerada pelos componentes de *definição dos candidatos* e *definição dos duplicados* , a componente de *Deteção de Duplicados*, executa a deteção de duplicados de

id	DO
1	(With The Beatles, title), (Beatles, artist)
2	(With Beatles, title), (The Beatles, artist)
3	(Live on Ten Legs, title), (Pearl Jam, artist)

Tabela 3.3: Exemplos de descrições de objectos.

forma automática. Como se pode observar pela Figura 3.3, esta componente é constituída por vários passos. Em primeiro lugar, no passo de *Execução da Interrogação sobre os Candidatos*, os objetos candidatos a duplicados são extraídos. De seguida, no passo de *Execução da Interrogação das Descrições*, para cada candidato, os dados relevantes são selecionados para serem transformados na representação de DO (como a que é exemplificada na Tabela 3.3), que ocorre no passo seguinte de *Geração das DOs*. Depois de serem geradas as DOs, a informação necessária acerca dos candidatos a duplicados está finalmente disponível, e desta forma é possível começar a comparar os pares de objetos, no passo de *Comparação*. As comparações são realizadas de acordo com o classificador de duplicados, definido no passo de *Definição de Duplicados*. Se estivermos a lidar com um conjunto de dados grande é imprescindível reduzir o número de comparações entre os candidatos a duplicados. Esta redução é realizada no passo de *Redução de Comparações*. Por fim, todos os objetos referentes à mesma entidade do mundo real são agrupados no passo de *Agrupamento de Duplicados*.

3.2.2 Deteção de Objetos Duplicados em Documentos XML

A *framework* apresentada na Secção 3.2.1 foi desenhada para detetar dados duplicados independentemente do seu modelo. Nesta secção será apresentado o DogmatiX, uma concretização da framework apresentada anteriormente para deteção de dados duplicados em bases de dados XML.

A principal característica do DogmatiX, é a sua abordagem de seleção das descrições dos objetos independente do domínio. A seleção das descrições de cada objeto é efetuada com base na intuição de que para um elemento e, os elementos mais próximos de e na árvore que representa a estrutura do documento XML, providenciam informação mais relevante para esse elemento. Tendo em conta o esquema representado na Figura 3.4, temos, por exemplo que, o título providencia informação mais relevante na descrição de um CD do que o nome do compositor das faixas. Esta intuição leva-nos à definição de duas heurísticas utilizadas para selecionar as descrições dos objetos:

- **k-closest:** Considera como descrição de um elemento e os próximos k elementos a seguir a e.

- **r-distance:** Considera como descrição de um elemento e, todos os elementos cuja profundidade no XML Schema não difere mais do que o raio r do elemento e;

Na Figura 3.5, são ilustrados exemplos da utilização destas heurísticas. Na árvore à esquerda, é utilizado a heurística *k-closest*, com $k = 3$. Assim, o valor dos três elementos seguintes ao elemento CD são escolhidos. Na árvore à direita, é utilizada a heurística *r-distance*, com $r = 1$. Assim, os elementos que não excedem um nível de profundidade relativamente ao CD são escolhidos.

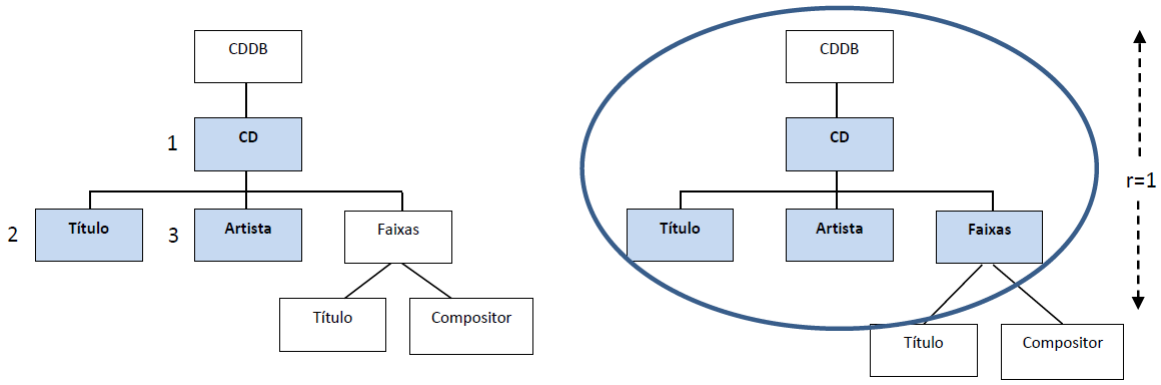

Figura 3.5: Heurísticas k-closest and r-distance [22].

Para além do DogmatiX, existem outras abordagens ao problema de deteção de dados em XML. Um dos métodos utiliza redes Bayesianas para a deteção de dados duplicados, baseando-se na suposição de que a probabilidade de dois elementos XML serem duplicados, depende da probabilidade dos seus valores e dos valores dos seus filhos serem duplicados [14].

Outro trabalho relacionado tem como principal objetivo reduzir o tempo de execução do processo de deteção de duplicados. Para isso reduz o número de comparações efetuadas, recorrendo a um processo de ordenação dos registos existentes na base de dados, de acordo com os seus conteúdos [17]. Desta forma, os elementos mais semelhantes ficam fisicamente mais próximos, numa lista ordenada. Assim, não é necessário comparar elementos que se encontrem afastados nesta lista, cabendo ao utilizador estipular o tamanho de uma janela que desliza ao longo da lista e que define os elementos que serão comparados entre si.

3.3 Fusão de Dados XML

O processo de fusão de dados estabelece regras para selecionar uma representação única de elementos duplicados. O processo de fusão de dados XML, explicado nesta secção e proposto

Estratégia	Tipo de Estratégia	Descrição
Pass It On	Ignorar	Mantém como valor todos os valores em conflito existentes
Consider All Possibilities	Ignorar	Combina todos os valores possíveis
Take the Information	Evitar	Escolhe os registos que contêm mais informação
No Gossiping	Evitar	Retorna somente registos coerentes
Trust Your Friends	Evitar	Escolhe o valor da fonte de dados mais fidedigna
Cry With the Wolves	Resolver	Escolhe o valor que está presente em maior número
Roll the Dice	Resolver	Escolhe um valor ao acaso de entre todos os que estão em conflito
Meet In the Middle	Resolver	Gera um valor novo que corresponde à média dos valores em conflito
Keep Up to Date	Resolver	Escolhe os dados mais recentes

Tabela 3.4: Estratégias para fusão de dados propostas em [3].

em [7], é baseado em outros trabalhos que descrevem métodos para fusão de dados relacionais. Estes assumem que os conflitos de dados podem ser incertezas ou contradições. Incertezas são conflitos entre um valor não nulo e um ou mais valores nulos. Para resolver este tipo de conflito, geralmente é escolhido o valor não nulo.

Contradições são conflitos de dados entre valores não nulos, o que torna a resolução deste tipo de problemas numa tarefa mais complicada de resolver. A Tabela 3.4, apresenta várias estratégias, propostas em [3], para a resolução da contradição de dados relacionais. Estas estratégias passam por *Ignorar*, *Evitar* ou *Resolver* o conflito. Ignorar um conflito significa que todos os valores contraditórios são apresentados, cabendo ao utilizador resolvê-lo manualmente. Evitar um conflito corresponde a escolher o valor de um dos registos em conflito (por exemplo, escolher o valor da fonte de dados supostamente mais fidedigna). Resolver um conflito corresponde a estabelecer um valor tendo em conta todos os outros registos e metadados das fontes de dados (por exemplo, estabelecer como valor o valor médio de todos os registos ou escolher o valor que foi alterado mais recentemente).

O modelo de fusão de dados XML proposto em [7] é baseado na definição de uma política de fusão que consiste num conjunto de regras para a resolução dos conflitos de dados. Este modelo é representado por um 5-tuplo $< R, T, K, P, L >$, onde:

- R é um conjunto de pares de candidatos a duplicados, onde cada candidato tem associado um grau de fiabilidade da fonte de onde proveio.
- T é o repositório dos dados consolidados.
- K é o conjunto de chaves que identificam cada elemento de T.

- P é o conjunto de regras que definem as estratégias para resolução de conflitos. Cada regra é um par $< \sigma, \Sigma >$ onde σ é uma expressão de caminho que representa o contexto que a regra cobre, e Σ é uma lista de estratégias para lidar com os conflitos. Por exemplo, se quisermos resolver um conflito sobre o ano do CD (como é apresentado na Listagem 3.1), a regra para a sua resolução poderá ser: *</cd[title = "With the Beatles"]/@year, [Keep Up to Date, Trust Your Friends]>*

  Se a primeira estratégia de Σ não consegue resolver o conflito, as estratégias seguintes são consideradas uma a uma, até que se consiga resolver o conflito ou a lista de regras seja totalmente percorrida. Estas estratégias são escolhidas pelo utilizador. Assim, se nenhuma regra de Σ consegue resolver o conflito, o utilizador pode iterar novamente sobre os dados com uma nova lista de estratégias.

```
<CD id = "dasd3nke" year = "1963">
  <title>With the Beatles</title>
  <artist>The Beatles</artist>
  <genre>Rock</genre>
</CD>
...
<CD id = "hgfb5bg" year = "1964">
  <title>With the Beatles</title>
  <artist>The Beatles</artist>
  <genre>Rock</genre>
</CD>
```

Listagem 3.1: Exemplo de dois registos duplicados em conflito.

- L armazena os dados descartados durante o processo de fusão e que não foram de encontro às estratégias escolhidas pelo utilizador. Desta forma, se o resultado final não for o expetável, é possível reconsiderar as escolhas que foram realizadas para resolução dos conflitos dos dados.

3.4 Discussão

Ao longo deste capítulo foram apresentados diferentes trabalhos relacionados com a limpeza de dados em XML. O *XClean* [21] é uma ferramenta para a limpeza de dados em XML. Os outros dois trabalhos apresentados, o *DogmatiX* [22] e o *XML Data Fusion* [7], pormenorizam, respetivamente, as tarefas de deteção de duplicados e de fusão de dados duplicados, incluídas nos processos de limpeza de dados. As maiores contribuições dadas pelo *DogmatiX* são os método para detetar duplicados, independentes do domínio dos dados utilizados e as heurísticas

apresentadas para a escolha dos elementos relevantes para comparar elementos candidatos a duplicados. O trabalho descrito em [7] descreve as várias estratégias para escolher os dados com melhor qualidade no processo de fusão de dados duplicados, de forma a que esta seleção dos dados possa ser realizada de forma automática.

3.5 Ferramentas

Existem, atualmente, algumas ferramentas capazes de realizar a limpeza de dados XML. As experimentadas no decorrer desta tese foram o XClean [21], o SSIS [1] e o Google Refine [2]. Foram ainda analisadas três ferramentas comerciais para limpeza de dados relacionais mas que aceitam documentos XML como fontes de dados (*Pentaho* [3] *, Informatica* [4] e *Trillium* [5]). Nestas ferramentas convencionais, o processo de limpeza de dados XML tem de ser precedido de uma conversão dos dados XML para o modelo relacional, de forma a se proceder à sua limpeza. Este passo extra, irá alterar a estrutura dos dados XML e a sua natureza hierárquica, o que se pode traduzir na perda de alguma informação relevante acerca dos dados. Estas três ferramentas não foram experimentadas pelo que a sua análise foi realizada através das suas especificações.

As ferramentas SSIS e Google Refine, apesar de aceitarem como dados de entrada documentos XML, apenas manipulam dados que se encontrem num modelo relacional. Como já foi referido, esta alteração do modelo dos dados pode criar algumas perdas de informação nos dados, como por exemplo a relação entre os dados na hierarquia que existe em documentos XML. Outro aspeto a ser tido em conta relativamente a estas duas ferramentas é o facto de serem pouco parametrizáveis. A impossibilidade de definir alguns parâmetros como as funções de similaridade a serem utilizadas e como os valores de *threshold* pode influenciar negativamente os resultados obtidos no processo de limpeza dos dados. A grande diferença existente entre estas duas ferramentas reside na impossibilidade de se criarem como dados de saída da ferramenta Google Refine, documentos XML, que podem ser criados com a ferramenta SSIS.

A ferramenta XClean e a utilização da linguagem XQuery com uma biblioteca de funções para limpeza de dados têm características mais semelhantes em relação à forma como tratam os dados. Estas duas abordagens manipulam diretamente os dados XML, não sendo necessária a transformação prévia de XML para o modelo relacional. O XClean e o XQuery têm bibliotecas que podem ser enriquecidas com novas funções, o que torna estas abordagens altamente

[1] http://msdn.microsoft.com/en-us/library/ms141026.aspx
[2] http://code.google.com/p/google-refine/
[3] http://www.pentaho.com/
[4] http://www.informatica.com/us/
[5] http://www.trilliumsoftware.com/home/solutions/enterpriseDQ/cleansing.aspx

	SSIS	**Google Refine**	**Pentaho**	**Informatica**	**Trillium**
Aceita XML como fonte de dados	Sim	Sim	Sim	Sim	-
Produz dados XML	Sim	Não	-	Sim	-
Transforma dados XML	Não	Não	-	Sim	-
Debugging	Sim	Não	-	-	Sim
Biblioteca de Funções	Sim	Sim	Sim	Sim	Sim
Interface Gráfica	Sim	Sim	Sim	Sim	Sim

Tabela 3.5: Comparação entre as diferentes ferramentas capazes de limpar dados XML.

parametrizáveis. Contudo, a linguagem XQuery (linguagem sobre a qual, os programas da ferramenta XClean são compilados), é uma linguagem menos eficiente que outras, como o Java ou C, pelo que o tempo de processamento de programas implementados em XQuery ou em XClean é claramente superior.

A Tabela 3.5 resume a comparação entre as ferramentas referidas neste capítulo.

Capítulo 4

Limpeza de Dados em XML

A maior parte do conhecimento científico referente à limpeza de dados foca-se nos dados relacionais. Quando os dados tratados são semi-estruturados, como é o caso dos dados armazenados em documentos XML, os processos para a deteção e correção dos seus problemas de qualidade são mais complexos. Para além dos problemas que podem surgir nos dados relacionais, como por exemplo, dados em falta, dados não normalizados ou dados duplicados, podem ainda existir outros referentes à inexistência da definição da estrutura do conjunto de dados. Sem a determinação da estrutura dos dados, estes são mais suscetíveis a problemas como a inconsistência da informação armazenada no documento XML, com a possibilidade de o mesmo tipo de objeto do mundo real a ter representações diferentes de elemento para elemento. Outra característica existente nos dados XML e que tem de ser tida em conta ao longo do seu processo de limpeza, é a existência de uma hierarquia que relaciona os elementos existentes e que dificilmente se pode traduzir no modelo relacional, sendo assim difícil utilizar as ferramentas que limpam dados relacionais para limpar dados XML.

O principal âmbito desta tese foca-se na criação de uma nova alternativa para a implementação de processos de limpeza de dados XML, recorrendo à linguagem XQuery. Antes de qualquer processo de limpeza de dados é necessário realizar uma auditoria aos dados para verificar problemas na sua qualidade. Para tal, foi desenvolvida uma taxonomia para a identificação de problemas de qualidade de dados XML e uma metodologia para resolvê-los tendo em conta as características existentes nos dados XML, referidas anteriormente. Da mesma forma que as ferramentas para limpeza de dados já existentes utilizam bibliotecas de funções que realizam tarefas de limpeza (como por exemplo, a normalização dos dados, a deteção de dados duplicados, a sua consolidação e o seu enriquecimento), foi criada uma biblioteca de funções que permite efetuar tarefas de limpeza sobre dados XML, com a linguagem XQuery. A existência de uma

extensão do XQuery com funções para limpeza de dados XML permite a reutilização de código comum a qualquer processo de limpeza de dados, tornando assim o XQuery, numa alternativa viável para a limpeza de dados XML.

Nesta secção, é descrita a taxonomia de problemas de qualidade XML utilizada no processo de limpeza de dados XML (Secção 4.1), bem como a metodologia adotada (Secção 4.2) e a biblioteca desenvolvida para facilitar a implementação de programas em XQuery para limpeza de dados XML (Secção 4.3). Na Secção 4.4, é apresentado o caso de estudo utilizado para a realização do processo de limpeza de dados XML em XQuery.

4.1 Taxonomia de Problemas na Qualidade de Dados

Nos últimos anos foram propostas diversas classificações de problemas de qualidade de dados que podem ocorrer em bases de dados relacionais. O objetivo destas taxonomias de qualidade de dados é compreender e classificar que problemas são passíveis de surgir numa base de dados, facilitando a sua identificação durante o processo de auditoria de qualidade de dados.

A forma como são classificados os problemas de qualidade de dados varia de taxonomia para taxonomia. Para Müller e Freytag [15], os problemas de qualidade de dados são classificados como anomalias sintáticas (como por exemplo, erros ortográficos ou erros no domínio dos valores), anomalias semânticas (como por exemplo, dados duplicados ou contraditórios) ou anomalias de cobertura (como são considerados os valores em falta). Esta classificação só cobre problemas de qualidade de dados existentes numa única fonte de dados. A taxonomia apresentada por Kim *et al.* [13] começa por dividir os problemas de qualidade de dados em dois conjuntos: os que têm dados em falta e aqueles que não têm dados em falta. Barateiro e Galhardas [2] fazem uma distinção entre os problemas de qualidade de dados ao nível da instância e os problemas ao nível do esquema dos dados. Por fim, nas taxonomias propostas por Oliveira *et al.* [16] e Rahm e Do [18], os problemas são divididos em problemas ocorridos numa fonte única de dados e problemas ocorridos em múltiplas fontes de dados. É a esta última forma de classificação que se aproxima a taxonomia que proponho nesta secção.

A taxonomia proposta diferencia os problemas que ocorrem em documentos XML com um único esquema daqueles que ocorrem em documentos com múltiplos esquemas. Esta divisão é feita, porque quando se trata de um conjunto de dados em que os seus conteúdos podem apresentar diferentes esquemas, outros problemas relacionados com a sua representação podem surgir (como por exemplo, elementos com diferentes estruturas ou elementos e atributos que apesar de representarem o mesmo tipo de objetos do mundo real, sejam definidos com nomes diferentes).

Problemas na Qualidade de Dados		Evitados pelo XSD (x.y.1)		Não Evitados Pelo XSD (x.y.2)	
		Docs XML c/ Esquema Único (I)	Docs XML c/ Múltiplos Esquemas (II)	Docs XML c/ Esquema Único (I)	Docs XML c/ Múltiplos Esquemas (II)
Valor (x.1)	Diferente Representação do Formato	✓	✓	-	-
	Diferentes Unidades de Medida	✓	✓	-	-
	Valor Ilegal	✓	✓	-	-
	Violação da Integridade da Entidade	✓	✓	-	-
	Erro Ortográfico	-	-	✓	✓
	Valor Incorreto	-	-	✓	✓
	Valores Embebidos	-	-	✓	✓
	Elemento/Atributo Mal Preenchido	-	-	✓	✓
	Valores Ambíguos	-	-	✓	✓
	Valor Mal Preenchido	-	-	✓	✓
Elemento (x.2)	Número de Elementos Inválido	✓	✓	-	-
	Dados em Falta	✓	✓	-	-
	Elementos com Diferentes Estruturas	✗	✓	-	-
	Conflito entre os Nomes	✗	✓	-	-
	Elementos Contraditórios	-	-	✓	✓
	Elementos Duplicados	-	-	✓	✓
	Violação da Dependência de Elementos	-	-	✓	✓
Atributo (x.3)	Dados em Falta	✓	✓	-	-
	Conflito entre os Nomes	✗	✓	-	-
	Violação da Dependência de Atributos	-	-	✓	✓

Tabela 4.6: Taxonomia de problemas na qualidade de dados em documentos XML.

Os problemas que ocorrem em documentos XML com um único esquema e os problemas que ocorrem em documentos com múltiplos esquemas são depois divididos em problemas ao nível do valor, ao nível do elemento e ao nível do atributo. Por fim, os problemas ao nível do valor, elemento e atributo são divididos em problemas que podem ser evitados pela definição do esquema do documento e em problemas que não são evitados pela definição do esquema, de uma forma semelhante ao que foi proposto por Barateiro e Galhardas na taxonomia de problemas de qualidade de dados relacionais, dividindo os problemas ao nível do valor da instância e ao nível do esquema [2]. A Tabela 4.6 ilustra a taxonomia que proponho.

I. Problemas que Ocorrem em Documentos com Esquema Único

I.1 Problemas ao Nível do Valor estão relacionados com o valor dos elementos e dos atributos do documento XML. Estes problemas podem ser divididos em problemas que podem ser evitados através da definição do esquema do documento XML (com recurso ao XML Schema Definition - XSD) e em problemas que a definição do esquema não evita.

No caso dos **problemas evitados pelo XSD (I.1.1)**, são considerados os seguintes:

– **Diferentes Representações do Formato:** quando num documento existem duas formas diferentes de representar o mesmo tipo de informação (por exemplo, o formato

de um preço pode ser representado por 3$50 ou por $3.50). Este problema pode ser evitado através do XSD, definindo expressões regulares que devem ser satisfeitas pelo valor.

– **Diferentes Unidades de Medida:** quando o mesmo tipo de informação é representado por diferentes unidades de medida (por exemplo, num elemento o preço pode ser apresentado em dólares enquanto que noutro, o preço pode ser apresentado em euros). Este problema pode ser evitado através do XSD, definindo expressões regulares que devem ser satisfeitas pelo valor.

– **Valor Ilegal:** quando um valor está fora dos valores possíveis de um dado domínio (por exemplo, data = 1989-13-32). Este problema pode ser evitado através do XSD, definindo o intervalo de valores a que o valor deve obedecer ou definindo os tipos de dados do atributo/elemento.

– **Violação da Integridade de Entidade:** quando um valor referente à chave é associado a mais do que um elemento. No exemplo ilustrado na Listagem 4.1, o valor do elemento *did* deveria ser único. Este problema pode ser evitado através do XSD, assinalando o elemento ou atributo como chave.

```
<cddb>
  <disc>
    <did>950ad90c</did>
    <artist>Pearl Jam</artist>
    <dtitle>Pearl Jam</dtitle>
    <category>rock</category>
    <genre>Grunge</genre>
    <year>1993</year>
  </disc>
  <disc>
    <did>950ad90c</did>
    <artist>Foo Fighters</artist>
    <dtitle>Foo Fighters</dtitle>
    <category>rock</category>
    <genre>Rock</genre>
    <year>1995</year>
  </disc>
</cddb>
```

Listagem 4.1: Exemplo de uma violação da integridade da entidade.

Em relação aos **problemas não evitados pelo XSD (I.1.2)**, esta taxonomia considera:

- **Erros Ortográficos:** quando o valor de um elemento ou atributo tem um erro ortográfico (por exemplo, "Beetles"em vez de "Beatles").
- **Valor Incorreto:** quando um valor é válido mas não satisfaz o valor real da entidade (por exemplo, o ano de lançamento de um CD dado como 1995 quando, na realidade, foi lançado em 1985).
- **Valores Embebidos:** quando dois tipos de dados diferentes estão embebidos num único valor (por exemplo, em "(track 1) - Alive"temos o número da faixa e o título do CD associados ao mesmo valor do elemento ou atributo).
- **Valor Mal Preenchido:** quando um valor é armazenado num campo incorreto (por exemplo, género = "Elvis Presley").
- **Valores Ambíguos:** quando os dados podem ser interpretados de várias formas. Esta situação pode ocorrer quando existe uma abreviatura (por exemplo, o nome “Homer J. Simpson” pode ser uma referência a “Homer Jeffrey Simpson” ou “Homer Jay Simpson”).
- **Valor em Falta num Elemento/Atributo Preenchido:** quando um elemento ou atributo está preenchido com um valor que não tem qualquer significado (por exemplo, ano = 9999).

I.2 Problemas ao nível do elemento são problemas que podem ocorrer em elementos de um documento XML. Estes podem ser os seguintes:

Problemas Evitados Pelo XSD (I.2.1)

- **Número Inválido de Elementos:** quando existe um número inválido de elementos (por exemplo, existir mais do que um título para um CD, quando deveria existir apenas um).
- **Dados em Falta:** quando um elemento que tem de estar necessariamente presente, não está.

Problemas Não Evitados Pelo XSD (I.2.2)

- **Dados Contraditórios:** quando existem vários elementos que representam o mesmo objeto do mundo real mas que contêm valores contraditórios. A Listagem 4.2 exemplifica este problema.
- **Elementos Duplicados:** Quando existem dois elementos diferentes que representam o mesmo elemento no mundo real. A Listagem 4.3 exemplifica este tipo de problema através de dois registos cujo título do cd é o mesmo, o nome do artista é o mesmo

(apesar de um estar escrito inteiramente em maiúsculas e o outro não), e parte das faixas correspondem.

```
<document>
  <disc>
    <did>950fg90c</did>
    <artist>Pearl Jam</artist>
    <dtitle>Pearl Jam</dtitle>
    <genre>Grunge</genre>
  </disc>
  <disc>
    <did>950cd78d</did>
    <artist>Pearl Jam</artist>
    <dtitle>Pearl Jam</dtitle>
    <genre>Rock</genre>
  </disc>
</document>
```

Listagem 4.2: Exemplo de elementos contraditórios num documento XML.

```
<cddb>
  <disc>
    <did>950ad90c</did>
    <artist>Pearl Jam</artist>
    <dtitle>Pearl Jam</dtitle>
    <tracks>
      <title>Go</title>
      <title>Animal</title>
      <title>Daughter</title>
    </tracks>
  </disc>
  ...
  <disc>
    <did>568bg67c</did>
    <artist>PEARL JAM</artist>
    <dtitle>Pearl Jam</dtitle>
    <tracks>
      <title>Go</title>
      <title>Animal</title>
      <title>Rearviewmirror</title>
    </tracks>
  </disc>
</cddb>
```

Listagem 4.3: Exemplo de dois elementos duplicados num documento XML.

– **Violação da Dependência de Elementos** quando dois ou mais elementos dependentes assumem valores que não satisfazem essa dependência. A Listagem 4.4 apresenta um elemento onde a idade não coincide com a data de nascimento.

Por outro lado, os **problemas ao nível do atributo (I.3)** são os problemas relacionados com os atributos de um documento XML. Estes podem ser:

Problemas Evitados pelo XSD (I.3.1)

– **Dados em Falta:** quando um atributo que deve ter um valor, não o tem.

```
<cddb>
  ...
  <artist name="J. Doe">
    <age>60</age>
    <bdate>21-12-1986</bdate>
  </artist>
  ...
</cddb>
```

Listagem 4.4: Exemplo de violação da dependência de dois elementos.

Problemas Não Evitados pelo XSD (I.3.2)

– **Violação da Dependência de Atributos:** quando dois ou mais atributos dependentes assumem valores que não satisfazem essa dependência. Na Listagem 4.5, é apresentado um exemplo deste problema, onde a idade não corresponde à data de nascimento.

```
<cddb>
  ...
  <artist name="J. Doe" age="60" bdate="21-12-1986"/>
  ...
</cddb>
```

Listagem 4.5: Exemplo de violação de dependências de dois atributos.

II. Problemas que Ocorrem em Documentos com Múltiplos Esquemas Os problemas de qualidade de dados existentes em documentos com um único esquema, descritos anteriormente, continuam a persistir quando consideramos documentos com múltiplos esquemas. Neste ponto do documento, apresento em detalhe apenas os problemas que podem ocorrer em documentos XML que possuem diversos esquemas.

Os problemas ao **nível do valor (II.1)** mantêm-se quando os documentos têm múltiplos esquemas. Em relação aos problemas ao **nível do elemento (II.2)** que ocorrem em documentos XML com múltiplos esquemas apenas acrescem aos problemas que ocorrem em documentos XML com um único esquema problemas **evitados Pelo XSD (II.2.1)**, que são os seguintes:

– **Conflitos entre os Nomes:** quando objetos diferentes são representados por etiquetas com o mesmo nome (sinónimos) ou quando o mesmo objeto é representado por etiquetas com nomes diferentes (homónimos).

```
<cddb>
  <disc>
    <did>950ad90c,a00ad90c,a40ad80c</did>
    <artist>Pearl Jam</artist>
    <dtitle>Pearl Jam</dtitle>
    <tracks>
      <track>
        <number>1</number>
        <title>Go</title>
      </track>
      <track>
        <number>2</number>
        <title>Animal</title>
      </track>
    </tracks>
  </disc>
</cddb>
```

Listagem 4.6: document1.xml - separa o número das faixas do seu nome.

```
<cddb>
  <disc>
    <did>950ad90c,a00ad90c,a40ad80c</did>
    <artist>Pearl Jam</artist>
    <dtitle>Pearl Jam</dtitle>
    <tracks>
      <track>1 - Go</track>
      <track>2 - Animal</track>
    </tracks>
  </disc>
</cddb>
```

Listagem 4.7: Ao contrário do document1.xml o número das faixas e o título constituem o mesmo elemento.

– **Elementos com Diferentes Estruturas:** quando um objeto é representado de formas diferentes em documentos com diferentes esquemas. Os exemplos ilustrados

na Listagem 4.6 e Listagem 4.7 representam dois documentos XML com diferentes esquemas mas que representam o mesmo objeto, com estruturas diferentes.

Em relação aos **problemas ao nível do atributo ocorridos em documentos XML com múltiplos esquemas**, estes são os mesmos que os que ocorrem em documentos com um esquema único, com adição do problema de **Conflitos entre os Nomes** que ocorre quando objetos diferentes são representados por um atributo com o mesmo nome (sinónimos) ou quando o mesmo objeto é representado por atributos com nomes diferentes (homónimos).

4.2 Metodologia de Limpeza de Dados XML

Numa base de dados relacional, um dos métodos para assegurar a sua limpeza, passa pela correção dos dados das camadas mais finas (ao nível dos atributos) para as camadas mais abrangentes (ao nível das relações). Assim, o processo de limpeza de dados relacionais é realizado através da correção dos valores dos atributos, numa primeira fase, seguida da correção dos tuplos, terminando com a correção das relações existentes na base de dados. Em cada fase da limpeza são asseguradas as tarefas de *correção*, que visam corrigir erros ortográficos, *normalização*, em que os elementos que representam o mesmo tipo de valor são colocados num formato comum, *deteção* e *eliminação de dados duplicados*, cujo resultado é a escolha de uma representação única para cada conjunto de dados duplicados e *enriquecimento dos dados*, em que é adicionada informação aos dados incompletos.

Dado que a hierarquia dos dados XML não é fixa (ao contrário dos dados relacionais que têm sempre três níveis de granularidade - atributos, tuplos e relações), a metodologia para a limpeza dos dados XML é diferente da metodologia para a limpeza de dados relacionais. Desta forma, considerando a forma hierárquica dos dados XML, em árvore, em que elementos, podem possuir outros elementos descendentes, é possível assegurar a limpeza dos elementos superiores (e, posteriormente, da base de dados XML), se os seus elementos descendentes e os seus atributos estiverem limpos.

Em cada passo destas iterações realizadas sobre os elementos, dos descendentes para os ascendentes, numa abordagem de baixo para cima na árvore que representa a base de dados XML, são realizadas as tarefas de correção, normalização, eliminação dos dados duplicados e enriquecimento dos dados.

A Figura 4.6 apresenta um exemplo da aplicação da metodologia de limpeza dos dados.

Utilizo como exemplo um conjunto de referências discográficas, em que cada referência tem

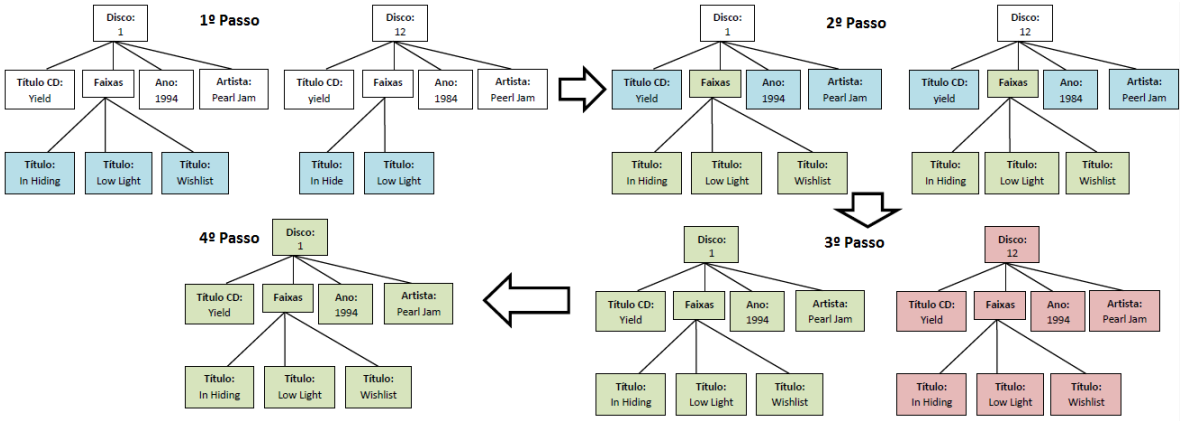

Figura 4.6: Exemplo da aplicação da metodologia de limpeza de dados XML.

informação acerca do seu título, do seu ano de lançamento, do artista que compôs o CD e o seu conjunto de faixas. Seguindo a metodologia apresentada, num primeiro passo deve ser corrigida a informação relativa ao conjunto de faixas que compõe o CD (i.e., os dados devem ser normalizados e os erros ortográficos devem ser corrigidos). Num segundo passo, a informação relativa aos artistas, aos títulos dos CDs e aos anos deve ser corrigida, normalizando os seus valores, e eliminando os elementos duplicados. Por fim, já com os seus elementos tratados procede-se à identificação e eliminação das referências discográficas duplicadas.

4.3 Biblioteca de Funções para Limpeza de Dados XML em XQuery

A principal proposta desta tese visa a implementação de um processo de limpeza de dados XML. Poder-se-iam implementar as tarefas de limpeza de dados através de linguagens de programação, como por exemplo o Java, ou de ferramentas para limpeza de dados, como por exemplo o SSIS da Microsoft [1]. Contudo, escolheu-se utilizar o XQuery, pois esta linguagem foi concebida para manipular diretamente dados XML, tendo em conta a sua estrutura. A utilização de ferramentas para limpeza de dados relacionais (que transformam dados XML em dados relacionais), tem como principal desvantagem o facto de estas alterarem a natureza hierárquica do XML, quando convertem os dados para o modelo relacional. Assim, com estas ferramentas, existe a possibilidade de perder alguma informação relevante dos dados que estão a ser limpos.

[1] http://msdn.microsoft.com/en-us/library/ms141026.aspx

As ferramentas existentes para a limpeza de dados são constituídas por conjuntos de operadores e bibliotecas de funções que facilitam as tarefas de limpeza, como a normalização e eliminação de dados duplicados. Assim, de forma análoga às ferramentas comerciais para limpeza de dados, foi também concebida uma biblioteca de funções de limpeza de dados XML para a linguagem XQuery.

A biblioteca de funções para limpeza de dados XML foi implementada sobre o processador de XQuery, *Zorba*, que suporta a versão mais atual desta linguagem (versão 3.0). Escolheu-se este processador de XQuery, face a outros processadores desta linguagem, porque esta tese tem sido desenvolvida no âmbito do projeto de desenvolvimento de software livre suportado pela *FLWOR Foundation* que tem como uma das suas contribuições a implementação do processador *Zorba*.

As funções estão organizadas de acordo com o seguinte conjunto de módulos[1]:

- **Semelhança entre Cadeias de Caracteres Baseada em Caracteres:** Este módulo providencia funções de semelhança que consideram as cadeias de caracteres como sequências de caracteres, atribuindo um valor que corresponde ao custo de transformar uma cadeia de caracteres na outra.
- **Semelhança entre Cadeias de Caracteres Baseada na Fonética:** Este módulo providencia funções de semelhança entre cadeias de caracteres, de acordo com a forma como estes são pronunciados.
- **Semelhança entre Cadeias de Caracteres Baseada em *Tokens:*** Este módulo providencia funções de semelhança entre cadeias de caracteres que são consideradas como conjuntos de *tokens*.
- **Semelhança entre Cadeias de Caracteres Híbrida:** Este módulo providencia funções para comparação de cadeias de caracteres, combinando as propriedades das funções de semelhança baseadas em *tokens* com as funções de semelhança baseadas em caracteres.
- **Definição de Similaridade:** Este módulo providencia funções que comparam a semelhança entre nós de um dado conjunto.
- **Consolidação:** Este módulo providencia funções para a consolidação dos dados, aplicando algumas regras sobre os dados de entrada que decidem qual a melhor representação de um dado elemento.

[1] http://www.zorba-xquery.com/html/modules/zorba/data-cleaning

- **Conversão:** Este módulo providencia funções para converter valores referentes a datas, valores temporais, moedas, unidades de medida, localizações geográficas, nomes e endereços postais.
- **Normalização:** Este módulo providencia funções para normalizar valores referentes a datas, valores temporais, moedas, unidades de medida, localizações geográficas, nomes e endereços postais.

4.4 Caso de Estudo

Todo o processo de limpeza de dados XML descrito ao longo deste capítulo teve o conjunto de dados CDDB [1] como principal caso estudado. Este é um conjunto de dados sintéticos que contém referências discográficas, ao qual foram adicionados problemas de qualidade de dados como duplicados, erros ortográficos, ou dados em falta. Cada referência discográfica contém informação relativa ao artista, ao título do CD, à categoria do CD, ao ano de lançamento do CD, ao seu género e os títulos das faixas. Algumas referências contêm, ainda, informação adicional como por exemplo o local de gravação do disco, ou o volume. A Listagem 4.8 apresenta o esquema destes dados.

Por forma a identificar os problemas existentes neste conjunto de dados, foi realizada uma tarefa de *profiling* de dados usando a taxonomia descrita na Secção 4.1. Os problemas identificados são apresentados na Tabela 4.7, sendo que para além da existência de referências discográficas duplicadas, existem também problemas ao nível do valor: diferentes representações do formato, erros ortográficos, valores ilegais, valores ambíguos e valores em falta em elementos preenchidos.

Com base nos problemas de qualidade de dados identificados, foi implementado um programa em XQuery, recorrendo à biblioteca de funções descrita na Secção 4.3, com o propósito de limpar estes dados de forma automática. O programa foi implementado com base na metodologia descrita na Secção 4.2. Assim, considerando o esquema em árvore dos dados CDDB, numa primeira tarefa, foram resolvidos os problemas dos nós folha: correção ortográfica e normalização dos dados relativos aos artistas, títulos do CD, categoria, género e títulos das faixas. A correção ortográfica foi realizada através de uma função de substituição de palavras com erros ortográficos (identificados no processo de avaliação da qualidade dos dados) pelas palavras corretas. Na função de normalização utilizada, os anos de lançamento foram todos transformados em números de 4 dígitos e os nomes dos artistas, os títulos dos cds e das faixas, as categorias

[1] http://www.hpi.uni-potsdam.de/naumann/projekte/repeatability/datasets/cd_datasets.html

e os géneros foram normalizados com uma função de capitalização de cadeias de caracteres.

```
<xsd:element name="disc">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="did" type="xsd:string" />
      <xsd:element name="artist" type="xsd:string" />
      <xsd:element name="dtitle" type="xsd:string" />
      <xsd:element name="year" type="xsd:int" />
      <xsd:element name="category" type="xsd:string" />
      <xsd:element name="genre" type="xsd:string" />
      <xsd:element name="cdextra" type="xsd:string" />
      <xsd:element name="tracks">
       <xsd:complexType>
          <xsd:sequence>
            <xsd:element maxOccurs="unbounded" name="title" type="xsd:string" />
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Listagem 4.8: XSD dos dados CDDB.

Numa segunda tarefa foram detetados e eliminados os problemas relacionados com a existência de artistas duplicados. Esta tarefa é processada da seguinte forma: em primeiro lugar, os valores duplicados são emparelhados, através da função de semelhança de cadeias de caracteres, *Jaro*; em segundo lugar, através de uma função de *clustering* (fecho transitivo), os valores aparentemente duplicados são agrupados em conjuntos de duplicados (*clusters*); por fim, é escolhida uma representação única destes valores duplicados através da função de consolidação *Most-Frequent* que escolhe como valor representativo o mais frequente entre os valores agrupados. Numa tarefa semelhante foram detetados e eliminados os problemas referentes aos discos duplicados. Nesta tarefa foi utilizada uma função de emparelhamento, em que são atribuídos pesos aos diferentes elementos que constituem o elemento CD. Os pesos atribuídos distinguem a importância que cada elemento tem na identificação de um CD. Desta forma, o conjunto de faixas é considerado o melhor elemento para distinguir um CD de outro, tendo por isso mais peso que os restantes (30%). Os artistas e o título, sendo também bons identificadores de um CD, têm também elevada importância na tarefa de emparelhamento de CDs (ambos com um peso de 25%). Esta função de emparelhamento de CDs, atribui ainda um peso de 10% ao ano de lançamento do CD, 5% ao género e também 5% à categoria. A função de semelhança utilizada para comparar os valores das faixas e do título do CD foi a função *Jaro*. As restantes

Nó XML	**Nível do Problema**	**Problema**	**Exemplo**
/cddb	-	-	-
/cddb/disc	Elemento	Dados Duplicados	Ver listagem 4.3
/cddb/disc/did	-	-	-
/cddb/disc/artist	Valor	Erro Ortográfico Dados Ambíguos	Ilse DeLangue Mozart, W.A.
	Elemento	Dados Duplicados	Mozart, W.A. / W.A. Mozart
/cddb/disc/dtitle	Valor	Erro Ortográfico Representação do Formato	Hibrid Theory Contrasts / contrasts
/cddb/disc/category	-	-	-
/cddb/disc/cdextra	Valor	Erro Ortográfico Representação do Formato	nEclipse YEAR 2004 / Year; 2004
/cddb/disc/year	Valor	Valor em Falta num Elemento Mal Preenchido Valor Incorreto Representação do Formato	9999 2070 60 / 1960
/cddb/disc/category	Valor	Representação do Formato	hiphop / Hip Hop
/cddb/disc/tracks	-	-	-
/cddb/disc/tracks/title	Valor	Valores Embebidos Valor Mal Preenchido Representação do Formato	Track 1 - Welcome / Welcome ? 1-Best Of You Best Of You

Tabela 4.7: Problemas de qualidade de dados identificados no conjunto de dados CDDB.

(género, ano de lançamento e categoria) foram comparadas através de igualdade dos valores. A função de consolidação utilizada para a escolha das representações únicas dos CDs duplicados foi a função *Most-Elements* que considera como elemento representativo o que tem mais mais elementos descendentes, ou seja, os que contem mais informação relativa ao CD.

O grafo que representa o fluxo do programa é apresentado na Figura 4.7. O código do programa encontra-se apresentado e explicado no anexo A.2.

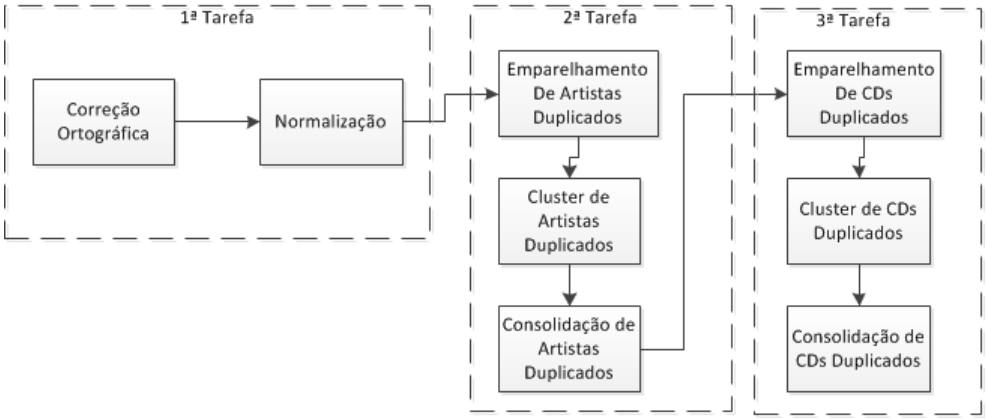

Figura 4.7: Grafo de transformações do programa implementado em XQuery para limpeza de dados XML.

Capítulo 5

Validação

Neste capítulo são descritas as experiências levadas a cabo para a validação da abordagem escolhida nesta tese para a limpeza de dados, ou seja um programa XQuery usando uma biblioteca de funções para limpeza de dados também desenvolvida em XQuery. A primeira experiência realizada teve como objetivo verificar qual o custo em termos de eficiência de se ter optado pela linguagem XQuery para o processo de limpeza de dados face à linguagem Java. A segunda experiência realizada compara as diferentes abordagens escolhidas para a implementação de um programa para limpar o conjunto de dados CDDB (XQuery, XClean e SSIS).

Este capítulo começa descrever as experiências realizadas sobre o XQuery e a biblioteca de funções XQuery para limpeza de dados XML, na Secção 5.1. Na Secção 5.2, é descrito o ambiente onde foram realizadas estas experiências, apresentando o *software* e o *hardware* utilizados. Na Secção 5.3, são apresentados os conjuntos de dados sobre os quais foram realizadas as experiências. Por fim, na Secção 5.4 são apresentados os resultados obtidos.

5.1 Experiências Realizadas

Realizaram-se duas experiências, em que numa primeira foi verificado o custo em termos de eficiência que advém da utilização da linguagem XQuery face à linguagem Java, relativamente à implementação de funções de semelhança entre cadeias de caracteres. A outra experiência realizada compara a implementação do programa de limpeza de dados XML, em XQuery com a implementação de programas sobre o mesmo conjunto de dados com o protótipo XClean e o software da Microsoft, SSIS, utilizando os mesmos critérios e funções.

Antes da realização destas experiências, já seria expectável que, na primeira experiência, a linguagem Java apresentasse tempos de execução mais reduzidos e que, na segunda experiência, o SSIS fosse a abordagem mais eficiente. Contudo, é relevante referir que estas experiências servem, também, para quantificar as diferenças em tempo de execução entre o XQuery e as restantes abordagens.

5.1.1 Experiência 1: Custo da Portabilidade

Uma das alternativas ao desenvolvimento da biblioteca de funções para limpeza de dados XML em XQuery, seria o seu desenvolvimento numa outra linguagem de programação como o Java ou C++. Contudo, perder-se-ia um dos principais objetivos da implementação da biblioteca de funções para limpeza de dados XML em XQuery, que é assegurar que esta é portável, podendo ser utilizada em qualquer processador XQuery que não exclusivamente o Zorba.

A portabilidade que é assegurada com o desenvolvimento da biblioteca em XQuery, apesar de constituir uma vantagem para a abordagem desenvolvida nesta tese, pode, contudo, prejudicar a sua eficiência comparativamente a bibliotecas de funções escritas em Java ou C++, uma vez que a linguagem XQuery é menos eficiente que as outras. Por forma a verificar o que se pode perder com a portabilidade, foi realizada a comparação entre o tempo de execução de algumas funções da biblioteca desenvolvida em XQuery com as suas versões implementadas em Java. As funções escolhidas para esta experiência tentam incluir o mais variado tipo de funções de semelhança de cadeias de caracteres (funções baseadas nos caracteres, baseadas nos tokens, híbridas e baseadas na fonética):

- **Funções de Semelhança Baseadas nos Caracteres:** Foram utilizadas as funções *Distância de Edição*, que devolve como resultado o número mínimo de transformações necessárias para que as cadeias de caracteres comparadas sejam iguais, *Jaro-Winkler*, que retorna um coeficiente baseado nas alterações necessárias para que as cadeias de caracteres comparadas sejam iguais, dando um peso à semelhança do início das cadeias de caracteres comparadas. Foi também utilizada a função *Needleman-Wunsch*, uma função semelhante à distância de edição mas à qual é atribuído um peso à operação de inserção ou remoção de um caracter no processo de transformação de uma cadeia de caracteres na outra que está a ser comparada.
- **Funções de Semelhança Baseadas em *tokens*:** Foram utilizadas as funções *Jaccard-tokens*, *Dice-tokens*, *Cosine-tokens*, que devolvem como resultado o coeficiente de semelhança entre os conjuntos de palavras extraídas das duas cadeias de caracteres comparadas.

- **Função de Semelhança Híbrida:** Foi utilizada a função de semelhança *Soft-Cosine-jaro-winkler*, que retorna o coeficente de semelhança Cosine entre os conjuntos de *tokens*, extraídos da cadeia de caracteres. Cada *token* é comparado através da função de semelhança Jaro-Winkler.

- **Função de Semelhança Baseada na Fonética:** Foi utilizada a função de semelhança *Soundex*, que retorna verdadeiro ou falso, de acordo com a semelhança das cadeias de caracteres tendo em conta a sua fonética.

5.1.2 Experiência 2: Comparação Entre as Diferentes Abordagens para Limpeza de Dados XML

Por forma a verificar a validade da biblioteca de funções para limpeza de dados em XQuery desenvolvida no decorrer desta tese, foi realizada uma comparação com outras abordagens existentes para a correção e limpeza de dados em XML. Outras duas formas de limpar dados XML de forma automática, são através da ferramenta SSIS da Microsoft e do protótipo XClean. A escolha do SSIS da Microsoft para realização desta experiência recai na facilidade do acesso a este software devido ao acordo entre o Instituto Superior Técnico e a Microsoft. O protótipo XClean foi o escolhido por ser o único protótipo limpar dados XML.

A experiência realizada para a comparação destas três abordagens, consistiu em identificar os artistas duplicados e criar um programa para limpar o conjunto de dados do documento CDDB. Foram medidos o tempo de execução de cada uma das abordagens, a complexidade do programa e a sua eficácia. A complexidade foi medida através do número de operadores necessários para a elaboração do programa em cada uma das abordagens. Uma vez que o conjunto de dados CDDB tem dados sujos gerados automaticamente, e que são conhecidos, é possível medir a eficácia de cada uma das abordagens utilizadas nesta experiência, verificando o número de pares de duplicados que foram mal identificados ou que não foram identificados.

5.2 Ambiente

As ferramentas utilizadas no processo de validação da biblioteca de funções para limpeza de dados em XML foram as seguintes.

- **XClean [21]:** O XClean, como já foi referido anteriormente nesta tese, é um protótipo de uma ferramenta de limpeza de dados XML, que trata dos problemas de qualidade de

dados dos documentos XML, através de programas escritos na linguagem da ferramenta, XQuery Program Language, que é compilada em XQuery. Assim, apesar do XClean ser apenas um protótipo, é relevante realizar a comparação entre a linguagem XQuery com a biblioteca de funções para limpeza de dados em XML e esta abordagem.

- **SQL Server Integration Services - SSIS**[1] **(v. 2008):** Esta plataforma da Microsoft é uma solução comercial concebida para transformação e integração de dados e serviços. Entre as diversas fontes de dados que podem ser tratadas com esta plataforma estão os dados XML, sendo, desta forma, uma possível solução a ser utilizada para resolver o problema da limpeza de dados em XML, abordado nesta tese.
- **Processador de XQuery Zorba**[2] **(v. 2.5.0):** Ainda que portável, a biblioteca de funções para limpeza de dados em XML desenvolvida no decorrer desta tese, foi implementada com recurso ao processador XQuery Zorba. A biblioteca de funções para limpeza de dados XML está integrada no processador de XQuery Zorba.

As características da máquina utilizada para realização das experiências (que serão apresentadas na Secção 5.3) são as seguintes: Processador Intel(R) Core(TM) i3 CPU M350 (4CPUs), 2.3GHz, com 4096MB de memória RAM e sistema operativo Windows 7 Profissional 64 bits.

5.3 Conjuntos de Dados

Esta secção descreve os conjuntos de dados utilizados nas experiências realizadas neste trabalho.

5.3.1 CDDB

Este é um conjunto de dados sintéticos, que contém referências discográficas, ao qual foram adicionados problemas de quailidade de dados como dados duplicados, erros ortográficos, ou dados em falta.

Cada referência discográfica é representada pelo elemento *disc* e contém a informação relativa ao artista (elemento *artist*), ao título do CD (elemento *dtitle*), à categoria do CD (elemento *category*), ao ano de lançamento do CD (elemento *year*) e ao seu género (elemento *genre*). O elemento *tracks* lista as faixas do CD, cujo título é representado pelo elemento *title*. Para além destes elementos, existe ainda um elemento, *cdextra*, que contém informação adicional,

[1] http://msdn.microsoft.com/en-us/library/ms141026.aspx
[2] http://www.zorba-xquery.com

como por exemplo o concerto onde foi gravado o CD (no caso dos CDs gravados ao vivo). O documento XML CDDB contém 10000 referências discográficas.

Este é o conjunto de dados utilizado na comparação de programas de limpeza de dados, desenvolvidos com três diferentes abordagens (SSIS, XClean e XQuery - os dois últimos com a biblioteca de funções de limpeza de dados). O seu esquema e os problemas identificados neste conjunto de dados, encontram-se detalhados na Secção 4.4.

5.3.2 *Match*

Para realizar a comparação entre algumas funções da biblioteca de limpeza de dados XML desenvolvida em XQuery com uma implementação em Java, foram utilizados ficheiros com cadeias de caracteres referentes a parques nacionais americanos (250 cadeias de caracteres) e nomes de pássaros (38 cadeias de caracteres). Estes conjuntos de dados, pertencentes ao conjunto de ficheiros denominado *Match*[1], foram criados para testar a biblioteca de funções de semelhança entre cadeias de caracteres, em Java, denominada *SecondString*.

A escolha destes dados tenta incluir cadeias de caracteres com características diversas, em termos de número de caracteres e de palavras. Como se pode observar pelas Figuras 5.8 a 5.11, as cadeias de caracteres do conjunto de dados referente a nomes de pássaros têm um tamanho inferior às cadeias de caracteres do conjunto de dados referente aos parques nacionais americanos. O número de palavras correspondentes aos nomes dos pássaros é também inferior ao número de palavras que correspondentes aos nomes dos parques nacionais americanos, que têm como valor máximo, respetivamente, 3 e 14 palavras.

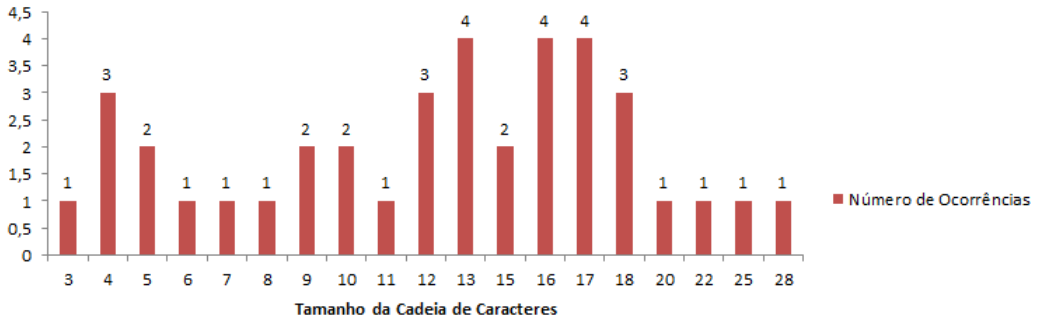

Figura 5.8: Tamanho em número de caracteres das cadeias de caracteres existentes no ficheiro que contém nomes de pássaros.

[1] http://www.cs.cmu.edu/ wcohen/data

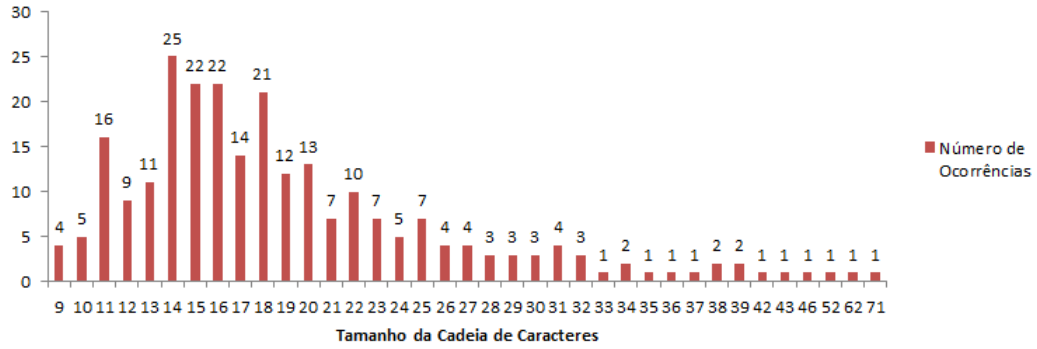

Figura 5.9: Tamanho em número de caracteres das cadeias de caracteres existentes no ficheiro que contém nomes de parques nacionais americanos.

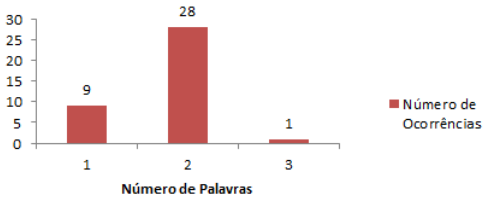

Figura 5.10: Tamanho em número de palavras das cadeias de caracteres existentes no ficheiro que contém nomes de pássaros.

5.4 Resultados

Nesta secção são apresentados os resultados das experiências realizadas neste trabalho. Na Secção 5.4.1, são demonstrados os resultados referentes à experiência que compara as funções de semelhança entre cadeias de caracteres escritas em XQuery com as funções escritas em Java. Os resultados obtidos na experiência que compara a implementação em XQuery com a biblioteca de funções para limpeza de dados XML com as outras abordagens possíveis são reportados na Secção 5.4.2.

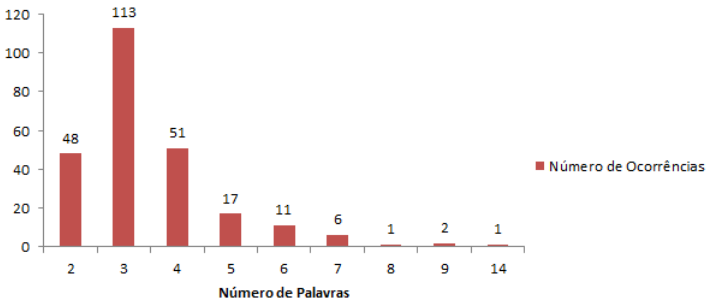

Figura 5.11: Tamanho em número de palavras das cadeias de caracteres existentes no ficheiro que contém nomes de parques nacionais americanos.

5.4.1 Experiência 1: Custo da Portabilidade

Os resultados apresentados nesta secção têm como objetivo demonstrar as consequências que advêm da escolha de uma biblioteca de funções portável para qualquer processador de XQuery.

Como é visível nos resultados descritos nas Figuras 5.13 e 5.14, os tempos de execução do XQuery são bastantes superiores aos tempos de execução das funções implementadas em Java, principalmente nas funções de semelhança entre cadeias de caracteres baseadas nos caracteres e nas funções de semelhança híbridas.

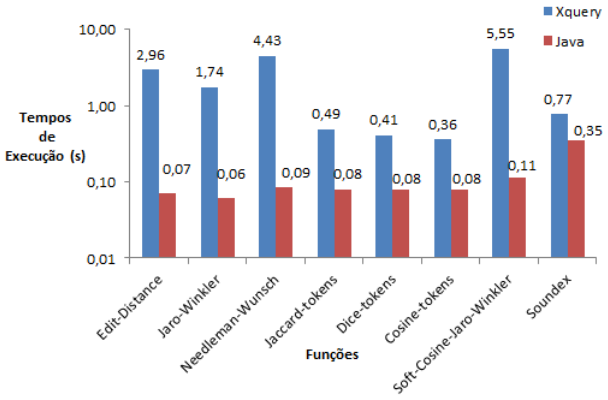

Figura 5.12: Tempo de execução das diferentes funções de similaridade sobre o conjunto de dados referente aos nomes de pássaros.

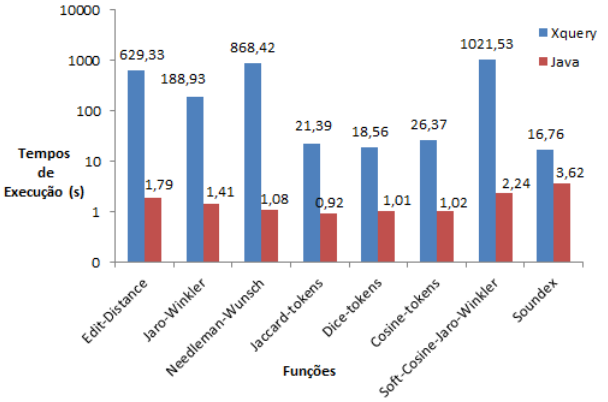

Figura 5.13: Tempo de execução das diferentes funções de similaridade sobre o conjunto de dados referente aos nomes de parques nacionais americanos.

As funções de semelhança de cadeias de caracteres baseadas em *tokens*, escritas em XQuery, apesar de atingirem tempos de execução semelhantes face às funções escritas em Java, podem chegar a atingir tempos 26 vezes superiores quando as cadeias de caracteres comparadas apresentam um maior número de palavras. Esta é uma diferença significativa, que se pode verificar pelos resultados obtidos sobre o conjunto de dados referentes aos nomes de parques nacionais, cujos caracteres contêm um maior número de palavras do que os dados referentes

aos nomes de pássaros (ver Figuras 5.10 e 5.11).

Os tipos de funções com uma performance mais semelhante são os que se baseiam na fonética das cadeias de caracteres, contudo a vantagem continua a recair sobre as funções escritas em Java que são em média quatro vezes mais rápidas que as funções escritas em XQuery, como se observa pelos tempos de execução da função *Soundex* indicados nas Figuras 5.12 e 5.13.

Assim, é possível concluir que a vantagem de se ter uma biblioteca que funcione para qualquer processador de XQuery, tem como ponto negativo a perda de eficiência demonstrada pelos superiores tempos de execução das funções implementadas em XQuery face às funções implementadas em Java.

5.4.2 Experiência 2: Comparação Entre as Diferentes Abordagens para Limpeza de Dados XML

Os resultados demonstrados nesta Secção focam-se na comparação entre três diferentes abordagens para a implementação de um programa de limpeza de dados XML: o XQuery com uma extensão de funções para limpeza de dados, o SSIS e XClean. Foram medidos os tempos de execução totais e parciais (apenas da deteção e correção dos artistas duplicados), a complexidade e a eficácia dos programas implementados. O grafo que representa o programa implementado em XQuery (Figura 4.7), encontra-se na Secção 4.4, e a implementação do programa SSIS é explicada no Anexo A.1.

Em relação aos tempos de execução verifica-se, pela Tabela 5.8, que a utilização do software SSIS é mais vantajosa, o que já seria expectável, uma vez que a linguagem XQuery é pouco eficiente comparativamente com a linguagem C#, sobre a qual está implementado o software SSIS. A complexidade do programa implementado com o SSIS, é contudo, superior. Esta comparação foi realizada tendo em conta os grafos de fluxo correspondentes aos programas implementados. Nota-se que o programa implementado com o SSIS (ver Figura A.14, do Anexo A.1) é mais complexo, pois necessita de mais operadores e transformações sobre os dados essencialmente para converter os dados XML para o modelo relacional. Como se verifica, no SSIS existe um operador para converter os dados XML para dados relacionais, *XML Source*. Estes dados são convertidos em três relações (*disc*, *tracks* e *title*), que necessitam de três operadores de ordenação essenciais para a junção destas três tabelas através de operações de *Merge Join*. Por fim, após a limpeza dos dados existe ainda um operador que converte os dados relacionais para dados XML. Com o XQuery estas operações não são necessárias uma vez que os dados XML são manipulados diretamente, diminuindo consideravelmente o grau de complexidade do programa.

	Tempo de Execução(s)	
Número de CDs	XQuery	SSIS
500	100.7	9.7
1000	459.3	17.0
2500	5198.8	41.8
5000	44679.8	106.0
10000	-	420.45

Tabela 5.8: Comparação dos tempos de execução, em segundos, entre o XQuery e o SSIS.

	Número de pares	
Número de CDs	XQuery	SSIS
500	7	12
1000	17	31
2500	22	42
5000	54	104
10000	-	135

Tabela 5.9: Número de pares de CDs duplicados não identificados ou mal identificados (XQuery vs SSIS).

Quando é comparada a eficácia do programa implementado no SSIS com a do programa implementado em XQuery (Tabela 5.9), verifica-se que na limpeza dos CDs do documento XML, o XQuery obtém melhores resultados. Contudo, a memória necessária para realizar o processo de limpeza dos dados com o XQuery é superior à memória utilizada pelo SSIS: O programa em XQuery não teve memória suficiente para corrigir o conjunto de dados com 10000 elementos.

Pelos resultados obtidos e reportados na Tabela 5.10, verifica-se que o programa implementado com o protótipo XClean é o menos vantajoso em termos de tempo de execução. O seu nível de complexidade é, contudo semelhante ao do programa XClean (como se pode verificar pelos programas implementados, nos Anexos A.2 e A.3). É importante referir que uma parte do tempo despendido na limpeza dos dados através do XClean é utilizada para a conversão do código escrito em XClean/PL para XQuery, atrasando assim o processo de limpeza de dados. Em relação à eficácia dos resultados (ver Tabela 5.11) verifica-se que as abordagens XClean e XQuery tomam valores bastante semelhantes.

As comparação realizada entre o XClean e o XQuery processa conjuntos de dados com apenas até 300 CDs. Esta redução do número de elementos deveu-se à incapacidade (em memória) por parte do programa escrito XClean de processar o conjunto de dados com 500 CDs.

	Tempo de Execução(s)	
Número de CDs	XQuery	XClean
50	7.3	18
100	19.4	36
200	135.2	152
300	358.5	389

Tabela 5.10: Comparação dos tempos de execução, em segundos, entre o XQuery e o XClean.

	Número de pares	
Número de CDs	XQuery	XClean
50	0	0
100	1	0
200	1	1
300	1	1

Tabela 5.11: Número de pares de CDs duplicados não identificados ou mal identificados (XQuery vs XClean).

Capítulo 6

Conclusões

Esta tese teve como tema principal a limpeza de dados XML. A limpeza de dados é um tema já com alguns tópicos de discussão relevantes no mundo científico, contudo, a maior parte destes tópicos refere-se à limpeza de dados em bases de dados relacionais. Apesar disso, alguns conceitos abordados na resolução do problema da limpeza de dados podem ser aplicados para qualquer modelo de dados. Assim, o conhecimento cientifico referente à limpeza de dados relacionais serviu de base para a definição de algumas contribuições realizadas nesta tese.

6.1 Sumário das Contribuições

Nesta tese, foram apresentadas as seguintes contribuições cientificas relacionadas com a limpeza de dados em documentos XML:

- **Taxonomia de Problemas na Qualidade de Dados:** A taxonomia de problemas na qualidade de dados foi criada de raiz para, no processo de auditoria dos dados XML, serem identificados os problemas de qualidade existentes. As taxonomias de qualidade de dados existentes classificam problemas de qualidade de dados do modelo relacional e não podem ser aplicadas em documentos XML devido às suas características, nomeadamente a sua estrutura hierárquica semi-estruturada.

  A taxonomia apresentada distingue os problemas que ocorrem em documentos XML com esquemas únicos dos que ocorrem em documentos XML com múltiplos esquemas, distingue os problemas ao nível do valor, ao nível do elemento e ao nível do atributo, e distingue, ainda dentro de cada um dos problemas anteriores, os que são evitados pelo XSD dos que

não são evitados pelo XSD.

- **Metodologia de Limpeza de Dados XML:** Foi proposta uma metodologia que permite a limpeza de dados XML tendo em conta a sua estrutura hierarquica, em árvore. Esta metodologia baseia-se num método *bottom-up*, dos nós filhos para os nós pais, em que se assegura que os nós superiores estão limpos se os seus nós filhos estiverem limpos. Assim, de forma recursiva dos nós filhos para os nós pais são realizadas tarefas de correção, normalização, deteção e eliminação de dados duplicados e enriquecimento dos dados que asseguram a limpeza dos nós superiores.

- **Biblioteca de Funções para Limpeza de Dados XML em XQuery:** Da mesma forma que as ferramentas para limpeza de dados existentes têm conjuntos de operadores e funções que facilitam o processo de limpeza de dados, foi criada uma biblioteca de funções para limpeza de dados XML em XQuery que foi utilizada na implementação do processo de limpeza de dados XML realizado nesta tese. A biblioteca inclui módulos de semelhança de cadeias de caracteres, de conversão e normalização de valores, de definição de semilaridade entre elementos do documento e de consolidação de dados, facilitando a implementação de programas de limpeza de dados.

  Esta biblioteca de funções para limpeza de dados XML é portável para qualquer processador de XQuery e o seu código é *open-source*.

- **Comparação entre XQuery e Java:** Neste trabalho optei por utilizar a linguagem XQuery para implementar o processo de limpeza de dados XML. As tarefas de identificação de valores duplicados foram realizadas com recurso aos módulos de semelhança de caracteres da bilbioteca de funções para limpeza de dados XML, escritas em XQuery. Estas funções escritas em XQuery têm como principal vantagem a sua portabilidade para diferentes processadores de XQuery, contudo as funções de semelhança de caracteres escritas em XQuery são menos eficientes do que as suas análogas escritas em Java. Desta forma, foi experimentada e reportada uma experiência que ajuda a perceber quanto se perde em termos de eficiência por se utilizar uma abordagem que passa pela utilização de funções escritas em XQuery face à utilização de funções escritas numa linguagem mais eficiente como o Java.

6.2 Trabalho Futuro

O trabalho realizado no decorrer desta tese de mestrado pode futuramente ser melhorado. O principal problema verificado durante as experiências realizadas foi a pouca eficiência em termos de tempo que as implementações em XQuery quando comparadas com implementações em Java. Desta forma um dos pontos que pode ser melhorado no futuro é a idealização e implementação de métodos capazes de optimizar a execução de programas escritos em XQuery.

Para além das experiências realizadas, outra experiência que poderia ser feita num trabalho futuro seria a implementação da biblioteca de funções de limpeza de dados embutida no processador de Xquery. No caso do processador Zorba, poder-se-iam implementar funções em C++, que provavelmente melhoraria a eficiência dos programas escritos em XQuery para limpar dados XML. Contudo, esta biblioteca deixaria de poder ser utilizada noutros processadores de XQuery, perdendo-se a sua portabilidade.

Seria também interessante realizar mais experiências com outros casos de estudo para além do CDDB e possivelmente alargar a comparação a outras ferramentes existentes.

Por fim, num trabalho futuro, poder-se-ia, ainda, continuar a melhorar a biblioteca de funções implementada, criando mais funções para os módulos existentes ou criando, por exemplo um módulo com funções para o enriquecimento dos dados, que não existe na biblioteca de funções e que é uma tarefa importante do processo de limpeza de dados.

Referências Bibliográficas

[1] Sihem Amer-Yahia, Chavdar Botev, Stephen Buxton, Pat Case, Jochen Dörre, Mary Holstege, Jim Melton, Michael Rys, and Jayavel Shanmugasundaram, *XQuery and XPath Full Text 1.0*, World Wide Web Consortium, May 2008.

[2] José Barateiro and Helena Galhardas, *A Survey of Data Quality Tools*, Datenbank-Spektrum **14** (2005), 15–21.

[3] Jens Bleiholder and Felix Naumann, *Conflict Handling Strategies in an Integrated Information System*, Workshop on Information Integration on the Web (IIWeb'06), held in conjunction with WWW 2006 Conference, 2006.

[4] Tim Bray, Jean Paoli, C. Michael Sperberg-McQueen, Eve Maler, and François Yergeau, *Extensible Markup Language (XML) 1.0 (Fifth Edition)*, World Wide Web Consortium, Recommendation REC-xml-20081126, November 2008.

[5] Peter Buneman and Wenfei Fan, *Constraints for semistructured data and XML*, SIGMOD Record **30** (2001), 200–1.

[6] Monica Caniupan, *Optimizing and implementing repair programs for consistent query answering in databases*, Tech. report, Ottawa, Ont., Canada, Canada, 2007, AAINR23289.

[7] Frantchesco Cecchin, Cristina Dutra de Aguiar Ciferri, and Carmem Satie Hara, *XML Data Fusion*, DaWak, 2010, pp. 297–308.

[8] Don Chamberlin, Jonathan Robie, Michael Dyck, and John Snelson, *XQuery 3.0: An XML Query Language*, (2013).

[9] Donald D. Chamberlin, Daniela Florescu, Jim Melton, Jonathan Robie, and Jérôme Siméon, *XQuery Update Facility 1.0*, World Wide Web Consortium, March 2011.

[10] Haitao Chen and Husheng Liao, *Integrity constraints for XML*, Software Engineering and Service Sciences (ICSESS), 2010 IEEE International Conference on, July 2010, pp. 331 –334.

[11] James Clark and Steve DeRose, *XML Path Language (XPath) version 1.0*, (1999), See `http://www.w3.org/TR/xpath.html`.

[12] Wenfei Fan, *XML constraints: Specification, analysis, and applications*, International Conference on Database and Expert Systems Applications (DEXA), 2005, pp. 805–809.

[13] Won Kim, Byoung-Ju Choi, Eui-Kyeong Hong, Soo-Kyung Kim, and Doheon Lee, *A Taxonomy of Dirty Data*, Data Mining Knowledge Discovery **7** (2003), 81–99.

[14] Luís Leitão, Pável Calado, and Melanie Weis, *Structure-based inference of XML similarity for fuzzy duplicate detection*, International Conference on Information and. Knowledge Management (CIKM), 2007, pp. 293–302.

[15] H. Müller and J.C. Freytag, *Problems, methods, and challenges in comprehensive data cleansing*, Tech. Report HUB-IB-164, Humboldt-Universität zu Berlin, Institut für Informatik, 2003.

[16] Paulo Oliveira, Fátima Rodrigues, Pedro Rangel Henriques, and Helena Galhardas, *A Taxonomy of Data Quality Problems*, 2nd Int. Workshop on Data and Information Quality (Porto, Portugal), June 2005, (in conjunction with CAiSE'05 conference), pp. 219–233.

[17] Sven Puhlmann, Melanie Weis, and Felix Naumann, *XML Duplicate Detection Using Sorted Neighborhoods*, International Conference on Extending Database Technology (EDBT), 2006, pp. 773–791.

[18] Erhard Rahm and Hong Hai Do, *Data Cleaning: Problems and Current Approaches*, IEEE Data Engineering Bulletin **23** (2000), 2000.

[19] Emanuel Santos, João Pavão Martins, and Helena Galhardas, *An Argumentation-based Approach to Database Repair*, European Conference on Artificial Inteligence (ECAI), 2010, pp. 125–130.

[20] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn, *XML Schema Part 1: Structures Second Edition*, World Wide Web Consortium, Recommendation REC-xmlschema-1-20041028, October 2004.

[21] Melanie Weis and Ioana Manolescu, *Declarative XML data cleaning with XClean*, Proceedings of the 19th international conference on Advanced information systems engineering (Berlin, Heidelberg), CAiSE'07, Springer-Verlag, 2007, pp. 96–110.

[22] Melanie Weis and Felix Naumann, *DogmatiX Tracks down Duplicates in XML*, ACM SIGMOD Int. Conference on Management of Data, Baltimore, Maryland, USA, June 14-16, 2005 (Fatma Özcan, ed.), ACM, 2005, pp. 431–442.

Apêndice A

Programas de Limpeza de Dados Implementados

A.1 SSIS

Neste anexo é explicado e ilustrado, através de um grafo representado na Figura A.14, o programa implementado com o SSIS para efetuar a limpeza de dados XML do conjunto de dados CDDB.

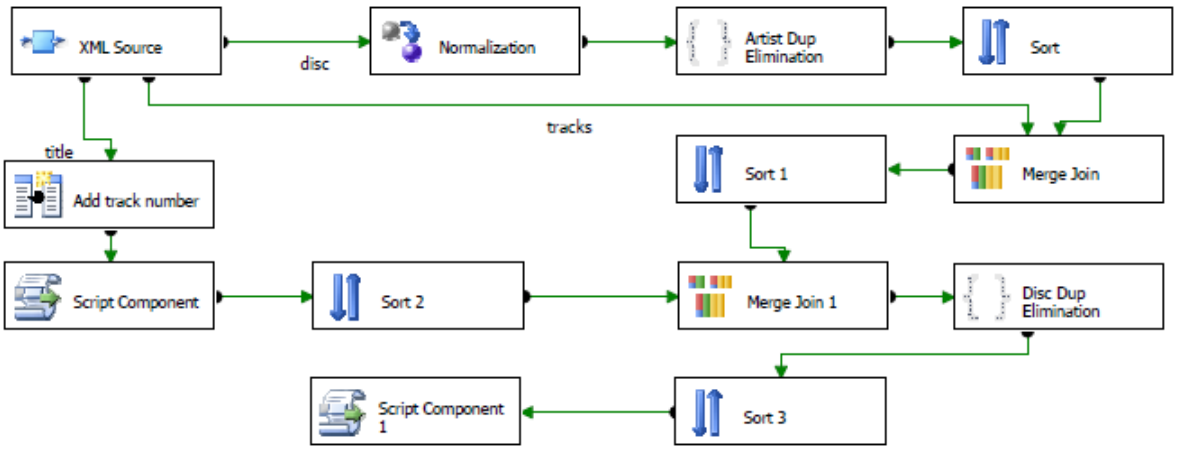

Figura A.14: Grafo que representa o fluxo do programa implementado com o software SSIS.

A primeira operação realizada pelo programa é a conversão dos dados XML para o modelo relacional através do operador *XML Source*. Nesta operação são criadas três tabelas: a tabela *disc*(artist, did, category, genre, cdtitle, category, cdextra, discID)a tabela *titles*(title, trackID) e a

tabela *tracks*(trackID, discID), que relaciona o identificador do elemento *disc*, com o identificador dos grupos de faixas.

De seguida, os valores dos elementos da tabela *disc* são normalizados através do operador *Normalization*, que utiliza uma função de capitalização sobre os valores dos artistas, dos títulos dos CDs e das categorias. Após esta normalização os artistas duplicados são identificados através do operador *Artist Dup Elimination*, que considera duplicados, os elementos que tenham um grau de semelhança superior ou igual a 80%, de acordo com a função de semelhança utilizada pelo SSIS (distância de edição). Este operador não elimina os dados duplicados. O seu resultado é a tabela *disc*, acrescida com uma coluna que representa os valores escolhidos para representar os grupos de artistas duplicados.

```
public void pexec(Input0Buffer Row, string value1){
   if (value != value1){
    count = 1;
    value = value1;
   }
}

public override void Input0_ProcessInputRow(Input0Buffer Row){
   if (count == 1){
    value = Row.tracksId.ToString();
    Row.tracknumber = count.ToString();
    count++;
   }
   else{
    pexec(Row, Row.tracksId.ToString());
    Row.tracknumber = count.ToString();
    count++;
   }
}
```

Listagem A.1: Código referente ao Script Component

Nesta fase do programa, ainda não existe nenhuma tabela que contenha toda a informação referente aos discos, uma vez que a fonte de dados foi dividida em três tabelas distintas. Desta forma, é necessário juntar estas tabelas através de uma operação de junção, para que se possa realizar a eliminação dos discos duplicados. A única opração de junção existente no SSIS é o *Merge Join*, que só pode ser aplicada sobre conjuntos de dados ordenados. Assim, é realizada uma primeira operação de *Sort*, que ordena os dados da tabela *disc* pelo valor do identificador dos discos. Com a tabela *disc* ordenada é possível realizar a operação de junção das tabelas *disc* e *tracks* - operador *Merge Join*. Em paralelo, são realizadas operações sobre a tabela *title*

que permitem a junção desta tabela com a nova tabela criada no operador *Merge Join*. Por forma a manter a ordenação das faixas, é adicionado o número da faixa à tabela *title*, através do operador *Add track number* - que cria uma coluna vazia chamada *trackNumber* - e através do operador *Script Component*, que preenche os valores desta coluna (ver Listagem A.1). Mais uma vez, para realizar a operação de junção das tabelas é necessário ordenar as tabelas *track* e a tabela resultante do operador *Merge Join*, desta vez pelo identificador dos conjuntos de faixas. As ordenações são realizadas nos operadores *Sort 1* e *Sort 2* e a junção é realizada no operador *Merge Join 1*.

```
public Boolean pexec(Input0Buffer Row, string value1)
    {
        //string value1 = Row.tracksId.ToString();
        if (value != value1)
        {
            xWriter.WriteEndElement();
            xWriter.WriteEndElement();
            count = 1;
            value = value1;
            return true;
        }
        else return false;
    }

    public override void Input0_ProcessInputRow(Input0Buffer Row)
    {

        if (OutputFileType == ".xml")
        {
            btime = Row.btime;
            if (count == 1)
            {
                value = Row.artist.ToString();
                xWriter.WriteStartElement("disc");

                if (!Row.dtitle_IsNull)
                {
                    xWriter.WriteStartElement("dtitle");
                    xWriter.WriteString(Row.dtitle.ToString());
                    xWriter.WriteEndElement();
                }
                if (!Row.artist_IsNull)
                {
                    xWriter.WriteStartElement("artist");
                    xWriter.WriteString(Row.artist.ToString());
```

```
                xWriter.WriteEndElement();
            }
            if (!Row.genre_IsNull)
            {
                xWriter.WriteStartElement("genre");
                xWriter.WriteString(Row.genre.ToString());
                xWriter.WriteEndElement();
            }
            if (!Row.category_IsNull)
            {
                xWriter.WriteStartElement("category");
                xWriter.WriteString(Row.category.ToString());
                xWriter.WriteEndElement();
            }
            if (!Row.year_IsNull)
            {
                xWriter.WriteStartElement("year");
                xWriter.WriteString(Row.year.ToString());
                xWriter.WriteEndElement();
            }
            xWriter.WriteStartElement("tracks");
            xWriter.WriteStartElement("title");
            xWriter.WriteString(Row.title.ToString());
            xWriter.WriteEndElement();
            count++;
        }
        else
        {
            if (!pexec(Row, Row.artist.ToString()))
            {
                xWriter.WriteStartElement("title");
                xWriter.WriteString(Row.title.ToString());
                xWriter.WriteEndElement();
                count++;
            }
        }
    }
}
```

Listagem A.2: Código referente ao Script Component 1

Criada a tabela que contém toda a informação relativa aos discos, procede-se à identificação e escolha de uma representação única dos discos duplicados. Este passo do programa é realizado no operador *Disc Dup Elimination*. A eliminação dos discos duplicados é realizada no operador

Sort 3, que para além de ordenar os discos pelo seu identificador elimina colunas duplicadas.

Após a transformação da tabela limpa num documento XML, pelo operador *Script Component 1* (ver Listagem A.2), o programa termina.

A.2 Programa XQuery

Neste anexo é apresentado e explicado o programa implementado em XQuery para limpeza de dados XML.

```
(:Functions:)

(:~
 : Takes as input an XML tree and corrects misspellings on
 :the values of some attributes/elements.
 :
 : @param $data The XML tree to be corrected.
 : @param $paths A set of XPath expressions refering the XML
 :elements/attributes to be corrected.
 : @param $wrong-values-dict A set of misspelled values.
 : @param $replaced-values-dict A set of corrected values.
 : @return The corrected data.
 :)
declare function spell:correction ($data as node(), $paths as xs:string*,
$wrong-values-dict as xs:string*, $replaced-values-dict as xs:string*) as node(){
 copy $doc := $data
 modify (
  for $n in $paths , $nv in refl:eval(concat ("$doc/", $n))
  let $text := lower-case($nv/text())
  where $text = $wrong-values-dict
  return replace value of node $nv
     with $replaced-values-dict[index-of($wrong-values-dict, $text)[1]]
 )
 return $doc
};


(:~
 : Normalizes the year values from a 2 character string to a 4 character string.
 :
 : @param $year The value to be corrected.
 : @return The normalized year.
 :)
declare function norm:year-normalization ($year as xs:string*) as xs:string*{
  if ( string-length ($year) <= 2 and $year != "")
  then
    if (number($year) > number(substring(string(year-from-date(current-date())), 3, 2)))
    then concat ("19", $year)
    else concat ("20", $year)
  else $year
```

```
};

(:~
 : Capitalizes a given string.
 :
 : @param $string The value to be capitalized.
 : @param $stop-word The list of stop-words (words that are not capitalized).
 : @return The string capitalized.
 :)
 declare function norm:capitalize ( $string as xs:string,
        $stop-words as xs:string*) as xs:string*{
  let $sw :=
    if (empty($stop-words))
    then ("a", "an", "the", "but", "as", "if", "and", "or", "nor", "of")
    else $stop-words
  let $ttokens := tokenize ($string, "␣")
  let $cap-string := concat(
      upper-case(substring($ttokens[1], 1, 1 ) ),
      lower-case(substring($ttokens[1], 2) ),
      "␣",
      string-join(
         for $tok in $ttokens[ position() > 1 ]
         return
         if (exists(index-of($sw, lower-case($tok))))
         then
         concat(lower-case($tok), "␣")
         else
         concat(
           upper-case(substring($tok, 1,1) ),
           lower-case(substring($tok, 2) ),
           "␣"
         )
      )
   )
  return substring($cap-string, 1, string-length($cap-string)-1 )
};

(:~
 : Takes as input an XML Tree and normalizes the values of its elements/attributes.
 :
 : @param $data The data to be normalized.
 : @return The data normalized.
 :)
declare function norm:normalization ($data as node()) as node(){
copy $doc := $data
```

```
modify(
  let $stop-words := ("a", "an", "the", "but", "as", "if", "and", "or", "nor", "of")
  for $disc in $doc//disc
   for $node in $disc//descendant::*
   where string(node-name($node)) != 'tracks'
    return if (string(node-name($node)) = 'year')
    then replace value of node $node
         with norm:year_normalization(string-join($node/year/text()))
    else replace value of node $node
         with norm:capitalize(string-join($node/text()), $stop-words)
)
return $doc
};

(:~
 : Identifies the similarity value of two duplicate candidate artists using jaro-winkler.
 :
 : @param $a1 The value of an artist.
 : @param $a2 The value of an artist.
 : @return Similarity value.
 :)
declare function dup-elimination:sim-artists ( $a1, $a2) {
  return if ($a1 = $a2) then 1 else simc:jaro($a1,$a2)
};

(:~
 : Identifies the similarity value of two duplicate candidate CDs.
 :
 : Weights assinged to the different elements:
 : genre -> 5%
 : category -> 5%
 : dtitle -> 25%
 : year -> 10%
 : artist -> 25%
 : track -> 30%
 :
 : @param $elm1 The value of an artist.
 : @param $elm2 The value of an artist
 : @return Similarity value.
 :)
declare function dup-elimination:sim-discs ( $elm1, $elm2 ) {
 let $genre-weight  := 0.05
 let $cat-weight    := 0.05
 let $dtitle-weight := 0.25
 let $year-weight   := 0.1
```

```
 let $artist-weight := 0.25
 let $tracks-weight := 0.3
 let $titles-filter := 4
 let $sim-genre     := if ( $elm1//genre = $elm2//genre) then $genre-weight else 0
 let $sim-category  := if ( $elm1//category = $elm2//category) then $cat-weight else 0
 let $sim-dtitle := simc:jaro($elm1//dtitle/text(), $elm2//dtitle/text())*$dtitle-weight
 let $sim-year      := if ( $elm1//year = $elm2//year) then $year-weight else 0
 let $sim-artist    := if ( $elm1//artist = $elm2//artist) then $artist-weight else 0
 let $sim-tracks    :=
   let $sim :=
   for $t1 at $p in $elm1//title/text()
   return max(
     for $t2 in $elm2//title/text()
     where abs(string-length($t1) - string-length($t2)) <= $titles-filter
     return simc:jaro($t1, $t2)
   )
   return if(count($sim) = 0) then 0 else (sum($sim) div count($sim) * $tracks-weight)
  return $sim-genre + $sim-year + $sim-dtitle + $sim-category + $sim-artist + $sim-tracks
};


(:~
 : Matches elements of two given XML trees.
 :
 : @param $e1 Set of elements to be matched.
 : @param $e2 Set of elements to be matched.
 : @param $t The threshold value.
 : @param $similarity-function The function to be applied in the pairng of the elements.
 : @return Pairs of duplicate elements and mismatch elements.
 :)
declare function dup-elimination:match ( $e1, $e2 , $t as xs:double,
$similarity-function ) {
let $match-pairs :=
for $pos1 in 1 to count($e1), $pos2 in $pos1 + 1 to count($e2)
let $elm1 := $e1[$pos1]
let $elm2 := $e2[$pos2]

let $sim := $similarity-function( $elm1, $elm2)
where $sim > $t
return <match>
<dup-elimination:elm pos = "{$pos1}">{$elm1}</dup-elimination:elm>
<dup-elimination:elm pos = "{$pos2}">{$elm2}</dup-elimination:elm>
<sim>{$sim}</sim>
</match>
```

```
let $mismatch-positions1 :=
  for $pos in 1 to count($e1)
  let $positions := distinct-values(data($match-pairs//dup-elimination:elm/@pos))
  return
    if($pos = $positions)
    then ()
    else $pos
let $mismatch1 :=
  for $p in $mismatch-positions1
  return
   <match>
    <dup-elimination:elm pos = "{$p}">{$e1[$p]}</dup-elimination:elm>
   </match>

return $match-pairs | $mismatch1
};


(:~
 : Applies the transitive-closure on the matched elements.
 :
 : @param $match The matched elements.
 : @return Clustered elements.
 :)
declare function dup-elimination:transitive-closure ($match){
variable $set1 as node()* := $match;
variable $set2 as node()* :=  ();
variable $distinct-values as xs:integer :=
  fn:count(set:distinct(data($set1//dup-elimination:elm/@pos)));

while (count($set1//dup-elimination:elm/@pos) != $distinct-values) {
   $set2 :=
     set:distinct(
     for $p1 in 1 to count($set1), $p2 in $p1+1 to count($set1)
      return
       if(data($set1[$p1]/dup-elimination:elm/@pos)
           = data($set1[$p2]/dup-elimination:elm/@pos))
       then
        <match>
         {set:deep-union($set1[$p1]/dup-elimination:elm, $set1[$p2]/dup-elimination:elm)}
        </match>
       else ()
            );
   $set1 := ($set2,
     (for $p1 in 1 to count($set1)
```

```
      return
      if (not(data($set1[$p1]//dup-elimination:elm/@pos)
              = data($set2//dup-elimination:elm/@pos)))
      then $set1[$p1] else()));
}
$set1
};

(:~
 : Applies a function that clusters the given data.
 :
 : @param $data The data to be clustered.
 : @param $clustering-function The clustering function to be applied on the data.
 : @return Clustered elements.
 :)
declare function dup-elimination:cluster ($data, $clustering-function){
  $clustering-function($data)
};

(:~
 : Applies a function the merges the given elements.
 :
 : @param $elements The elements to be merged.
 : @param $consolidation-function The consolidation function to be applied on the data.
 : @return The new values.
 :)
declare function dup-elimination:merge ($cluster, $consolidation-function){
  $consolidation-function($cluster)
};


(:~
 : Removes the duplicate artists.
 :
 : @param $data The data with duplicate artists.
 : @param $t The threshold value that imposes which are the duplicate artists.
 : @return The data without duplicate artists.
 :)
declare function dup-elimination:artist-dups-elimination ($data as node(),
$t as xs:double){
copy $doc := $data
modify(
  let $artists :=
    for $art in $doc//artist
    return
```

```
      if (not(exists($art/text()))) then "␣"
      else ($art/text())
  let $match :=
    dup-elimination:match($artists, $artists ,$t, dup-elimination:sim-artists#2)
  let $clusters :=
    dup-elimination:cluster($match, dup-elimination:transitive-closure#1)
  let $dup-elements :=
    set:distinct($clusters//dup-elimination:elm)
  for $dup in $clusters//dup-elimination:elm
  let $replace-value :=
    dup-elimination:merge (data($dup/..//dup-elimination:elm), con:most-frequent#1)
  return
    if (fn:exists($doc//disc[position() = number($dup/@pos)]//artist))
    then replace value of node $doc//disc[position() = number($dup/@pos)]//artist[1]
         with $replace-value
  else ()
 )
return $doc
};

(:~
 : Removes the duplicate CDs.
 :
 : @param $data The data with duplicate CDs.
 : @param $t The threshold value that imposes which are the duplicate CDs.
 : @return The data without duplicate CDs.
 :)
declare function dup-elimination:cd-dup-elimination ($data as node(),
$t as xs:double) as node()*{
copy $doc := $data
modify(
  let $match :=
    dup-elimination:match ($data//disc, $data//disc, $t, dup-elimination:sim-discs#2)
  let $clusters :=
    dup-elimination:cluster($match, dup-elimination:transitive-closure#1)
  let $elements-to-leave :=
    for $dups in $clusters
    return dup-elimination:merge ($dups//dup-elimination:elm, con:most-elements#1)
  for $disc in $doc//disc
  where not(empty($disc//did/text())) and
        not(exists(index-of($elements-to-leave//did/text(), $disc//did/text()))) and
        not(empty($disc))
  return delete node $disc
)
return $doc
```

```
};

(:~
 : Removes duplicates existing in a dataset.
 :
 : @param $data Duplicate data.
 : @param $t The threshold value that imposes which are the duplicate values.
 : @param $dup-elimination-function The function that elimnates duplicate.
 : @return The data without duplicate values.
 :)
declare function dup-elimination:dup-elimination ($data, $t, $dup-elimination-function){
 $dup-elimination-function($data, $t)
};

(:Data cleaning program:)
let $data := doc("cddb.xml")
let $no-misspellings :=
  spell:correction ($data, ("/disc/artist", "/disc/dtitle"),
                   ('hibrid␣theory', 'va', 'various', 'v.a', 'v.a.'),
                   ('Hybrid␣Theory', 'Various␣Artists', 'Various␣Artists',
                            'Various␣Artists', 'Various␣Artists'))
let $normalized-data := norm:normalization ($no-misspellings)
let $no-art-dup-data :=
  dup-elimination:dup-elimination($normalized-data,
                                  0.8, dup-elimination:artist-dups-elimination#2)
let $no-dups :=
dup-elimination:dup-elimination ($no-art-dup-data,
                                  0.8, dup-elimination:cd-dup-elimination#2)
return $no-dups
```

Listagem A.3: Programa para limpeza de dados XML em Xquery

O programa de limpeza de dados implementado em XQuery possui quatro blocos de tarefas de correção dos problemas de qualidade de dados do conjunto de dados CDDB: Correção Ortográfica, normalização dos dados, eliminação dos artistas duplicados e eliminação dos CDs duplicados.

Na tarefa de correção ortográfica dos dados, são alterados os valores dos elementos *artist* e *dtilte* que correspondem a cadeias de caracteres que contêm erros ortográficos. Estes valores errados foram identificados através de uma análise manual dos dados e são corrigidos através da função *spell:correction*.

A normalização dos dados realizada através da função *norm:normalization*, normaliza os valores dos elementos *artist*, *dtitle*, *genre*, *title*, *year*. Os elementos *artist*, *dtitle*, *genre*, *title* sofrem um processo de transformação que capitaliza os seus valores através da função *norm:capitalize*. O valor dos elementos *year* é normalizado através da função *norm:year-normalization*, que coloca os valores representados com quatro dígitos.

Na tarefa de eliminação de artistas duplicados é realizado, numa primeira fase, o emparelhamento dos valores duplicados, através da função *dup-elimination:match*, que através da função *jaro*, emparelha os artistas cujo coeficiente de semelhança é superior ou igual a 80%. Numa segunda fase, os pares identificados através da função *dup-elimination:match* são aglomerados em *clusters*, através de uma função que aplica o fecho transitivo, *dup-elimination:transitive-closure*. Para escolher uma representação única para cada *cluster*, é aplicada a função de consolidação *con:most-frequent*, que escolhe como valor representante o mais frequente.

Por fim, a tarefa de eliminação dos CDs duplicados é realizada de forma semelhante à tarefa de eliminação dos artistas duplicados. A grande diferença prende-se pela função de emparelhamento, que no caso dos CDs atribui pesos aos diferentes elementos que o constituem, de acordo com o seu grau de importância para a identificação de um CD. Desta forma, o peso atribuido ao valor do género e da categoria do CD é 5%, o peso atribuído ao valor do título e do artista do CD é 25%, o peso atribuído ao ano de lançamento do CD é de 10% e o peso atribuído ao conjunto das faixas é de 30%. Dois CDs são considerados duplicados se o resultado da função do seu emparelhamento for superior a 80%. A função de consolidação utilizada para a escolha das representações únicas dos CDs duplicados, é a função *con:most-frequent*.

A.3 Programa XClean

Neste anexo é descrito e apresentado o programa de limpeza de dados XML implementado com o XClean

```
XCLEAN
CANDIDATES
{ doc("cddb.xml")/cddb/disc} INTO $cdCands SCHEMA cdCand
;

SCRUB $cd IN {$cdCands}
WITH xcl:spell-correction($cd.cdCand/disc/artist/text())
AS scArtist
INTO $scrubbedCD1;

SCRUB $cd IN {$scrubbedCD1}
WITH xcl:spell-correction($cd.cdCand/disc/dtitle/text())
AS scDtitle
INTO $scrubbedCD2;

SCRUB $cd IN {$scrubbedCD2}
WITH xcl:capitalize($cd.cdCand/disc/artist/text())
AS nArtist
INTO $scrubbedCD3;



SCRUB $cd IN {$scrubbedCD3}
WITH xcl:capitalize($cd.cdCand/disc/dtitle/text())
AS nDTitle
INTO $scrubbedCD4;

SCRUB $cd IN {$scrubbedCD4 }
WITH xcl:scrubDateTo4Digits($cd.cdCand/disc/year/text())
AS nYear
INTO $scrubbedCD5;

SCRUB $cd IN {$scrubbedCD5 }
WITH xcl:capitalize($cd.cdCand/disc/genre/text())
AS nGenre
INTO $scrubbedCD6;

FILTER PAIRS ($cd1 AS cd1, $cd2 AS cd2) OF {$scrubbedCD6 } INTO $cdDups
WHERE {

        (( xcl:jaro($cd1.nArtist/text(), $cd2.nArtist/text()) * 0.25)
```

```
        + ( xcl:jaro($cd1.nDTitle/text(), $cd2.nDTitle/text()) * 0.25)
        + ( xcl:jaro($cd1.nYear/text(), $cd2.nYear/text()) * 0.1)
        + ( xcl:jaro($cd1.nGenre/text(), $cd2.nGenre/text()) * 0.05)
        + ( xcl:jaro($cd1//category/text(), $cd2//category/text()) * 0.05)
        + ( xcl:tracks-sim($cd1//title/text(), $cd2//title/text())* 0.3)) >= 0.80
        };

XVIEW TRANSFORM
$pair IN {$cdDups}
INTO $newDups
BY
(
{
        let $cd1 := $pair.cd1
        return <disc>
        \{
                $cd1->cdCand/disc/@xcid,
                <artist>\{$cd1->nArtist/text()\}</artist>,
                <dtitle>\{$cd1->nDTitle/text()\}</dtitle>,
                <year>\{$cd1->nYear/text()\}</year>,
                <genre>{$cd1->nGenre/text()}</genre>,
                <category>{$cd1->cdCand/disc/category/text()}</category>,
                <tracks>\{$cd1->cdCand/disc/tracks/element()\}</tracks>
        \}
        </disc>

} AS cd1,

{
    let $cd2 := $pair.cd2
    return <disc>
        \{
                $cd2->cdCand/disc/@xcid,
                <artist>\{$cd2->nArtist/text()\}</artist>,
                <dtitle>\{$cd2->nDTitle/text()\}</dtitle>,
                <year>\{$cd2->nYear/text()\}</year>,
                <genre>{$cd2->nGenre/text()}</genre>,
                <category>{$cd2->cdCand/disc/category/text()}</category>,
                <tracks>\{$cd2->cdCand/disc/tracks/element()\}</tracks>
        \}
        </disc>
} AS cd2

);
```

```
CLUSTER CLASSIFICATION USING xcl:cddbTC({$newDups})
INTO $cdClusters SCHEMA cdClust;

FUSE CLUSTERS USING xcl:most-frequent({$cdClusters} )
INTO $fusedCDs SCHEMA origCD, cdFused
```

Listagem A.4: Programa de limpeza de dados XML com XClean

O programa implementado em XClean para a limpeza dos dados CDDB, começa por extrair os dados que serão limpos através do operador *CANDIDATES*. Numa tarefa seguinte os erros ortográficos existentes nos dados são corrigidos e os seus valores são normalizados. Estas tarefas são desempenhadas pelos operadores *SCRUB* que através das funções *xcl:spell-correction* corrigem os erros ortográficos existentes nos elementos *artist* e *dtitle*, através das funções *xcl:capitalize* normalizam os valores dos elementos *artist*, *dtitle* e *genre* e que através da função *xcl:scrubDateTo4Digits* normaliza os valores dos elementos *year*.

Numa tarefa seguinte os elementos *disc* são emparelhados através do operador *FILTER PAIRS*. Cada elemento tem um peso atribuido conforme a sua importância na identificação dos CDs: o peso atribuido ao valor do género e da categoria do CD é 5%, o peso atribuído ao valor do título e do artista do CD é 25%, o peso atribuído ao ano de lançamento do CD é de 10% e o peso atribuído ao conjunto das faixas é de 30%.

Após o emparelhamento dos pares, são construídas as representações dos dados (*XVIEW TRANSFORM*), que serão agrupadas em *clusters* através do operador *CLUSTER CLASSIFICATION*, utilizando a função de fecho transitivo *xcl:cddbTC*.

A representação única é escolhida através do operador de fusão, *FUSION*, que escolhe o elemento mais frequente de cada *cluster*, utilizando a função *xcl:most-frequent*.