Detecção e Correcção de Problemas de Qualidade dos Dados: Modelo, Sintaxe e Semântica

Paulo Jorge Machado Oliveira

UMinho | 2008

Universidade do Minho
Escola de Engenharia

Paulo Jorge Machado Oliveira

Detecção e Correcção de Problemas de Qualidade dos Dados: Modelo, Sintaxe e Semântica

Setembro 2008

Universidade do Minho
Escola de Engenharia

Paulo Jorge Machado Oliveira

Detecção e Correcção de Problemas de Qualidade dos Dados: Modelo, Sintaxe e Semântica

Tese de Doutoramento
Informática / Tecnologia da Programação

Trabalho efectuado sob a orientação de
Professor Doutor Pedro Rangel Henriques
Professora Doutora Maria de Fátima Rodrigues

Setembro 2008

Universidade do Minho, ___/___/_____

Assinatura: _____

Ao meu Filho, David

À minha Esposa, Susana

AGRADECIMENTOS

Um conjunto alargado de pessoas e instituições contribuiu, de uma forma mais ou menos acentuada, para a realização desta dissertação de doutoramento. A todos gostaria de expressar o meu agradecimento pela ajuda que me concederam. Desde já apresento as minhas desculpas se, por lapso, me esquecer de referir alguém.

Naturalmente, a primeira palavra vai para aqueles que mais "sofreram" com a realização deste trabalho: o meu filho David e a minha esposa Susana. Ao meu filho David, que na sua infinita inocência nunca percebeu a razão pela qual o pai tinha de estar quase sempre a "tabalhar" e nunca tinha o tempo que gostaria para "bincar". Por ter aguentado as minhas ausências, mantendo sempre uma alegria nos olhos quando dizia "é o pai". Sem o saber, deu-me sempre o necessário incentivo para que terminasse aquilo que a dada altura me parecia um infindável trabalho. À leal, amiga e companheira Susana, expresso os meus maiores agradecimentos. Nunca esquecerei todo o apoio emocional que me concedeu, todas as condições que me proporcionou, todas as minhas ausências que suportou e de tudo aquilo que abdicou, em meu favor, ao longo destes quatro anos para que eu pudesse realizar este trabalho. Sem dúvida que esta é mais uma prova de amor! Por todo este amor e carinho; por todos os incentivos; pela confiança transmitida; e, pela amizade e ajuda concedida, aqui fica o meu muito, muito obrigado. Uma vez que estas são as pessoas mais importantes da minha vida é pois a elas que dedico este trabalho.

Também gostaria de destacar os meus pais, a quem expresso toda a minha gratidão pelo facto de sempre me terem incutido a importância de estudar e terem proporcionado a possibilidade de obter uma licenciatura. Estou ciente que se não fossem vocês, nunca teria conseguido chegar até aqui.

Um profundo agradecimento e reconhecimento vai, obviamente, para os meus orientadores, Professora Doutora Fátima Rodrigues e Professor Doutor Pedro Henriques, pelo tempo que me dedicaram; pela disponibilidade que sempre manifestaram; pela capacidade científica que evidenciaram; pelas críticas, sugestões e correcções que me efectuaram; pela orientação que me concederam; pela competência e interesse que demonstraram; e, não menos importante, pelos incentivos que sempre me deram. À Professora Doutora Fátima Rodrigues agradeço, também, o financiamento da aquisição de material informático necessário à realização deste trabalho de doutoramento. Ao Professor Pedro Henriques agradeço, também, pelo financiamento que concedeu às minhas deslocações a conferências, sempre que eu o solicitei.

À Professora Doutora Helena Galhardas, docente do Instituto Superior Técnico, pela disponibilidade que, desde o primeiro momento em que foi contactada, sempre mostrou; pela colaboração prestada neste trabalho; pelas valiosas criticas e sugestões que fez; pelo tempo que me concedeu; pela cedência de bibliografia; e, pelo interesse que demonstrou neste trabalho.

Ao Grupo de Engenharia do Conhecimento e Apoio à Decisão (GECAD) do ISEP-IPP, na pessoa do seu Director, Professor Doutor Carlos Ramos, que me facultou o acesso aos meios informáticos existentes nas instalações do referido grupo e financiou as minhas deslocações a conferências, sempre que eu o solicitei.

Ao meu colega do GECAD Nuno Silva, pelo interesse demonstrado no andamento do trabalho; pelos seus conselhos e incentivos; pelo seu tempo e disponibilidade; pela cedência de material (bibliografia; MAFRA *Toolkit*); e, pelas frutuosas discussões que tivemos acerca de certos aspectos do trabalho.

Ao ISEP e ao IPP, todos os meios que me disponibilizou e o apoio institucional que me concedeu na candidatura a uma bolsa PRODEP. Igualmente, agradeço ao PRODEP o financiamento concedido, o que me isentou do pagamento de propinas e, especialmente, permitiu a dispensa de serviço docente. Todos estes aspectos contribuíram muito favoravelmente para a realização do presente trabalho.

À Cristina, Goretti, Helena e Paula da Secretaria do Departamento de Informática da Universidade do Minho, pela sua simpatia, disponibilidade e auxílio prestado em diversos assuntos.

Por último, mas não menos importante, agradeço a Deus a vida e a saúde que me concedeu, bem como a todos aqueles que me são queridos, durante a realização deste trabalho.

A todos exprimo os meus muito sinceros agradecimentos.

Porto, Setembro de 2008

RESUMO

Os dados possuem uma importância crescente na actual sociedade da informação e comunicação. Por este motivo é muito importante a Detecção e Correcção (DC) dos seus problemas de qualidade. Os resultados obtidos a partir dos dados são negativamente influenciados pelos seus problemas de qualidade. Isto é conhecido como o princípio do "lixo entra, lixo sai". Uma taxionomia de Problemas de Qualidade dos Dados (PQD) constitui uma das contribuições que resulta do trabalho de doutoramento apresentado nesta dissertação. Os problemas encontram-se organizados pelo Nível de Granularidade (NG) do modelo relacional em que ocorrem (*e.g.*: atributo; tuplo; relação). O âmbito do trabalho encontra-se restrito a este tipo de dados.

Outra contribuição constitui a definição de um modelo concebido exclusivamente para a DC dos PQD, *i.e.*, para a Limpeza de Dados (LD). O modelo encontra-se em total consonância com a tese defendida de que os PQD têm de ser identificados e, de imediato, solucionados, seguindo uma sequência predefinida baseada numa aproximação ascendente (*i.e.*, *bottom-up*) por NG. Os primeiros PQD manipulados (*i.e.*, detectados e corrigidos) são os que ocorrem no NG mais elementar do modelo relacional (*i.e.*, o atributo). A sequência termina com os problemas que ocorrem no NG de maior complexidade (*i.e.*, múltiplas relações de diferentes fontes de dados).

O utilizador especifica as operações de DC dos PQD incluídos na taxionomia com base em duas linguagens declarativas, especialmente desenvolvidas para esse efeito. A sintaxe de cada linguagem é formalizada nesta dissertação. A semântica operacional inerente à operação que conduz à detecção ou correcção de cada PQD encontra-se também formalizada nesta dissertação, para todas aquelas situações em que esta pode ser fornecida.

O modelo proposto de LD e as formalizações sintácticas e semânticas das operações de DC constituem a base do protótipo desenvolvido, baptizado com a denominação *SmartClean*. O *SmartClean* constitui mais uma contribuição que resulta do trabalho realizado, provavelmente a que possui maior visibilidade. Tendo como objectivo testar o *SmartClean* e, consequentemente, demonstrar a validade do modelo de LD subjacente, o protótipo é utilizado num estudo de caso. Os resultados alcançados confirmam a sua aplicabilidade, validade e utilidade na DC dos PQD.

Como última contribuição deste trabalho, nesta dissertação é ainda apresentada uma aproximação que suporta a interoperabilidade das operações de DC entre bases de dados diferentes. Isto é alcançado através da especificação das operações a um nível conceptual que as isola do nível do esquema, permitindo facilmente a sua reutilização em bases de dados diferentes.

ABSTRACT

The data have a growing importance in today's information and communication society. For this reason, the Detection and Correction (DC) of their quality problems is very important. The results obtained from data are negatively influenced by their quality problems. This is known as the "garbage in, garbage out" principle. The taxonomy of Data Quality Problems (DQP) is one of the contributions that results from the PhD work presented in this dissertation. The problems are organized by Granularity Level (GL) of the relational model in which they occur (*e.g.*: attribute; tuple; relation).The scope of the work is restricted to this kind of data.

Another contribution is the definition of a model exclusively designed for the DC of the DQP, *i.e.*, for data cleaning. The model is in total agreement with the thesis defended that the DQP must be identified and immediately solved, following a predefined sequence based on an ascending (*i.e.*, *bottom-up*) approach by GL. The first DQP manipulated (*i.e.*, detected and corrected) are those occurring at the most elementary GL of the relational model (*i.e.*, the attribute). The sequence ends with the problems occurring at the most complex GL (*i.e.*, multiple relations from different data sources).

The user specifies the DC operations of the DQP included in the taxonomy based on two declarative languages, specially developed for that purpose. The syntax of each language is formalized in this dissertation. The underlying operational semantic of the operation that leads to the detection or correction of each DQP is also formalized in this dissertation, for all those situations where it can be provided.

The proposed model for data cleaning and the syntactic and semantic formalizations of the DC operations are the basis of the developed prototype, baptized with the denomination *SmartClean*. *SmartClean* is another contribution that results from the work done and it is probably the one that has the largest visibility. Having the test of *SmartClean* as a goal and, therefore, the demonstration of the validity of the underlying data cleaning model, the prototype is used in a case study. The results achieved confirm its applicability, validity and usefulness in the DC of DQP.

As a last contribution of this work, an approach that supports the interoperability of the DC operations among different databases is also presented in this dissertation. This is achieved through the specification of the operations at a conceptual level that isolates them from the schema level, allowing them to be easily used in different databases.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE ACRÓNIMOS

Em consequência da utilização frequente de algumas designações, procedeu-se à sua substituição por acrónimos. Em cada capítulo, a primeira utilização do acrónimo é sempre acompanhada pelo respectivo significado. Mesmo assim, com o intuito de facilitar a leitura da dissertação, fornece-se a seguinte lista onde podem ser consultados os significados de todos os acrónimos utilizados.

Acrónimo	**Significado**
AD	Análise de Dados
BD	Base de Dados
DC	Detecção e Correcção
DP	*Data Profiling*
ED	Enriquecimento de Dados
ETC	Extracção, Transformação e Carregamento
FD	Fonte de Dados
FDU	Função Definida pelo Utilizador
GTQD	Gestão Total da Qualidade dos Dados
LD	Limpeza de Dados
NG	Nível de Granularidade
OC	Operação de Correcção
OD	Operação de Detecção
PQ	Problema de Qualidade
PQD	Problema de Qualidade dos Dados
QD	Qualidade de Dados
SI	Sistema de Informação
SQL	*Structured Query Language*
TD	Transformação de Dados

1 INTRODUÇÃO

Neste capítulo começa-se por efectuar o enquadramento da área em que se situa o presente trabalho de doutoramento. Seguidamente, são apresentados os problemas e limitações das actuais soluções de melhoria da Qualidade dos Dados (QD) que motivam e justificam a realização do trabalho. Na sequência destes problemas e limitações são apresentados os objectivos que se pretendem atingir com a sua realização. Após a apresentação destes, é enunciada a tese defendida nesta dissertação. As contribuições alcançadas com a realização do trabalho são enumeradas de seguida. Por último, é apresentada a estrutura da dissertação, o que inclui uma descrição sucinta do conteúdo de cada capítulo.

1.1 Enquadramento

As organizações actuais operam e competem numa sociedade dominada pelas tecnologias da informação e comunicação. Os dados armazenados em formato digital constituem um dos pilares em que assenta esta sociedade. Os *dados* representam propriedades de entidades ou objectos do mundo real num formato que pode ser armazenado, acedido, elaborado por um procedimento computacional e comunicado através de uma rede [Batini e Scannapieco, 2006]. Nas últimas décadas assistiu-se a um aumento exponencial do armazenamento de dados nas mais diversas áreas da actividade humana, potenciado por um conjunto de evoluções tecnológicas. Entre todas as evoluções, aquela que mais contribui para esta realidade foi a *Internet*. As tecnologias de armazenamento de dados também tiveram a sua quota-parte de responsabilidade neste crescimento. A nível de *software*, o desenvolvimento das Bases de Dados (BD) relacionais veio permitir o armazenamento e acesso aos dados de uma forma fácil e eficiente, o que muito justifica o seu sucesso e divulgação comercial[1]. A nível de *hardware*, as inovações e melhorias registadas nos dispositivos de recolha (*e.g.*: leitor de código de barras) e armazenamento (*e.g.*: CD; DVD) permitiram tempos de acesso cada vez mais rápidos e capacidades de armazenamento cada vez maiores, acompanhadas por uma diminuição dos custos inerentes.

[1] O âmbito do trabalho de doutoramento retratado nesta dissertação encontra-se limitado, exclusivamente, a este modelo de representação de dados.

Apesar de armazenarem continuamente dados e, cada vez mais, em maiores quantidades, apenas recentemente as organizações públicas e privadas começaram a aperceber-se do seu real valor. Os dados passaram a ser vistos como um recurso importante que possibilita o aumento da produtividade, eficiência e competitividade da organização [Oliveira *et al.*, 2004]. Na visão anterior, os dados não eram mais do que um mero histórico de actividades. Como consequência da nova visão, a exploração dos dados passou a assumir um papel cada vez mais importante. Por exemplo, a consolidação de dados a partir de *fontes de dados*[2] dispersas num armazém de dados permite às organizações executarem operações de análise multi-dimensional de dados e, assim, obterem informações que são de importância estratégica e táctica para as suas actividades [Ballou e Tayi, 1999] [Inmon *et al.*, 1998]. Além das ferramentas de análise multi-dimensional, existem outras que procuram tirar partido dos dados existentes, como as que efectuam mineração de dados (em inglês: *data mining*) e gestão do relacionamento com o cliente (em inglês: *customer relationship management*). Todas estas ferramentas requerem um elevado grau de QD, uma vez que estes influenciam a qualidade dos resultados produzidos. Se os dados forem de má qualidade, isso reflecte-se nos resultados produzidos (principio "lixo entra, lixo sai") [Sattler e Schallehn, 2001]. Os resultados são usados como suporte para a tomada de decisão, logo se forem de má qualidade isso influencia de forma negativa as decisões tomadas. Um elevado número de organizações já experimentou os efeitos adversos de decisões baseadas em dados de má qualidade [Huang *et al.*, 1999]. Num número cada vez maior de organizações acredita-se mesmo que a QD é condição essencial para o seu sucesso [Wang *et al.*, 1998].

Normalmente, as preocupações com a QD surgem, fundamentalmente, nos contextos referidos (*i.e.*, análise multi-dimensional de dados; mineração de dados; e, gestão de relacionamento com o cliente). No entanto, o contexto de maior relevância diz respeito aos próprios dados quando considerados isoladamente. De facto, mesmo que os dados não sejam utilizados nesses contextos, *i.e.*, sejam apenas usados no contexto dos sistemas de informação que os armazenam, a necessidade de dados com qualidade constitui sempre um requisito. No entanto, constata-se que uma boa parte dos dados, na generalidade das fontes, apresenta problemas de qualidade [Oliveira *et al.*, 2004]. À luz da perspectiva adoptada neste trabalho, estes Problemas de Qualidade dos Dados (PQD) correspondem à falta de completude, correcção ou consistência dos valores dos dados[3]. Entre outras possibilidades, estes PQD correspondem a valores em falta em atributos[4] de

[2] Nesta dissertação, o termo *fonte de dados* significa base de dados ou conjunto de ficheiros.
[3] Os PQD estão a ser abordados de acordo com uma perspectiva estrita que considera apenas estas três vertentes ou dimensões. Existe uma outra perspectiva, de âmbito lato, que considera muitas outras dimensões, a generalidade das

preenchimento obrigatório, violações de sintaxe ou domínio, erros ortográficos ou representações diferentes dos mesmos valores (*i.e.*, utilização de sinónimos) ou entidades do mundo real (*i.e.*, existência de duplicados).

Nas suas formas mais simples, os PQD manifestam-se, quotidianamente, nas mais diversas actividades. Muitos e variados exemplos poderiam ser dados, mas considere-se apenas as seguintes situações: (*i*) uma carta que não chega ao destinatário devido a erros existentes nos dados que compõem o endereço (*e.g.*: na rua; no n.º de polícia; ou no código postal); (*ii*) o envio de correspondência ou *e-mails* em duplicado, como consequência da existência de redundância na BD subjacente.

Os PQD acarretam custos elevados às organizações [Eckerson, 2002]. Estes custos resultam dos prejuízos causados (*e.g.*: envio de correspondência em duplicado) e dos valores monetários que é necessário despender para proceder à sua Detecção e Correcção (DC). No entanto, o impacto da fraca QD não é apenas quantificável em termos de custos. Na realidade, o efeito da fraca QD pode afectar as organizações ao nível operacional, táctico e estratégico [Redman, 1996]. Ao nível operacional, a fraca QD tem reflexo negativo na satisfação do cliente. Os clientes esperam que os seus dados sejam correctamente mantidos, pelo que a existência de erros sistemáticos pode motivar que estes deixem de o ser. Ao nível táctico, a fraca QD influencia a qualidade das decisões. As decisões dependem dos dados e como frequentemente uma só decisão obriga à análise de grandes quantidade de dados, é improvável que todos possuam uma elevada qualidade. Ao nível estratégico, há necessidade de dados internos e externos para que seja possível definir a estratégia de longo prazo. A falta de dados completos, relevantes, actualizados e precisos acerca de clientes, concorrentes e tecnologias pode constituir um obstáculo à definição de uma estratégia sólida.

Historicamente, a existência de PQD sempre foi uma realidade. No entanto, nos sistemas de informação monolíticos do passado, os dados eram manipulados em ambientes que proporcionavam um melhor controlo destes. Com o surgimento das redes locais e da *Internet*, os

---

quais de natureza subjectiva (*e.g.*: a interpretabilidade dos dados). Os defensores desta perspectiva advogam que estas dimensões também são importantes na análise dos PQD. Ambas as perspectivas são apresentadas no Capítulo 2.

[4] Nesta dissertação certos termos são utilizados de forma indiferenciada. Isto acontece com os termos *atributo* e *campo*, representando ambos uma propriedade de um objecto ou entidade do mundo real. O mesmo acontece com os termos *tuplo* e *registo*, em que ambos correspondem a um número pré-definidos de atributos (ou campos), representando um objecto ou entidade do mundo real. Por último, os termos *relação*, *tabela* e *ficheiro* são, também, usados indistintamente, correspondendo a um conjunto de tuplos (ou registos), representando vários objectos ou entidades do mundo real do mesmo tipo.

sistemas de informação passaram a poder interactuar entre si. Os dados passaram a ser manipulados em ambientes bem mais "turbulentos", frequentemente sem a implementação de mecanismos adequados que garantissem a sua qualidade. A existência de PQD acentuou-se fortemente perante este novo panorama tecnológico.

Face à importância dos dados na actual sociedade da informação e comunicação é imperioso que os PQD sejam identificados e solucionados. Inúmeros projectos centrados nos dados falham por diferentes motivos, mas a generalidade dos estudos apontam como principal causa a fraca QD [English, 1999]. Actualmente, começa a haver uma consciencialização crescente da importância da melhoria da QD, o que passa pela DC dos PQD. Esta consciencialização é reforçada pela constatação de que quanto maior a qualidade, maior o valor e utilidade dos dados.

1.2 Problemas e Limitações das Actuais Soluções de Melhoria da Qualidade dos Dados

A melhoria da QD implica a realização de dois tipos de actividades distintas: (*i*) DC dos PQD existentes; (*ii*) alteração ao processo gerador desses dados, de modo a eliminarem-se as reais causas da existência dos PQD. No âmbito deste trabalho de doutoramento, apenas o primeiro tipo de actividades constitui objecto de estudo, embora se reconheça a importância do segundo na eliminação definitiva dos PQD.

Ainda que existam outras possibilidades para a DC automática[5] dos PQD (a apresentar no Capítulo 2 – Secção 2.7), a *Limpeza de Dados* (LD) por se basear unicamente nas especificações do próprio utilizador[6], constitui a forma mais efectiva de alcançar esse objectivo. As áreas académica e comercial têm contribuído com diversas soluções informáticas para a LD. No entanto, estas soluções apresentam alguns problemas e limitações relevantes.

Os protótipos[7] resultantes dos trabalhos de investigação realizados, ainda que se auto-intitulem de LD, possuem potencialidades limitadas a este nível. No geral, estes protótipos estão

[5] O termo *automático* refere-se unicamente à execução do processo de DC, uma vez que este é efectuado por meios exclusivamente automáticos. Naturalmente, um processo de DC (no global) nunca é completamente automático, pois requer o envolvimento do utilizador, quanto mais não seja para analisar os problemas detectados e validar as correcções efectuadas. Nesta perspectiva, o termo mais adequado é *semi-automático*.

[6] O termo *utilizador* é empregue nesta dissertação com um sentido para além do tradicional. Assim, utilizador representa um indivíduo com bons conhecimentos sobre as características dos dados em questão e, em particular, dos potenciais PQD que os podem afectar e das acções correctivas a efectuar para os solucionar.

[7] Um *protótipo* constitui um sistema informático de carácter experimental.

verdadeiramente vocacionados é para a Transformação de Dados (TD). É certo que na literatura estes dois termos são, muitas vezes, usados de forma indiferenciada ou mesmo em conjunto. Ainda que relacionados, em rigor, o seu significado não é o mesmo. Sendo o significado de LD o apresentado (*i.e.*, DC dos PQD), a TD visa colocar os dados segundo um outro esquema de representação, consentâneo com a finalidade pretendida. A transformação faz com que os dados que se encontram segundo o esquema *X*, fiquem de acordo com o esquema *Y*. Normalmente, a TD tem como objectivo permitir que se explore melhor os dados existentes (*e.g.*: permitindo a realização de novas análises de dados). Nesta perspectiva, a realização da transformação é, também, uma actividade que conduz à melhoria da QD. Naturalmente, pode aproveitar-se o momento em que os dados são transformados para detectar e corrigir os problemas existentes nos próprios valores (*i.e.*, efectuar LD). No entanto, a TD pode não incluir operações de limpeza, *i.e.*, apenas efectua a migração dos valores de um esquema para outro. Neste caso considera-se que não existem PQD que mereçam ser identificados e solucionados. No entanto, a LD ainda que possa ser efectuada durante a TD, é perfeitamente independente desta. Pode querer melhorar-se somente a QD, mas sem realizar qualquer transformação no seu esquema de representação. Neste caso, os dados são "limpos" directamente na própria fonte onde residem.

Os protótipos existentes possuem diversas potencialidades intrínsecas a nível de transformação, mas são limitados a nível de potencialidades de limpeza. Por vezes, estas potencialidades resumem-se à detecção e eliminação de registos duplicados (*i.e.*, que representam a mesma entidade ou objecto do mundo real). As potencialidades de que aqui se falam são as reais e não as virtuais, uma vez que a materialização destas implica a inclusão de funções definidas pelo utilizador. Esta característica comum dos protótipos de investigação é conhecida por *extensibilidade*. Se para a DC de alguns PQD (*e.g.*: violação de uma restrição de integridade específica do domínio) não há alternativa que não seja suportar a inclusão de funções definidas pelo utilizador, na generalidade dos problemas tal não acontece. Quer isto dizer que para estes problemas a DC pode ser suportada de base/raiz. Contudo, isto não acontece nos protótipos de investigação. Apenas virtualmente permitem a DC dos PQD, devidamente escudados nas suas potencialidades de extensibilidade. Note-se que o desenvolvimento das funções implica o dispêndio de recursos e não está ao alcance de qualquer utilizador, mas apenas dos que possuem conhecimentos a nível de programação. De facto, os protótipos de investigação estão muito orientados para o utilizador especialista. No entanto, mesmo para estes, desenvolver algumas funções ou, pelo menos, encontrar a forma mais eficiente de o fazer, pode não ser trivial. O desenvolvimento das funções que permitem a DC dos PQD, apenas deve ser deixado em aberto quando não há, mesmo, outra hipótese (*e.g.*: como no caso das restrições de integridade que são

específicas de cada situação particular). Por outro lado, nos protótipos de investigação quase sempre a LD é realizada exclusivamente no contexto da TD. Isto significa que não é possível "limpar" os dados directamente na própria *fonte de dados*. É sempre necessário efectuar uma TD, mesmo que não se pretenda efectuar uma alteração ao seu esquema. Neste caso, há que simular essa transformação (*i.e.*, do esquema X para o esquema X) para se poder realizar a LD. A definição da sequência de execução das operações de limpeza a efectuar, sob a forma de um grafo dirigido de transformações, constitui uma responsabilidade do utilizador.

No geral, as ferramentas comerciais, ditas de LD, baseiam o seu funcionamento numa sequência composta pelos seguintes passos: (*i*) *atomização*, que consiste em dividir/separar os valores dos atributos do tipo *string* nos seus vários componentes atómicos, preparando-os para os passos seguintes (*e.g.*: separação da morada em nome da rua e número de polícia); (*ii*) *uniformização*, que consiste em colocar os valores de acordo com um standard comum; (*iii*) *validação*, que consiste em identificar valores inválidos (*e.g.*: por comparação com outros valores; com base em restrições de integridade a que os valores têm de obedecer); (*iv*) *detecção de duplicados*, que consiste em identificar registos que representam a mesma entidade ou objecto do mundo real; e, (*v*) *consolidação de duplicados*, que consiste em agrupar os conjuntos de duplicados identificados num só registo. O utilizador não tem possibilidade de efectuar qualquer tipo de alteração a esta sequência. À semelhança do que acontece com os protótipos de investigação, a realização de LD é efectuada num contexto que implica TD. A TD está presente, desde logo, no primeiro passo da sequência (*i.e.*, atomização). Os dados são colocados de acordo com um esquema de representação predefinido. Só após a realização deste passo é que os demais passos da sequência podem ser executados. A TD resultante da atomização só não é efectuada caso os dados já estejam de acordo com o esquema predefinido, o que normalmente não acontece. Após a realização da sequência de passos que conduzem à melhoria da qualidade, os dados ficam representados segundo esse novo esquema. Normalmente, o novo esquema de dados não é adequado à finalidade que se pretende (*e.g.*: ao esquema de dados utilizado no sistema de informação de onde originam), pelo que os dados têm de ser sujeitos a um novo processo de transformação (*e.g.*: concatenação da rua e do número de policia em morada). Não é claro como este novo processo de transformação é suportado pelas ferramentas de LD, pelo que muito provavelmente terá de ser o utilizador a efectuá-lo de alguma forma (*e.g.*: recorrendo a programação explícita; usando uma ferramenta comercial de TD). Tal como nos protótipos de investigação, não é possível proceder à melhoria da QD directamente na própria fonte onde estes se encontram.

Por outro lado, a sequência rígida preconizada pelas ferramentas comerciais não parece ser a mais adequada para manipular certos PQD. A título de exemplo, considere-se o caso das restrições de integridade. A verificação de certas restrições só deve ser efectuada após a detecção e eliminação de duplicados. Caso contrário podem ser reportadas falsas violações, motivadas pela existência dos registos duplicados. Na sequência apresentada, este tipo de verificações é efectuado antes da detecção e eliminação de registos duplicados (*i.e.*, na fase de validação). Ainda que em certas situações a sequência se mostre adaptada, há outras situações em que claramente não o é. Ainda a propósito deste assunto, as ferramentas comerciais suportam a verificação de restrições de integridade definidas pelo utilizador, em função das necessidades particulares de cada caso. Tipicamente, as restrições de integridade a verificar são definidas com base em linguagens relativamente simples, que incluem operadores lógicos e de comparação. No entanto, estas linguagens não suportam a especificação de restrições de integridade com lógicas subjacentes demasiado complexas. Nestas situações, não há outra hipótese que não seja a implementação da sua verificação através de uma linguagem de programação convencional. No entanto, as ferramentas comerciais de LD normalmente não possuem capacidades de extensibilidade, *i.e.*, não permitem a inclusão de funções definidas pelo utilizador. Isto pode ser explicado pelo seu cariz orientado ao utilizador não perito, contrariamente ao que acontece nos protótipos de investigação. Por outro lado, ainda que em teoria pretendam abarcar todos os PQD, na prática as ferramentas comerciais evidenciam uma cobertura limitada, a nível de DC dos diversos problemas. Como não possuem potencialidades de extensibilidade, não conseguem cobrir outros problemas para além daqueles que suportam de base. As funcionalidades oferecidas pelas diferentes ferramentas comerciais acabam por ser muito similares. A isto não será alheio a concorrência existente entre as empresas que as detêm, o que se reflecte na constante procura de oferecerem iguais potencialidades às existentes nas ferramentas concorrentes.

Como corolário dos problemas e limitações identificados nas actuais soluções académicas e comerciais pode afirmar-se que é concedida uma cobertura deficiente aos PQD. O resumo dos problemas e limitações aqui apresentado encontra-se devidamente fundamentado nos Capítulos 3 e 4 desta dissertação.

1.3 Objectivos do Trabalho

O presente trabalho de doutoramento explora os problemas e limitações apontados às actuais soluções de melhoria da QD. Em particular, os objectivos que se pretendem alcançar com a sua realização são:

- Formalização da semântica das operações que conduzem à DC dos PQD em que tal suporte é susceptível de ser, desde logo, fornecido. Estas formalizações constituem a base para uma posterior implementação e respectiva automatização das operações.
- Concepção e desenvolvimento de um modelo orientado, especificamente, para a DC dos PQD relacionais que obedeça aos seguintes requisitos genéricos:
  - Defina uma sequência de manipulação para os PQD que: (*i*) supere os problemas da sequência utilizada pelas ferramentas comerciais, reflectindo as dependências existentes entre os diversos problemas; (*ii*) na medida do possível, não dependa da especificação manual por parte do utilizador, ao invés do que sucede nos protótipos de investigação.
  - Suporte a execução das operações de LD directamente na própria fonte, sem obrigar à realização de TD.
- Implementação de um protótipo de LD que materialize e permita validar o modelo proposto.

1.4 Tese Defendida

Nesta dissertação de doutoramento defende-se a seguinte tese: A adopção de uma sequência de manipulação predefinida, em que os PQD são detectados e, de imediato, corrigidos, seguindo uma abordagem ascendente (*i.e.*, *bottom-up*), em que se começa nos problemas que se manifestam no nível de granularidade mais elementar do modelo relacional (*i.e.*, ao nível do atributo) e termina nos que ocorrem no nível de granularidade de maior complexidade (*i.e.*, ao nível de múltiplas relações pertencentes a diferentes fontes de dados), constitui uma forma adequada e eficaz de identificar e solucionar os PQD existentes.

1.5 Contribuições

O trabalho desenvolvido e apresentado nesta dissertação não tem a pretensão de fornecer soluções definitivas, mas apenas contribuir com uma pequena parcela de conhecimento que permita algum avanço técnico-científico na área de estudo em questão. Como consequência desse trabalho, resultaram as contribuições que a seguir se apresentam:

- **Taxionomia de PQD relacionais** – A taxionomia organiza os PQD em função do nível de granularidade do modelo relacional (*e.g.*: atributo; tuplo) em que estes ocorrem. Comparativamente às anteriormente existentes, a taxionomia proposta encontra-se alicerçada numa abordagem sólida que conduziu à identificação dos diferentes problemas

que afectam os dados, evidenciando ser mais genérica e abrangente nos PQD incluídos. Esta taxionomia foi inicialmente apresentada em [Oliveira *et al.*, 2004], tendo sofrido posteriores desenvolvimentos que foram apresentados em [Oliveira *et al.*, 2005a] e [Oliveira *et al.*, 2006d]. A taxionomia permitiu analisar a cobertura das ferramentas existentes orientadas para a DC dos PQD, o que conduziu à identificação de algumas lacunas exploradas neste trabalho de doutoramento. Entre outras, a taxionomia proposta serviu para analisar a cobertura de algumas ferramentas de *data profiling*. Os resultados desta análise foram apresentados em [Oliveira *et al.*, 2006a].

Um outro aspecto que também a diferencia em relação às taxionomias anteriormente existentes prende-se com o fornecimento de definições formais para cada PQD, eliminando possíveis subjectividades típicas das definições textuais. As definições formais, fruto do seu rigor, possuem mais elementos informativos do que as textuais, evidenciando o que é necessário para seja possível automatizar a detecção de cada problema. A formalização dos PQD que compõe a taxionomia foi inicialmente apresentada em [Oliveira *et al.*, 2005c].

- **Modelo para a DC dos PQD** – Nesta dissertação concebe-se e define-se um modelo direccionado, especificamente, para a LD que visa suportar a DC de todos os PQD incluídos na taxionomia. Este modelo encontra-se materializado numa arquitectura de uma ferramenta informática. O modelo assenta numa aproximação sequencial ascendente (*i.e.*, *bottom-up*) de detecção e imediata correcção dos PQD, concebida de acordo com os níveis de granularidade que compõem a taxionomia, desde o nível mais elementar (*i.e.*, atributo) até ao de maior complexidade (*i.e.*, múltiplas relações de diferentes fontes de dados). Um mecanismo de execução incremental das operações de DC faz também parte do modelo. As operações de DC a efectuar são especificadas pelo utilizador, com base em duas linguagens declarativas. As potencialidades de extensibilidade incluídas no modelo permitem cobrir a DC dos problemas para os quais não é possível fornecer, à partida, tal suporte devido à sua especificidade (*e.g.*: violações de restrições de integridade). O modelo permite que as operações sejam efectuadas directamente na própria fonte, dispensado a realização de TD. Este modelo foi inicialmente proposto em [Oliveira *et al.*, 2005b]. Os novos desenvolvimentos entretanto ocorridos foram apresentados em [Oliveira *et al.*, 2006b].
- **Protótipo para a DC de PQD** – Um protótipo, denominado de *SmartClean*, é apresentado, tendo sido desenvolvido de acordo com a arquitectura proposta para a DC dos PQD. O protótipo é usado num caso real de melhoria da QD, permitindo os resultados validar a arquitectura e, consequentemente, o modelo de LD subjacente. Simultaneamente, os resultados demonstram a aplicabilidade do *SmartClean* na DC dos PQD.

- **Aproximação que suporta a interoperabilidade das operações de DC** – A especificação das operações de LD ao nível do esquema foi a abordagem empregue até aos dias de hoje. No entanto, diversas alterações são necessárias para reutilizar as mesmas operações noutra BD com um esquema diferente. Na realidade, muitas operações são genéricas o suficiente para serem aplicadas noutras BD. Estas operações podem estar limitadas a BD pertencentes ao mesmo domínio ou podem ser tão genéricas que até são independentes deste. Nesta dissertação apresenta-se uma aproximação que suporta a interoperabilidade das operações de DC entre BD diferentes. Isto é alcançado através de um nível ontológico que suporta a especificação conceptual das operações ao nível de domínios, conceitos e propriedades. Este nível de abstracção isola as operações do esquema de dados e permite, facilmente, a sua reutilização noutras BD. Em função dos mapeamentos definidos pelo utilizador entre os elementos do esquema de uma BD e os elementos da ontologia, automaticamente são identificadas e instanciadas (ao nível do esquema dos dados) as operações conceptuais de LD susceptíveis de serem executadas nessa BD. Este aproximação foi genericamente apresentada em [Oliveira *et al.*, 2006b]. Posteriormente, foi apresentada com mais detalhe em [Oliveira *et al.* 2006c]. Ainda assim, esta vertente do trabalho merece ser melhor explorada, pelo que será objecto de novos desenvolvimentos num futuro próximo.

1.6 Estrutura da Dissertação

Além da apresentação da estrutura da dissertação, no actual capítulo é efectuado o enquadramento dos PQD. Os problemas e limitações das actuais soluções de LD são evidenciados, uma vez que constituem a razão de ser deste trabalho. Os objectivos que nortearam a sua realização são enunciados. A tese que se encontra subjacente é também apresentada, assim como as contribuições que resultaram da realização do trabalho.

No Capítulo 2 – Qualidade de Dados é dada uma panorâmica geral sobre a temática. Como o próprio título da dissertação o indica, o trabalho desenvolvido no âmbito do doutoramento enquadra-se, genericamente, no tema da QD, daí que seja importante conceder-lhe uma atenção especial. Neste capítulo apresentam-se todos os aspectos considerados relevantes para a sua compreensão e enquadra-se o trabalho desenvolvido.

No Capítulo 3 – Limpeza de Dados é dado destaque a esta actividade do processo de melhoria da QD. O trabalho reportado nesta dissertação centra-se, especificamente, sobre esta actividade. Assim, justifica-se perfeitamente o relevo que lhe é dado, em detrimento das restantes actividades. O objectivo deste capítulo é o de apresentar as diversas vertentes (*e.g.*: a questão da

detecção e eliminação de duplicados) consideradas relevantes para a compreensão do significado de LD e do seu actual estado da arte. Por outro lado, no Capítulo 3 são expostas, de forma fundamentada, as lacunas já apontadas no presente capítulo. Estas lacunas são exploradas no âmbito deste trabalho de doutoramento.

No Capítulo 4 – Problemas de Qualidade dos Dados é apresentada, pormenorizadamente, uma taxionomia de PQD. Esta taxionomia foi desenvolvida no âmbito deste trabalho de doutoramento, evidenciando cobrir um maior número de problemas do que as anteriormente existentes. A taxionomia é complementada com um conjunto de definições matemáticas que acrescentam rigor às definições textuais dos problemas. A taxionomia é usada para avaliar a cobertura dada aos PQD, a nível de DC, pelas actuais ferramentas informáticas de LD. Desta forma, fundamentam-se as deficiências de cobertura já referidas no actual capítulo e devidamente exploradas neste trabalho.

No Capítulo 5 – Formalização das Operações de Detecção, como o nome o indica, apresenta-se a formalização, a nível sintáctico e semântico, da operação de detecção de cada PQD que integra a taxionomia exposta no capítulo anterior. As formalizações são apresentadas por nível de granularidade, obedecendo à forma de organização estabelecida pela própria taxionomia.

No Capítulo 6 – Formalização das Operações de Correcção apresenta-se a formalização sintáctica e semântica do outro tipo de operações que faz parte da LD (*i.e.*, as operações de correcção), cuja finalidade é solucionar os PQD identificados pelas operações de detecção. Para cada problema da taxionomia em que é possível conceder-lhe, desde logo, um suporte automatizado à sua resolução, apresenta-se a formalização da respectiva operação de correcção. Esta pode envolver alterações aos valores dos atributos ou mesmo a eliminação de tuplos. Tal como no capítulo anterior, as formalizações destas operações também se encontram organizadas por nível de granularidade.

No Capítulo 7 – Modelo Proposto para a Limpeza de Dados dá-se continuidade aos dois capítulos anteriores, através da apresentação do modelo que se defende para a LD. O modelo proposto articula a execução alternada dos dois tipos de operações formalizadas (*i.e.*, detecção e correcção) por, e em cada, nível de granularidade. A execução das operações obedece a uma determinada sequência preestabelecida. A arquitectura que materializa este modelo é descrita, nomeadamente os componentes (ou módulos) que a integram e a forma como estes interactuam entre si. O protótipo que esta arquitectura originou, designado de *SmartClean*, é também descrito de forma sucinta.

No Capítulo 8 – Estudo de Caso de Limpeza de Dados procede-se à exposição do estudo de caso de LD efectuado com o intuito de testar e validar a ferramenta *SmartClean* e, consequentemente, o modelo proposto e as operações de LD subjacentes (*i.e.*, de detecção e correcção). O caso é baseado numa BD relativa ao domínio das dádivas/colheitas de sangue. As operações de LD realizadas cobrem todos os níveis de granularidade de ocorrência de PQD, desde o mais elementar (*i.e.*, o atributo), até ao de maior complexidade (*i.e.*, múltiplas relações de diferentes fontes de dados).

No Capítulo 9 – Interoperabilidade das Operações de Limpeza de Dados é apresentada uma abordagem original que suporta a interoperabilidade das operações de DC em diferentes BD. A interoperabilidade é alcançada através da representação conceptual das operações de LD ao nível da ontologia do domínio subjacente. A abordagem constitui uma extensão ao modelo de LD apresentado no Capítulo 7 e que futuramente virá a ser implementada e integrada no protótipo *SmartClean*.

Finalmente, no Capítulo 10 – Conclusão procura-se demonstrar que todos os objectivos que estiveram na base da realização deste trabalho de doutoramento foram alcançados. Uma comparação entre trabalho relacionado e o trabalho efectuado é apresentada. As limitações inerentes a este são, também, evidenciadas. Por último, são perspectivadas algumas hipóteses de desenvolvimentos futuros, com o intuito de dar seguimento ao trabalho efectuado até ao momento.

Além dos 10 capítulos, a dissertação é ainda composta por dois apêndices. No Apêndice A – Ferramentas Comerciais de Melhoria da QD, apresenta-se uma descrição sucinta de algumas ferramentas comerciais utilizadas no processo de melhoria da QD. No âmbito deste trabalho de doutoramento procedeu-se à sua análise e experimentação. No Apêndice B – Álgebra Relacional, apresenta-se uma breve descrição sobre os operadores de álgebra relacional utilizados na formalização semântica das operações de detecção e correcção presentes, respectivamente, nos Capítulos 5 e 6 desta dissertação.

Para uma leitura mais rápida desta dissertação, orientada apenas ao trabalho efectuado no âmbito deste doutoramento, sugere-se que se avance os capítulos referentes ao estado da arte (*i.e.*, Capítulos 2 e 3). Para um leitor menos conhecedor das áreas da Qualidade de Dados, no geral, e da Limpeza de Dados, em particular, aconselha-se a leitura dos Capítulos 2 e 3 desta dissertação.

2 QUALIDADE DE DADOS

Este capítulo dá uma visão geral sobre a temática da Qualidade de Dados (QD). Assim, começa por contextualizar e vincar a importância da QD, realçando o facto de ser uma área de investigação multidisciplinar. O termo QD possui significados diferentes consoante a perspectiva de abordagem. Na literatura é possível encontrar duas perspectivas, uma estrita (*i.e.*, mais específica) e a outra lata (*i.e.*, mais abrangente). A perspectiva estrita começa por ser apresentada em primeiro lugar, sendo depois efectuada a exposição da perspectiva lata. De seguida, são apresentados os principais contextos onde as preocupações com a QD se tornam particularmente notórias. As causas e consequências da fraca QD são descritas posteriormente. A gestão da QD é objecto de atenção especial na secção seguinte, sendo apresentada uma metodologia que visa a melhoria contínua da QD. No âmbito da gestão da QD são, ainda, apresentadas as formas de avaliar objectivamente e subjectivamente a qualidade, bem como as abordagens empregues na sua melhoria. Seguidamente, são apresentadas e comparadas três taxionomias de Problemas de Qualidade dos Dados (PQD), ao nível dos valores dos atributos. Por último, são descritas as diversas actividades susceptíveis de serem efectuadas durante um processo de melhoria da qualidade, centrado nos dados e que, designadamente, são: *data profiling*; análise de dados; transformação de dados; limpeza de dados; e, enriquecimento de dados.

2.1 Introdução

A informação e o conhecimento de extraordinário valor que se podem extrair dos dados evidenciam a importância destes. Os dados constituem o principal suporte na tomada de decisão [Redman, 2004]. No entanto, os dados nunca são tão perfeitos como se esperaria e gostaria que fossem [Kubica e Moore, 2003]. A fraca QD é uma realidade, sejam estes pertencentes a organizações públicas ou privadas [Fisher e Kingma, 2001]. Nenhuma organização se encontra imune à existência de Problemas de Qualidade (PQ) nos seus dados. Em virtude do princípio "lixo entra, lixo sai", a pouca QD influencia negativamente a informação e o conhecimento que se conseguem obter [Lee *et al.*, 2000a].

Na realidade, um dos maiores problemas que as organizações enfrentam actualmente está relacionado com a pouca qualidade dos seus dados, o que constitui um obstáculo à sua efectiva utilização. Num número crescente de organizações que consideram os dados como um dos seus activos mais importantes, a opção por elevados níveis de QD constitui um investimento estratégico para que se preserve o valor deste activo [Missier *et al.*, 2003]. Apenas se conseguem alcançar vantagens competitivas a partir dos dados, quando estes possuem um elevado nível de qualidade. Assim, a QD constitui um aspecto crítico a ter em conta na gestão dos sistemas de informação. Apesar da sua importância e valor, a QD é normalmente negligenciada [Lopes e Babbitt, 1999].

Numa perspectiva de investigação, a QD tem sido objecto de estudo em diversas ciências, das quais se salientam a estatística, a gestão e a informática. Os investigadores das ciências estatísticas foram os primeiros a debruçar-se sobre esta problemática nos finais da década de cinquenta, tendo apresentando uma teoria probabilística (denominada de *record linkage* [Newcombe *et al.*, 1959]) para a identificação de duplicados em conjunto de dados estatísticos. No início da década de oitenta, os investigadores das ciências da gestão abordaram o problema numa perspectiva de gestão da qualidade do sistema de manufactura de dados, estabelecendo uma analogia com os sistemas de manufactura de produtos. Apenas no início da década de noventa, os investigadores de diversas áreas das ciências informáticas começaram a debruçar-se sobre o problema de melhorar a QD armazenados electronicamente (*e.g.*: em bases de dados; em armazéns de dados). Ao longo das últimas décadas, estas ciências desenvolveram modelos e metodologias para a QD, por vezes com algumas sobreposições. Todas estas propostas contribuíram para que a QD se afirmasse como área de investigação.

A QD também tem merecido atenção comercial, o que se comprova pelo elevado número de empresas que centram as suas actividades nesta área. Estas empresas têm contribuído para o desenvolvimento da área, propondo variadas ferramentas informáticas que suportam as diversas actividades envolvidas no processo de melhoria da QD.

O termo QD pode ser erradamente interpretado como qualidade do modelo que se encontra subjacente aos dados (*e.g.*: qualidade do modelo de dados na representação fiel do mundo real; qualidade do modelo de dados relacional face à teoria da normalização). O termo que caracteriza este tipo de preocupações qualitativas é *qualidade do modelo de dados* [Genero *et al.*, 2002]. O termo QD também pode ser erradamente interpretado como qualidade do esquema subjacente aos dados (*e.g.*: qualidade do esquema em assegurar a unicidade dos valores do atributo; qualidade do esquema em assegurar a integridade referencial). O termo *qualidade do esquema dos dados* [Wohed,

2000] é o que corresponde a este tipo de preocupações qualitativas. Naturalmente, a fraca qualidade do modelo de dados ou do esquema dos dados reflecte-se na qualidade dos próprios dados (*i.e.*, dos valores). Por exemplo: (*i*) a existência de violações de dependências funcionais só acontece em modelos de dados que não respeitam a teoria da normalização (*i.e.*, não estão na terceira forma normal); (*ii*) podem ocorrer violações de integridade referencial em esquemas de dados relacionais que não asseguram a manutenção da integridade. A qualidade do modelo de dados e a qualidade do esquema de dados não fazem parte do âmbito deste trabalho de doutoramento.

Normalmente, quando se fala em QD associa-se o conceito a dados do tipo estruturado. Apesar deste trabalho de doutoramento se cingir a este tipo de dados e mais especificamente aos que são do subtipo relacional, o conceito de qualidade é transversal a todos os tipos de dados existentes, como os de tipo semi-estruturado (*e.g.*: XML) e não estruturado (*e.g.*: documentos textuais).

2.2 Perspectivas Diferentes de Qualidade dos Dados

O conceito de QD pode ser interpretado de forma diferente, em função da perspectiva de abordagem, *i.e.*, estrita ou lata. A definição de QD à luz da perspectiva lata inclui outros aspectos não considerados na perspectiva estrita. Nas duas subsecções seguintes apresenta-se cada uma das perspectivas.

2.2.1 Perspectiva Estrita

Quando se fala em QD o termo é, quase sempre, mencionado de acordo com a perspectiva estrita do mesmo. Assim, QD significa completude, consistência e correcção dos valores armazenados. Por completude, entende-se a existência de todos os valores considerados necessários. Por consistência, entende-se a inexistência de conflitos ou contradições entre os valores. Por correcção, entende-se a inexistência de erros nos valores. Naturalmente, o conceito de correcção está intimamente dependente de cada situação particular. Aquilo que é considerado correcto numa situação, pode não o ser noutra. Estes são os critérios segundo os quais a QD é tradicionalmente analisada. Estes critérios de análise são normalmente designados de *dimensões* da QD. Estas dimensões evidenciam preocupações com a qualidade apenas ao nível dos valores armazenados nas tabelas. Normalmente, esta é perspectiva que os investigadores da área da informática adoptam na definição do conceito, reflectindo a sua visão objectiva sobre o que é a QD.

Em qualquer Fonte de Dados (FD) é quase certo existirem PQ de completude, consistência e correcção dos dados. Mais de metade dos registos criminais informáticos dos Estados Unidos da América foram classificados como estando incorrectos, incompletos ou ambíguos [Strong *et al.*, 1997]. Ao nível da FD individual, são inúmeros os PQ que podem afectar os dados, entre os quais: valores em falta em atributos de preenchimento obrigatório; representações não uniformes dos mesmos dados (*e.g.*: uso de abreviaturas); valores incompletos; múltiplos registos relativos à mesma entidade do mundo real; erros ortográficos; e, inconsistências entre os valores [Lee *et al.*, 1999] [Low *et al.*, 2001].

A importância da qualidade dos valores dos dados foi originalmente identificada em 1950, aquando dos censos nos Estados Unidos da América [Kilss e Alvey, 1985]. No entanto, apenas recentemente se começou a assistir a uma consciencialização de que uma parte importante dos dados, na generalidade das fontes, apresenta PQ. Ao nível do atributo, diversos estudos reportam taxas de ocorrência que vão desde os 0.5% até aos 30% [Redman, 1998]. Naturalmente, há situações onde esta taxa é ainda maior (*e.g.*: um gestor de uma empresa reportou que 60% dos dados transferidos para o armazém de dados não passaram na verificação das regras de integridade que se julgava estarem a ser garantidas pelos sistemas informáticos existentes [Orr, 1998].)

2.2.2 Perspectiva Lata

A luz desta perspectiva, a QD abarca muito mais do que a mera completude, consistência e correcção dos valores. Esta posição tem sido defendida por diversos investigadores (*e.g.*: [Strong, 1997]; [English, 1999]; [Salaun e Flores, 2001]; [Wang e Strong, 1996]; [Ballou *et al.*, 1998]; [Orr, 1998]) que argumentam que a QD deve considerar outros critérios ou dimensões. Apesar de não ignorarem a importância das três dimensões usuais, classificando-as mesmo como as de maior importância [Ballou *et al.*, 1998], argumentam que estas, por si só, não abarcam todas as preocupações que os utilizadores dos dados denotam. Assim, consideram-se outras dimensões, como a actualidade, a acessibilidade e a interpretabilidade, de modo a que se possa caracterizar globalmente a QD. Essencialmente, esta perspectiva tem origem nos investigadores oriundos da área da gestão.

Os inúmeros investigadores que defendem esta posição procuraram definir e caracterizar o significado de QD (*e.g.*: [Wang *et al.*, 1995]). Na realidade, é possível encontrar-se quase tantas definições de QD, quantos os artigos existentes sobre o assunto [Gertz *et al.*, 2004]. Apesar da diversidade de definições, há um consenso de que a QD deve ser definida na óptica dos

consumidores dos dados, daí que a generalidade das definições envolva o conceito de "adequação ao uso" (*e.g.*: [Redman, 1996]; [Shankaranarayanan *et al.*, 2003]; [Wang e Strong, 1996]). Uma definição vulgarmente encontrada na literatura é a seguinte: dados com qualidade são aqueles que são adequados ao uso que o utilizador lhes pretende dar [Wang e Strong, 1996]. Esta perspectiva tem em consideração a finalidade para a qual os dados vão ser usados. No entanto, a definição é por natureza subjectiva, o que faz com que um determinado conjunto de dados possa ser considerado como possuindo qualidade para uma determinada finalidade, mas não ter a qualidade suficiente para uma outra finalidade. Por exemplo, uma lista com nomes e endereços dos habitantes de uma determinada cidade pode ser considerada como tendo qualidade para um utilizador que pretende publicitar um determinado produto. A mesma lista pode ser considerada como tendo pouca qualidade, para um utilizador que pretende contactar apenas os médicos dessa cidade. A QD varia em função do contexto em que estes são usados e dos requisitos a que devem obedecer.

Em [Orr, 1998] também é sugerido que a QD está dependente do uso que os utilizadores dão aos dados, uma vez que estes são quem, em última instância, avaliam se os dados correspondem às suas necessidades. Estas definições implicam que a qualidade seja definida pelo consumidores dos dados, o que significa que não pode ser avaliada de forma independente de quem os usa [Strong *et al.*, 1997]. A qualidade é considerada como análoga de satisfação. A qualidade também pode ser definida como o respeito pelas ou o superar das expectativas dos consumidores. Estes conceitos são originários da área da qualidade dos produtos, tendo estes sido adaptados pelos investigadores para caracterizar o que é a QD. A qualidade de um produto depende, em primeira instância, do seu processo de concepção e produção [Wand e Wang, 1996]. No entanto, um produto perfeito, sob estes pontos de vista, possui pouco valor se não se adequa à finalidade que o consumidor pretende. Assim, as características do produto devem ir de encontro às necessidades do cliente.

2.2.2.1 A Multi-Dimensionalidade da Qualidade de Dados

Diversos estudos empíricos evidenciam que a QD é um conceito multi-dimensional (*e.g.*: [Wang e Strong, 1996]; [Dasu e Johnson, 2003]; [Ives *et al.*, 1983]; [Wang e Kon, 1993]; [Fox *et al.*, 1994]; [Wand e Wang, 1996]; [Wang e Strong, 1996]; [Shanks e Darke, 1998]; [Kahn *et al.*, 2002]), uma vez que na sua definição diferentes dimensões concorrem. Nestes estudos foram efectuadas diversas propostas de dimensões ou critérios que reflectem aspectos importantes da QD para os consumidores e que, na opinião dos seus autores, merecem ser analisados e medidos. As dimensões da QD representam propriedades dos dados (*e.g.*: correcção) [Scannapieco *et al.*, 2004].

A escolha de um conjunto de dimensões, sobre as quais será efectuada a avaliação da QD, constitui o passo inicial de qualquer iniciativa de melhoria da qualidade.

Frequentemente, a QD é expressa sob a forma de uma hierarquia de *categorias*, sendo estas refinadas nas *dimensões* da qualidade [Luebbers *et al.*, 2003]. Um PQD é definido como uma qualquer deficiência encontrada numa ou mais dimensões da qualidade, o que torna os dados impróprios para a finalidade pretendida [Baskarada e Koronios, 2006].

Nos pontos seguintes, referem-se sucintamente seis propostas de dimensões da QD, vulgarmente encontradas na literatura da área.

- **Proposta de Wang e Strong** – Em [Wang e Strong, 1996] são propostas quinze dimensões para a QD, distribuídas por quatro categorias, de acordo com o apresentado na Tabela 2.1. Entre as várias propostas de dimensões da QD, esta é a mais referenciada.

Tabela 2.1 – Dimensões da QD propostas em [Wang e Strong, 1996]

Categoria	Objectivo	Dimensões
Intrínseca	Dimensões referentes à natureza dos próprios dados	Correcção; objectividade; credibilidade; reputação
Representação	Dimensões relativas ao formato e significados dos dados	Interpretabilidade; facilidade de compreensão; representação concisa; representação consistente
Contextual	Dimensões relativas ao contexto de manipulação dos dados	Relevância; valor acrescentado; actualidade; completude; volume de dados
Acessibilidade	Dimensões relativas à segurança e acessibilidade dos dados	Acessibilidade; segurança de acesso

- **Proposta de Wand e Wang** – Em [Wand e Wang, 1996] são propostas as seguintes dimensões da QD: correcção; completude; consistência; actualidade; e, credibilidade.
- **Proposta de Redman** – Em [Redman, 1996] as dimensões da QD estão distribuídas por três categorias: visão conceptual dos dados (cinco dimensões); valores dos dados (quatro dimensões); e, formatos dos dados (oito dimensões).
- **Proposta de Jarke** – Em [Jarke, 1999] as dimensões da QD estão organizadas em função dos papéis dos intervenientes num ambiente de armazém de dados: qualidade de desenho e administração (seis dimensões); qualidade da implementação do software (seis dimensões); qualidade de uso dos dados (cinco dimensões); e, QD armazenados (cinco dimensões).

- **Proposta de Bovee** – Em [Bovee, 2003] são propostas quatro dimensões para a QD: acessibilidade; interpretabilidade; relevância; e, credibilidade.
- **Proposta de Naumann** – Em [Naumann, 2002] são propostas 21 dimensões distribuídas pelas seguintes categorias: conteúdo; técnica; intelectual; e, instanciação.

Efectuando uma comparação entre as anteriores propostas de dimensões, constata-se que não há um consenso sobre qual deve ser o conjunto de dimensões da QD a considerar. A correcção e a completude dos dados são as que recolhem maior unanimidade. Por outro lado, constata-se que não há uma definição comum para cada dimensão. Por vezes, as diferentes propostas até usam o mesmo nome para dimensões com significados diferentes. Na realidade, nalguns casos, há um completo desacordo entre as propostas quanto ao significado de uma determinada dimensão específica (*e.g.*: credibilidade). Mesmo uma dimensão com um significado relativamente óbvio como é o caso da correcção, não possui uma definição unânime. Também se constata a utilização de nomes diferentes com significados iguais ou semelhantes entre as diferentes propostas de dimensões de QD (*e.g.*: a dimensão *clareza de definição* proposta em [Redman, 1996] é semelhante à dimensão *interpretabilidade* que surge nas propostas de [Wang e Strong, 1996], [Bovee, 2003] e, [Naumann, 2002]).

2.2.2.2 Sinopse

À luz da perspectiva lata, a QD é um conceito complexo definido por múltiplas dimensões, algumas das quais muito subjectivas. De forma mais concreta, a QD pode ser definida como um conjunto de dimensões que incluem a correcção, a completude, a consistência, a actualidade, a interpretabilidade e a acessibilidade. Estas correspondem às dimensões partilhadas pela generalidade das propostas. Apesar de existirem várias dimensões diferentes nas várias propostas, é possível identificar um conjunto comum de dimensões que definem a essência do conceito de QD. Todas as restantes dimensões incluídas nas diversas propostas, ou capturam aspectos secundários, ou são muito específicas do contexto.

Apesar da diversidade de definições que é possível encontrar na literatura, foram seleccionadas as que a seguir se apresentam, para definir o significado de cada uma das dimensões atrás referidas.

- **Correcção** – Representa o armazenamento correcto dos valores relativos às propriedades dos objectos do mundo real, *i.e.*, consiste em possuir valores certos [Pipino *et al.*, 2002].
- **Completude** – Representa o armazenamento de todos os dados considerados importantes, de modo a que a ausência ou a insuficiência de detalhe dos dados não condicione a tomada de decisão [Pipino *et al.*, 2002] [Lee e Strong, 2003].

- **Consistência** – Refere-se à inexistência de anomalias sintácticas nos valores dos dados e à inexistência de conflitos/contradições entre os valores [Müller e Freytag, 2002].
- **Actualidade** – Corresponde à exigência dos dados estarem suficientemente actualizados para a execução das tarefas a efectuar [Lee e Strong, 2003].
- **Acessibilidade** – Representa a facilidade e rapidez com que os dados podem ser acedidos pelos utilizadores [Pipino *et al.*, 2002].
- **Interpretabilidade** – Envolve a existência de informação suplementar e meta-dados sobre os dados, com o objectivo de clarificar o seu significado, o que garante a sua utilização de forma adequada [Brackstone, 1999].

2.3 Contextos de Preocupação com a Qualidade dos Dados

Essencialmente, as preocupações com a QD manifestam-se nos seguintes contextos [Galhardas *et al.*, 2000b]:

- Detecção e Correcção (DC) dos PQD (*e.g.*: valores em falta; violações de sintaxe; registos duplicados) existentes ao nível da fonte de dados individual (*e.g.*: base de dados ou ficheiros). Um exemplo de uma situação em que há a necessidade de detectar e corrigir os erros e anomalias existentes na FD ocorre aquando da implementação de um projecto de gestão de relacionamento com o cliente (vulgarmente designado pela sigla CRM – *Customer Relationship Management*).
- Migração de dados pouco estruturados ou não estruturados para uma FD estruturada (*e.g.*: migração dos dados que se encontram num ficheiro de texto para uma tabela de uma base de dados) que facilite a sua manipulação. A implementação de um projecto de *descoberta de conhecimento em bases de dados*[8] (vulgarmente conhecido por *mineração de dados*) constitui um exemplo de uma situação que obriga a que os dados sejam colocados segundo uma representação estruturada.
- Integração de dados provenientes de múltiplas fontes numa nova FD (*e.g.*: criação de um *armazém de dados*[9]). Neste caso, as preocupações com os PQD aumentam substancialmente. É necessário não só solucionar os problemas que possam existir individualmente em cada FD, como também os problemas que possam existir entre estas. Frequentemente, as fontes contêm dados redundantes (*e.g.*: múltiplos registos relativos à

[8] A *descoberta de conhecimento em bases de dados* é um processo analítico concebido para explorar grandes volumes de dados, na procura de padrões consistentes e relacionamentos sistemáticos entre os atributos.
[9] Um *armazém de dados* consiste num conjunto centralizado de dados, normalmente recolhidos a partir de fontes diferentes e concebido com a finalidade de auxiliar na tomada de decisão.

mesma entidade do mundo real) em diferentes representações (*e.g.*: unidades de medida diferentes). Os PQD constituem uma barreira à integração de dados, uma vez que dificultam a identificação dos registos referentes à mesma entidade do mundo real. A reunião de dados provenientes de diferentes sítios *web* constitui um outro exemplo de integração de dados.

2.4 A Fraca Qualidade dos Dados

A fraca QD pode ficar a dever-se a inúmeros factores, repercutindo-se negativamente em diversas situações. Cada um destes aspectos é abordado numa das seguintes subsecções.

2.4.1 Causas

Na literatura é possível encontrarem-se inúmeros exemplos dos diversos factores que contribuem para a fraca QD. Um dos principais factores para a existência de PQD consiste na sua incorrecta introdução [Umar *et al.*, 1999]. Essencialmente, a introdução incorrecta dos dados fica a dever-se à existência de erro humano [Kimball *et al.*, 1998]. Os seguintes exemplos ilustram algumas situações causadoras de PQ [Lee *et al.*, 2000]: uso indiscriminado de abreviaturas como forma de acelerar a velocidade de introdução; erros na digitação/transcrição dos valores; omissões de valores em atributos supostamente obrigatórios; introdução deliberada de valores errados (*e.g.*: valores por defeito; valores fictícios); e, introdução involuntária de valores errados, causados por formulários de recolha de dados confusos.

Outros factores não dependentes de erro humano directo, também são causadores de PQ, como [Lee *et al.*, 2000] [Chaudhuri e Dayal, 1997] [Fayyad *et al.*, 1996]: erros de medição causados por um sensor defeituoso; erro na transmissão dos dados; erro de processamento (*bug*) do sistema informático; e, erro na transformação ou integração dos dados.

A um outro nível, certos factores inerentes à própria organização também contribuem para a existência de PQ, como [Koronios *et al.*, 2005]: estruturas inadequadas que assegurem o reporte completo, preciso e atempado dos dados; inadequação das regras, do treino ou dos procedimentos que a recolha de dados envolve; inconsistências entre os diversos serviços que efectuam a recolha dos dados; e, falta de consciencialização da importância de dados correctos e completos em quem efectua a sua introdução.

A existência ou não de certos PQD (*e.g.*: valores em falta) depende do grau de controlo exercido pelo esquema dos dados e das restrições que controlam os valores permitidos. Nas FD que não

possuem esquema (*e.g.*: ficheiros de texto), há poucas restrições aos valores que podem ser armazenados. Nestes casos, há uma alta probabilidade de existirem vários PQD. Nas FD que possuem esquema (*e.g.*: bases de dados), a existência de certos PQ resulta da ausência de restrições adequadas (*e.g.*: restrições de integridade). A sua inexistência pode ficar a dever-se à pouca qualidade do esquema de dados subjacente ou ao número insuficiente de restrições implementadas para não sobrecarregar o controlo de integridade dos dados.

2.4.2 Consequências

Em [Parssian *et al.*, 2004] demonstra-se o impacto da fraca QD nos resultados produzidos por um sistema de informação. A fraca QD influencia negativamente a sua interpretação, os modelos e as análises criadas a partir destes e compromete as decisões tomadas com base nestes [Kubica e Moore, 2003]. Não é fácil aos gestores tomarem decisões lógicas e bem fundamentadas, com base em informação e conhecimento obtidos a partir de dados afectados por PQ [Low *et al.*, 2001]. Normalmente, a insuficiência de qualidade nos dados leva à tomada de más decisões. Más decisões podem causar desde pequenos inconvenientes operacionais, passar pela diminuição dos lucros e ir até à completa paragem das actividades da empresa [Ballou *et al.*, 2003] [Chengalur-Smith *et al.*, 1999] [Huang *et al.*, 1999]. Estas más decisões conduzem a uma perda de credibilidade no sistema de informação. Por outro lado, a fraca QD tem sérias implicações na satisfação do cliente e, como tal, na sua manutenção.

Em [Olson, 2003] aponta-se a falta de QD como causa para o aumento dos custos (*e.g.*: na implementação de novos sistemas informáticos). Globalmente, estes custos económicos cifram-se em milhões de dólares [Strong *et al.*, 1997] (o mesmo é dizer, milhões de euros). Estes custos reflectem apenas aqueles que são mensuráveis, resultantes do esforço de DC dos problemas [Redman, 2004]. No entanto, os custos da fraca QD não se cingem, meramente, aos que são tangíveis.

Ainda que de forma indirecta, a fraca QD também compromete a capacidade de compreender novas dinâmicas e novos contextos de negócio, novos requisitos dos clientes e a forma de responder a novas oportunidades [Lee *et al.*, 2000b] A fraca QD também contribui para a degradação da imagem pública da organização. Todas estas situações acabam por representar custos para a organização, ainda que sejam muito difíceis, senão mesmo impossíveis, de quantificar. Estes são os custos intangíveis da fraca QD. Em [Redman, 2004] argumenta-se que estes custos representam uma fatia de montante semelhante aos custos tangíveis.

No seu conjunto, os custos tangíveis e intangíveis da fraca QD representam uma percentagem significativa dos proveitos organizacionais. Em [Redman, 2004] aponta-se para um valor que ronda os 20%.

2.5 Gestão da Qualidade dos Dados

Em [Wang, 1998] defende-se a metodologia *Gestão Total da Qualidade dos Dados* (GTQD) (em inglês: *Total Data Quality Mangement*), para a melhoria contínua da qualidade. Na literatura há outras metodologias que visam gerir a QD (*e.g.*: *Sistema da Qualidade Seis Sigma* (em inglês: *Six Sigma Quality System*) [Pande *et al.*, 2000]), no entanto, esta é a mais divulgada. A metodologia GTQD constitui uma adaptação da *Gestão Total da Qualidade* (em inglês: *Total Quality Management*) à área da gestão da QD, com base na abordagem *produto informação* [Wang, 1998].

De acordo com a metodologia, o resultado produzido por um sistema de informação é considerado como sendo um produto, *i.e.*, o *produto informação*, que visa satisfazer as necessidades dos clientes ou consumidores da informação. Da mesma forma que um produto físico tem critérios de qualidade associados, um *produto informação* também tem associados critérios de qualidade [Wang, 1998]. A adopção do termo *produto informação* visa enfatizar o facto dos resultados obtidos a partir dos dados possuírem valor, sendo este transferido para os seus consumidores. O conjunto de sistemas que processam os dados em bruto e os disponibilizam aos seus consumidores sob a forma de produtos de informação é considerado um processo de manufactura de informação. Esta metodologia nasce de uma analogia com o tradicional fabrico de produtos, no qual o sistema produtivo actua sobre materiais em bruto, para gerar os produtos. Apesar do número exacto e das designações das dimensões variarem de investigador para investigador, desde os meados dos anos 90 que a essência desta metodologia possui uma elevada aceitação entre a comunidade de investigação da área.

Tal como se ilustra na Figura 2.1, a metodologia GTQD envolve as seguintes fases na melhoria contínua da QD: *definir*; *avaliar*; *analisar*; e, *melhorar*. O ciclo é análogo ao ciclo de melhoria da qualidade proposto em [Deming, 1986], composto pelas seguintes fases: *planear*; *executar*; *controlar*; e, *actuar*.

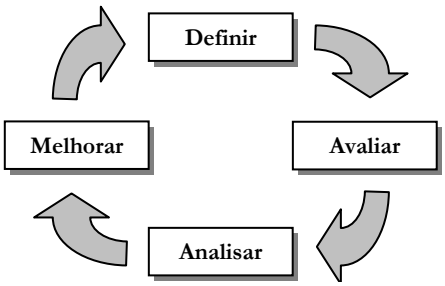


Figura 2.1 – Ciclo da GTQD

A primeira fase do ciclo consiste na definição das dimensões da QD e dos requisitos que estas devem assegurar aos consumidores dos dados. A fase de avaliação visa a produção de métricas que quantifiquem a QD nas dimensões consideradas. A metodologia sugere algumas métricas simples para a medição da QD. O resultado da fase de avaliação, *i.e.*, um conjunto de medidas da QD referentes às diferentes dimensões, serve de base à fase de análise. Nesta fase, analisa-se o processo de manufactura de informação para: (*i*) identificação de configurações problemáticas; (*ii*) detecção das causas das falhas a nível de qualidade; e, (*iii*) avaliação de políticas alternativas para a melhoria da qualidade. Por último, a fase de melhoramento corresponde às iniciativas que procuram eliminar as causas dos problemas e assim melhorar a QD.

2.5.1 Avaliar a Qualidade dos Dados

A avaliação da QD efectuada no contexto da metodologia GTQD ou de uma qualquer outra metodologia de gestão é essencial. Para que tal seja possível é necessário seleccionar-se um conjunto de dimensões e estabelecer-se os relacionamentos entre estas [Bovee *et al.*, 2003]. A avaliação da qualidade é definida como o processo de atribuir valores numéricos ou categóricos a cada dimensão, usando para o efeito métricas da QD [Gertz *et al.*, 2004].

A avaliação da QD envolve dois tipos de avaliações: objectiva e subjectiva. Ambas as avaliações são necessárias para quantificar a QD. De um modo geral, a QD pode ser definida como sendo o valor combinado resultante dos valores das medições efectuadas no conjunto de dimensões da qualidade consideradas [Naumann, 2002].

A *avaliação objectiva ou imparcial* diz respeito à avaliação da QD que é obtida a partir dos próprios dados, *i.e.*, sem considerar outros factores como os utilizadores ou a finalidade para que vão ser usados. Este tipo de avaliação pode ser independente ou dependente do contexto [Pipino *et al.*, 2002]. A *avaliação independente do contexto* reflecte o estado dos dados sem necessitar de qualquer conhecimento adicional sobre o seu domínio, podendo ser directamente efectuada em qualquer

conjunto de dados (*e.g.*: percentagem de valores nulos no atributo). A *avaliação dependente do contexto* obriga a que sejam explicitadas as restrições específicas do conjunto de dados em questão (*i.e.*, do domínio; da organização) (*e.g.*: percentagem de valores que violam o domínio do atributo).

A *avaliação subjectiva ou contextual* considera a utilidade dos dados na satisfação das necessidades dos utilizadores ou para a finalidade que vão ser usados [Shankaranarayan, 2005]. Esta avaliação é efectuada com base no *feedback* dos indivíduos que usam os dados, *i.e.*, dos consumidores dos dados. Este *feedback* é recolhido através de inquéritos aos consumidores dos dados. A avaliação subjectiva da qualidade reflecte as necessidades dos consumidores dos dados [Wang *et al.*, 1998].

Na avaliação da QD, as métricas desenvolvidas podem resultar de três abordagens distintas [Pipino *et al.*, 2002]:

- **Rácio simples** – Diversas métricas da QD assumem esta forma (*e.g.*: consistência; completude). A métrica é definida de acordo com o apresentado na fórmula seguinte. Por exemplo, a completude ao nível do atributo (*i.e.*, coluna de uma tabela) é definida em função do número de valores em falta nesse atributo.

$$1 - \frac{numero\ de\ casos\ com\ o\ problema\ de\ qualidade}{número\ total\ de\ casos}$$

- **Operação mínimo ou máximo** – Na manipulação de dimensões que envolvem a agregação de múltiplos indicadores da qualidade (variáveis) pode usar-se a operação mínimo ou máximo. O mínimo ou o máximo é determinado a partir da lista que contém os valores normalizados referentes a cada indicador de qualidade considerado. A operação mínimo é conservadora, uma vez que atribui à dimensão um valor agregado igual ao menor valor que se encontra nos indicadores da qualidade. Ao invés, a operação máximo representa uma interpretação liberal dos diversos indicadores de qualidade da dimensão. Estes indicadores podem ser calculados recorrendo ao rácio simples.
- **Média ponderada** – Caso haja um perfeito conhecimento da importância de cada indicador de qualidade na avaliação global de uma determinada dimensão, então o uso da média ponderada dos diversos indicadores é adequado. Cada factor de ponderação deve situar-se entre zero e um e o seu somatório tem de ser igual a um.

A avaliação da qualidade constitui uma tarefa difícil, pelos seguintes motivos:

- As dimensões da qualidade subjectivas não podem ser avaliadas automaticamente (*e.g.*: acessibilidade; interpretabilidade), obrigando à realização de inquéritos aos utilizadores.

- Em fontes com volumes de dados muito elevados, a avaliação objectiva da qualidade pode ter de ser efectuada numa amostra dos dados, o que condiciona a precisão dos resultados obtidos.

Em [Pipino *et al.*, 2002], [Shankaranarayanan e Watts-Sussman, 2003] e [Wang, 1998] são apresentadas ferramentas de software específicas que permitem avaliar a QD.

2.5.2 Melhorar a Qualidade dos Dados

A melhoria da QD efectuada no âmbito da metodologia GTQD ou de qualquer outra metodologia do género pode ser alcançada por duas abordagens distintas [Redman, 1996]:

- **Baseada nos dados** – Esta abordagem envolve a detecção automática dos PQD e a sua correcção automática ou manual. O principal inconveniente reside no facto de constituir apenas uma solução temporária, não evitando problemas futuros. Assim, é necessário repetir periodicamente todo o processo de melhoria da QD. A consequência disto é o aumento contínuo dos custos operacionais, uma vez que são necessários recursos humanos e computacionais para detectar e corrigir os problemas. Por outro lado, a abordagem baseada nos dados frequentemente não soluciona satisfatoriamente certos PQD (*e.g.*: valores em falta). A ocorrência destes problemas deve ser evitada logo na sua origem. Assim, de modo a solucionar definitivamente os PQD é necessário identificar as suas reais causas e redesenhar os processos em função destas. Os métodos de detecção de duplicados *record linkage* [Newcombe *et al.*, 1959] e *vizinhança ordenada* [Hernández e Stolfo, 1998] inserem-se neste tipo de abordagem.
- **Baseada no processo** – Esta abordagem centra a sua atenção nas actividades que geram os PQD, originando a reengenharia do processo que pode ir desde a captura dos dados até ao desenho da própria base de dados. As principais actividades que esta abordagem envolve são: monitorar o ciclo de vida dos dados para localizar o aparecimento dos PQ; e, usar controlo de qualidade estatístico e de gestão do processo para assegurar a QD desejada. Esta abordagem possui a vantagem de solucionar os PQD a longo prazo, ainda que possa ser bem mais dispendiosa do que a abordagem baseada nos dados. Na literatura é possível encontrar diversos métodos que seguem esta abordagem, melhorando a QD por intermédio de alterações ao processo, como: funções do processamento de informação [Redman, 1996]; desenvolvimento da função qualidade [Akao, 1990]; sistema de produção de informação de *Ballou* [Ballou *et al.*, 1998]; e, arquitectura IP-MAP [Shankaranarayanan *et al.*, 2000]. A introdução de alterações ao processo, tendo em vista a resolução definitiva dos PQD, fica fora do âmbito deste trabalho de doutoramento.

Na melhoria da QD ambas as abordagens são necessárias [Umar *et al.*, 1999]. Ferramentas e técnicas devem ser usadas para solucionar os PQ que existem nos dados, conjuntamente com alterações ao processo que erradiquem as verdadeiras causas da sua existência. Apesar da importância de que se reveste, a melhoria da QD recorrendo a qualquer uma das abordagens (*i.e.*, dados ou processo) constitui uma tarefa de elevado grau de dificuldade [Naumann, 2002].

2.6 Problemas de Qualidade dos Dados

Neste trabalho de doutoramento a QD cinge-se à correcção, completude e consistência dos valores armazenados (o significado de cada uma destas dimensões encontra-se na Secção 2.2.2.2). Assim contextualizado, o conceito de QD corresponde literalmente ao sentido estrito do termo. Estando definido o significado de QD no âmbito deste trabalho, importa conhecer os problemas que afectam os dados.

A existência de dados com qualidade não é a regra, mas sim a excepção. Normalmente, os dados estão afectados por diversos PQ. Entre outros PQD, os dados podem conter: violações de domínio; sinónimos; violações de restrição de integridade; registos duplicados; e, violações à integridade referencial. Na literatura da área foi possível encontrar três taxionomias de PQD ao nível dos valores dos dados. A forma como os PQD se encontram organizados em cada uma destas taxionomias é apresentada nos pontos seguintes. Uma comparação entre as três taxionomias é efectuada imediatamente a seguir à sua apresentação.

- **Taxionomia de Rahm e Do** – Nesta taxionomia [Rahm e Do, 2000] é feita uma distinção entre PQD mono-fonte e multi-FD. Os problemas mono-fonte e multi-fonte encontram-se divididos nos que se relacionam com o esquema dos dados e nos que se relacionam com as instâncias. Os problemas relacionados com o esquema podem ser evitados melhorando o seu desenho, a sua transformação ou a sua integração. Os problemas nas instâncias correspondem a erros e inconsistências nos dados que não podem ser evitados através do esquema. Nos problemas mono-fonte é efectuada uma distinção entre os problemas que ocorrem no: (*i*) atributo (*e.g.*: valor em falta; erro ortográfico); (*ii*) registo (*e.g.*: violação de dependência funcional); (*iii*) tipo de registo (*e.g.*: registos duplicados); e, (*iv*) FD (*e.g.*: referência errada). Não são fornecidas informações sobre a abordagem usada na identificação dos problemas.
- **Taxionomia de Kim *et al.*** – Nesta taxionomia [Kim *et al.*, 2003] é apresentada uma relação bastante completa de PQD, sendo descrita a lógica subjacente à sua elaboração. Os seus autores adoptam uma abordagem hierárquica descendente, aumentando sucessivamente o grau de detalhe dos problemas. O ponto de partida consiste numa

classificação genérica dos PQD em três grandes classes: (*i*) valores em falta; (*ii*) valores existentes mas errados (*e.g.*: violação de integridade referencial; violação de unicidade); e, (*iii*) valores existentes e correctos mas não utilizáveis (*e.g.*: utilização de abreviaturas; unidades de medidas diferentes). Este último caso advém de não ter sido adoptado um padrão de representação comum aquando da introdução dos valores. A taxionomia surge da decomposição hierárquica destas três classes genéricas de manifestação de PQD.

- **Taxionomia de Müller e Freytag** – Nesta taxionomia [Müller e Freytag, 2003], os PQD são classificados genericamente com sendo sintácticos, semânticos e de cobertura. Os problemas sintácticos dizem respeito aos aspectos relacionados com sintaxe e valores usados na representação das entidades (*e.g.*: violações de sintaxe). Os problemas semânticos impedem os dados de serem uma representação não redundante e não ambígua do mundo real (*e.g.*: registos duplicados). Os problemas de cobertura estão relacionados com a quantidade de entidades e propriedades das entidades do mundo real que, de facto, se encontram armazenadas na tabela (*e.g.*: valores em falta). Nesta taxionomia, além de muito genéricos, os PQD encontram-se limitados aos que ocorrem ao nível de uma só tabela, daí que muitos outros estejam em falta. Nada é dito sobre a forma como se chegou aos problemas apresentados em cada grupo (*i.e.*, problemas sintácticos; problemas semânticos; e, problemas de cobertura).

Quando se comparam os problemas que constituem cada uma das taxionomias constata-se que estes são diferentes. As diferenças não são só ao nível dos termos usados para denominar cada PQD. Apesar de existir um conjunto de problemas comuns, *i.e.*, pertencendo a mais do que uma taxionomia, também há problemas que apenas surgem numa destas (*e.g.*: o PQD *contexto incompleto* apenas é mencionado em [Kim *et al.*, 2003]). Quer isto dizer que nenhuma das três taxionomias é completa, *i.e.*, engloba todos os PQ que afectam os dados. A reunião dos problemas que constam das três taxionomias permite obter um conjunto bastante vasto de PQD, mas que pode não ser completo. Não é fácil detectar eventuais problemas que possam estar em falta neste conjunto quando a sua identificação não resulta de um método sistemático, mas sim de uma compilação de problemas. A propósito disto, a única taxionomia que explicita claramente o método usado na identificação dos problemas é a de [Kim *et al.*, 2003]. Uma vez que neste trabalho de doutoramento se pretende conceber e desenvolver um modelo de DC que dê suporte aos diversos PQD, a sua identificação exaustiva reveste-se de uma especial importância. Estes motivos levaram à elaboração de uma nova taxionomia, cuja apresentação será efectuada no Capítulo 4. Por esta razão, não se enumerou detalhadamente os problemas que fazem parte de cada uma das três taxionomias.

2.7 Processo de Melhoria da Qualidade Baseado nos Dados

A melhoria da QD centrada exclusivamente nos dados constitui um processo que envolve a realização de actividades de detecção e actividades de correcção. As actividades de detecção têm como objectivo auxiliar a identificar alguns PQD, sem ser necessária qualquer intervenção humana. Esta característica é importante nas seguintes situações: (*i*) o utilizador não está disponível/interessado em especificar as operações de detecção a efectuar; (*ii*) ainda que o pretenda, o utilizador não tem capacidades/conhecimentos para efectuar essa especificação. As actividades susceptíveis de serem efectuadas para detecção dos PQD são: *Data Profiling* (DP) e *análise de dados* (AD). As actividades de correcção visam solucionar os PQD detectados ou previamente já conhecidos. As actividades susceptíveis de serem efectuadas para correcção dos PQD são: *transformação de dados* (TD); *limpeza de dados* (LD); e, *enriquecimento de dados* (ED). No caso particular da LD, a detecção dos PQD também é suportada, mas com um cariz diferente da efectuada no DP e na AD. A LD obriga a que as operações a realizar sejam especificadas pelo utilizador. Na realidade, a generalidade dos PQD apenas pode ser identificada com base em conhecimento específico do domínio em causa.

Um processo de melhoria da QD envolve a realização de uma ou mais das anteriores cinco actividades e implica que pelo menos uma dessas actividades seja de correcção. O processo pode ser iterativo, *i.e.*, uma ou mais actividades podem repetir-se várias vezes. Isto acontece em virtude de poder não ser possível atingir o nível de QD pretendido com uma única iteração. Muitas vezes é necessário analisar os resultados obtidos, refinar as decisões tomadas para melhorar a QD e re-executar, novamente, a actividade ou actividades em causa. A intervenção do utilizador pode, ainda, ser necessária para solucionar certos PQD. Nalguns casos apenas o utilizador pode, manualmente, solucionar os problemas, uma vez que a sua resolução depende de uma análise individualizada. Estes PQD não são susceptíveis de serem manipulados de forma automática.

Nas subsecções seguintes são descritas as diversas actividades que o processo de melhoria da QD compreende. Este trabalho de doutoramento enquadra-se, fundamentalmente, na actividade de LD. As restantes actividades são apresentadas com o intuito de dar uma visão geral de tudo o que está envolvido no processo de melhoria da qualidade.

2.7.1 Data Profiling

O DP consiste em aplicar técnicas de análise com o objectivo de determinar o conteúdo, estrutura e qualidade dos dados existentes [Olson, 2003]. O DP é diferente da AD efectuada com

fins empresariais/comerciais por sistemas de apoio à decisão e sistemas de mineração de dados. Este tipo de sistemas permite adquirir conhecimento *a partir* dos dados. Por sua vez, o DP permite adquirir conhecimento *acerca* dos próprios dados, *i.e.*, sobre as suas características e padrões. Estes meta-dados são essenciais para a identificação dos PQD. Na sua geração são usadas técnicas analíticas e de descoberta. O DP deve ser a primeira actividade a executar no processo de melhoria da QD. Qualquer iniciativa baseada nos dados (*e.g.*: criação de um armazém de dados; projecto de mineração de dados; projecto de integração de dados) deve iniciar-se com DP.

O DP gera meta-dados precisos e completos, efectuando a sua reengenharia com base nos dados existentes. O outro resultado produzido pelo DP consiste na identificação dos registos e valores que violam os meta-dados correctos. As discrepâncias identificadas constituem um ponto de partida para a investigação das causas subjacentes. As ferramentas de DP possuem capacidades interactivas de navegação que permitem analisar os dados e visionar as instâncias onde ocorrem os problemas. Os principais passos efectuados no DP são [Olson, 2003]:

- **Análise das propriedades do atributo** – A criação do perfil do atributo auxilia a verificar se os dados que nele se encontram correspondem às expectativas. Neste passo são analisados os valores que se encontram em cada atributo (*i.e.*, coluna da tabela) e inferidas as suas características, como: tipo de dados e respectivos tamanhos; valores ou intervalo de valores existentes; cardinalidade; e, percentagem de valores nulos. Este tipo de análise também inclui o recurso a técnicas de reconhecimento de padrões sintácticos que permitem verificar se os valores de um atributo possuem o formato esperado (*e.g.*: se um atributo só contém valores numéricos; se possui tamanhos consistentes). Algumas estatísticas básicas acerca dos valores do atributo são também geradas (*e.g.*: valores mínimos e máximos; média; mediana; moda; desvio padrão). As estatísticas são úteis em todos os tipos de dados, especialmente nos de tipo numérico. A criação do perfil do atributo também produz frequências de ocorrência. As frequências de ocorrência permitem analisar a distribuição dos valores existentes nos dados. O perfil do atributo auxilia a identificar alguns PQD (*e.g.*: o facto da cardinalidade do atributo *sexo* ser superior a dois indica a existência de um PQD).
- **Análise de estrutura** – Permite criar um perfil de dependências através da análise dos valores entre registos. Os valores de cada atributo são comparados com os valores que se encontram nos restantes atributos. A análise de estrutura permite inferir todas as dependências funcionais que possam existir entre os atributos de uma tabela. Durante o processo são identificadas *pseudo-dependências* que são verdadeiras na maioria dos casos, mas não o são sempre. Normalmente, isto indica a existência de um PQD (*i.e.*, violação de dependência funcional). As chaves primárias também são identificadas durante este

passo. A análise de estrutura também permite criar um perfil de redundâncias, através da comparação dos valores entre tabelas da mesma ou de fontes diferentes. Da comparação, resulta a identificação dos atributos que contêm a mesma informação, mas nomes diferentes (*i.e.*, sinónimos) e atributos que apesar de terem o mesmo nome, contêm dados com significados diferentes (*i.e.*, homónimos). A análise de estrutura também auxilia a identificar quais os atributos redundantes e que podem ser eliminados, bem como os que são necessários para conectar os dados entre as tabelas, *i.e.*, as chaves estrangeiras necessárias à manutenção da integridade referencial.

- **Análise das restrições de integridade (regras de negócio)** – Uma restrição de integridade especifica uma condição que tem de ser verdadeira, envolvendo um ou mais atributos de uma ou mais tabelas (*e.g.*: os funcionários tem de ter pelo menos 16 anos de idade). Estas regras são recolhidas a partir do perito do domínio, convertidas para uma lógica executável e testadas nos dados. Uma restrição de integridade pode envolver: verificações de domínio; validações, usando tabelas de pesquisa; ou fórmulas específicas. Como resultado, são identificados os registos que violam as restrições de integridade. Uma ferramenta robusta de DP suporta a construção, o armazenamento e a validação das restrições de integridade específicas de cada organização.

Existem várias ferramentas comerciais de DP (*e.g.*: *Datiris Profiler* [Dataris, 2007]; *ETI Data Profiler* [ETI, 2007b]; *Trillium Discovery* [Trillium Software, 2007b]; *dfPower Profile* [DataFlux, 2007b]; *HIQuality Inspect* [Human Inference, 2007b]). No Apêndice A – Secção A.1 apresenta-se uma descrição da ferramenta *Datiris Profiler*. A informação técnica disponível sobre as diversas ferramentas permite concluir que todas denotam, sensivelmente, as mesmas potencialidades. As principais diferenças residem na interface com o utilizador e na capacidade de acederem a diferentes tipos de FD. A revisão de literatura efectuada não conduziu à identificação de qualquer protótipo de investigação de DP.

2.7.2 Análise de Dados

A AD baseia-se nos dados para inferir padrões, relacionamentos e regras subjacentes a estes. Desta forma, a AD permite detectar e solucionar automaticamente alguns PQ (*e.g*, violações de dependências funcionais; violações de restrições de integridade). O recurso à AD justifica-se pela existência de operações de negócio cada vez mais complexas e opacas [Dasu *et al.*, 2003]. De referir que os termos *descoberta nos dados* e *mineração da QD* (em inglês: *data quality mining*) [Hipp *et al.*, 2001] por vezes também são usados como sinónimos de AD. Esta última designação tem origem no facto da AD poder ser vista como uma variante da mineração de dados. A AD envolve duas tarefas: inferência de estrutura e detecção de anormalidades [Luebbers *et al.*, 2003].

Na inferência de estrutura são usados métodos estatísticos (*e.g.*: regressão linear) e algoritmos de mineração de dados (*e.g.*: árvores de decisão; redes neuronais; indução de regras) [Luebbers *et al.*, 2003]. Por *estrutura*, entende-se qualquer forma de regularidade (*i.e.*, padrão, regra ou relacionamento) que possa ser encontrado (*e.g.*: um conjunto de regras ou árvores de decisão que criem uma decomposição estatística válida em subclasses). Para que se possa inferir uma estrutura é, no entanto, necessário que a QD seja a suficiente. Por outro lado, a inferência da estrutura, também depende da adequação dos algoritmos de mineração de dados ao domínio em questão.

Na detecção de anormalidades são identificadas todas aquelas situações que são excepcionais (*e.g.*: valores inconsistentes; valores em falta). Tudo aquilo que não se enquadra no padrão normal é assinalado como potencial PQD e, com base naquele, é proposta uma provável correcção (*e.g.*: valor que elimina uma inconsistência; previsão de um valor em falta) [Luebbers *et al.*, 2003]. Nem sempre o pressuposto anterior faz sentido, *i.e.*, aquilo que aparenta ser uma anormalidade até pode estar correcto e ser de grande importância em análises subsequentes. Assim, a correcção dos valores tem de ser sempre supervisionada por um perito do domínio.

Existem várias ferramentas comerciais que efectuam AD (*e.g.*: *WizWhy* [WizSoft, 2007a]; *WizRule* [WizSoft, 2007b]). No Apêndice A – Secção A.2 descreve-se a ferramenta *WizRule*. O protótipo *Bellman* [Dasu *et al.*, 2002] e o desenvolvido na *Ken State University* [Maletic & Marcus, 2000a] são os representantes da comunidade académica. Apesar das ferramentas de AD sugerirem correcções, normalmente estas não dispensam o uso de ferramentas de LD para a realização das correcções. O principal objectivo das ferramentas de AD centra-se na detecção dos potenciais problemas e não na sua correcção.

2.7.3 Transformação de Dados

A TD representa o conjunto de operações (*e.g.*: integração; agregação; filtragem; concatenação; divisão) que um conjunto de dados que se encontra segundo o esquema $\mathcal{X}$ (de origem) tem de ser sujeito para que fique de acordo com o esquema $\mathcal{Y}$ (de destino). Este tipo de operações resulta em modificações ao nível do esquema dos dados, materializadas em transformações ao nível dos valores. A migração de dados e a criação de armazéns de dados são dois cenários típicos que obrigam à realização de operações de TD. A necessidade destas operações pode resultar, entre outras possibilidades, do seguinte:

- O conjunto de atributos utilizados nos esquemas de dados para caracterizar cada entidade do mundo real é diferente (*e.g.*: o atributo *sexo* existe no esquema de destino, mas não existe no esquema de origem).

- Tipos de dados diferentes são usados para representar o mesmo atributo (*e.g.*: o atributo *data_factura* é do tipo data no esquema de destino, enquanto que no esquema de origem é do tipo *string*).
- Modelos de dados diferentes (*e.g.*: no esquema de origem, o atributo *localidade* integra a tabela *clientes*, enquanto que no esquema de destino, o mesmo atributo faz parte da tabela *código_postal*).

A TD implica que sejam efectuados *mapeamento de dados* que estabelecem associações entre os atributos do esquema de origem e os atributos do esquema de destino e sejam especificadas as transformações que devem ocorrer entre estes. Frequentemente, estes mapeamentos são dificultados pela necessidade de transformações complexas, envolvendo associações entre atributos do tipo *um-para-muitos* e *muitos-para-um*.

Normalmente, a TD é efectuada no contexto mais abrangente da criação de armazéns de dados, envolvendo também operações de extracção e carregamento. Este processo é, vulgarmente, conhecido por *extracção, transformação e carregamento*. Na fase de extracção, os dados são "retirados" das fontes onde se encontram. A generalidade dos projectos de armazéns de dados procede à integração de dados provenientes de múltiplas fontes, eventualmente com modelos de representação de dados diferentes (*e.g.*: bases de dados relacionais; ficheiros de texto; ficheiros binários). Na fase de carregamento, os dados transformados são "colocados" no armazém de dados. Nalguns armazéns de dados procede-se meramente à substituição dos dados que possam existir pelos novos dados. Noutros armazéns de dados mais evoluídos é mantido um histórico que regista todas as alterações efectuadas, suportando carregamentos incrementais periódicos dos dados.

No mercado encontram-se disponíveis inúmeras ferramentas de TD. A título meramente exemplificativo referem-se as seguintes: *dfPower Quality* [DataFlux, 2007a]; *TS Quality* [Trillium Software, 2007a]; *HIQuality* [Human Inference, 2007c]; e *Oracle Data Integrator* [Oracle Data Integrator, 2007]. Esta última é apresentada no Apêndice A – Secção A.3. Os protótipos de investigação *Ajax* [Galhardas *et al.*, 2001], *Arktos II* [Vassiliadis *et al.*, 2003] e *Potter's Wheel* [Raman e Hellerstein, 2001] constituem os representantes da comunidade académica que suportam a realização de TD. Estes protótipos encontram-se descritos no capítulo seguinte (Secção 3.5.1).

2.7.4 Limpeza de Dados

A LD (em inglês: *data cleaning*; *data cleansing*; *data scrubbing*) visa a DC dos PQ que afectam os valores dos dados [Milano *et al.*, 2005]. A LD pode ser vista como um processo que envolve a

realização sequencial dos seguintes passos: (*i*) especificação dos PQD cuja existência se pretende verificar; (*ii*) detecção automática dos PQD que existem nos dados; e, (*iii*) correcção automática ou manual dos PQD identificados.

A fronteira entre LD e TD nem sempre é clara. Na literatura da área acontece, por vezes, os termos serem usados conjuntamente ou mesmo indistintamente. No âmbito desta dissertação, efectua-se uma distinção entre ambas uma vez que em rigor não são o mesmo. A TD visa a realização de operações que alteram os dados ao nível do esquema (*i.e.*, os dados transitam do esquema $\mathcal{X}$ para o esquema $\mathcal{Y}$). Naturalmente, estas alterações ao nível do esquema repercutem-se nos valores dos próprios dados (*e.g.*: um atributo (*e.g.*: nome) do esquema de origem pode corresponder a dois atributos (*e.g.*: nomes próprios e apelidos) no esquema de destino). A LD visa manipular os PQ que afectam os dados ao nível dos seus valores, não produzindo modificações ao seu esquema de representação (*i.e.*, os dados permanecem de acordo com o esquema $\mathcal{Z}$).

Alguns autores atribuem um destaque especial à eliminação de duplicados, considerando-a como mais uma actividade do processo de melhoria da QD (*e.g.*: [Barateiro e Galhardas, 2005]). Neste trabalho considera-se que os duplicados estão abrangidos nesta actividade de LD, não merecendo tal destaque. A própria definição de LD adoptada (*i.e.*, DC dos PQ que afectam os valores dos dados) justifica a opção tomada. Afinal de contas, os duplicados são apenas mais um PQD que deve ser adequadamente manipulado pela LD.

No mercado há inúmeras ferramentas comerciais que efectuam LD. Entre estas, encontram-se as seguintes: *WinPure Clean and Match* [WinPure, 2007]; *ETI Data Cleanser* [ETI 2007a]; *dfPower Quality* [Data Flux, 2007a]; *TS Quality* [Trillium Software, 2007a]; e *HIQuality* [Human Inference, 2007a]. Estas ferramentas são apresentadas no capítulo seguinte (Secção 3.5.2). Além destas, foram também analisadas as seguintes ferramentas de LD: *WizSame* [WizSoft, 2007c]; *matchIT* [helpIT, 2007]; e, *Centrus Merge/Purge* [Group 1 Software, 2007]. Por não terem sido consideradas tão representativas e relevantes quanto as anteriores, foram remetidas para o Apêndice A (respectivamente, Secção A.4, Secção A.5, Secção A.6 e Secção A.7).

A comunidade académica também contribuiu para a LD com alguns protótipos. Entre estes, é possível encontrar os seguintes: *Ajax* [Galhardas *et al.*, 2001]; *Arktos II* [Vassiliadis *et al.*, 2003]; *IntelliClean* [Low *et al.*, 2001]; *Potter's Wheel* [Raman e Hellerstein, 2001]; e *FraQL* [Sattler e Schallehn, 2001]. Estes protótipos de investigação são apresentados no capítulo seguinte (Secção 3.5.1).

2.7.5 Enriquecimento de Dados

Frequentemente, os dados estão incompletos ou desactualizados. O ED (em inglês: *data enrichment*) consiste em melhorar a sua precisão e completude, com base nos dados existentes noutras fontes, internas ou externas à organização. A nível de fontes externas, há diversas empresas que se dedicam à comercialização de dados, principalmente sobre indivíduos e organizações empresariais. Independentemente da origem das FD de referência, o objectivo é aumentar o valor dos dados actuais da organização.

As fontes normalmente usadas no ED são: (*i*) informação postal, usada na validação e correcção de endereços; (*ii*) dados geográficos (*e.g.*: latitude; longitude), usados na análise de localizações; (*iii*) os próprios dados da organização se dispersos por diferentes sistemas, com diferentes níveis de QD; e (*iv*) dados demográficos, usados para melhor caracterizar os indivíduos (*e.g.*: sexo; idade; rendimento mensal).

O ED resulta nos seguintes benefícios:

- Permite identificar e corrigir valores que aparentemente estão correctos, mas que na realidade não o estão (*e.g.*: um endereço postal que apesar de sintacticamente correcto corresponde a uma morada inexistente).
- Fornece dados complexos como a latitude e longitude geográficas, cuja introdução não pode ser efectuada através dos métodos tradicionais.
- Aumenta o valor intrínseco dos dados sem sobrecarregar o seu processo de recolha ou introdução. Saliente-se que o ED pode obrigar a que sejam efectuadas mudanças ao nível do esquema, *i.e.*, a que novos atributos sejam acrescentados de modo a que os novos valores possam ser armazenados.

Um aspecto de extraordinária importância no ED é o da elevada qualidade da FD de referência. Os dados estão a ser confrontados com uma FD que necessariamente tem de ter um nível de qualidade superior. Caso contrário, do processo não resulta a esperada melhoria da QD. O principal problema do ED reside na inexistência de um mecanismo de notificação das alterações que ocorrem na própria FD de referência. Assim, é necessário proceder, não só ao enriquecimento dos dados novos como também, de forma periódica, dos que já tinham sido anteriormente enriquecidos.

No mercado, há diversas ferramentas comerciais que procedem ao ED (*e.g.*: *dfPower Quality* [DataFlux, 2007a]; *TS Quality* [Trillium Software, 2007a]; *HIQuality* [Human Inference, 2007h]). Estas ferramentas baseiam-se nos seus próprios dados que funcionam como uma espécie de base

de conhecimento para efectuarem o ED. Também permitem o acesso a FD de referência comercializadas por terceiros. O protótipo de investigação *Ajax* [Galhardas *et al.*, 2001] originário da comunidade académica é o único que suporta a realização de ED.

2.8 Conclusão

Neste capítulo abordou-se genericamente a temática da QD. As duas interpretações que o termo suscita foram apresentadas. Estas interpretações foram identificadas na literatura existente e resultam fundamentalmente de perspectivas de abordagem diferentes, uma estrita e a outra lata. Quando o conceito é abordado de forma estrita, a QD significa completude, correcção e consistência dos valores dos dados. Quando o conceito é abordado de forma lata, QD significa muito mais do que estas três vertentes, *i.e.*, são consideradas outras dimensões que se advogam também importantes na QD (*e.g.*: actualidade; acessibilidade; interpretabilidade) e que lhe conferem um carácter muito mais multi-dimensional. A esta perspectiva falta ainda alguma maturidade científica, o que se pode comprovar nas inúmeras propostas de dimensões da QD e na falta de uniformização terminológica que existe entre estas. A isto não será alheio certamente a sua relativa juventude. Nesta dissertação, o conceito de QD corresponde à perspectiva estrita do mesmo, *i.e.*, cinge-se à correcção, completude e consistência dos valores. Mesmo assim, entendeu-se que a temática da QD não ficaria completa sem a inclusão da perspectiva lata, até porque o seu número de defensores é bastante significativo.

Neste capítulo foram também expostas e comparadas três taxionomias que abarcam os PQD ao nível dos valores dos dados. A principal conclusão alcançada é que nenhuma das taxionomias é completa. Esta conclusão resulta do facto de nenhuma das taxionomias cobrir todos os PQD apresentados nas restantes. Uma vez que constitui objectivo deste trabalho o desenvolvimento de um modelo de LD que suporte os vários PQD, daí que uma taxionomia completa assuma especial relevância. A elaboração de uma nova taxionomia tornou-se assim num requisito a cumprir.

As diversas actividades passíveis de serem efectuadas num processo de melhoria da qualidade (*i.e.*, DP; AD; TD; LD; e, ED) baseado unicamente nos dados foram descritas. Apesar deste trabalho de doutoramento se centrar na LD, esta actividade foi apresentada resumidamente, mantendo uma coerência de apresentação com as demais actividades do processo de melhoria da QD. No entanto, há que conceder necessariamente uma atenção especial à LD, o que será efectuado no capítulo seguinte.

3 LIMPEZA DE DADOS

Este capítulo debruça-se detalhadamente sobre a Limpeza de Dados (LD). Este trabalho de doutoramento enquadra-se nesta actividade do processo de melhoria da Qualidade dos Dados (QD). É por este motivo que é dado destaque a esta actividade, em detrimento das restantes apresentadas no capítulo anterior. Assim, após se introduzir o tema, começa-se por apresentar os principais contextos em que a LD assume especial importância. Seguidamente, são expostos os requisitos a que uma actividade de LD tem de obedecer. Na secção seguinte apresentam-se, com algum detalhe, os aspectos mais importantes relacionados com o principal Problema de Qualidade dos Dados (PQD) que a LD enfrenta: os duplicados. No contexto desta problemática descreve-se: os métodos de detecção de duplicados considerados mais relevantes; o âmbito dos métodos de detecção; as métricas utilizadas na avaliação da semelhança entre valores; a avaliação da eficácia dos métodos de detecção; e, as abordagens susceptíveis de serem utilizadas na eliminação de duplicados. Por fim, descreve-se o actual *estado da arte* a nível de soluções informáticas que permitem um tratamento automatizado à LD. Para o efeito, é exposto um conjunto de ferramentas representativas, oriundas de duas comunidades distintas: académica/investigação e empresarial/comercial. Em ambos os casos as ferramentas são comparadas, tendo por base um conjunto de características seleccionadas para o efeito.

3.1 Introdução

A solução natural para os Problemas de Qualidade (PQ) que possam existir passa por analisar os dados com o objectivo de os identificar e solucionar [Maletic e Marcus, 2000b]. Efectuar esta tarefa através da análise visual dos dados é fastidioso, moroso e propício à não identificação dos PQD [Guyon *et al.*, 1996]. Obviamente, não é uma tarefa praticável em grandes volumes de dados. Como tal, esta tarefa tem de ser automatizada o mais possível. Foi neste contexto que surgiu a LD (em inglês: *data cleaning*; *data cleansing*; *data scrubbing*).

A LD tem sido objecto de um interesse crescente ao longo dos últimos anos, apesar de ser uma área relativamente recente. A definição original de LD envolvia apenas a detecção e eliminação de duplicados. Actualmente, a definição de LD tem um âmbito mais lato. Apesar de não existir uma

definição universal, as várias definições que podem ser encontradas na literatura apresentam semelhanças. Nesta dissertação adopta-se a que é proposta em [Milano *et al.*, 2005] devido ao seu caracter genérico e abrangente. Assim, a LD é definida como o processo de detecção e eventual correcção automática dos PQ que afectam os dados. Por PQ entende-se erros e anomalias que existem ao nível dos valores dos dados. A definição do que é um PQD está, no entanto, dependente de cada caso particular (*e.g.*: domínio; Base de Dados (BD)). Por este motivo, uma solução de LD que suporte de forma totalmente automática, *i.e.*, sem intervenção humana, todos os PQD é impossível de alcançar [Dasu *et al.*, 2003]. A LD é vista como um processo composto pelas seguintes fases: definir os tipos de PQD a pesquisar; pesquisar e identificar os PQD; e, corrigir os problemas encontrados. Cada uma das fases constitui por si só um problema complexo. Os investigadores desta área têm proposto métodos e técnicas que se enquadram em cada uma das fases (*e.g.*: métodos de detecção e eliminação de duplicados; técnicas de preenchimento de valores em falta).

3.2 Contextos de Aplicação

A realização de LD faz sentido em qualquer contexto em que a inexistência de PQD seja um requisito. Em particular, a LD justifica-se nos mesmos contextos genéricos em que surgem preocupações com a QD (ver Capítulo 2 – Secção 2.3), *i.e.*: (*i*) Detecção e Correcção (DC) de erros e anomalias existentes ao nível da Fonte de Dados (FD) individual; (*ii*) migração de dados pouco estruturados ou não estruturados para uma FD estruturada; e, (*iii*) integração de dados provenientes de múltiplas fontes numa nova FD.

Há, no entanto, dois contextos específicos em que a LD se reveste de uma especial importância: armazéns de dados e descoberta de conhecimento em BD. Esta importância reflecte-se no facto da LD ser parte integrante de cada um destes processos. A LD integra o primeiro passo, designado de *pré-processamento*, do processo de descoberta de conhecimento em BD (ou mineração de dados). Neste contexto, a LD é efectuada para que o princípio "lixo entra, lixo sai" não se verifique [Lee *et al.*, 1999]. No contexto dos armazéns de dados, a LD antecede a fase de carregamento e constitui um importante passo de preparação que assegura um elevado nível de QD [Feekin e Chen, 2000]. Um estudo efectuado pelo *Metagroup* revela que 41% dos projectos de armazéns de dados falham, apontando a QD como a principal causa identificada [Luebbers *et al.*, 2003]. A implementação com sucesso de um armazém de dados só pode ser conseguida por intermédio da boa QD [Wang *et al.*, 2003]. A problemática da QD em ambientes de armazéns de dados tem sido objecto de estudo em diversos trabalhos académicos (*e.g.*, em [Costa, 2006]). De

notar que por vezes, também se recorre à criação de armazéns de dados como forma de solucionar alguns dos PQ existentes [Lee *et al.*, 2000b].

Em suma, a LD antecede a aplicação das tecnologias de dados de alto nível actualmente existentes, tendo como finalidade a DC dos PQD que possam existir. No entanto, a tarefa de limpar os dados é das mais difíceis e onerosas. Estima-se que entre 50 a 70% do tempo total necessário à implementação de um projecto de descoberta de conhecimento em BD ou criação de um armazém de dados seja dispendido na fase de LD [Liu, 1999].

Nos últimos anos estas duas áreas (*i.e.*, descoberta de conhecimento em BD e armazéns de dados) receberam muita atenção entre a comunidade científica da área das BD. No passado, os esforços de investigação centraram-se, quase em exclusivo, no problema da integração de esquemas de dados. O objectivo passava por reconciliar diferenças ao nível do esquema. A integração de esquemas foi investigada em inúmeros trabalhos que propuseram arquitecturas de integração (*e.g.*: [Calvenese, 1999], [Metais *et al.*, 1997]); sistemas mediadores (em inglês: *mediator systems*) (*e.g.*: [Papakonstantinou, 1997], [Bressan et al., 1997]); e, resolução de conflitos ao nível do esquema (*e.g.*: [Batini *et al.*, 1986], [Lee e Ling, 1997]). Comparativamente, o próximo passo lógico, *i.e.*, identificar e solucionar problemas ao nível dos valores tem merecido pouca atenção entre a comunidade científica.

3.3 Requisitos

Especificar e implementar um processo de LD que alcance bons resultados envolve alguns requisitos:

- Em primeiro lugar, um processo de LD não pode ser executado sem o envolvimento de um perito do domínio nas suas várias fases. Quer a detecção quer a correcção dos PQD não podem ser automatizadas na totalidade. Na realidade, para que se possa concretizar um processo de LD é necessário fornecer-se o conhecimento necessário à DC dos problemas. Este conhecimento apenas pode ser transmitido por peritos do domínio [Dasu *et al.*, 2003].
- Em segundo lugar, em virtude da natureza iterativa do processo de LD é essencial ter-se a possibilidade de modificar toda a especificação e implementação do processo de LD.
- Em terceiro lugar, como os peritos do domínio são um recurso caro é importante automatizar o máximo de trabalho possível, antes destes serem chamados a intervirem no

processo de limpeza. É necessário fornecer-lhes o máximo de informação possível, de modo a que possam tomar decisões bem fundamentadas.

- Por último, quando o volume de dados envolvidos é elevado, o processo de LD torna-se muito dispendioso computacionalmente. Este aspecto torna-se particularmente crítico quando a LD envolve a detecção e eliminação de duplicados. As novas tecnologias disponibilizam uma maior capacidade de processamento e uma maior velocidade, o que permite efectuar os processos de LD em tempos aceitáveis, mesmo com volumes de dados muito elevados. Mesmo assim, a utilização de técnicas de optimização mostra-se necessária de modo a manter este custo computacional em níveis aceitáveis.

3.4 O Problema dos Duplicados na Limpeza de Dados

Um duplicado é um registo que representa uma entidade do mundo real já representada noutro registo [Verykios *et al.*, 2003]. Os duplicados podem ser exactos (*i.e.*, rigorosamente iguais) ou aproximados (*i.e.*, com ligeiras diferenças de representação). Frequentemente, as BD contêm redundâncias deste tipo [Guyon *et al.*, 1996]. Este problema coloca-se não só ao nível da tabela individual (*i.e.*, identificar e eliminar duplicados no interior de uma tabela), como também ao nível de múltiplas tabelas (*i.e.*, identificar e eliminar registos que representam a mesma entidade em tabelas diferentes) [Bilenko e Mooney, 2003] [Sarawagi e Bhamidipaty, 2002] [Verykios *et al.*, 2003]. Este último caso assume especial importância na integração de dados provenientes de fontes diferentes.

A detecção e eliminação de registos duplicados constitui a tarefa da LD a que tem sido dada maior atenção. Há pelo menos cinquenta anos que se reconhece que o problema é importante [Monge, 2000]. Esta realidade é comprovada pelo elevado número de publicações, o que constitui um indicador da sua importância. O problema foi originalmente identificado em [Newcombe *et al.*, 1959], tendo este autor usado o termo *record linkage* para o designar. O âmbito do trabalho centrou-se na identificação de registos médicos relativos ao mesmo indivíduo, mas referentes a períodos de tempo diferentes. Depois deste, muitos outros trabalhos de investigação debruçaram-se, debruçam-se e certamente continuarão a debruçar-se sobre esta problemática da detecção e eliminação de duplicados. No entanto, não há um termo universalmente aceite que a caracterize, como se pode constatar pelas diferentes designações encontradas na literatura além da inicialmente proposta em [Newcombe *et al.*, 1959]: problema da identidade do objecto (em inglês: *object identity problem*) [Monge, 2000] [Naumann e Häussler, 2002]; problema de fundir/remover (em inglês: *merge/purge problem*) [Hernández e Stolfo, 1995] [Hernandez e Stolfo,

1998]; detecção de duplicados [Monge e Elkan, 1997] [Sarawagi e Bhamidipaty, 2002]; correspondência de referências (em inglês: *reference matching*) [McCallum *et al.*, 2000]; agrupamento e correspondência de nomes de entidades (em inglês: *entity name clustering and matching*) [Cohen e Richman, 2002]; problema da identificação da instância [Maletic e Marcus, 2000b]; identificação da entidade (em inglês: *entity identification*) [Cholvy e Moral, 2001]; e, identificação do objecto [Tejada *et al.*, 2001] [Tejada *et al.*, 2002].

A existência de registos duplicados pode ser uma consequência directa de outros PQ nos dados, como: valores em falta; valores incorrectos (*e.g.*: motivados por erros de introdução ou problemas de actualização); inexistência de uma uniformidade de representação (*e.g.*: acrónimos; abreviaturas); e, valores incompletos. A existência de valores incompletos é usual em atributos cujo tipo é textual [Gravano *et al.*, 2003]. Os sistemas de gestão de BD relacionais não permitem a introdução de registos que contenham valores duplicados nas chaves primárias. No entanto, como os valores dos atributos da chave primária podem estar afectados por PQD (*e.g.*: valores incorrectos devido a erros de digitação), não é possível garantir a não existência de registos duplicados. Mesmo não existindo estes PQD, a mesma entidade ou objecto do mundo real pode ter sido armazenada com valores de chave primária diferentes (*e.g.*: o mesmo cliente encontra-se armazenado em dois registos com um número de cliente diferente). Como consequência destas situações, vários registos podem representar a mesma entidade ou objecto do mundo real, *i.e.*, serem equivalentes do ponto de vista semântico ainda que não o sejam sintacticamente [Lee *et al.*, 1999] [Monge, 2000] [Bilenko e Mooney, 2003]. Por exemplo, um determinado registo pode armazenar dados relativos a um indivíduo em que este está identificado pelo primeiro e último nome (*e.g.*: António Silva). Um outro registo pode armazenar dados sobre o mesmo indivíduo em que este está identificado pelas iniciais dos nomes próprios e último apelido (*e.g.*: A. C. Silva).

Entre outros problemas, a existência de duplicados influencia negativamente a realização de análises estatísticas. Quando não se procede à sua remoção, frequências de distribuição e operações de agregação produzem resultados errados e enganadores. Por outro lado, podem impedir que algoritmos de mineração de dados descubram padrões importantes ou induzi-los na descoberta de padrões errados [Bilenko e Mooney, 2003].

3.4.1 Métodos de Detecção de Duplicados

A detecção de duplicados envolve a identificação de registos semelhantes que correspondem à mesma entidade ou objecto do mundo real. Na detecção de duplicados, um dos principais problemas prende-se com a decisão de quando é que dois registos devem ser considerados

duplicados, apesar das diferenças que possam existir entre estes. Há duas estratégias distintas para determinar a existência de registos duplicados. A primeira é designada de *exacta* ou *determinística*, sendo usada quando há um identificador na tabela ou tabelas que permite identificar os duplicados existentes (*e.g.*: o número de contribuinte). A segunda estratégia de identificação de registos duplicados designa-se de *aproximada*. A principal dificuldade na detecção de duplicados surge quando não existe um atributo que possa ser usado na identificação dos duplicados. Mesmo quando este existe, os seus valores podem estar afectados por PQD resultantes de variadas situações (*e.g.*: erros de digitação, erros de transcrição) [Verykios *et al.*, 2003]. Como tal, a identificação dos registos duplicados poderá não ser possível através de uma operação de igualdade (*i.e.*, *join*) sobre os valores desse atributo. Assim, são usados métodos que estabelecem equivalências aproximadas entre os valores. O procedimento de equivalência pode tornar-se mais robusto pela inclusão de vários atributos, o que atenua os PQD que possam existir ao nível dos atributos individuais.

As duas principais fases envolvidas na identificação de registos duplicados de um qualquer método aproximado de identificação de duplicados são: (*i*) pesquisa dos potenciais duplicados, o que envolve uma série de comparações registo a registo com base em critérios de igualdade ou semelhança entre os valores de determinados atributos (*e.g.*: nome, morada, número de contribuinte); e, (*ii*) decidir se um determinado par de registos corresponde ou não à mesma entidade do mundo real. A fase de pesquisa procura efectuar o menor número de comparações possíveis, mas procurando garantir a identificação de todos os duplicados existentes. Por exemplo, uma parte dos métodos de detecção existentes efectua comparações apenas entre registos vizinhos após estes terem sofrido uma ordenação. Mesmo assim, a nível do tempo de execução, a detecção de duplicados constitui uma operação computacional dispendiosa. A fase de decisão procura concluir se dois registos correspondem ou não à mesma entidade, tendo em conta as semelhanças e diferenças existentes entre os valores dos seus atributos.

O método tradicional de detectar registos duplicados exactos numa tabela passa por efectuar a sua ordenação e, de seguida, verificar se registos consecutivos são iguais [Monge, 2000]. Os registos duplicados garantidamente ficam próximos uns dos outros, independentemente do atributo usado na ordenação. Na literatura é possível encontrar diversas propostas de métodos de detecção mais sofisticados que suportam, não só duplicados exactos, mas também duplicados aproximados. Um outro aspecto comum a todos estes métodos prende-se com a existência de optimizações que visam reduzir o número Cartesiano de comparações necessário à detecção dos

duplicados. Nas subsecções seguintes apresentam-se os métodos de detecção que possuem maior divulgação.

3.4.1.1 Método do Produto Cartesiano

O método baseado no produto Cartesiano envolve comparar cada registo da tabela com todos os outros registos. Na detecção de todos os duplicados aproximados que possam existir numa tabela, este é o método que apresenta maior fiabilidade [Lee *et al.*, 1999]. O problema é que se trata de um método lento e ineficiente, pois obriga à realização de um número quadrático de comparações. Sendo n o número de registos de uma tabela, o número de comparações que resulta do produto Cartesiano é de $O(n^2)$. Numa tabela composta por um número muito elevado de registos (*e.g.*: na ordem dos milhões), muito dificilmente constitui uma opção viável. Note-se que além do elevado número de comparações, há que acrescer também ao custo computacional o custo de cada operação de comparação. Normalmente, a detecção de que um registo constitui um duplicado aproximado de outro, também é uma operação dispendiosa sob o ponto de vista computacional.

3.4.1.2 Método da Vizinhança Ordenada

O *método da vizinhança ordenada* [Hernández e Stolfo, 1995] [Hernández e Stolfo, 1998] visa a detecção de registos duplicados exactos ou aproximados. A ideia que se encontra subjacente a este método é a seguinte: ordenar os registos com base numa determinada chave de modo a colocar os potenciais duplicados próximos uns dos outros. De seguida, efectuar um conjunto de comparações apenas entre registos vizinhos. Assim, evita-se o número excessivo de comparações que resulta do produto Cartesiano entre os registos. A aplicação do método é precedida pela reunião dos registos provenientes das diversas fontes numa só tabela e envolve a execução sequencial dos seguintes passos:

- **Criar chave de ordenação** – Para cada registo existente na tabela é criada uma chave de ordenação. Em cada registo, esta chave é formada pela sequência de valores dos atributos seleccionados ou de partes desses valores. A escolha dos elementos constituintes da chave, bem como da ordem pela qual aparecem, é da responsabilidade do utilizador. Não há qualquer regra que defina como deve ser constituída. Esta é uma tarefa que envolve conhecimento específico sobre os dados em causa. Além de se conhecer as características dos dados é também necessário ter uma ideia dos problemas os afectam. A eficácia do método depende de uma escolha adequada da chave de ordenação. Se tal não for alcançado, da ordenação não resulta uma sequência de registos em que os duplicados aproximados se encontram próximos entre si. Como consequência, certos duplicados

acabam por não ser identificados como tal. A título exemplificativo, considere-se os registos apresentados na Tabela 3.1. Neste caso, a chave é constituída pela concatenação de partes dos diferentes atributos: as primeiras três letras do primeiro nome, são concatenadas com as primeiras três letras do segundo nome, seguidas pela concatenação dos quatro primeiros dígitos do código postal e terminando com a concatenação das três primeiras letras do apelido. A adopção de uma chave seguindo esta sequência pode ser justificada da seguinte forma: os primeiros dois nomes são usuais pelo que a probabilidade de ocorrência de erros nos primeiros caracteres é reduzida, daí serem considerados os principais elementos discriminativos da chave de ordenação.

Tabela 3.1 – Exemplo de chaves de ordenação

Nome	Apelido	Código Postal	Chave de Ordenação
David André	Loureiro	4415-206	DAVAND4415LOU
David André	Loreiro	4415 206	DAVAND4415LOR
Davide André	Loureiro	4415	DAVAND4415LOU
David José	Lourenço	4415-206	DAVJOS4415LOU

- **Ordenar os registos** – Os registos que fazem parte da tabela são ordenados em função da chave criada no passo anterior.
- **Identificar duplicados** – Envolve deslocar uma janela de tamanho fixo sobre o conjunto de registos ordenados e realizar comparações entre estes para identificar os duplicados. O tamanho ou dimensão da janela é definido pelo utilizador, tendo impacto directo nos duplicados identificados. As comparações necessárias à identificação dos duplicados são efectuadas apenas entre os registos que se encontram no interior desta janela. Supondo que n é o número total de registos da tabela e que t é o tamanho da janela de comparação, cada novo registo que entra na janela é comparado com os anteriores $t-1$ registos para verificar se há duplicados seus. Quanto maior o tamanho da janela, maior o número de registos duplicados identificados. A entrada de um novo registo na janela de comparações faz com que o último saia do seu interior, tal como se ilustra na Figura 3.1. O número de comparações efectuadas usando este método é apenas de $O(nt)$. A comparação entre os registos para detecção dos duplicados é efectuada com base num conjunto de axiomas (regras), enunciados pelo perito do domínio sob a forma de uma *teoria equacional* (em inglês: *equational theory*) que definem as situações em que há equivalência/correspondência entre os registos. A teoria equacional é expressa usando uma linguagem declarativa de alto nível. Na avaliação da equivalência são incluídos outros atributos, além dos usados na chave de ordenação. Os pares de registos identificados como duplicados são o resultado visível do método.

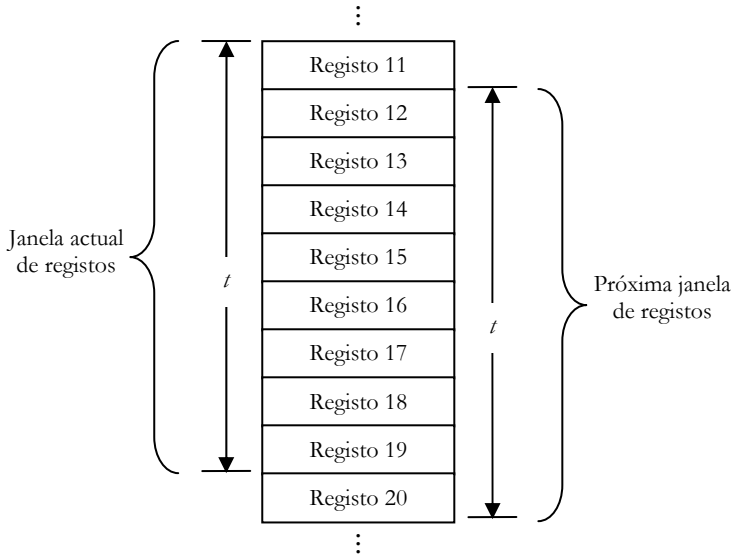


Figura 3.1 – Janela "deslizante"

O principal inconveniente deste método prende-se com a forte dependência da existência de proximidade entre os registos duplicados após a ordenação que, por sua vez, depende da chave utilizada. Uma vez que a chave de ordenação é extraída directamente dos dados e que estes podem apresentar outros PQD além da existência de duplicados, há a probabilidade de que as próprias chaves também espelhem esses problemas. Caso os registos duplicados estejam distantes após a ordenação é improvável que se situem na mesma janela de comparações, o que significa que não serão identificados como duplicados. Uma possível solução passa por aumentar o tamanho da janela. O problema é que isto implica um aumento da complexidade computacional face ao maior número de comparações a realizar, o que se traduz num acréscimo do tempo de execução.

3.4.1.3 Método da Vizinhança Ordenada com Multi-Passagem

O *método da vizinhança ordenada com multi-passagem* [Hernández e Stolfo, 1995] [Hernández e Stolfo, 1998] considera que uma só iteração/passagem do método da vizinhança ordenada não é suficiente para detectar todos registos duplicados. Por exemplo, suponha-se que um funcionário se encontra representado em dois registos da tabela, num com o número de segurança social 132451702 e no outro com o número 312451702 (há transposição dos dois primeiros números). Suponha-se também que o número de segurança social é usado como principal elemento da chave de ordenação. Após a realização da ordenação certamente que os dois registos ficarão distantes um do outro, pelo que não serão identificados como duplicados. Como forma de superar este problema, são efectuadas múltiplas passagens deste método (criar chave de

ordenação; ordenar os registos; e, identificar duplicados), usando chaves diferentes em cada uma das várias iterações (*e.g.*: na primeira passagem o atributo *morada* é usado como parte principal da chave, enquanto que na segunda passagem é utilizado o atributo *nome*) com uma janela de comparações de reduzida dimensão. Reconhece-se que uma só chave não garante que todos os registos duplicados fiquem próximos uns dos outros. O objectivo subjacente a este método é o de reduzir as hipóteses de não serem identificados todos os duplicados.

Cada iteração do método origina um conjunto de pares de registos duplicados. Sobre este conjunto é aplicada a transitividade matemática, *i.e.*, se o registo r_1 é um duplicado do registo r_2 e o registo r_2 é um duplicado do registo r_3, então por transitividade o registo r_1 também é um duplicado do registo r_3. Desta forma, os registos r_1 e r_3 são considerados duplicados, embora não se encontrem simultaneamente em nenhuma das janelas de comparação criadas durante as diferentes iterações. O resultado final resulta da reunião de todos os pares de registos duplicados identificados em cada iteração (sendo excluídos os que surgem mais do que uma vez), acrescentado dos pares que neste conjunto podem ser inferidos por transitividade.

Os testes realizados pelos autores evidenciam que a combinação dos resultados de múltiplas passagens com uma janela de reduzida dimensão, aumenta a taxa de recuperação de duplicados comparativamente a uma única passagem com uma janela de tamanho superior. Uma janela de reduzida dimensão também contribui para a diminuição do número de registos incorrectamente classificados como duplicados, *i.e.*, para a diminuição da taxa de falsos positivos. O incremento no tempo de execução registado nos testes realizados não é significativo. Para se alcançar resultados equivalentes com o método original seria necessário um tempo de execução muito superior.

Apesar de tudo, este método apresenta o mesmo problema que o método original, *i.e.*, a elaboração das chaves de ordenação. Encontrar as chaves adequadas que coloquem os registos duplicados próximos uns dos outros pode revelar-se uma tarefa árdua. Caso os valores dos atributos não obedeçam a um padrão, as diferenças de representação fazem com que registos duplicados sejam colocados distantes entre si, mesmo recorrendo a chaves de ordenação diferentes. O outro problema está relacionado com a transitividade. A sua aplicação gera resultados correctos se os pares de registos correspondem a verdadeiros duplicados, *i.e.*, se representam a mesma entidade do mundo real. No entanto, como se está a manipular duplicados aproximados, há a possibilidade dos registos serem erradamente classificados como duplicados (*i.e.*, falsos positivos). A aplicação da transitividade nestes registos conduzirá à propagação do erro e à classificação de outros registos como duplicados quando não o são.

Este método encontra-se formalizado nesta dissertação no âmbito das operações de detecção que conduzem à identificação de tuplos duplicados (Capítulo 5 – Secção 5.5.3).

3.4.1.4 Algoritmo Fila de Prioridades

A janela de comparações usada no método da vizinhança ordenada possui uma dimensão fixa. Se uma determinada entidade do mundo real está representada em inúmeros registos de uma tabela, pode acontecer que alguns destes duplicados não sejam detectados em virtude do número insuficiente de comparações. Por outro lado, se há poucos duplicados das entidades representadas na tabela ou mesmo nenhuns, então estão a ser efectuadas comparações que não são necessárias.

O *algoritmo fila de prioridades* (em inglês: *priority queue algorithm*) [Monge, 2000] [Monge e Elkan, 1997] permite uma adaptação à dimensão e homogeneidade dos potenciais duplicados identificados à medida que a tabela vai sendo percorrida, aumentando ou diminuindo a janela de comparações. Para tal, a janela de comparações de tamanho fixo é substituída por uma *fila de prioridades*. O algoritmo percorre sequencialmente a tabela ordenada, recorrendo à fila de prioridades composta por conjuntos de registos pertencentes aos últimos segmentos (ou *clusters*) de duplicados identificados. Cada membro de um segmento é um duplicado aproximado de qualquer outro membro. A fila de prioridades não pode exceder um determinado número máximo de conjuntos de registos. Cada conjunto contém um ou mais registos de um segmento. O conjunto que representa o segmento em que foi acrescentado o membro mais recente, possui a maior prioridade da lista e assim sucessivamente.

Para cada registo *r* da tabela, o algoritmo começa por verificar se *r* já é membro de algum dos segmentos representados na fila de prioridades. A verificação é feita comparando o segmento representativo a que pertence *r* com os representativos dos segmentos existentes na fila de prioridades. Caso alguma das comparações tenha sucesso isso significa que *r* já é membro de um segmento, representado por um conjunto de registos na fila de prioridade. Este conjunto passa a ser o que detém maior prioridade. Caso *r* não seja membro de um segmento existente na fila de prioridades é efectuada uma pesquisa que compara *r* com os registos existentes na fila de prioridades. Esta pesquisa itera por cada conjunto de registos da lista de prioridades, começando por aquele com maior prioridade. Em cada conjunto, o algoritmo itera pelos seus diversos elementos. Caso uma semelhança seja detectada, o segmento a que pertence *r* é combinado com o segmento correspondente ao conjunto de registos onde se detectou a semelhança, recorrendo a uma operação de reunião. O registo *r* também poderá ser adicionado ao conjunto de registos que

representa o segmento na fila de prioridades. Caso *r* seja muito similar ao registo com o qual foi detectada semelhança não é necessária a sua inclusão nesse conjunto. Caso contrário, *i.e.*, se for apenas moderadamente similar, então a inclusão de *r* auxiliará na detecção futura de outros membros do segmento.

Caso a comparação entre *r* e um dos elementos de um conjunto de registos da lista de prioridades representativos de um segmento resulte num valor de semelhança muito baixo, a pesquisa avança para o próximo conjunto de registos existente na lista de prioridades. A filosofia é a seguinte: caso não haja semelhança na comparação efectuada, então subsequentes comparações com os outros membros do mesmo segmento também não detectarão a existência de semelhança.

Por último, caso *r* seja comparado com todos os elementos de todos os conjuntos de registos existentes na fila de prioridades e não seja detectada semelhança, então *r* deve ser membro de um novo segmento, actualmente não representado na fila de prioridades. Neste caso, é criado um novo conjunto na lista de prioridades representativo do novo segmento em que *r* é o único elemento. A prioridade deste novo conjunto é a mais elevada. Caso o limite máximo do número de conjuntos da lista de prioridades seja excedido, o conjunto com a menor prioridade é removido da fila de prioridades.

A definição do número máximo de segmentos representados na fila de prioridade, bem como o desenvolvimento de heurísticas que permitam seleccionar o representativo de cada segmento constituem responsabilidades do utilizador. Ambos os aspectos influenciam fortemente a identificação dos registos duplicados.

3.4.1.5 Record Linkage

Em [Fellegi e Sunter, 1969] é apresentado um modelo formal probabilístico para o conceito de *record linkage*, anteriormente introduzido em [Newcombe *et al.*, 1959] [Newcombe e Kennedy, 1962]. Nestes trabalhos definiu-se o *record linkage* como o problema probabilístico de decidir que pares resultantes do produto Cartesiano devem ser classificados como duplicados, apesar dos PQD que possam existir nos seus atributos. Fellegi e Sunter formalizaram esta ideia através da definição de regras que dividem as comparações efectuadas em três conjuntos de pares de registos: duplicados; não duplicados; e, possivelmente duplicados.

O trabalho de Fellegi e Sunter permitiu estimar parâmetros chave. Suponha-se que se pretende identificar os duplicados que existem numa tabela ou entre duas tabelas. Considere-se que as tabelas se encontram simbolicamente representadas por $\mathcal{A}$ e $\mathcal{B}$. No caso de se tratar de uma só

tabela, $\mathcal{A}$ e $\mathcal{B}$ representam a mesma tabela. O objectivo é classificar os pares resultantes do produto Cartesiano $\mathcal{A} \times \mathcal{B}$ e colocá-los em $\mathcal{D}$ que representa simbolicamente o conjunto de verdadeiros duplicados ou em $\mathcal{N}$ que simboliza o conjunto dos verdadeiros não duplicados. Dando rigor aos conceitos introduzidos em [Newcombe *et al.*, 1959], foram introduzidos rácios de probabilidade do tipo:

$$\mathcal{R} = \frac{\mathcal{P}(\gamma \in \Gamma \mid \mathcal{D})}{\mathcal{P}(\gamma \in \Gamma \mid \mathcal{N})}$$

Na expressão, γ representa um padrão arbitrário de concordância, no espaço de comparação Γ. Por exemplo, Γ pode consistir de padrões representando concordâncias simples no maior componente do nome e no maior componente da morada. Cada $\gamma \in \Gamma$ pode conter a frequência relativa com que determinados componentes do nome (*e.g.*: José, Silva, ZZZ) ocorrem. O rácio $\mathcal{R}$ representa o grau de equivalência ou correspondência entre os pares. O resultado da comparação é dado pelas seguintes regras:

- Se $\mathcal{R} > \mathcal{L}_\mu$, então classificar os pares como duplicados
- Se $\mathcal{L}_\lambda \leq \mathcal{R} \leq \mathcal{L}_\mu$, então classificar os pares como possíveis duplicados, ficando a aguardar a análise do utilizador
- Se $\mathcal{R} < \mathcal{L}_\lambda$, então classificar os pares como não duplicados

Os limiares $\mathcal{L}_\lambda$ e $\mathcal{L}_\mu$ são determinados pelas fronteiras de erro, conhecidas *a priori* na identificação de falsos duplicados e falsos não duplicados. Frequentemente, os limiares são designados de graus de admissão superior e inferior [Winkler, 2004]. Se $\gamma \in \Gamma$ consistir essencialmente de concordâncias, então as possibilidades de ocorrer entre os duplicados são bem maiores do que ocorrer entre os não duplicados. Isto faz com que o valor do rácio $\mathcal{R}$ seja elevado. Por outro lado, se $\gamma \in \Gamma$ é composto essencialmente de discordâncias, então o valor do rácio $\mathcal{R}$ é reduzido. As regras acima apresentadas separam o conjunto $\gamma \in \Gamma$ em três áreas disjuntas. A área $\mathcal{L}_\lambda \leq \mathcal{R} \leq \mathcal{L}_\mu$ corresponde a uma área de indecisão que obriga a uma análise do utilizador.

Os pares com um valor de $\mathcal{R}$ superior ao limiar superior são considerados duplicados. Os pares com um valor de $\mathcal{R}$ inferior ao limiar inferior são considerados não duplicados. Os restantes pares são considerados como potenciais duplicados. Se $\mathcal{L}_\lambda = \mathcal{L}_\mu$, então as regras decisão podem ser usadas para separar os registos que estão num conjunto daqueles que não estão.

As probabilidades $\mathcal{P}$(concordância nome$\mid\mathcal{D}$), $\mathcal{P}$(concordância morada$\mid\mathcal{D}$), $\mathcal{P}$(concordância nome$\mid\mathcal{N}$) e $\mathcal{P}$(concordância morada$\mid\mathcal{N}$), são designadas de *probabilidades marginais*. Em função da concordância entre atributos comuns esta teoria baseia-se nestas probabilidades marginais para

determinar se há correspondência ou não entre dois registos. $P(\ldots|\mathcal{D})$ e $P(\ldots|\mathcal{N})$ são designadas, respectivamente, de probabilidades de $\mathcal{D}$ e $\mathcal{N}$. Os logaritmos dos rácios das probabilidades associados aos atributos individuais (*i.e.*, probabilidades marginais) são designados de *pesos de concordância individual*. Estes pesos representam a contribuição de cada atributo comum na classificação dos pares de registos. As probabilidade de $\mathcal{D}$ e $\mathcal{N}$ são também designadas de *parâmetros de correspondência* [Winkler, 2003].

Um dos inconvenientes deste método consiste na necessidade de conhecimento *a priori* sobre os valores das probabilidades marginais. Em [Jaro, 1989] são apresentados dois algoritmos que permitem estimar os seus valores de forma automática. O método também parte do pressuposto que os registos podem ser armazenados em memória primária e que a realização do produto Cartesiano entre estes é exequível [Gravano *et al.*, 2003]. Em BD de elevada dimensão este pressuposto pode não ser realista.

3.4.2 Âmbito dos Métodos de Detecção de Duplicados

Quanto ao âmbito, os métodos de detecção de duplicados podem ser classificados como sendo:

- **Dependentes do domínio** – Estes métodos obrigam à existência de conhecimento específico do domínio para que a detecção de duplicados possa ser levada a cabo. Uma abordagem muito usada na detecção de duplicados consiste em especificar um conjunto de regras que definem as situações perante as quais dois registos são considerados duplicados (em [Hernández e Stolfo, 1998] este conjunto de regras é designado de *teoria equacional*). Estas regras podem envolver a invocação a funções de semelhança definidas pelo utilizador e o uso de limiares de semelhança que permitem distinguir os duplicados, dos não duplicados. Noutro tipo de abordagem, a detecção é efectuada exclusivamente com base em funções definidas pelo utilizador [Hellerstein *et al.*, 1999]. Estas funções determinam o grau de semelhança entre atributos individuais, procedem à sua combinação para formar um valor de semelhança global ao nível do registo e recorrem a limiares de semelhança definidos pelo utilizador para concluir se dois registos são ou não duplicados. Em ambas as abordagens, especificar as situações em que dois registos devem ser considerados ou não duplicados pode revelar-se uma tarefa altamente complexa.
- **Independentes do domínio** – Estes métodos não requerem conhecimento específico do domínio para efectuar a detecção de duplicados. Em [Monge e Elkan, 1997], considera-se que os registos são compostos exclusivamente por sequências de caracteres alfanuméricos (*i.e.*, *strings*). A detecção de registos duplicados é efectuada com base em métricas de análise de semelhança entre *strings*. Ainda que atractivos, face à sua

independência em relação ao domínio, a precisão destes métodos não atinge valores convincentes na identificação dos duplicados.

- **Mistos** – Uma forma de contornar a especificação das regras e o desenvolvimento das funções que permitem concluir se dois registos são ou não duplicados, consiste em recorrer a algoritmos de aprendizagem automática [Cohen e Richman, 2001]. O objectivo é treinar um classificador que permita distinguir entre duplicados e não duplicados. O algoritmo recebe um *conjunto de treino*[10] composto por pares de duplicados e de não duplicados. Um conjunto com diversos tipos de funções de semelhança, específicas do domínio em questão, é também fornecido por um perito do domínio. O algoritmo de aprendizagem socorre-se dos exemplos para automaticamente encontrar a melhor forma de combinar e definir os limiares de semelhança dos diversos atributos. O sucesso desta abordagem depende muito do fornecimento de um conjunto de treino abrangente, composto por pares de registos que sejam reveladores das subtilezas das duplicações e não duplicações existentes. Este processo não é nada trivial, uma vez que implica que o utilizador pesquise manualmente em tabelas de grandes dimensões os pares de registos que interessa incluir no conjunto de treino. Surpreendentemente, o que se reveste ainda de maior dificuldade é encontrar um conjunto de registos não duplicados que possam ser confundidos com registos duplicados [Sarawagi e Bhamidipaty, 2000]. A recolha manual de um número elevado de registos duplicados não permitirá alcançar uma elevada precisão, a não ser que um especial cuidado seja colocado na selecção de um conjunto representativo de não duplicados. A precisão do classificador depende do conjunto de treino que lhe deu origem [Missier *et al.*, 2003]. Ainda que o classificador seja criado automaticamente com base na amostra, não estando dependente do fornecimento directo de conhecimento específico do domínio por parte do utilizador, a selecção da amostra apropriada é uma competência do perito e, como tal, específica de cada domínio. Por este motivo se considera que este tipo de métodos é misto.

3.4.3 Métricas de Avaliação da Semelhança

A avaliação da semelhança entre dois valores de um atributo é um problema importante na integração de dados e na LD [Gravano *et al.*, 2001]. A semelhança entre valores do tipo numérico é determinada quase sempre com base no módulo da sua diferença. Quanto menor for este resultado, maior a semelhança entre os valores dos atributos. No caso de valores cujo tipo de dados é *string*, o cenário não recolhe igual unanimidade. A generalidade das métricas usadas para

[10] Um *conjunto de treino* é uma amostra de tamanho limitado, extraída a partir dos dados existentes.

determinar a semelhança insere-se num dos dois seguintes grupos: *métricas baseadas em caracter* e *métricas baseadas no espaço vectorial.*

As *métricas baseadas em caracter* baseiam-se em operações de edição ao nível do caracter. Neste grupo, a *distância de edição*, também conhecida por *distância de Levenshtein*, é a métrica usualmente adoptada para avaliação da semelhança entre valores do tipo *string*. A *distância de edição ou Levenshtein* [Levenshtein, 1966] entre duas *strings* corresponde ao menor número de operações de edição simples (*i.e.*, inserções, eliminações e substituições) que é necessário efectuar ao nível dos caracteres individuais para transformar uma *string* na outra (*e.g.*: o valor da distância de edição entre as *strings* "limpeza de dados" e "lipeza dos dados" é igual a um). Esta métrica é uma generalização da *distância de Hamming* [Hamming, 1950], uma vez que esta obriga a que as *strings* possuam igual número de caracteres, sendo permitidas apenas operações de substituição. Entre duas strings, a distância de *Hamming* representa o número total de posições em que os símbolos que as ocupam são diferentes. Uma variante da distância de edição também usada na análise da semelhança entre *strings* é a *distância de Damerau-Levenshtein* [Damerau, 1964] [Levenshtein, 1966]. Nesta métrica, à semelhança da distância de edição, a diferença entre duas *strings* é dada pelo menor número de operações necessárias para transformar uma das *strings* na outra. Uma operação corresponde a uma inserção, eliminação ou substituição de um só caracter. No entanto, contrariamente ao que acontece na distância de edição, a transposição de dois caracteres corresponde a uma só operação de edição em vez de duas. Outras métricas também usadas, ainda que menos representativas, são a distância de *Jaro* [Jaro, 1989] e uma evolução desta última designada de distância de *Jaro-Winkler* [Winkler, 1999].

O segundo grupo de métricas, originário da comunidade científica da área da recuperação de informação, transforma as *strings* em representações vectoriais sobre as quais são executadas análises de semelhança. A métrica de maior representatividade neste grupo é a da *semelhança baseada no co-seno dos vectores* [Baeza-Yates e Ribeiro-Neto, 1999]. As *strings* são representadas sob a forma de vectores unitários num espaço bidimensional. O valor do co-seno do ângulo formado pelos dois vectores representa o grau de semelhança existente entre as *strings*. Caso os vectores se sobreponham, o ângulo formado por estes é zero. Daqui resulta um valor do co-seno igual a um, o que significa que as *strings* são iguais. No caso do ângulo ser 90º, o valor do co-seno é igual a zero. Isto significa que não existe qualquer semelhança entre as *strings*. Estas duas situações representam os extremos. Todas as outras situações intermédias correspondem a graus variáveis de semelhança entre as duas *strings*. Um outro método de comparação de *strings* também inspirado em representações vectoriais é baseado em *n-grams* [Hylton, 1996]. Um *n-gram* é uma

representação vectorial que inclui todas as combinações de ***n*** caracteres de uma *string*. A representação *n-gram* de uma *string* corresponde a um vector não nulo para cada *substring* de ***n*** caracteres que dela faz parte, armazenando o número de vezes que cada *substring* ocorre. O algoritmo de comparação de *strings* forma vectores *n-gram* para ambas as *strings*, subtraindo cada vector ao outro. A diferença entre os dois vectores é então comparada com um determinado limiar para determinar se há ou não equivalência entre as *strings*.

As métricas baseadas em caracter funcionam bem a estimar a distância entre *strings* que difiram devido a erros ortográficos ou a abreviaturas. No entanto, perdem precisão e o seu custo computacional aumenta consideravelmente perante *strings* de maior dimensão [Bilenko e Mooney, 2003]. Por outro lado, não se comportam bem perante a transposição de *tokens* no interior da *string* (*e.g.*: "Hipermercados Continente" e "Continente Hipermercados") [Gravano *et al.*, 2003]. As métricas baseadas no espaço vectorial menosprezam a ordem pela qual os *tokens* surgem, considerando as *strings* como um conjunto de *tokens*. Como tal, fornecem bons resultados mesmo perante a movimentação de *tokens* ou a inserção de novos *tokens*. Estas métricas demonstram ser muito eficientes, produzindo bons resultados mesmo quando manipulam *strings* de grandes dimensões [Bilenko e Mooney, 2003].

As métricas podem ser usadas, não só para determinar a semelhança entre *strings*, mas também para avaliar a semelhança global entre dois registos. No caso do cálculo da semelhança entre registos é necessário combinar as semelhanças entre os valores de cada atributo. A contribuição da semelhança entre os valores de cada atributo para a semelhança global entre os registos varia em função do grau informativo. Como tal, é necessário atribuir pesos aos atributos de acordo com a sua importância na determinação da semelhança entre os registos. Em função destes pesos e dos valores das semelhanças individuais de cada atributo é calculado o valor global da semelhança entre os registos. A generalidade dos algoritmos considera dois registos duplicados se a semelhança entre estes se encontra acima de um determinado limiar [Bilenko e Mooney, 2003].

3.4.4 Avaliação da Eficácia da Detecção de Duplicados

Alguns registos duplicados podem não ser identificados enquanto outros registos que não representam a mesma entidade podem ser erradamente classificados como duplicados. Estes pares de registos incorrectamente classificados são denominados de *falsos positivos* [Lee *et al.*, 1999]. As métricas abaixo apresentadas são vulgarmente usadas (*e.g.*: em [Hernández e Stolfo, 1995];

[Lee *et al.*, 2000a]; e, [Scannapieco, 2004]) para estimar a eficácia das estratégias de detecção de duplicados. Naturalmente, apenas se pode avaliar a eficácia se existir um *conjunto de teste*[11] sobre o qual as estratégias de detecção de duplicados possam ser aplicadas. A eficácia de uma estratégia de detecção de duplicados tem implicação directa no grau de melhoria da QD.

- **Precisão** – Representa a relação entre o número de duplicados correctamente detectados e o número total de duplicados existentes nos dados, de acordo com a seguinte fórmula:

$$\text{Precisão} = \frac{\text{Número de duplicados correctamente detectados}}{\text{Número total de duplicados existentes}}$$

- **Falsos positivos** – Representa a relação entre o número de duplicados incorrectamente detectados e o número total de duplicados detectados, de acordo com a seguinte fórmula:

$$\text{Falsos Positivos} = \frac{\text{Número de duplicados incorrectamente detectados}}{\text{Número total de duplicados detectados}}$$

3.4.5 Abordagens Utilizadas na Eliminação de Duplicados

A correcção deste PQD (*i.e.*, existência de duplicados) envolve remover os registos duplicados de modo a que cada entidade do mundo real fique representada num só registo. Na remoção automática dos duplicados podem ser seguidas duas vias distintas. Um dos registos é considerado correcto (*e.g.*: o mais recente) e todos os seus duplicados considerados errados. Neste caso, o objectivo é eliminar da tabela todos os registos duplicados de um determinado registo. Esta via é proposta, por exemplo, em [Hernández e Stolfo, 1998]. Uma alternativa mais razoável passa por considerar que cada registo constitui uma fonte parcial de dados. Frequentemente, os dados são apenas parcialmente redundantes e os diferentes registos podem complementar-se, fornecendo informação adicional acerca de uma entidade. Por vezes, os valores que se encontram nos atributos dos registos duplicados são contraditórios. Neste caso, o objectivo é proceder à consolidação dos registos duplicados, efectuando a reconciliação dos valores conflituosos (*e.g.*: entre os vários valores conflituosos de um atributo apenas "sobrevive" o mais completo) e originando um registo mais completo e preciso que substitui os existentes. Esta via é seguida, por exemplo, em [Hylton, 1996] e [Cochinwala *et al.*, 2001]. Naturalmente, a remoção de duplicados também pode ser feita manualmente pelo utilizador, seguindo qualquer uma das estratégias anteriores.

[11] Um *conjunto de teste* é uma amostra de tamanho limitado na qual se conhece *a priori* os resultados a obter.

3.5 Ferramentas de Limpeza de Dados

Os actuais sistemas de BD relacionais (*e.g.*: *Oracle*; *SQL Server*; *Access*) suportam a realização de algumas operações simples de LD. Essencialmente, estas operações traduzem-se em verificações à integridade dos dados. A integridade dos dados pode ser analisada ao nível: da tabela (*i.e.*, inexistência de duplicações da mesma entidade); referencial (*i.e.*, inexistência de violações à integridade referencial dos dados); e, do atributo (*e.g.*: inexistência de valores duplicados numa determinada coluna da tabela; inexistência de valores que violem o domínio do atributo). As verificações são efectuadas através de inquéritos à BD. Claramente, este tipo de suporte não se mostra adequado para manipular a generalidade dos PQD. Assim, surgiram ferramentas que visam especificamente a LD. Estas ferramentas têm origem em duas comunidades distintas: académica/científica e empresarial/comercial. Nas duas subsecções seguintes (*i.e.*, 3.5.1 e 3.5.2) são apresentadas as ferramentas consideradas mais representativas, oriundas de cada uma das comunidades.

3.5.1 Protótipos de Investigação

Os trabalhos de investigação efectuados pela comunidade científica da área resultaram no desenvolvimento de alguns protótipos de LD. Em [Barateiro e Galhardas, 2005] é apresentada uma lista muito completa. Alguns destes protótipos estão direccionados para a manipulação de um único PQD. Face à sua especificidade, em [Oliveira *et al.*, 2004] estes são classificados como sendo de *âmbito especializado*. Exemplos deste género de protótipos são os que visam unicamente a detecção e eliminação de duplicados. O *Projecto Flamingo* [Jin *et al.*, 2005] [Jin et al., 2004] é um exemplo de um protótipo vocacionado apenas para o problema dos duplicados. Outros protótipos de investigação procuram manipular simultaneamente vários PQD. Face à sua abrangência, estes são classificados como sendo de *âmbito genérico* [Oliveira *et al.*, 2004]. No contexto deste trabalho de doutoramento pretende-se desenvolver um modelo de LD que suporte, globalmente, os vários PQD. Como tal, nas secções seguintes apresentam-se apenas os protótipos de investigação de âmbito genérico, ilustrativos do actual *estado da arte*.

3.5.1.1 Ajax

Na base do *Ajax* [Galhardas *et al.,* 2000a] [Galhardas *et al.,* 2001] encontra-se uma arquitectura flexível e extensível que separa os níveis lógico e físico de um processo de transformação de dados. O nível lógico suporta a especificação das operações, enquanto que o nível físico é responsável pela sua implementação. O seu principal objectivo consiste em transformar dados de

uma ou várias fontes (integração de dados) para um determinado esquema alvo. Os eventuais PQD são solucionados durante o processo de transformação (*e.g.*: eliminação de registos duplicados). O processo de transformação recebe um conjunto de fluxos de dados, possivelmente errados ou inconsistentes e origina um conjunto de fluxos de dados formatados, correctos e consistentes.

A lógica de um processo de transformação ou LD é representada sob a forma de um grafo dirigido de transformações sobre fluxos de dados. Cada transformação recebe fluxos de dados de uma ou mais transformações precedentes e origina um fluxo de dados que "alimenta" uma ou mais transformações subsequentes.

Uma linguagem declarativa (inspirada em *Structured Query Language* (SQL)) permite especificar as transformações de dados de forma compacta e simplifica a sua manutenção. A linguagem é composta pelo seguinte conjunto de operadores lógicos:

- ***SQL view*** – Equivale a um inquérito SQL típico, com potencialidades adicionais de verificação de restrições de integridade. Também suporta a realização de *joins* e *unions*.
- ***Map*** – Ao nível do registo permite expressar mapeamentos entre um fluxo de dados de entrada e um ou mais fluxos de dados de saída.
- ***Match*** – Efectua um *join* aproximado entre dois fluxos de dados de entrada, atribui um valor de semelhança a cada par de registos resultante do produto Cartesiano e origina um só fluxo de dados de saída. Os registos são comparados usando funções de semelhança nos valores de um ou mais atributos. Esta operação é essencial para a detecção de registos duplicados.
- ***Cluster*** – Agrupa os registos com elevados valores de semelhança em *clusters*, tendo por base um determinado critério (*e.g.*: transitividade).
- ***Merge*** – Subdivide o fluxo de entrada de dados em função de um conjunto de atributos de agrupamento. De seguida, cada subdivisão é agregada num único registo, recorrendo a uma função de agregação definida pelo utilizador.

A linguagem de especificação não é totalmente declarativa, uma vez que permite a invocação de código procedimental (*e.g.*: funções definidas pelo utilizador). Aliás, é precisamente esta característica que confere à linguagem potencialidades de extensibilidade (*e.g.*: invocação de funções específicas do domínio).

A semântica de cada transformação envolve a geração de excepções aquando da ocorrência de situações anormais (erros ou inconsistências). As excepções constituem a base de um ambiente

interactivo com o utilizador na manipulação dessas situações. Os registos causadores de uma situação excepcional durante uma transformação são marcados para posterior análise por parte do utilizador. A originalidade advém das excepções serem toleradas e não obrigarem à paragem do processo de transformações. O *Ajax* também permite a análise dos resultados intermédios produzidos ao longo do grafo de transformações.

O último aspecto relevante consiste na existência de um mecanismo *genealógico dos dados* (em inglês: *data lineage*) que permite a obtenção de explicações. Em cada transformação, o utilizador pode aceder aos registos que estiveram na base da geração de um determinado registo. Este mecanismo permite analisar o processo de transformações, auxiliando o utilizador a compreender os resultados alcançados. Isto possibilita identificar a origem de transformações incorrectas e a refinar o processo de transformações.

3.5.1.2 Arktos II

O *Arktos II* [Vassiliadis *et al.,* 2003] é uma ferramenta gráfica baseada numa arquitectura para a modelação e execução de processos de Extracção, Transformação e Carregamento (ETC) destinados à criação de armazéns de dados. Um processo de ETC consiste numa sequência de passos que extraem dados relevantes das fontes, efectuam a sua transformação para o formato pretendido, procedem à sua limpeza e, por último, executam o seu carregamento para o armazém de dados. A arquitectura define um modelo conceptual para: (*i*) o fluxo de dados que se estabelece desde as fontes até ao armazém de dados; e, (*ii*) a composição das actividades sob a forma de grafo e respectivas semânticas. No modelo, as actividades são abstracções lógicas das operações físicas executadas no processo de ETC. Uma actividade constitui uma unidade atómica de trabalho, representando um passo na sequência de operações do processo de ETC. Uma vez que a finalidade de uma actividade é efectuar processamento sob um fluxo de dados, cada uma destas encontra-se conectada a tabelas de entrada e de saída de uma ou mais BD.

O modelo conceptual define os seguintes níveis de instanciação/especialização que suportam a representação das actividades de ETC:

- **Meta-modelo** – Este é o nível mais elevado de instanciação, constituindo a forma mais genérica de representação das actividades. A este nível, cada actividade é descrita por diversos elementos. Os de maior importância na caracterização da actividade são:
  - **Nome** – Termo usado para identificar cada actividade de forma única.
  - **Tabela(s) de entrada** – Uma ou mais tabelas de entrada pertencentes a uma ou mais BD.

- **Tabela de saída** – Tabela de saída na qual são colocados os resultados produzidos pela actividade.
- **Semântica operacional** – Descrição declarativa da lógica subjacente à actividade. O formalismo de representação usado é uma linguagem declarativa de programação em lógica denominada de LDL [Tsur e Zaniolo, 1986].

- ***Template*** – Este nível especializa o anterior, fornecendo um conjunto de actividades que suportam as tarefas mais usuais de ETC. As potencialidades de extensibilidade permitem que actividades especificadas pelo utilizador possam ser adicionadas às existentes de base. As actividades encontram-se distribuídas pelos seguintes grupos:
  - **Filtros** – Verificam se determinada condição é respeitada ou não pelos registos. Os filtros que se encontram disponíveis são: violação de chave primária; violação de unicidade; violação de integridade referencial; existência de nulo; violação de domínio; e, sintaxe inválida.
  - **Transformações unárias** – Deste grupo faz parte a actividade *push* que simplesmente transfere os dados para a tabela de saída e as usuais actividades de *agregação* e *aplicação de função*. Encontram-se também disponíveis as seguintes actividades, responsáveis por transformações específicas para a criação de armazéns de dados: *atribuição de chave de substituição*; *normalização*; e, *desnormalização*.
  - **Transformações binárias** – Neste grupo encontram-se as actividades de *join*, *union* e *difference*.
- **Instanciação** – O modelo conceptual suporta a instanciação e reutilização das actividades *template*. A instanciação destas actividades ocorre neste nível, originando actividades adaptadas a cada processo específico de ETC. Estas actividades são aplicadas sobre os dados em questão.

Novos desenvolvimentos deste trabalho encontram-se em [Simitsis *et al.,* 2005], no qual é apresentada uma proposta teórica para a optimização lógica do processo de ETC.

3.5.1.3 IntelliClean

Na base do *IntelliClean* [Lee *et al.,* 2000a] [Low *et al.,* 2001] encontra-se uma arquitectura baseada em conhecimento para a LD, com especial ênfase na eliminação de duplicados. As operações são especificadas sob a forma de regras, sendo a sua aplicação efectuada pelo motor de inferência de um sistema pericial. A arquitectura decompõe o processo de LD nas seguintes fases distintas:

- **Pré-processamento** – Nesta fase são corrigidos os problemas sintácticos existentes nos dados. Entre estes encontram-se as uniformizações de sintaxes e a adopção de representações consistentes para abreviaturas e acrónimos. Na realização das correcções

são usadas funções de conversão e tabelas de pesquisa. O resultado desta fase é um conjunto consistente de registos que será usado na fase seguinte.

- **Processamento** – Envolve a avaliação das regras de LD sobre os registos pré-processados. Estes registos "alimentam" o mecanismo de inferência do sistema pericial (os registos são os factos). As regras especificam as acções a realizar perante a ocorrência de determinadas situações. Estas podem conter predicados complexos e referências a funções externas, tanto no antecedente como no seu consequente. Sempre que os registos respeitam as condições definidas nas regras, estas são activadas ("disparadas"). As acções especificadas no consequente das regras são então executadas. Os tipos de operações de LD suportados encontram-se organizados nas seguintes categorias de regras:
  - **Identificação de duplicados** – Especificam as condições a que é necessário obedecer para que dois registos possam ser considerados duplicados. As regras podem invocar funções que: comparam a semelhança textual; efectuam manipulação de texto; ou, determinam a equivalência entre registos.
  - **Fusão** – Definem a forma de manipulação dos registos duplicados. Por exemplo, uma regra de fusão pode especificar que apenas prevalece o registo com menor número de atributos vazios, sendo os restantes eliminados. Caso não tenham sido definidas regras de fusão, os registos duplicados podem ser manipulados manualmente pelo utilizador durante a próxima fase.
  - **Actualização** – Especificam a forma como os dados são actualizados perante a ocorrência de uma determinada situação. Este tipo de regras pode ser usado para definir a forma de preenchimento dos valores em falta nos atributos.
  - **Alerta** – Definem as situações em que o utilizador é notificado em virtude da ocorrência de determinado evento. Este tipo de regras pode ser usado para avisar o utilizador da ocorrência de violações de restrições de integridade.
- **Verificação e validação humana** – As acções realizadas durante as duas fases anteriores são registadas num ficheiro, documentando as operações efectuadas. Nesta fase, é necessária intervenção humana para o analisar. A validação da base de regras é efectuada com base neste ficheiro. O ficheiro permite verificar a consistência e precisão das acções efectuadas, o que pode resultar na introdução de correcções.

O *método da vizinhança ordenada com multi-passagem* [Hernández e Stolfo, 1998] é adoptado para a detecção de registos duplicados. O objectivo é evitar uma comparação exaustiva entre cada par de registos resultante do produto Cartesiano entre estes. Em cada instante de tempo, as regras incidem apenas sobre o conjunto de registos que se encontra residente na memória de trabalho do sistema pericial.

A identificação de registos duplicados é efectuada com base num novo método, baseado na transitividade perante a existência de incerteza. Este método permite reduzir o número de registos incorrectamente classificados como duplicados (*i.e.*, falsos positivos). A precisão das operações de identificação de duplicados é melhorada através da introdução do conceito de factor de certeza das regras (um número real compreendido entre zero e um). Durante a determinação da transitividade, o factor de certeza resultante do grupo de registos em análise é comparado com um limiar definido pelo utilizador. Se o factor de certeza é inferior ao limiar, então os registos não são considerados como duplicados.

Uma arquitectura baseada em conhecimento (representado sob a forma de regras) facilita a existência de um mecanismo de explicações. Cada acção pode ser explicada através da análise da sua sequência de execução. A sequência elucida sobre o raciocínio seguido e as alterações efectuadas. As explicações são especialmente úteis para refinar a base de regras, efectuada no seguimento da fase de verificação e validação humana.

3.5.1.4 Potter's Wheel

O *Potter's Wheel* [Raman e Hellerstein, 2001] baseia-se numa arquitectura interactiva simples para a transformação e LD. A arquitectura integra simultaneamente transformação e detecção dos PQD. Os utilizadores procedem gradualmente à especificação e análise do efeito das transformações num interface gráfico do tipo *folha de cálculo*. A principal originalidade consiste na alternância entre DC dos problemas.

Com base numa amostra de dados são inferidas automaticamente as sintaxes subjacentes aos diversos atributos. As sintaxes inferidas são, então, usadas na detecção de problemas sintácticos. A detecção é efectuada em segundo plano na última versão transformada dos dados. O utilizador desenvolve e refina transformações para os problemas à medida que estes lhe são assinalados. A especificação das transformações é efectuada interactivamente sob a forma de transformações simples aplicadas sobre uma amostra de dados. O utilizador pode observar imediatamente o efeito de uma transformação nos registos visíveis no ecrã. Caso o seu efeito não seja o pretendido, pode ser anulada facilmente. Não é necessário aguardar pela transformação de todos os registos da tabela para se analisar as consequências.

A arquitectura deste protótipo inclui um mecanismo genérico e extensível que permite a incorporação de algoritmos dependentes do domínio, para detecção de problemas sintácticos e semânticos. Alguns algoritmos simples de carácter geral encontram-se já implementados, permitindo a detecção dos problemas mais comuns.

O *Potter's Wheel* suporta os seguintes tipos de problemas:

- **Sintácticos** – Como consequência de diferenças ao nível da sintaxe dos atributos (*e.g.*: 31/05/04 e 2004/05/31).
- **Ao nível do esquema** – Resultantes de deficientes estratégias de integração de dados. Por exemplo, nalguns casos os valores de um determinado atributo encontram-se a nulo e noutros casos estão correctamente preenchidos.
- **Violação de restrições de integridade** – A detecção deste tipo de problemas requer algoritmos específicos do domínio. A detecção pode ser efectuada ao nível de um único registo ou envolvendo vários registos (*e.g.*: o somatório dos valores das linhas de uma factura não são iguais ao valor da factura correspondente).

Um conjunto de operadores, designado de *transformadores*, encontra-se disponível, permitindo a realização de transformações usuais ao nível do esquema. Os transformadores são simples e fáceis de especificar através do ambiente gráfico, não envolvendo linguagens complexas de transformação. Um elevado número de situações de transformação encontra-se coberto à custa da composição dos diferentes transformadores. Os transformadores efectuam um balanceamento entre três objectivos: (*i*) poder das operações de transformação sem recurso a programação explícita por parte do utilizador; (*ii*) facilidade de especificação gráfica; e, (*iii*) flexibilidade de aplicação interactiva. Os utilizadores recorrem aos transformadores à medida que são necessários, o que normalmente acontece após a detecção de algum problema. Como consequência, a sequência de transformações resultante possui frequentemente transformadores redundantes ou não optimizados. O *Potter's Wheel* converte esta sequência numa sequência de execução mais eficiente e efectua a sua compilação num programa optimizado de transformação de dados. A execução deste programa é depois efectuada sob a forma de *batch*, efectuando as transformações em todos os registos da tabela.

Os transformadores encontram-se organizados nas seguintes classes:

- **Transformadores de valor individual** – Aplicam uma função ao valor que se encontra num atributo em cada um dos registos existentes. Além das transformações baseadas em expressões regulares e operações aritméticas, permitem a invocação de funções definidas pelo utilizador para realizar transformações específicas.
- **Transformadores um para um** – Utilizados em operações ao nível do atributo (transformador vertical), transformando um registo num novo registo. A esta classe pertencem os seguintes transformadores: eliminar; copiar; inserir; fundir; separar (em função de uma posição ou expressão regular); e, dividir (em função de uma condição).

- **Transformadores um para muitos** – Usados em transformações de múltiplos registos (transformações horizontais), permitindo manipular heterogeneidades ao nível do esquema. Visam as situações em que a informação se encontra distribuída entre os valores dos atributos e o esquema da BD. Os transformadores que fazem parte desta classe são:
  - **Seleccionar registos** – Seleccionam registos com base numa condição (expressão regular, operação aritmética ou função) definida pelo utilizador.
  - **Agregar atributos** – Tendo por base um conjunto de atributos de um registo são produzidos múltiplos registos resultantes da agregação destes atributos num só e da replicação de toda a restante informação.
- **Transformadores muitos para muitos** – Transformam múltiplos registos num ou mais registos. Nesta classe apenas se encontra definido o seguinte transformador: desagregar atributos. Este transformador é usado para mover informação dos valores dos atributos para os nomes dos atributos.

3.5.1.5 FraQL

O *FraQL* [Sattler *et al.,* 2000] [Sattler e Schallehn, 2001] define uma arquitectura para as tarefas envolvidas na preparação de dados (integração, transformação, limpeza e redução de dados), tendo por base uma linguagem declarativa que permite o acesso e manipulação de dados armazenados em múltiplas fontes. Tendo por base um modelo de dados objecto-relacional, a linguagem é uma extensão ao SQL com características que permitem cobrir as necessidades particulares inerentes à preparação de dados. A implementação das extensões como primitivas de BD permite tirar partido das potencialidades intrínsecas dos sistemas de gestão de BD actuais. A principal vantagem da utilização de uma linguagem deste tipo, que combina mecanismos de preparação dos dados e potencialidades poderosas de inquérito a várias fontes heterogéneas, consiste numa integração virtual na qual é possível executar operações de transformação e limpeza sem afectar o conjunto de dados original. Desta forma, é possível ensaiar e avaliar diferentes estratégias sem ser necessário proceder ao carregamento e à materialização explícita dos dados.

Além de potencialidades de integração, transformação e redução de dados aqui não abordadas, este protótipo concede uma atenção especial à LD, suportando a realização das seguintes operações:

- **Conversão de valores** – São efectuadas conversões simples de valor recorrendo a funções predefinidas (*e.g.*: funções de normalização de valores numéricos) ou definidas pelo próprio utilizador.

- **Resolução de inconsistências entre os valores** – As inconsistências são solucionadas recorrendo a funções de reconciliação e agregação definidas pelo próprio utilizador.
- **Detecção e eliminação de duplicados** – O processo decompõe-se em duas fases. Na primeira, são identificados os registos que provavelmente se referem à mesma entidade do mundo real. Na segunda fase, os registos duplicados são consolidados (fundidos) num só. A detecção de duplicados é efectuada com base num critério de semelhança especificado pelo utilizador e em operadores de SQL estendido de *join*, união e agrupamento. A consolidação é efectuada com base em funções definidas pelo utilizador (*e.g.*: especificando que o valor mais recente é o que prevalece).
- **Preenchimento de valores em falta** – Algumas estratégias para o preenchimento de valores em falta estão disponíveis. A de maior simplicidade envolve a remoção de todos os registos que estejam afectados com valores em falta, com todos os inconvenientes que daí advêm. Uma outra alternativa passa por calcular o valor do atributo em falta com base nos restantes que estão preenchidos (*e.g.*: média ou mediana dos valores do atributo).

3.5.1.6 Análise Comparativa

Na Tabela 3.2 é apresentada uma comparação entre os cinco protótipos de investigação, com base num conjunto de características seleccionadas para o efeito. Os significados dos acrónimos usados são: BDR – BD Relacional; e, FT – Ficheiro de Texto.

Tabela 3.2 – Comparação das características dos protótipos de investigação

	Ajax	***Arktos II***	***IntelliClean***	***Potter's Wheel***	***FraQL***
Tipos de fontes suportadas	BDR, FT	BDR	BDR	BDR, FT	BDR
Preocupações optimização	Sim	Sim	Não	Sim	Não
Capacidades extensibilidade	Sim	Sim	Sim	Sim	Sim
Suporte a excepções	Sim	Sim	Não	Não	Não
Interface com o utilizador	Não gráfico	Gráfico	Não gráfico	Gráfico	Não gráfico
Potencialidades de anulação	Sim	Não	Não	Sim	Não
Execução incremental	Sim	Não	Não	Não	Não
Def. seq. exec. operações	Utilizador	Utilizador	Automática	Utilizador	Utilizador

O significado de cada característica considerada é apresentado de seguida:

- **Tipo de fontes suportadas** – Tipo de fontes a que o protótipo consegue aceder.

- **Preocupações de optimização** – Uso de técnicas com o objectivo de diminuir o tempo de execução das operações.
- **Capacidades de extensibilidade** – Possibilidade de incorporar novas funções definidas pelo utilizador na biblioteca de funções existente.
- **Suporte a excepções** – Uma excepção ocorre quando uma operação de limpeza falha num determinado registo. A manipulação de excepções pode ser efectuada por duas vias: (*i*) manualmente, através de uma interface com o utilizador adequada; ou, (*ii*) automaticamente, ignorando ou eliminando os registos excepcionais ou reportando-os para ficheiro/tabela.
- **Interface com o utilizador** – Tipo de ambiente usado para interagir com o utilizador. Nos ambientes que não são gráficos, recorre-se a linguagens declarativas para especificar as operações de LD. Os resultados da LD são colocados em tabelas ou ficheiro, competindo ao utilizador proceder à sua análise.
- **Potencialidades de anulação** – Possibilidade de anular uma operação caso os seus efeitos se mostrem indesejáveis.
- **Execução incremental** – Possibilidade de executar o processo de LD por incrementos. Caso haja necessidade de voltar a executar a sequência de operações de LD (após uma execução desta), esta não necessita de o ser na íntegra. A sequência pode ser executada apenas a partir de um determinado ponto e aplicar-se unicamente aos tuplos que se encontrem numa determinada situação (*i.e.*, obedeçam a uma determinada condição).
- **Definição da sequência de execução das operações** – Forma de definição da sequência de execução das operações de LD. Dependendo da ferramenta de LD, a sequência de operações é definida manualmente pelo utilizador ou automaticamente pela própria ferramenta.

Os resultados apresentados na tabela permitem tecer os seguintes comentários:

- Todos os protótipos permitem aceder a BD relacionais, uma vez que se trata do modelo de representação de dados que maior divulgação possui actualmente.
- A maioria dos protótipos denota preocupações a nível de optimização, tendo adoptado técnicas que visam diminuir o tempo de execução das operações.
- Todos os protótipos evidenciam potencialidades de extensibilidade, *i.e.*, permitem que funções definidas pelo utilizador sejam adicionadas às respectivas bibliotecas de funções.
- Apenas dois dos protótipos fornecem um suporte adequado às excepções que podem ocorrer durante a execução das operações.

- A maioria dos protótipos analisados recorre a interfaces com utilizador que não são gráficos.
- Não existem potencialidades de anulação na maioria dos protótipos, o que em certas operações pode fazer com que se tornem irremediavelmente definitivas.
- À excepção do *Ajax*, nenhum outro protótipo de investigação evidencia potencialidades de executar incrementalmente as operações de LD. Este protótipo permite que após uma primeira execução do grafo dirigido de transformações seja possível proceder novamente à sua execução, mas apenas a partir de um determinado ponto. Esta característica revela-se particularmente útil na manipulação das excepções que possam ter ocorrido aquando da execução anterior. Após a correcção manual destas ou de se ter refinado a operação de transformação, os tuplos envolvidos podem ser submetidos às restantes operações que tinham ficado pendentes.
- Apenas o *IntelliClean* define automaticamente a sequência de execução das operações de LD. Em todos os outros protótipos essa responsabilidade fica a cargo do utilizador. No entanto, no *IntelliClean* o tipo de operações de LD suportado encontra-se muito orientado para a detecção e eliminação de duplicados.

3.5.2 Ferramentas Comerciais

Há diversas ferramentas comerciais de LD disponíveis no mercado. À semelhança do que acontece com os protótipos de investigação, há ferramentas comerciais que estão vocacionadas para a manipulação de um único PQD. Estas ferramentas designam-se de *âmbito especializado* [Oliveira *et al.*, 2004]. As ferramentas que visam somente a detecção e eliminação de registos duplicados constituem um exemplo deste tipo de ferramentas específicas. As ferramentas *WizSame* [WizSoft, 2007c]; *matchIT* [helpIT, 2007]; e, *Centrus Merge/Purge* [Group 1 Software, 2007] (consultar Apêndice A) são exemplos de ferramentas vocacionadas apenas para o problema dos duplicados. Há outras ferramentas comerciais que procuram manipular um conjunto alargado de PQD, *i.e.*, dizem-se de *âmbito genérico* [Oliveira *et al.*, 2004]. Como no âmbito deste trabalho de doutoramento se pretende desenvolver um modelo que dê suporte aos vários problemas que afectam a QD, nas subsecções seguintes apresentam-se apenas as ferramentas de LD de âmbito genérico. Entre as várias alternativas disponíveis no mercado, seleccionou-se um conjunto de cinco ferramentas representativo das potencialidades de LD que estas ferramentas actualmente oferecem. As descrições apresentadas nas subsecções seguintes são baseadas na informação técnica disponível sobre as ferramentas nos *sites* das empresas proprietárias e na experimentação das próprias versões de demonstração quando tal foi possível.

3.5.2.1 WinPure Clean & Match

A ferramenta *WinPure Clean and Match* [WinPure, 2007] suporta a realização de um conjunto de operações de LD das quais se destaca a detecção e eliminação de duplicados. Além de permitir o acesso a ficheiros de texto, a ferramenta permite aceder a dados armazenados nos formatos *Microsoft Access* e *Microsoft Excel*. As operações de limpeza podem ser efectuadas numa FD ou em duas FD diferentes em simultâneo. Neste último caso, é possível a detecção e eliminação dos duplicados existentes entre estas.

A ferramenta permite ao utilizador visualizar e "navegar" nos dados de uma ou duas tabelas (em simultâneo). As operações suportadas pela ferramenta são:

- **Detecção de valores em falta** – É possível obterem-se estatísticas, sob a forma textual e gráfica, que auxiliam a identificar os valores em falta nos atributos seleccionados.
- **Limpeza no atributo** – Permite corrigir todos os valores incorrectos de um atributo, com base na exemplificação da correcção a efectuar num dos seus valores.
- **Limpeza de texto** – Permite remover caracteres indesejáveis num atributo do tipo textual (*e.g.*: espaços em branco; repetições de certos caracteres; e, caracteres numéricos em atributos exclusivamente textuais).
- **Conversor de caracteres** – Permite converter os valores que se encontram no atributo seleccionado para maiúsculas, minúsculas ou para a representação adequada (*e.g.*: no caso de serem nomes ou endereços).
- **Limpeza de *e-mail*** – Permite identificar endereços de *e-mail* incorrectos no atributo seleccionado, apresentando sugestões de correcção quando possível.

Após a realização das operações anteriores, procede-se à detecção e eliminação dos registos duplicados que possam existir. Para o efeito, o utilizador começa por especificar se a manipulação de duplicados é para ser efectuada ao nível da tabela ou a um nível entre tabelas. De seguida, o utilizador define os atributos a usar na detecção dos registos duplicados. No caso da manipulação de duplicados ser efectuada entre tabelas é necessário estabelecer mapeamentos entre os atributos seleccionados das duas tabelas. Caso os nomes dos atributos sejam iguais, estes mapeamentos podem ser automaticamente estabelecidos. Caso contrário, o utilizador tem de os estabelecer manualmente. No passo seguinte, o utilizador escolhe o tipo de análise que deve ser efectuada, *i.e.*, exacta ou aproximada, definindo qual o conteúdo semântico de cada atributo usado na identificação dos duplicados. Actualmente, apenas três possibilidades estão disponíveis: nome de pessoa; nome de empresa; e, linha de endereço. Esta informação é importante para a ferramenta seleccionar automaticamente os algoritmos mais adequados à detecção dos duplicados. Não foi

possível recolher qualquer informação adicional sobre estes algoritmos. Os registos duplicados são identificados em função da igualdade ou semelhança dos seus valores. Os registos que supostamente representam a mesma entidade do mundo real, *i.e.*, que são duplicados, são apresentados agrupadamente. Ao utilizador é permitido: (*i*) excluir registos dos grupos de duplicados, o que significa que não concorda com a sua inclusão no grupo; (*ii*) eliminar registos duplicados, manualmente ou automaticamente, com base num determinado critério (*e.g.*: critério aleatório); (*iii*) proceder à consolidação dos duplicados, dando origem a um só registo que agrega os vários valores; e, (*iv*) efectuar a filtragem dos registos duplicados.

A ferramenta também permite que o utilizador filtre (*e.g.*: registos duplicados; registos não duplicados) e exporte dados para outro ficheiro com o mesmo ou outro formato. Os formatos permitidos são os já referidos.

3.5.2.2 ETI Data Cleanser

A ferramenta *ETI Data Cleanser* [ETI, 2007a] possui um conjunto de funcionalidades de LD bastante completo. Estas funcionalidades permitem a realização das seguintes tarefas: separar os valores dos atributos nos seus componentes atómicos; proceder à uniformização dos valores; verificar a validade dos valores; identificar a existência de duplicados; e, consolidar cada grupo de duplicados num único registo.

A ferramenta extrai automaticamente os diversos elementos (*e.g.*: rua; número de polícia/porta; localidade) que se encontram num determinado atributo (*e.g.*: morada) em função do seu conteúdo semântico. Estes elementos são armazenados em novos atributos. A separação dos elementos é efectuada com base num conjunto de regras predefinidas na ferramenta. Além de lhes ser possível efectuar alterações, também é permitida a introdução de novas regras de modo a cobrirem situações específicas. A separação dos valores em componentes atómicos permite obter uma representação mais manuseável e adequada à realização de outros tipos de operações de limpeza (*e.g.*: detecção e eliminação de duplicados).

A ferramenta também suporta operações de uniformização e formatação dos valores (*e.g.*: abreviaturas; sintaxe de representação). As operações de uniformização são efectuadas de acordo com as regras definidas pelo utilizador ou com base em dados de referência adquiridos a entidades externas. Estas operações também contribuem para aumentar o número de duplicados identificados.

A ferramenta suporta a realização de tarefas de validação à correcção dos valores. Estas tarefas baseiam-se em dados de referência adquiridos a terceiros. Estes dados são usados para corrigir, suprir a ausência e completar os valores da FD em causa. Na identificação dos registos que nas duas FD se referem às mesmas entidades do mundo real são usadas técnicas de detecção de duplicados aproximados.

A detecção de registos duplicados na FD ou entre FD é também suportada por esta ferramenta. O utilizador define o critério que conduz à identificação dos duplicados. Para o efeito, a ferramenta disponibiliza uma interface gráfica de fácil utilização. Na identificação dos registos duplicados são utilizadas técnicas de correspondência/equivalência aproximada. Isto significa que são toleradas diferenças, não sendo necessário que os registos sejam rigorosamente iguais para que possam ser classificados como duplicados.

Por último, a ferramenta também possibilita a consolidação de cada grupo de duplicados num único registo. Esta consolidação pode ser efectuada manualmente pelo utilizador (*i.e.*, caso a caso) ou automaticamente com base num critério de sobrevivência dos valores (*e.g.*: o valor mais completo; o valor mais recente) definido para cada atributo. De cada grupo de duplicados resulta um novo registo que substitui todos os outros.

3.5.2.3 TS Quality

A ferramenta *TS Quality* [Trillium Software, 2007a] é composta por três módulos, cada um dos quais com as seguintes finalidades complementares de LD: uniformização; validação de endereços postais; e, detecção e eliminação de duplicados. Em cada módulo, a ferramenta disponibiliza de base/raiz um conjunto de regras de LD cuja utilização pode ser efectuada de imediato. O utilizador pode efectuar alterações a estas regras bem como introduzir novas regras, de modo a corresponderem às suas necessidades específicas. Por motivos de facilidade de compreensão e edição por parte do utilizador, as regras são especificadas sob a forma de tabelas textuais.

O *módulo de uniformização* visa uniformizar os valores dos atributos. Esta uniformização pode ser efectuada de forma totalmente automática, *i.e.*, não obrigando a que o utilizador especifique previamente as tarefas a efectuar. Para o efeito, são usadas as regras de uniformização que existem de base no módulo, tendo em conta o país de origem dos dados. As potencialidades de uniformização automática são especialmente notórias em dados sobre clientes, designadamente ao nível de nomes e endereços e também ao nível de dados sobre produtos, como códigos de produtos, marcas e modelos. Apesar de existir no módulo um conhecimento aprofundado sobre

este género de dados, uma especificação apropriada das regras de uniformização permite uniformizar qualquer outro género de dados. De facto, é permitida a especificação manual de regras de uniformização bem como a alteração das existentes de base, de modo a cobrir situações específicas de LD. O módulo possui potencialidades de interpretar, compreender e reorganizar os valores, tendo em conta o contexto em que se encontram. Em função do contexto, o significado de um mesmo valor pode diferir. As regras permitem detectar palavras-chave e padrões no valor de cada atributo, de que resulta a identificação dos elementos relevantes merecedores de serem armazenados em novos atributos. A realização destas operações de decomposição advém da existência de atributos sobrecarregados de valores. Estas operações melhoram não só o valor dos dados, como também aumentam a taxa de recuperação de registos duplicados.

O *módulo de validação de endereços postais* permite validar endereços postais de diversos países, ainda que o grau de validação concedido varie de país para país. O módulo identifica automaticamente o país de origem do endereço e aplica-lhe o conjunto de regras de validação postal respectivo, entre os diversos conjuntos que possui de base. Esta potencialidade é muito útil em FD que armazenam endereços referentes a países diferentes, mas nas quais não há qualquer indicação sobre o país a que os endereços dizem respeito. As regras permitem validar e, quando possível, corrigir e até mesmo completar os atributos referentes a endereços postais (*e.g.*: morada; código postal; localidade).

O *módulo de detecção e eliminação de duplicados*, como o nome o indica, permite detectar e eliminar a existência de registos duplicados ao nível da FD ou entre FD. O módulo fornece de base algumas regras genéricas de identificação de duplicados, disponíveis para uma utilização imediata. No entanto, a identificação de registos duplicados é efectuada essencialmente com base em regras especificadas pelo utilizador que podem ser armazenadas para utilizações futuras. As regras permitem identificar se os registos são duplicados, não duplicados ou potenciais duplicados. A existência de ligeiras diferenças entre os registos não impede a sua classificação como duplicados. A eliminação automática dos duplicados envolve a consolidação dos vários registos identificados como duplicados num só. Para o efeito, o utilizador especifica em cada atributo qual o critério de sobrevivência dos valores (*e.g.*: o mais recente; o mais completo), o que resulta na criação de um novo registo que substitui todos os duplicados detectados. A consolidação dos registos também pode ser efectuada manualmente pelo próprio utilizador através da interface gráfica que este módulo disponibiliza.

3.5.2.4 dfPower Quality

As principais funcionalidades oferecidas pela ferramenta *dfPower Quality* [DataFlux, 2007a] consistem na uniformização de valores e na detecção e eliminação de registos duplicados.

A ferramenta permite a realização de operações de uniformização nos valores dos atributos (*e.g.*: abreviaturas; formatos). Um conjunto de regras de uniformização é disponibilizado de raiz. Além de lhes ser possível efectuar alterações, também é permitida a introdução de novas regras de modo a cobrirem necessidades específicas de LD. A ferramenta também possui potencialidades de separar automaticamente os múltiplos valores que erradamente possam estar armazenados num único atributo do tipo textual (*string*). O mecanismo existente é capaz de "compreender" os seus significados semânticos, colocando-os em atributos individuais. Por exemplo, a identificação automática dos elementos referentes a um endereço, ao nome de um fornecedor e a um código de produto requer apenas o dispêndio de uns breves instantes na sua configuração através de uma interface gráfica do tipo *point-and-click*.

A ferramenta permite validar a correcção dos valores dos atributos por comparação com outros valores, da mesma ou de outra FD utilizada como referencial. Efectuar verificações às restrições de integridade e identificar as violações que a estas ocorrem também é possível. Um conjunto genérico de restrições de integridade é disponibilizado pela ferramenta para utilização imediata. Além destas restrições poderem ser alteradas, também é possível a definição de novas restrições de integridade cujo respeito se pretenda verificar. A ferramenta também disponibiliza uma biblioteca de transformações de dados bastante completa que suporta a realização de diversas operações de LD.

A ferramenta permite identificar e eliminar registos duplicados ao nível da FD ou entre FD. Os duplicados são identificados mesmo que existam diferenças de formatação e erros ortográficos nos valores dos atributos. A realização prévia de operações de uniformização (*e.g.*: formatos; abreviaturas; códigos numéricos) melhora significativamente os resultados desta tarefa. A identificação de duplicados baseia-se em inúmeras regras existentes de raiz na própria ferramenta. O utilizador pode alterá-las ou adicionar novas regras, de modo a corresponderem às especificidades da identificação de duplicados em causa. Os registos identificados como duplicados são consolidados num só com base em regras de sobrevivência dos valores, definidas pelo utilizador para cada atributo. A consolidação dos registos duplicados também pode ser efectuada, caso a caso, pelo próprio utilizador, recorrendo à interface gráfica disponível para o efeito.

3.5.2.5 HIquality

A ferramenta *HIquality* [Human Inference, 2007a] é composta por diversos módulos, cada um vocacionado para uma finalidade específica. No estudo das potencialidades desta ferramenta de LD foram analisados os seguintes módulos: *HIquality Name*; *HIquality Address*; *HIquality Identify*; e *HIquality Merge*.

O módulo *HIquality Name* [Human Inference, 2007d] efectua operações de limpeza em atributos que armazenam nomes de pessoas, assegurando uma notação correcta e completa. O módulo suporta a limpeza de nomes de diversos países. O conhecimento existente na base de conhecimento do módulo sobre cada país permite validar, abreviar, uniformizar e converter este género de dados. A base de conhecimento é usada para verificar todos os elementos que fazem parte de um nome (*e.g.*: apelidos; nomes próprios; títulos académicos), sendo considerados inválidos os que não façam parte desta. O conhecimento também é usado para corrigir discrepâncias nos formatos dos nomes e revelar potenciais conflitos com atributos relacionados (*e.g.*: sexo do individuo). O módulo é configurável permitindo uma adaptação às necessidades específicas deste género de dados. Estas necessidades resultam, entre outras, de regras gramaticais e de requisitos específicos de cada organização. As seguintes operações de LD encontram-se disponíveis: (*i*) *conversão de formatos* – converte os nomes em função de uma estrutura predefinida (*e.g.*: dividir um nome nos seus vários componentes; uniformizar os nomes com base em regras); (*ii*) *capitalização* – colocação da primeira letra de cada palavra que constitui o nome do indivíduo em maiúsculas; (*iii*) *abreviação de nomes* – este tipo de operação visa garantir que os nomes se ajustam ao espaço físico que lhes está destinado (*e.g.*: num envelope) ou ao tamanho do atributo onde serão armazenados; e, (*iv*) *validação de nomes* – valida os diversos componentes do nome de uma pessoa (*e.g.*: nome próprio; apelido) de acordo com as regras específicas do país a que este pertence ou de acordo com as regras específicas da organização.

O módulo *HIquality Address* [Human Inference, 2007e] efectua operações de limpeza em atributos que armazenam endereços postais. Este módulo permite validar, uniformizar, corrigir e completar endereços, colocando-os de acordo com o pretendido. Todos os tipos de elementos que constam de um endereço (*e.g*, nome do edifício; nome da localidade; código postal) são validados, sendo assegurado que se encontram segundo a ordem adequada. Caso algum problema seja detectado, os valores são: (*i*) corrigidos, sempre que possível onde for necessário; (*ii*) completados, caso se encontrem apenas parcialmente preenchidos; e, (*iii*) uniformizados os seus formatos. Entre outras possibilidades, isto envolve a correcção de erros ortográficos, a substituição de códigos postais incompletos ou a inclusão de códigos postais em falta. O módulo também permite a introdução

de configurações específicas, o que lhe permite responder a necessidades particulares de limpeza nos endereços postais. Uma interface gráfica suporta e simplifica este processo de configuração. Uma base de conhecimento fornece conhecimento sobre os standards, formatos e sistemas de códigos postais usados nos endereços específicos de cada país. De modo a garantir a fiabilidade do processo, a base de conhecimento específica de cada país é actualizada frequentemente pela empresa proprietária da ferramenta. O grau de funcionalidades oferecidas pelo módulo varia de país para país. Os formatos de uniformização possíveis em cada país estão disponíveis sob a forma de *templates*.

O módulo *HIquality Identify* [Human Inference, 2007f] identifica potenciais registos duplicados, mesmo que os seus valores não sejam exactamente iguais (*e.g.*: devido à existência de: erros ortográficos; abreviaturas; acrónimos; sequências diferentes de palavras). Uma base de conhecimento armazena o conhecimento necessário à identificação dos registos duplicados (*e.g.*: conhecimento linguístico) sob a forma de regras e factos. O conhecimento é crucial, não só para a identificação dos registos verdadeiramente similares, como para a correcta identificação do seu grau de semelhança. As comparações são efectuadas envolvendo os diversos atributos dos registos (*e.g.*: nomes próprios; apelidos; nomes de empresas; datas). Apesar do módulo poder ser usado em dados de inúmeros países, as melhores precisões na identificação de duplicados são alcançadas nos dados originários de um conjunto reduzido de países (*i.e.*, Holanda; Alemanha; Bélgica; França; e, Reino Unido). Isto acontece em virtude do vasto conhecimento que existe na base de conhecimento do módulo sobre os dados destes países. O processo de detecção de duplicados pode ser configurado manualmente, de modo a responder às necessidades específicas de detecção de duplicados. A configuração é efectuada pelo utilizador, recorrendo a uma interface gráfica de fácil utilização na qual são especificadas as regras que conduzem à identificação dos duplicados.

O módulo *HIquality Merge* [Human Inference, 2007g] procede à consolidação de dados, possibilitando a fusão dos registos identificados pelo módulo anterior como duplicados. O módulo permite que os grupos de duplicados possam ser automaticamente consolidados num só registo, com base em critérios especificados pelo utilizador, *i.e.*, regras de sobrevivência. Estas regras definem com exactidão a forma como os registos duplicados são consolidados, o que permite efectuar esta tarefa de forma totalmente automática. Assim, as decisões tomadas na consolidação dos registos são consistentes entre si, uma vez que obedecem às regras de sobrevivência estipuladas. O módulo disponibiliza de base um conjunto apreciável de regras de sobrevivência que cobrem inúmeras situações. Entre estas, encontram-se as seguintes: considerar

o valor do atributo do registo mais recente; considerar o valor mais completo do atributo. No entanto, também podem ser definidas pelo utilizador outras regras de sobrevivência específicas de cada situação concreta. Em alternativa, o módulo também suporta a consolidação manual dos grupos de duplicados. Neste caso é da responsabilidade do utilizador especificar, atributo a atributo, os valores que "sobrevivem". A interface gráfica do módulo permite ao utilizador: visualizar os registos duplicados; especificar os critérios de sobrevivência dos valores; e, consolidar manualmente os registos duplicados.

3.5.2.6 Análise Comparativa

Na Tabela 3.3 apresenta-se uma comparação entre as cinco ferramentas comerciais apresentadas usando, para o efeito, o mesmo conjunto de características considerado na comparação dos protótipos de investigação (ver Secção 3.5.1.6). Os significados dos acrónimos usados na tabela são: BDR – BD Relacional; FT – Ficheiro de Texto; e, DT – Diversos Tipos de FD (*e.g.*: BD relacionais; ficheiros de texto; ficheiros binários; e, fontes semi-estruturadas).

Tabela 3.3 – Comparação das características das ferramentas comerciais

	WinPure	***ETI Data Cleanser***	***TS Quality***	***dfPower Quality***	***HI Quality***
Tipos de fontes suportadas	BDR, FT	DT	DT	DT	BDR, FT
Preocupações de optimização	Sim	Sim	Sim	Sim	Sim
Capacidades de extensibilidade	Não	Não	Não	Não	Não
Suporte a excepções	Não	Sim	Sim	Sim	Sim
Interface com o utilizador	Gráfico	Gráfico	Gráfico	Gráfico	Gráfico
Potencialidades de anulação	Não	Não	Não	Não	Não
Execução incremental	Não	Não	Não	Não	Não
Def. seq. execução operações	Autom.	Autom.	Autom.	Autom.	Autom.

O significado de cada característica considerada é o mesmo do que o apresentado na Secção 3.5.1.6. Os resultados apresentados na tabela permitem tecer as seguintes considerações:

- As ferramentas comerciais analisadas maioritariamente cobrem um leque alargado de diferentes tipos de FD (*e.g.*: ficheiros binários; documentos XML).
- Todas as ferramentas evidenciam preocupações a nível de optimizar a execução das operações de LD. Este facto é especialmente evidente na detecção de registos duplicados.
- Nenhuma das ferramentas denota potencialidades de extensibilidade. A possibilidade de incorporar funções definidas pelo utilizador não se coaduna com o espírito destas

ferramentas orientadas para o utilizador não perito. O objectivo é permitir o seu uso sem obrigar a que os utilizadores possuam conhecimentos a nível de programação.

- Maioritariamente, as ferramentas analisadas toleram a ocorrência de excepções durante a realização das operações de LD. Os registos em que as operações não podem ser levadas a cabo são armazenados numa tabela, sendo reportados ao utilizador no final.
- Todas as ferramentas providenciam uma interface gráfica com o utilizador de modo a proporcionarem um ambiente intuitivo e de fácil utilização. Isto reflecte o seu cariz orientado ao utilizador não perito.
- Nenhuma ferramenta evidencia potencialidades de anular automaticamente as operações de correcção efectuadas. Para anular uma operação é necessário que o utilizador defina manualmente uma nova operação de LD. No entanto, nem todas as operações efectuadas são susceptíveis de serem anuladas desta forma, *i.e.*, algumas operações tornam-se irremediavelmente definitivas.
- Nenhuma ferramenta possui a capacidade de executar incrementalmente as operações de LD. Nas ferramentas comerciais analisadas, sempre que se executam as operações de LD suportadas, essa execução é feita, novamente, na íntegra, englobando todos os tuplos das relações envolvidas.
- Em todas as ferramentas analisadas a definição da sequência de execução das operações de LD é efectuada automaticamente. Essencialmente, estas ferramentas executam um conjunto de passos predefinidos que são configuráveis pelo utilizador para a DC dos PQD. Mais uma vez, este tipo de abordagem reflecte o cariz destas ferramentas orientadas para o utilizador não perito.

3.6 Conclusão

Neste capítulo abordou-se, em pormenor, a actividade do processo de melhoria da QD sobre a qual se centra este trabalho de doutoramento, *i.e.*, a LD. Tendo como objectivo a DC dos PQD (com base nas especificações fornecidas pelo utilizador), a LD é importante em diversos contextos, entre os quais se destacam: a descoberta de conhecimento em BD; os armazéns de dados; e, a integração de dados.

O termo LD é muitas vezes interpretado com o sentido de detecção e eliminação de duplicados. Se no passado esta interpretação até era válida, actualmente possui um significado mais vasto, abrangendo os diversos PQ que afectam os dados. Ainda assim, a detecção e eliminação de duplicados, bem como tudo aquilo que com esta se relaciona (*e.g.*: métricas de avaliação da semelhança), tem centrado a maioria dos esforços de investigação na área da LD. Esta realidade

reflecte-se nesta dissertação, como se pode comprovar pela dimensão da secção dedicada ao tema.

Neste capítulo foi descrito o actual *estado da arte* das ferramentas informáticas que suportam a realização de operações de LD. Para o efeito, foram seleccionadas ferramentas representativas originárias de duas comunidades distintas: académica e empresarial. As descrições foram efectuadas com um certo grau de detalhe para evidenciar as suas potencialidades de LD.

Os protótipos de investigação propostos pela comunidade académica, ainda que possuam potencialidades de LD, na sua essência encontram-se direccionados para a transformação de dados. Estes protótipos transformam dados que se encontram sob um determinado esquema $\mathcal{X}$ para um outro esquema $\mathcal{Y}$, em função de um grafo de transformações especificado pelo utilizador. Durante o processo de transformação suportam a realização de algumas operações de LD. O suporte concedido à LD acaba por ser reduzido uma vez que as ferramentas não foram concebidas especificamente para esta finalidade. O único protótipo vocacionado exclusivamente para a LD é o *IntelliClean* [Low *et al.,* 2001]. Contrariamente ao que acontece em todos os outros, este protótipo efectua as operações de LD na própria FD, não obrigando a transformações ao nível do esquema. No entanto, está limitado quase em exclusivo ao problema dos duplicados.

As ferramentas comerciais não são tecnicamente tão transparentes quanto os protótipos de investigação. Não é possível obter qualquer informação relativamente aos algoritmos usados (*e.g.*: na detecção de duplicados). Naturalmente, estes "segredos" resultam da concorrência que existe entre as empresas que desenvolvem as ferramentas. Esta mesma concorrência faz com que as empresas procurem disponibilizar as mesmas funcionalidades do que as concorrentes. As descrições apresentadas das ferramentas comerciais analisadas acabam por ser muito semelhantes. Por vezes, até dá ideia de se estar a descrever novamente a mesma ferramenta. Isto permite generalizar que outras ferramentas possuirão sensivelmente as mesmas funcionalidades. Basicamente, na generalidade das ferramentas, a LD corresponde à realização sequencial dos seguintes passos: atomização (*i.e.*, separação dos valores dos atributos nos seus elementos atómicos); uniformização; validação; detecção de duplicados; e, consolidação de duplicados. Esta sequência não é susceptível de ser alterada pelo utilizador, o que reflecte a filosofia destas ferramentas orientadas para o uso pelo *utilizador não perito*. Tal como acontece nos protótipos de investigação, não se consegue dissociar a LD de transformação de dados. A transformação de dados ocorre logo no primeiro passo da sequência (*i.e.*, aquando da atomização). Só depois dos dados respeitarem o esquema de representação predefinido pela ferramenta é que os restantes passos da sequência são efectuados. É muito pouco provável que este passo possa ser

dispensado, *i.e.*, que os dados já estejam de acordo com esse esquema de representação. Após a realização da LD os dados resultantes encontram-se segundo um esquema diferente do inicial. Implicitamente, considera-se que o novo esquema é o ideal para efectuar posteriores manipulações dos dados. No entanto, este pressuposto muitas vezes não se verifica, pelo que é necessário voltar a colocar os dados de acordo com o esquema original. Esta tarefa não é suportada nas ferramentas comerciais analisadas. À semelhança dos protótipos de investigação, no geral estas ferramentas também não permitem efectuar LD directamente na própria fonte, *i.e.*, de forma autónoma da transformação de dados.

Por outro lado, a sequência de passos adoptada também evidencia não ser a mais adequada para manipular certos PQD. No Capítulo 1 – Secção 1.2 exemplificou-se a inadequação desta sequência na verificação da violação de certas restrições de integridade. Ainda a propósito da verificação da violação de restrições de integridade, as ferramentas analisadas suportam a definição de novas restrições em função das necessidades específicas de LD. No entanto, a definição destas restrições é baseada em linguagens limitadas sob o ponto de vista da expressividade, daí que não suportem lógicas subjacentes complexas. Nestes casos, a única solução seria recorrer a uma linguagem de programação. O problema é que nas ferramentas comerciais analisadas nenhuma evidenciou potencialidades de extensibilidade. Por outro lado, ainda que de âmbito genérico, estas ferramentas aparentam ser limitadas na cobertura que dão a nível de DC aos diversos PQ que podem afectar os dados. De facto, este é um aspecto importante que fica a faltar analisar.

A avaliação da cobertura dada, não só pelas ferramentas comerciais, mas também pelos protótipos de investigação só pode ser efectuada após a apresentação da taxionomia de PQD proposta. Estes dois assuntos serão abordados no capítulo seguinte. As razões que justificam a criação de uma nova taxionomia já foram enunciadas no capítulo anterior.

O estudo do *estado da arte* da LD apresentado neste capítulo influenciou decisivamente o trabalho desenvolvido no âmbito deste doutoramento. Em particular, as lacunas e deficiências identificadas contribuíram para a formulação dos objectivos do trabalho (enunciados no Capítulo 1 – Secção 1.3).

PROBLEMAS DE QUALIDADE DOS DADOS 4

Este capítulo debruça-se detalhadamente sobre os problemas de qualidade que afectam dados relacionais. Assim, começa-se por propor uma taxionomia[12] de Problemas de Qualidade dos Dados (PQD), na qual os problemas estão agrupados por Nível de Granularidade (NG) de ocorrência. Os diferentes NG considerados são: atributo; tuplo; relação; múltiplas relações; e, múltiplas Fontes de Dados (FD). Seguidamente, a taxionomia é comparada com os trabalhos relacionados, evidenciando ser mais genérica e abrangente que estes na cobertura dos PQD. Posteriormente, para cada problema da taxionomia são apresentadas definições formais que eliminam possíveis subjectividades inerentes às definições textuais. Por outro lado, como são mais rigorosas possuem mais informação do que as definições textuais, especificando tudo aquilo que é necessário para automatizar a sua detecção. Finalmente, a taxionomia proposta é usada para avaliar a cobertura, a nível de Detecção e Correcção (DC), que as actuais soluções de Limpeza de Dados (LD) dão aos PQD. A avaliação é efectuada nas soluções oriundas de duas áreas distintas: académica/científica e comercial/empresarial. Uma vez que as ferramentas comerciais de LD se baseiam nas ferramentas de *data profiling* para a detecção de certos problemas, a taxionomia proposta também é usada para avaliar a cobertura destas.

4.1 Introdução

Actualmente, a importância e utilidade dos dados é largamente reconhecida. Os dados são um activo chave na melhoria da eficiência e eficácia, num ambiente organizacional actual de grande dinamismo e competitividade. No entanto, também é sabido que geralmente os dados se encontram afectados por diversos problemas de qualidade [Orr, 1998]. Tal como se evidenciou no Capítulo 2, diferentes interpretações podem ser dadas à expressão PQD. Relembre-se que no contexto deste trabalho de doutoramento, PQD refere-se aos problemas que se manifestam ao nível dos valores dos dados. Estes problemas afectam todos os projectos que se baseiam nos dados (*e.g.*: mineração de dados; análise multi-dimensional de dados; gestão do relacionamento

[12] Uma *taxionomia* é um sistema de classificação que organiza entidades/conceitos [Giovani, 2005].

com o cliente). A fraca qualidade dos dados condiciona negativamente a qualidade dos resultados que se obtêm nos processos baseados na sua exploração/análise [Sattler e Schallehn, 2001] (princípio "lixo entra, lixo sai"). Como tal, os PQD necessitam de ser detectados e corrigidos, *i.e.*, os dados têm de ser "limpos".

No âmbito deste trabalho importa identificar quais são os diversos problemas de qualidade que afectam os dados. Fundamentalmente, ter um perfeito conhecimento dos PQD é importante pelo seguinte:

- Possibilita desenvolver um modelo de LD que, de uma forma global e integrada, lhes dê o necessário suporte a nível de DC.
- Permite avaliar e compreender a cobertura que lhes é dada pelas soluções informáticas actualmente existentes, a nível de DC.
- Permite orientar os esforços de investigação e desenvolvimento, realçando aqueles que merecem uma maior atenção, *i.e.*, se um determinado PQD não possui suporte a nível de detecção ou correcção, isso significa que deve ser objecto de uma atenção especial.

A identificação dos diversos problemas que afectam a qualidade dos dados resultou na elaboração de uma taxionomia cuja apresentação é efectuada na secção seguinte.

4.2 Taxionomia de Problemas de Qualidade dos Dados

A Figura 4.1 ilustra a estrutura de organização dos dados segundo o modelo relacional. Neste modelo, os dados encontram-se armazenados numa ou mais FD (*e.g.*: Bases de Dados (BD); sistema de ficheiros). Cada FD é composta por *relações*, podendo existir *associações* (*i.e.*, relacionamentos) entre estas. Uma *relação* é constituída por vários *tuplos*, sendo cada um destes formado por um número pré-definido de *atributos*. Esta estrutura resulta na seguinte hierarquia de NG dos dados: múltiplas FD; múltiplas relações; relação; tuplo; e, atributo.

A taxionomia de PQD ao nível da instância [Oliveira *et al.*, 2005a] foi criada baseada neste modelo. Cada NG foi minuciosamente analisado, tendo os problemas sido identificados com base nos diferentes elementos que fazem parte do modelo (*e.g.*: tipos de dados; domínios dos atributos; relacionamentos existentes; características usuais dos valores armazenados). A análise foi efectuada de forma independente de qualquer BD específica, o que lhe confere um carácter genérico e universal. Desta forma, a taxionomia não é meramente representativa dos PQD

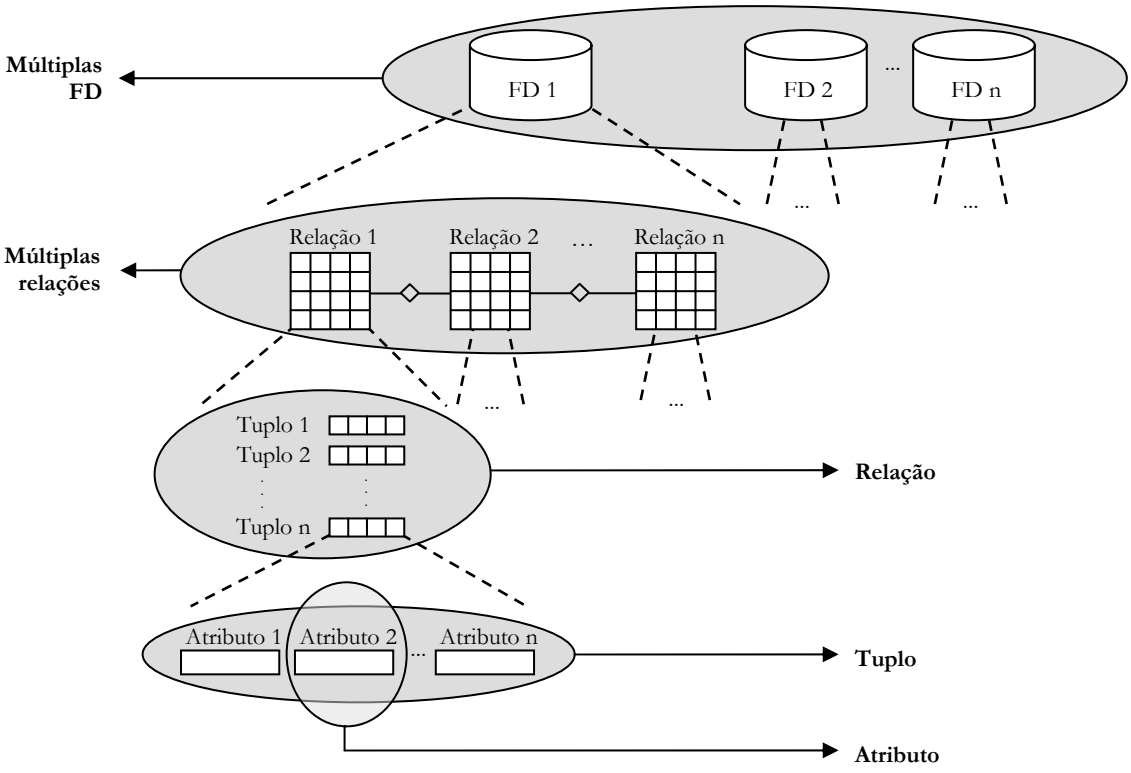


Figura 4.1 – Estrutura de organização dos dados segundo o modelo relacional

identificados numa BD ou num conjunto de BD, em particular. Se assim fosse, certamente seria uma taxionomia incompleta e não representativa dos diversos PQD que ocorrem.

A aproximação adoptada foi ascendente, *i.e.*, começou-se por analisar o NG mais elementar (*i.e.*, atributo) e terminou-se no NG mais complexo (*i.e.*, múltiplas FD). O objectivo da aproximação consistiu em identificar, em primeiro lugar, os problemas mais concretos e de fácil detecção, deixando para o fim os mais genéricos e de difícil detecção. Nas secções seguintes encontram-se os PQD identificados em cada NG (Secção 4.2.1 – Problemas ao nível do atributo; Secção 4.2.2. – Problemas ao nível do tuplo; Secção 4.2.3 – Problemas ao nível da relação; Secção 4.2.4 – Problemas ao nível de múltiplas relações/múltiplas FD).

Os motivos que levaram ao desenvolvimento desta nova taxionomia, quando já existem outras na literatura, já foram devidamente explicados no Capítulo 2 – Secção 2.6. Na identificação dos problemas que compõem a taxionomia, começou-se "a partir do zero", *i.e.*, ignorando os problemas incluídos nas taxionomias relacionadas. Este trabalho resultou numa nova taxionomia, independente das já existentes, uma vez que a sua criação não foi minimamente influenciada por aquelas. A taxionomia cobre os problemas que afectam os dados representados num formato tabular, *i.e.*, que se encontram pelo menos na primeira forma normal. Exceptuando dados do tipo

multimédia que obrigam a um tipo especial de manipulação, todos os outros tipos de dados (*i.e.*, caracter, *string*, numérico, data/hora, enumerado) foram considerados na identificação dos PQD.

Seguidamente, inicia-se a apresentação da taxionomia com a exposição dos PQD que ocorrem no NG mais elementar, *i.e.*, ao nível do atributo.

4.2.1 Problemas ao Nível do Atributo

Na identificação dos PQD que ocorrem no atributo efectuaram-se dois tipos de análises: ao valor individual do atributo e ao conjunto de valores do atributo. Os problemas identificados em cada contexto são apresentados nas secções seguintes.

4.2.1.1 Contexto do Valor Individual

Nesta secção apresentam-se os PQD que envolvem o valor individual dos atributos, tal como se ilustra na Figura 4.2.

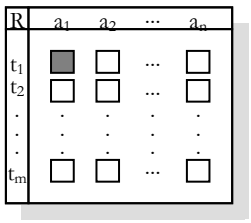


Figura 4.2 – Valor de um atributo (num tuplo)

Na identificação dos PQD que ocorrem neste contexto, utilizou-se o critério representado na Figura 4.3. O significado de cada PQD identificado é clarificado de seguida:

- **Valor em falta** – Ausência de valor num atributo que é de preenchimento obrigatório (*e.g.*: falta de preenchimento do atributo *nome_cliente*).
- **Violação de sintaxe** – O valor não respeita a sintaxe estabelecida para o atributo (*e.g.*: o atributo *cod_produto* contém o valor *ABC123* em vez de *ABC–123*).
- **Erro ortográfico** – O valor contém um erro ortográfico acidental (*i.e.*, de digitação) ou não (*i.e.*, o utilizador não soube escrever correctamente a palavra) (*e.g.*: o atributo localidade contém o valor *Pedrozo* em vez de *Pedroso*).
- **Violação de domínio** – O valor não faz parte do conjunto de valores válidos do atributo (*e.g.*: no atributo *quantidade_encomendada* encontra-se um valor negativo). Quando se está perante um atributo do tipo enumerado textual, uma violação de domínio pode corresponder a um:

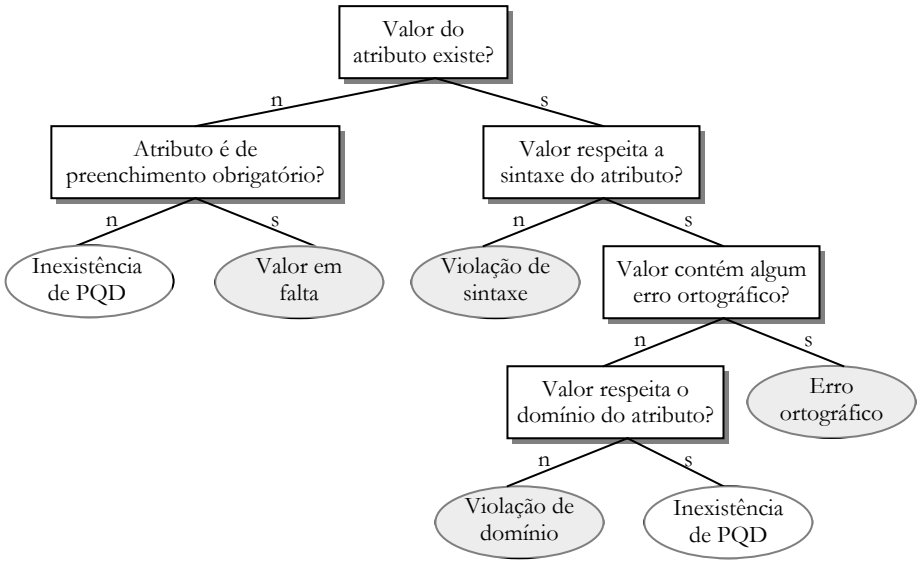


Figura 4.3 – Árvore de decisão para identificação dos PQD que ocorrem no valor individual do atributo

- **Valor sobrecarregado** – O valor do atributo contém informação além do que é pretendido (*e.g.*: no atributo *localidade* além da localidade encontra-se também o concelho: *Pedroso – Vila Nova de Gaia*).
- **Valor incompleto** – O valor do atributo contém informação aquém do que é pretendido (*e.g.*: no atributo *localidade* encontra-se o valor *Feira* quando deveria ser *Santa Maria da Feira*).

4.2.1.2 Contexto Multi-Valor

Nesta secção apresentam-se os PQD que envolvem os valores de um atributo, quando estes são considerados conjuntamente, como ilustrado na Figura 4.4.

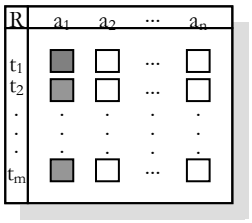


Figura 4.4 – Valores de um atributo (em múltiplos tuplos)

A Figura 4.5 representa o critério utilizado na identificação dos PQD que ocorrem neste contexto.

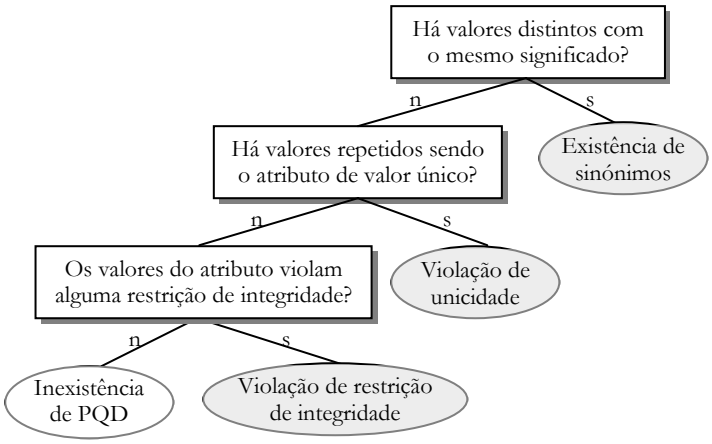


Figura 4.5 – Árvore de decisão para identificação dos PQD que ocorrem nos vários valores de um atributo

O significado de cada PQD identificado é clarificado de seguida:

- **Existência de sinónimos** – No atributo são usados valores diferentes mas com igual significado (*e.g.*: no atributo *profissão* um tuplo contém o valor *Professor* enquanto outro tuplo contém o valor *Docente*).
- **Violação de unicidade** – Um atributo de valor único possui valores iguais em mais do que um tuplo (*e.g.*: dois clientes possuem o mesmo número de identificação fiscal).
- **Violação de restrição de integridade** – Uma restrição de integridade ao nível dos valores do atributo não é respeitada (*e.g.*: o somatório dos valores que se encontram no atributo *valor_percentual* não é igual a *100%*, violando assim uma restrição existente).

4.2.2 Problemas ao Nível do Tuplo

Nesta secção apresenta-se o PQD que envolve os valores dos atributos de um tuplo, quando considerados conjuntamente, de acordo com o ilustrado na Figura 4.6.

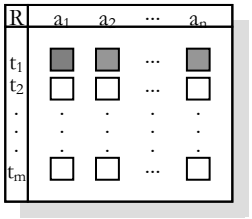


Figura 4.6 – Valores dos atributos de um tuplo

Na identificação do PQD que ocorre neste nível usou-se o critério representado na Figura 4.7.

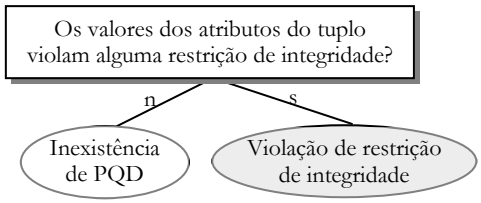


Figura 4.7 – Árvore de decisão para identificação do PQD que ocorre ao nível do tuplo

O significado do único PQD identificado é a seguir apresentado:

- **Violação de restrição de integridade** – Uma restrição de integridade ao nível dos valores dos atributos do tuplo não é respeitada (*e.g.*: num determinado tuplo o valor do atributo *total_produto* não é igual ao valor resultante da multiplicação entre os valores dos atributos *preço_unitário* e *quantidade*, violando assim uma restrição existente).

4.2.3 Problemas ao Nível da Relação

Nesta secção apresentam-se os PQD que envolvem valores de vários atributos em múltiplos tuplos de uma relação, quando considerados conjuntamente, como se ilustra na Figura 4.8.

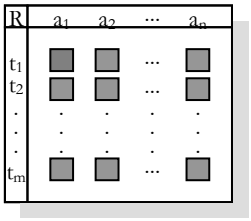


Figura 4.8 – Valores dos atributos em múltiplos tuplos

A Figura 4.9 representa o critério utilizado na identificação dos PQD que ocorrem a este nível. O significado de cada PQD identificado é clarificado de seguida:

- **Violação de dependência funcional** – Uma dependência funcional envolvendo dois ou mais atributos é violada pelos valores de um tuplo (*e.g.*: considerando que existe uma dependência funcional entre os atributos *código_postal* e *localidade*, os seguintes dois tuplos representando clientes constituem uma violação de dependência funcional: *cliente*(…, 4415-206, 'Pedroso', …) e *cliente*(…, 4415-206, 'Vila Nova de Gaia', …)).
- **Circularidade entre tuplos num auto-relacionamento** – Representa uma situação de ciclo entre dois (circularidade directa) ou mais tuplos (circularidade indirecta) de uma relação na qual existe um auto-relacionamento (*e.g.*: considerando que um produto pode ser subproduto num outro produto e que esta informação se encontra armazenada na relação *produtos* com o seguinte esquema: *produtos*(*cod_produto*, …, *cod_subproduto*), dos tuplos *produtos*('xpto', …, 'ypto') e *produtos*('ypto', …, 'xpto') resulta uma circularidade directa.

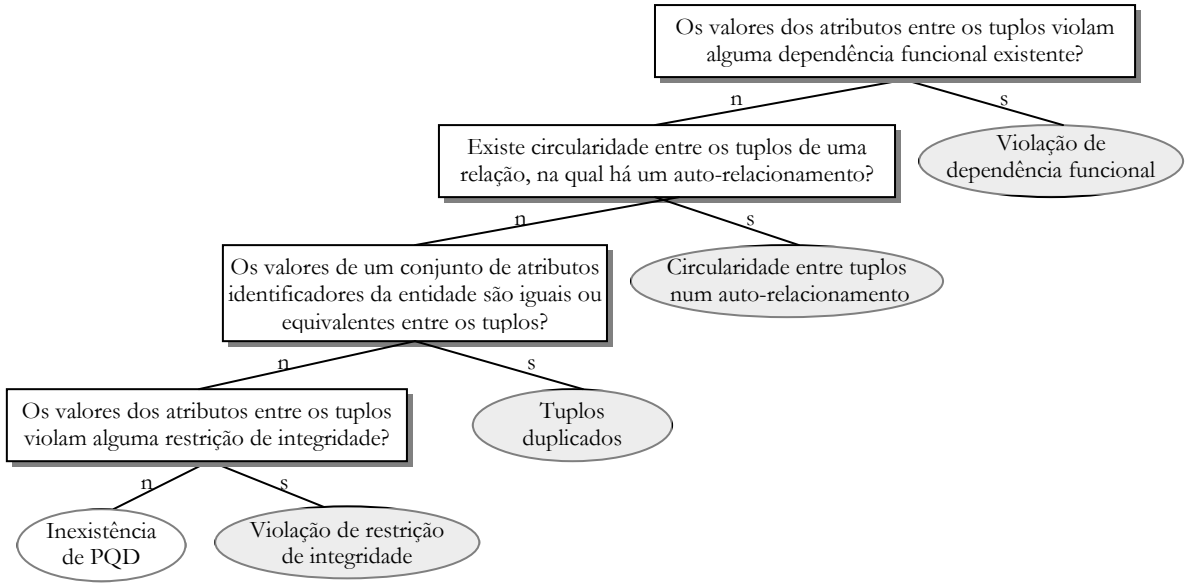


Figura 4.9 – Árvore de decisão para identificação dos PQD que ocorrem ao nível da relação

- **Tuplos duplicados** – A mesma entidade ou objecto do mundo real encontra-se representada em mais do que um tuplo. A classificação de dois tuplos como sendo duplicados pode ainda ser mais detalhada, de acordo com o que a seguir se apresenta. Para ilustrar cada situação de duplicidade, suponha-se a existência de uma relação *clientes* com o seguinte esquema: *clientes*(*nr_cliente*, *nome*, *morada*, *nr_identificação_fiscal*).
  - **Duplicados iguais** – Os tuplos são rigorosamente iguais, diferindo apenas nos valores dos atributos que constituem a chave primária da relação (caso esta exista) (*e.g.*: o tuplo *clientes*(10, 'António Carlos Carvalho', 'Rua do Sol, 123', 205723467) é um duplicado exacto do tuplo *clientes*(72, 'António Carlos Carvalho', 'Rua do Sol, 123', 205723467)).
  - **Duplicados aproximados** – Os tuplos são aproximadamente iguais, não sendo as diferenças nos valores dos atributos significativas que impeçam a sua classificação como tal (*e.g.*: o tuplo *clientes*(10, 'António Carlos Carvalho', 'Rua do Sol, 123', 205723467) é um duplicado aproximado do tuplo *clientes*(72, 'António C. Carvalho', 'Rua do Sol - 123', 205723467)). Os atributos que integram a chave primária da relação (caso esta exista) não são considerados na avaliação das diferenças.
  - **Duplicados inconsistentes** – Ainda que os tuplos sejam duplicados um do outro, um ou mais atributos que não fazem parte da chave primária da relação contêm valores que são conflituosos entre si (*e.g.*: o tuplo *clientes*(10, 'António Carlos Carvalho', 'Rua do Sol, 123', 205723467) é um duplicado inconsistente do tuplo *clientes*(72, 'António Carlos Carvalho', 'Rua das Flores, 987', 205723467)).
- **Violação de restrição de integridade** – Uma restrição de integridade ao nível dos valores dos atributos de múltiplos tuplos não é respeitada (*e.g.*: a quantidade de existências

em stock de produtos da família *X* não é superior a *100*, violando assim uma restrição existente).

4.2.4 Problemas ao Nível de Múltiplas Relações/Fontes de Dados

Nesta secção apresentam-se os PQD que envolvem os valores de múltiplas relações pertencentes a uma ou mais FD, quando considerados conjuntamente, tal como se ilustra na Figura 4.10. Entre as relações podem existir ou não associações.

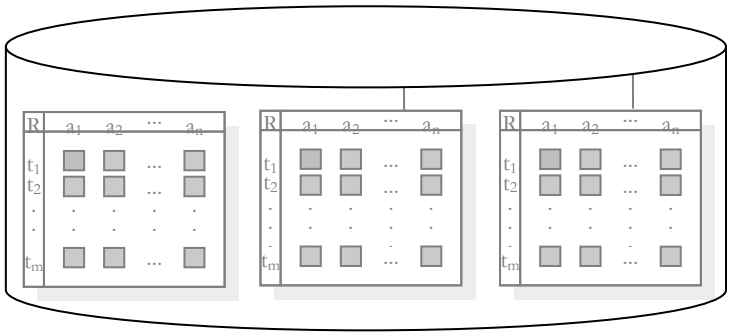


Figura 4.10 – Valores dos atributos em múltiplos tuplos de múltiplas relações (de uma ou mais FD)

O critério utilizado na identificação dos PQD que ocorrem a estes níveis encontra-se representado na Figura 4.11. Seguidamente, clarifica-se o significado de cada PQD identificado:

- **Heterogeneidade de sintaxes** – Utilização de sintaxes diferentes entre as relações para representar a mesma propriedade (*e.g.*: na relação *X* o atributo *data_encomenda* é representado sob a forma *dia/mês/ano*, enquanto que na relação *Y* o atributo *data_enc* é representado sob a forma *ano/mês/dia*).
- **Heterogeneidade de unidades de medida** – Utilização de unidades de medida distintas entre relações diferentes para representar a mesma propriedade (*e.g.*: na relação *Facturas* da FD 1, o atributo *Total_Factura* representa euros, enquanto que na relação *Facturas* da FD 2, o atributo *Tot_Fact* representa dólares).
- **Existência de sinónimos** – Utilização de valores diferentes entre as relações, mas com igual significado (*e.g.*: na relação *X* é usado o termo *Professor*, enquanto que na relação *Y* é usado o termo *Docente*).
- **Existência de homónimos** – Uso de valores iguais entre relações diferentes, mas com significado distinto (*e.g.*: na relação *Produtos* da FD 1 o termo *vela* é usado com o sentido de componente automóvel, enquanto que na relação *Produtos* da FD 2 o mesmo termo significa material de iluminação).

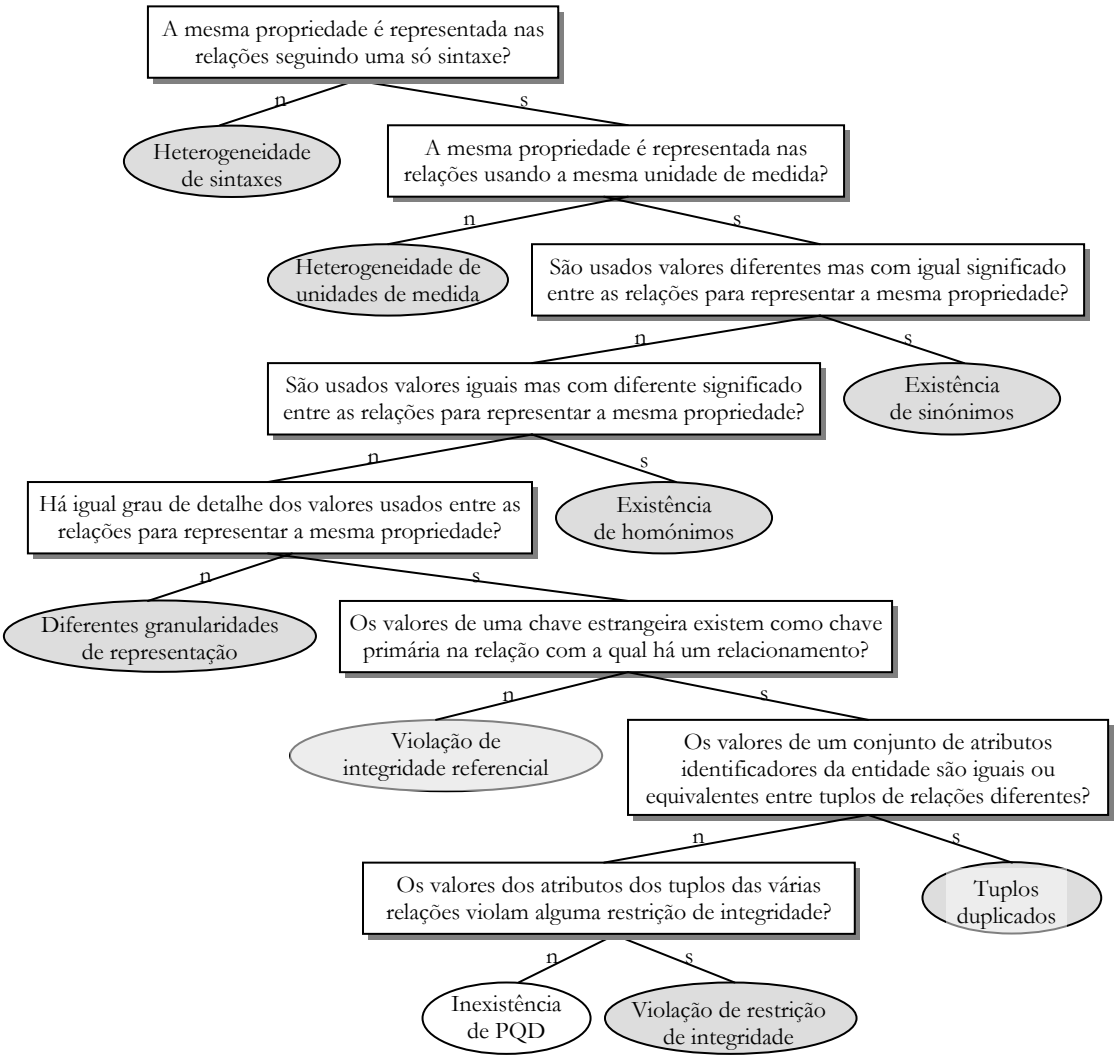


Figura 4.11 – Árvore de decisão para identificação dos problemas que ocorrem ao nível de múltiplas relações

- **Diferentes granularidades de representação** – A mesma propriedade do mundo real encontra-se representada com diferentes níveis de detalhe entre as relações (*e.g.*: para representar a propriedade estado civil, na relação *Pacientes* da FD 1 são usados os valores: *casado*, *solteiro*, *viúvo* e *divorciado*; enquanto que na relação *Doentes* da FD 2 são usados os valores: *casado*, *solteiro* e *outro*).

- **Violação de integridade referencial** – Os valores dos atributos que formam uma chave estrangeira numa relação não surgem como valores de chave primária na relação com a qual existe um relacionamento (*e.g.*: um tuplo da relação *clientes* contém o valor *5100* no atributo *código_postal*, sendo este inexistente na relação *códigos_postais*).

- **Tuplos duplicados –** A mesma entidade ou objecto do mundo real encontra-se representada em diferentes relações. À semelhança do que acontece ao nível da relação, a classificação de dois tuplos como sendo duplicados pode ainda ser mais detalhada, em função do que a seguir se apresenta. Para ilustrar convenientemente as situações,

suponha-se a existência das relações *clientes1* e *clientes2*, ambas com esquema: (*nr_cliente*, *nome*, *morada*, *nr_identificação_fiscal*).

- **Duplicados iguais** – À excepção dos valores dos atributos que constituem a chave primária de cada uma das relações (caso estas existam), todos os outros valores são rigorosamente iguais (*e.g.*: o tuplo *clientes1*(10, 'António Carlos Carvalho', 'Rua do Sol - 123', 205723467) é um duplicado exacto do tuplo *clientes2*(72, 'António Carlos Carvalho', 'Rua do Sol - 123', 205723467)).
- **Duplicados aproximados** – Ainda que existam diferenças entre os valores de alguns atributos, estas não são significativas que impeçam a classificação dos dois tuplos como sendo duplicados (*e.g.*: o tuplo *clientes1*(10, 'António Carlos Carvalho', 'Rua do Sol, 123', 205723467) é um duplicado aproximado do tuplo *clientes2*(72, 'António C. Carvalho', 'Rua do Sol - 123', 205723467)). Na avaliação das diferenças não são considerados os atributos que integram as chaves primárias das relações (caso estas existam).
- **Duplicados inconsistentes** – Apesar dos tuplos serem duplicados um do outro, um ou mais atributos que não fazem parte das chaves primárias (caso estas existam) das relações contêm valores que são contraditórios entre si (*e.g.*: o tuplo *clientes1*(10, 'António C. Carvalho', 'Rua do Sol, 123', 205723467) é um duplicado inconsistente do tuplo *clientes2*(72, 'António C. Carvalho', 'Rua das Flores, 987', 205723467)).

- **Violação de restrição de integridade** – Uma restrição de integridade envolvendo valores provenientes de múltiplas relações não é respeitada (*e.g.*: o somatório dos valores dos produtos que fazem parte de uma determinada factura (armazenados na relação *detalhes_facturas*) não é igual ao total da factura (armazenado na relação *facturas*), o que viola uma restrição enunciando que têm de ser iguais).

Optou-se por apresentar os PQD dos dois níveis conjuntamente numa só secção, uma vez que estes são os mesmos. Quando se consideram os valores de várias relações os problemas são iguais, independentemente destas pertencerem a uma ou mais FD. Há, no entanto, uma preponderância para que certos problemas ocorram mais frequentemente entre relações de FD diferentes, do que de uma só fonte. Isto acontece com a heterogeneidade de sintaxes e unidades de medida, os tuplos duplicados, a existência de homónimos e a existência de diferentes granularidades de representação. Ao nível de uma só FD, a situação habitual é: existir homogeneidade nas sintaxes e unidades de medida utilizadas; um determinado conjunto de entidades do mundo real (*e.g.*: clientes) estar armazenado numa só relação (*i.e.*, assim não existem duplicados entre relações); não serem usados valores com significados distintos consoante a relação; e, ser usado o mesmo grau de detalhe na representação dos valores. Quer isto dizer que

habitualmente ao nível de uma FD estes problemas não existem entre as relações. No entanto, não é possível excluir, de todo, a sua existência. A título exemplificativo, suponha-se que uma determinada BD contém, além de outras relações, duas relações onde se armazenam respectivamente os clientes e os antigos clientes de uma empresa. Ao fim de cinco anos sem ser efectuada qualquer transacção comercial, o cliente é transferido da tabela de clientes para a tabela de antigos clientes. Se os tuplos não forem removidos da tabela de clientes, passam a constar de ambas as tabelas. Quer isto dizer que é possível encontrar tuplos duplicados entre duas tabelas pertencentes à mesma BD. De igual forma, poderiam ser fornecidos exemplos que ilustrariam a ocorrência dos outros PQD entre relações de uma FD.

4.2.5 Resumo

Na Tabela 4.1 apresenta-se uma síntese dos PQD que compõem a taxionomia, a respectiva dimensão da QD em que cada problema se insere (de acordo com as definições apresentadas no Capítulo 2 - Secção 2.2.2.2) e os respectivos NG onde ocorrem.

Tabela 4.1 – PQD por NG

PQD	Dimensão da QD	NG				
		Atributo		Tuplo	Relação	Múltiplas relações/ FD
		Valor individual	Multi-valor			
Valor em falta	Completude	×				
Violação de sintaxe	Consistência	×				
Erro ortográfico	Correcção	×				
Violação de domínio	Correcção	×				
Violação de unicidade	Consistência		×			
Existência de sinónimos	Consistência		×			×
Violação de restrição de integridade	Consistência		×	×	×	×
Violação de depend. Funcional	Consistência				×	

Tabela 4.1 – PQD por NG (continuação)

PQD	Dimensão da QD	NG				
		Atributo		Tuplo	Relação	Múltiplas relações/ FD
		Valor individual	Multi-valor			
Tuplos duplicados	Consistência				×	×
Circularidade entre tuplos num auto relacionamento	Consistência				×	
Existência de homónimos	Consistência					×
Violação de integr. referencial	Consistência					×
Dif. granularidades de representação	Consistência					×
Heterogeneidade de sintaxes	Consistência					×
Heterogeneidade de unidades de medida	Consistência					×

Não se poderia concluir esta apresentação resumida dos PQD que fazem parte da taxionomia proposta, sem efectuar os seguintes comentários a dois supostos problemas mencionados nalguma literatura da área:

- A existência de valores consideravelmente diferentes dos restantes, *i.e.*, que não se enquadram no padrão normal (designados de *desvios*) é classificada como sendo um PQD. No âmbito desta dissertação, PQD é tudo aquilo que de facto constitui um erro, anomalia, redundância ou inconsistência. Um desvio não corresponde necessariamente a uma destas situações. Um desvio representa um potencial PQD, mas que na prática pode não o ser. Assim, se um valor não viola o domínio do atributo, mesmo que substancialmente diferente dos restantes, considera-se que este não constitui um PQD. Um desvio pode ser classificado como sendo uma *violação de restrição de integridade*, caso esta possa ser enunciada (*e.g.*: o peso de um individuo não pode exceder em mais de 200% a média dos pesos de todos os indivíduos). Em consonância com a decisão de não considerar os desvios como PQD e, assim, de não os incluir na taxionomia proposta, estão as outras três taxionomias relacionadas (publicadas em: [Rahm e Do, 2000], [Kim *et al.*, 2003] e [Müller e Freytag, 2003]) onde estes também não surgem como PQD.

- A existência de diferenças aleatórias nos valores de um atributo, entre dois tuplos diferentes, mas respeitantes à mesma entidade do mundo real é designada de *ruído*. Entre outras possibilidades, o ruído pode surgir como consequência de diferenças nas medições efectuadas ou causado por diferentes estratégias de arredondamento dos valores. Este problema não consta da taxionomia proposta, uma vez que se encontra incluído nos PQD *tuplos duplicados inconsistentes* ou *violação de restrição de integridade*. Se a diferença entre os valores excede um determinado limiar de tolerância definido pelo utilizador, em função da situação, está-se perante *tuplos duplicados inconsistentes* ou *violação de restrição de integridade*. Caso contrário, não se considera existir PQD em virtude das diferenças entre os valores serem desprezáveis. À semelhança da taxionomia proposta, o ruído também não surge como PQD nas taxionomias relacionadas.

4.3 Comparação com Trabalho Relacionado

Na Tabela 4.2 estabelece-se uma correspondência entre os PQD da taxionomia proposta nesta dissertação (primeira coluna) e os PQD incluídos nas taxionomias relacionadas (restantes colunas).

Tabela 4.2 – Taxionomia proposta *versus* outras taxionomias

PQD	[Rahm e Do, 2000]	[Kim *et al.*, 2003]	[Müller e Freytag, 2003]
Valor em falta	Valor em falta	Valor em falta	Valor em falta
Violação de sintaxe	Transposição de valores	Disposições diferentes	Erro sintáctico
Erro ortográfico	Erro ortográfico	Erro ortográfico	×
Violação de domínio	Valor ilegal	Violação do intervalo de valores	Violação restrição de integridade
Valor sobrecarregado	Outros valores embutidos	Valores extra	×
Valor incompleto	×	×	×
Violação de unicidade	Violação de unicidade	Valor duplicado	Violação restrição de integridade
Existência de sinónimos	Dif. representações dos valores	Valores alternativos /abreviados	Irregularidade
Violação de restrição de integridade	Violação de depend. entre atributos	Inconsistência entre múltiplas tabelas	Violação restrição de integridade

Tabela 4.2 – Taxionomia proposta *versus* outras taxionomias (continuação)

PQD	[Rahm e Do, 2000]	[Kim *et al.*, 2003]	[Müller e Freytag, 2003]
Violação de dependência funcional	Violação de depend. entre atributos	Valores mutuamente inconsistentes	Violação restrição de integridade
Circularidade entre tuplos num auto-relacio.	×	×	×
Tuplos duplicados	Registos duplicados	×	Duplicados
Dupl. iguais	Registos duplicados	×	Duplicados
Dupl. aproximados	Registos duplicados	×	Duplicados
Dupl. inconsistentes	Registos contraditórios	×	Duplicados
Existência de homónimos	×	Contexto incompleto	×
Dif. granularidades de representação	Diferentes níveis de agregação	×	×
Violação de integridade referencial	Violação integridade referencial	Violação integridade referencial	Violação restrição de integridade
Heterogeneidade de sintaxes	×	×	×
Heterogeneidade de unidades de medida	Dif. significados dos valores	Diferentes unidades de medida	×

A comparação entre os PQD existentes nas taxionomias relacionadas e os que fazem parte da taxionomia proposta permite concluir o seguinte:

- Nem sempre as designações usadas para classificar os PQD são as mesmas. Na taxionomia proposta procurou-se que as designações estivessem de acordo com a terminologia usada na área das BD (*e.g.*: o problema *violação de dependência funcional* corresponde ao problema *valores mutuamente inconsistentes* na taxionomia de [Kim *et al.*, 2003]).
- Alguns problemas incluídos na taxionomia proposta são mais genéricos e abrangentes do que os seus equivalentes nas taxionomias relacionadas. Desta forma, tem cobertura um maior número de situações. (*e.g.*: o problema *violação de sintaxe* engloba o problema *transposição de valores* usado na taxionomia de [Rahm e Do, 2000] – uma violação de sintaxe representa muitas mais situações do que uma mera transposição de valores).

- Todos os PQD existentes nas taxionomias relacionadas possuem um PQD equivalente na taxionomia proposta. Em virtude do seu elevado grau de detalhe, a taxionomia de [Kim *et al.*, 2003] apresenta um número superior de problemas do que a aqui proposta. Contudo, todos os PQD que se encontram nesta taxionomia possuem um equivalente na taxionomia proposta, o que permite concluir que esta última é mais genérica e abrangente. Por exemplo, o PQD *violação de domínio* da taxionomia proposta equivale aos problemas *violação do intervalo de valores* e *violação do conjunto de valores* na taxionomia de [Kim *et al.*, 2003]. Nestes casos, apenas um dos problemas foi apresentado na Tabela 4.2 para não a sobrecarregar excessivamente com informação.
- A taxionomia proposta apresenta novos PQD que não constam nas taxionomias relacionadas, designadamente: *valor incompleto; circularidade entre tuplos num auto-relacionamento*; e, *heterogeneidade de sintaxes*.
- A taxionomia de [Rham e Do, 2000] é a que cobre o maior número de PQD entre as taxionomias relacionadas. Comparativamente a esta, a taxionomia proposta apresenta quatro problemas adicionais: *valor incompleto*; *circularidade entre tuplos num auto-relacionamento*; *existência de homónimos*; e, *heterogeneidade de sintaxes*.

A forma de organização dos PQD na taxionomia proposta é similar à da taxionomia de [Rham e Do, 2000]. Assim, à semelhança da taxionomia proposta, na taxionomia de [Rham e Do, 2000] é feita uma distinção entre PQD mono-fonte e multi-FD. No entanto, ao nível de uma FD não é feita uma separação entre os problemas que ocorrem na relação e entre relações. Nos problemas mono-fonte e multi-fonte, a taxionomia de [Rham e Do, 2000] efectua uma divisão entre os PQD que se relacionam com o esquema dos dados e os que se relacionam com os valores. Na taxionomia proposta nesta dissertação não é feita esta separação, uma vez que está direccionada unicamente para os PQD que ocorrem ao nível dos valores dos dados.

A abordagem usada na elaboração da taxionomia proposta foi exaustiva e sistemática. Ainda assim, não é possível provar formalmente que a taxionomia é completa, *i.e.*, que cobre garantidamente todos os problemas que afectam os dados. No entanto, a comparação com trabalho relacionado evidencia que todos os problemas apresentados nas taxionomias anteriores possuem equivalente na taxionomia proposta. Por outro lado, a taxionomia proposta apresenta novos PQD que não possuem correspondência em nenhuma dessas taxionomias. Estas duas constatações permitem concluir que se trata de uma taxionomia mais completa do que as anteriormente existentes. Isto contribui para aumentar a convicção de que não está a faltar nenhum PQD, *i.e.*, que a taxionomia proposta cobre todos os problemas susceptíveis de afectar os dados.

4.4 Definição Formal dos Problemas de Qualidade dos Dados

À semelhança do efectuado na Secção 4.2, os PQD apresentados nas três taxionomias relacionadas [Rahm e Do, 2000] [Kim *et al.*, 2003] [Muller e Freytag, 2003] são definidos através de descrições textuais. Por mais cuidado que exista na sua elaboração, estas descrições podem suscitar dúvidas quanto ao seu verdadeiro significado, mesmo quando acompanhadas de exemplos. O uso de definições formais apresenta-se como uma solução adequada para a especificação clara e rigorosa dos PQD, não sendo sujeita a ambiguidades de interpretação.

Além da clareza e do rigor, as definições formais são também de maior utilidade, uma vez que contêm informação não existente nas definições textuais. Uma definição formal torna explícito: (*i*) que diz respeito a um determinado tipo de dados (*e.g.*: *string*); (*ii*) os meta-dados necessários à detecção de um determinado problema (*e.g.*: o domínio do atributo); (*iii*) a expressão matemática que define objectivamente o PQD, podendo ser traduzida para linguagem computacional, o que permite automatizar a sua detecção; e, (*iv*) a necessidade eventual de uma função que permita identificar o PQD (*e.g.*: a detecção de um erro ortográfico obriga à existência de uma função de verificação ortográfica).

Nas secções seguintes são apresentadas as definições formais para os PQD identificados em cada NG, de acordo com o proposto em [Oliveira *et al.*, 2005c]. A apresentação das definições formais é antecedida pela exposição da notação adoptada.

4.4.1 Notação

Nesta secção apresenta-se a notação que será usada na definição formal dos PQD. No essencial, esta notação segue a que é apresentada em [Atzeni e Antonellis, 1983]. O *esquema de uma relação* consiste de um nome $\mathcal{R}$ (*i.e.*, o nome da relação) juntamente com uma lista $\mathcal{A} = a_1,a_2,\ldots,a_n$ de nomes de atributos distintos e representa-se por $\mathcal{R}(a_1,a_2,\ldots,a_n)$ ou, simplesmente, por $\mathcal{R}(\mathcal{A})$. O inteiro n representa o *grau do esquema da relação*. Uma FD $\mathcal{F}$ é composta por um conjunto de m esquemas de relação $\mathcal{R}_1(\mathcal{A}_1)$, $\mathcal{R}_2(\mathcal{A}_2),\ldots,$ $\mathcal{R}_m(\mathcal{A}_m)$. Um *domínio* d é um conjunto de valores atómicos. Dado um conjunto $\mathcal{D}$ de domínios e um conjunto $\mathcal{A}$ de nomes de atributos, assume-se que existe um função $\mathcal{D}om: \mathcal{A} \rightarrow \mathcal{D}$ que associa os atributos aos seus domínios. A função $\mathcal{D}om$ aplicada ao conjunto de nomes de atributos é representada por $\mathcal{D}om(\mathcal{A}) = \mathcal{D}om(a_1,a_2,\ldots,a_n) = \mathcal{D}om(a_1) \times \mathcal{D}om(a_2) \times \ldots \times \mathcal{D}om(a_n)$. Uma *instância da relação* (ou, mais simplesmente, *relação*) é um conjunto finito $r \subseteq \mathcal{D}om(a_1) \times \mathcal{D}om(a_2) \times \ldots \times \mathcal{D}om(a_n)$, sendo representado por $r(a_1,a_2,\ldots,a_n)$ ou,

simplesmente, por $r(\mathcal{A})$. Cada elemento de r é designado de *tuplo*, sendo representado por t. Um tuplo t pode ser visto como uma função que associa um valor de $\mathcal{D}om(a_1)$ a a_1, um valor de $\mathcal{D}om(a_2)$ a a_2,… e um valor de $\mathcal{D}om(a_n)$ a a_n. O valor do atributo a no tuplo t é representado por $v(t,a)$. Os valores no tuplo t dos atributos $a_1,a_2,\ldots,a_n$, isto é, $v(t,a_1)$, $v(t,a_2)$,…, $v(t,a_n)$ são representados por $v(t,\mathcal{A})$. Por analogia, os valores que o atributo a assume nos tuplos existentes são representados por $v(\mathcal{T},a)$.

4.4.2 Problemas ao Nível do Atributo

Nesta secção apresenta-se a definição formal dos PQD identificados ao nível do atributo. A forma de apresentação anteriormente usada na Secção 4.2.1 é novamente seguida. Assim, começa-se por expor as definições dos problemas que ocorrem no contexto do valor individual do atributo. De seguida, são expostas as definições dos problemas que resultam do conjunto dos seus valores.

4.4.2.1 Contexto do Valor Individual

De seguida, são apresentadas as definições formais dos problemas que ocorrem no valor individual do atributo:

- **Valor em falta** – Seja $\mathcal{S}$ um conjunto de nomes de atributos definido como: $\mathcal{S} = \{a \mid a \in \mathcal{R}(\mathcal{A}) \wedge a$ é de preenchimento obrigatório$\}$, *i.e.*, $\mathcal{S} \subseteq \mathcal{R}(\mathcal{A})$. Existe um *valor em falta* no atributo $a \in \mathcal{S}$ se e só se (*sse*): $\exists\ t \in r: v(t,a) = null$.
- **Violação de sintaxe** – Seja $\mathcal{G}(a)$ a sintaxe do atributo a, dada por uma gramática ou expressão regular. Seja $\mathcal{L}(\mathcal{G}(a))$ a linguagem gerada pela gramática ou expressão regular. Há uma *violação de sintaxe* no atributo $a \in \mathcal{R}(\mathcal{A})$ *sse*: $\exists\ t \in r: v(t,a) \notin \mathcal{L}(\mathcal{G}(a))$.
- **Erro ortográfico** – Seja *spell* uma função de verificação ortográfica que recebe um *token* e pesquisa a sua existência num dicionário d. Caso a palavra conste do dicionário, a função devolve a própria palavra. No caso contrário, a função efectua uma análise de semelhança e devolve a mais próxima. Caso a análise de semelhança não permita a obtenção de um resultado, a função nada devolve. Seja *tokenize* uma função que recebe um valor e devolve os *tokens* que o compõem. Seja $\mathcal{S}$ o conjunto de *tokens* assim definido: $\mathcal{S} = \{tokenize(v(t,a)) \mid t \in r \wedge a \in \mathcal{R}(\mathcal{A})\}$. Existe um *erro ortográfico* no atributo $a \in \mathcal{R}(\mathcal{A})$ *sse*: $\exists\ s \in S : spell(s, d) \neq s$.
- **Violação de domínio** – Diz-se que há uma *violação de domínio* no atributo $a \in \mathcal{R}(\mathcal{A})$ *sse*: $\exists\ t \in r: v(t,a) \notin \mathcal{D}om(a)$.

Uma violação de domínio num atributo cujo domínio é do tipo enumerado textual, pode corresponder a um:

- **Valor sobrecarregado** – Seja $v'(t,a)$ uma *substring* de $v(t,a)$ e $\mathcal{S}$ um conjunto de nomes de atributos, assim definido: $\mathcal{S} = \{a \mid a \in \mathcal{R}(\mathcal{A}) \wedge \mathcal{D}om(a)$ é enumerado textual$\}$. Existe um *valor sobrecarregado* no atributo $a \in \mathcal{S}$ *sse*: $\exists\, t \in r : v(t,a) \notin \mathcal{D}om(a) \wedge v'(t,a) \in \mathcal{D}om(a)$.
- **Valor incompleto** – Seja $v(t,a)$ uma *substring* de $v'(t,a)$ e $\mathcal{S}$ um conjunto de nomes de atributos, assim definido: $\mathcal{S} = \{a \mid a \in \mathcal{R}(\mathcal{A}) \wedge \mathcal{D}om(a)$ é enumerado textual$\}$. Existe um *valor incompleto* no atributo $a \in \mathcal{S}$ *sse*: $\exists\, t \in r : v(t,a) \notin \mathcal{D}om(a) \wedge v'(t,a) \in \mathcal{D}om(a)$.

4.4.2.2 Contexto Multi-Valor

Seguidamente, apresentam-se as definições formais dos problemas que ocorrem quando se consideram os vários valores de um atributo:

- **Violação de unicidade** – Seja $\mathcal{S}$ o conjunto de nomes de atributos, assim definido: $\mathcal{S} = \{a \mid a \in \mathcal{R}(\mathcal{A}) \wedge a$ é um atributo de valor único$\}$. Existe uma *violação de unicidade* no atributo $a \in \mathcal{S}$ *sse*: $\exists\, t_1, t_2 \in r : v(t_1,a) = v(t_2,a) \wedge t_1 \neq t_2$.
- **Existência de sinónimos –** Seja *synonyms* uma função que recebe dois *tokens* e um dicionário de sinónimos d, verifica se os *tokens* são sinónimos com base no dicionário e devolve um valor lógico (*i.e.*, verdadeiro ou falso). Seja *tokenize* uma função que recebe um valor e devolve os *tokens* que o compõem. Sejam $\mathcal{S}$ e $\mathcal{T}$ conjuntos de *tokens* assim definidos: $\mathcal{S} = \{tokenize(v(t_1,a)) \mid t_1 \in r \wedge a \in \mathcal{R}(\mathcal{A})\}$ e $\mathcal{U} = \{tokenize(v(t_2,a)) \mid t_2 \in r \wedge a \in \mathcal{R}(\mathcal{A})\}$, sendo $t_1 \neq t_2$. Diz-se que se está perante a *existência de sinónimos* no atributo $a \in \mathcal{R}(\mathcal{A})$ *sse*: $\exists\, s \in \mathcal{S}, u \in \mathcal{U} : synonyms(s, u, d) = verdadeiro$.
- **Violação de restrição de integridade** – Seja *check* uma função que recebe os valores do atributo, verifica se uma determinada restrição de integridade é respeitada e devolve um valor lógico. Existe uma *violação de restrição de integridade* no atributo $a \in \mathcal{R}(\mathcal{A})$ *sse*: $check(v(\mathcal{T},a)) = falso$.

  Nota: Tal como definido na Secção 4.4.1, $v(\mathcal{T},a)$ representa o conjunto de valores do atributo a, considerando todos os tuplos existentes.

4.4.3 Problemas ao Nível do Tuplo

A definição formal do PQD identificado ao nível dos valores dos atributos de um tuplo é, a seguir, apresentada:

- **Violação de restrição de integridade** – Seja *check* uma função que recebe um conjunto de valores dos atributos de um tuplo, verifica se uma dada restrição de integridade x é respeitada e devolve um valor lógico. Seja $\mathcal{S}$ o conjunto de nomes de atributos assim definido: $\mathcal{S} = \{a \mid a \in \mathcal{R}(\mathcal{A}) \wedge a$ é usado na formulação de $x\}$, *i.e.*, $\mathcal{S} \subseteq \mathcal{R}(\mathcal{A})$. Existe uma *violação de restrição de integridade* no tuplo $t \in r$ *sse*: $check(v(t,\mathcal{S})) = falso$.

  Nota: Tal como definido na Secção 4.4.1, $v(t,\mathcal{S})$ representa os valores dos atributos do conjunto $\mathcal{S}$ no tuplo t.

4.4.4 Problemas ao Nível da Relação

Nesta secção apresentam-se as definições formais dos PQD identificados ao nível da relação, *i.e.*, que envolvem os valores de vários atributos, em múltiplos tuplos de uma relação.

- **Violação de dependência funcional** – Seja a_2 um atributo cujos valores dependem funcionalmente dos valores de outros atributos. O conjunto de nomes destes atributos é definido como: $\mathcal{S} = \{a_1 \mid a_1, a_2 \in \mathcal{R}(\mathcal{A}) \wedge$ valor de a_2 depende funcionalmente do valor de $a_1\}$. Existe uma *violação de dependência funcional* na relação r *sse*: $\exists\ t_1, t_2 \in r\ \forall\ a_1 \in \mathcal{S} : v(t_1,a_1) = v(t_2,a_1) \wedge v(t_1,a_2) \neq v(t_2,a_2)$.

- **Circularidade entre tuplos num auto-relacionamento** – Sejam $\mathcal{S}$ e $\mathcal{T}$ conjuntos de nomes de atributos, assim definidos: $\mathcal{S} = \{a \mid a \in \mathcal{R}(\mathcal{A}) \wedge a$ faz parte da chave primária$\}$ e $\mathcal{T} = \{a \mid a \in \mathcal{R}(\mathcal{A}) \wedge a$ faz parte da chave estrangeira que define o auto-relacionamento$\}$. Seja $\mathcal{U}$ o conjunto que contém os valores das chaves primárias de todos os tuplos existentes na relação r, assim definido: $\mathcal{U} = \{v(t, \mathcal{S}) \mid t \in r\}$. Seja u o valor de uma chave primária: $u \in \mathcal{U}$. Seja $\mathcal{V}$ o conjunto que, começando no tuplo identificado pela chave primária u, contém os valores das chaves estrangeiras de todos os outros tuplos com os quais há auto-relacionamentos, assim definido: $\mathcal{V} = \{ v(t_1, \mathcal{T}) \mid v(t_1, \mathcal{S}) = v(t_2, \mathcal{T}) \wedge t_1, t_2 \in r\}$. Existe *circularidade entre tuplos num auto-relacionamento* na relação r *sse*: $u \in \mathcal{V}$.

- **Tuplos duplicados** – Seja $\mathcal{S}$ o conjunto de nomes de atributos assim definido: $\mathcal{S} = \{a \mid a \in \mathcal{R}(\mathcal{A}) \wedge a$ é usado na identificação de duplicados$\}$, *i.e.*, $\mathcal{S} \subseteq \mathcal{R}(\mathcal{A})$. Seja $similarity_s$ a função que determina a semelhança entre dois valores do atributo $s \in S$, devolvendo-a sob a forma de um número real entre 0 e 1. Seja θ_s um número real compreendido entre 0 e 1, representativo do limiar a partir do qual se considera que os valores do atributo $s \in S$ são equivalentes. Existem *tuplos duplicados* na relação r *sse*: $\exists\ t_1, t_2 \in r\ \forall\ s \in \mathcal{S} : v(t_1,a_1) = v(t_2,a_2) \vee similarity_s(v(t_1,a),v(t_2,a)) \geq \theta_s \wedge t_1 \neq t_2$.

  A classificação de dois tuplos t_1 e t_2 ($t_1,t_2 \in r$) como sendo duplicados pode ainda ser mais detalhada de acordo com o que se apresenta de seguida. Seja $\mathcal{U}$ o conjunto de nomes de atributos assim definido: $\mathcal{U} = \{a \mid a \in \mathcal{R}(\mathcal{A}) \wedge a$ não faz parte da chave primária$\}$.

- **Duplicados iguais** – Diz-se que os tuplos são *duplicados iguais sse*: $\forall\ a \in \mathcal{U} : v(t_1,a) = v(t_2,a)$.
- **Duplicados aproximados** – Caso não sejam iguais, diz-se que são *duplicados aproximados sse*: $\forall\ a \in \mathcal{U} : v(t_1,a) = v(t_2,a) \vee similarity_s(v(t_1,a),v(t_2,a)) \geq \theta_s$
- **Duplicados inconsistentes** – Caso as anteriores situações não se verifiquem, diz-se que os tuplos são *duplicados inconsistentes* dado que: $\exists\ a \in \mathcal{U} : similarity_s(v(t_1,a),v(t_2,a)) < \theta_s$

- **Violação de restrição de integridade** – Seja *check* uma função que recebe valores de atributos de todos os tuplos da relação, verifica se uma determinada restrição x é respeitada e devolve um valor lógico. Seja $\mathcal{S}$ o conjunto de nomes de atributos assim definido: $\mathcal{S} = \{a \mid a \in \mathcal{R}(\mathcal{A}) \wedge a$ é usado na formulação de $x\}$, *i.e.*, $\mathcal{S} \subseteq \mathcal{R}(\mathcal{A})$. Existe uma *violação de restrição de integridade* na relação r *sse*: $check(v(\mathcal{T},\mathcal{S})) = falso$.

  Nota: $v(\mathcal{T},\mathcal{S})$ representa os valores dos atributos que pertencem a $\mathcal{S}$, em todos os tuplos da relação.

4.4.5 Problemas ao Nível de Múltiplas Relações/Fontes de Dados

Nesta secção são apresentadas as definições formais dos PQD identificados ao nível de múltiplas relações, quer estas pertençam a uma ou mais FD.

- **Heterogeneidade de sintaxes** – Seja $\mathcal{G}(a)$ a sintaxe do atributo a, dada por uma gramática ou expressão regular. Sejam a_1 e a_2 os nomes de dois atributos representativos da mesma propriedade do mundo real. Existe uma *heterogeneidade de sintaxes* entre as relações r_1 e r_2 *sse*: $\exists\ a_1 \in \mathcal{R}_1(\mathcal{A}_1), a_2 \in \mathcal{R}_2(\mathcal{A}_2) : \mathcal{G}(a_1) \neq \mathcal{G}(a_2)$.
- **Heterogeneidade de unidades de medida** – Seja *convert* uma função que recebe um valor na unidade de medida x, efectua a sua conversão para a unidade de medida y e devolve este resultado. Sejam a_1 e a_2 os nomes de dois atributos representativos da mesma propriedade do mundo real. Sejam t_1 e t_2 dois tuplos representativos da mesma entidade do mundo real. Existe *heterogeneidade de unidades de medida* entre as relações r_1 e r_2 *sse*: $\forall\ t_1 \in r_1\ \exists\ a_1 \in \mathcal{R}_1(\mathcal{A}_1), a_2 \in \mathcal{R}_2(\mathcal{A}_2), t_2 \in r_2 : convert(v(t_1,a_1)) = v(t_2,a_2)$.
- **Existência de sinónimos –** Seja *synonyms* uma função que recebe dois *tokens* e um dicionário de sinónimos d, verifica se os *tokens* são sinónimos com base no dicionário e devolve um valor lógico. Seja *tokenize* uma função que recebe um valor e devolve os *tokens* que o compõem. Sejam a_1 e a_2 os nomes de dois atributos representativos da mesma propriedade do mundo real. Sejam $\mathcal{S}$ e $\mathcal{T}$ conjuntos de *tokens* assim definidos: $\mathcal{S} = \{tokenize(v(t_1,a_1)) \mid t_1 \in r_1 \wedge a_1 \in \mathcal{R}_1(\mathcal{A}_1)\}$ e $\mathcal{U} = \{tokenize(v(t_2,a_2)) \mid t_2 \in r_2 \wedge a_2 \in$

$\mathcal{R}_2(\mathcal{A}_2)\}$. Diz-se que se está perante a *existência de sinónimos* entre as relações r_1 e r_2 *sse*: $\exists\ s \in \mathcal{S}, u \in \mathcal{U}$: *synonyms*(s, u, d) = *verdadeiro*.

- **Existência de homónimos –** Sejam a_1 e a_2 os nomes de dois atributos representativos da mesma propriedade do mundo real. Diz-se que se está perante a *existência de homónimos* entre as relações r_1 e r_2 *sse*: $\exists\ t_1 \in r_1, t_2 \in r_2, a_1 \in \mathcal{R}_1(\mathcal{A}_1), a_2 \in \mathcal{R}_2(\mathcal{A}_2) : v(t_1,a_1) = v(t_2,a_2)$.

- **Diferentes granularidades de representação –** Sejam a_1 e a_2 os nomes de dois atributos representativos da mesma propriedade do mundo real. Sejam $\mathcal{S}$ e $\mathcal{T}$ conjuntos que contêm os valores distintos de cada um dos atributos, assim definidos: $\mathcal{S} = \{v(t_1,a_1) \mid t_1 \in r_1, a_1 \in \mathcal{R}_1(\mathcal{A}_1)\}$ e $\mathcal{T} = \{v(t_2,a_2) \mid t_2 \in r_2, a_2 \in \mathcal{R}_2(\mathcal{A}_2)\}$. Existem *diferentes granularidades de representação* entre as relações r_1 e r_2 *sse*: $|\mathcal{S}| \neq |\mathcal{T}|$.

  Nota: $|\mathcal{S}|$ e $|\mathcal{T}|$ representam, respectivamente, as cardinalidades dos conjuntos $\mathcal{S}$ e $\mathcal{T}$.

- **Violação de integridade referencial** – Considere-se a existência de um relacionamento entre os esquemas $\mathcal{R}_1(\mathcal{A}_1)$ e $\mathcal{R}_2(\mathcal{A}_2)$. Sejam $\mathcal{S}$ e $\mathcal{T}$ conjuntos de nomes de atributos assim definidos: $\mathcal{S} = \{a \mid a \in \mathcal{R}_1(\mathcal{A}_1) \wedge a$ faz parte da chave primária$\}$ e $\mathcal{T} = \{a \mid a \in \mathcal{R}_2(\mathcal{A}_2) \wedge a$ faz parte da chave estrangeira que estabelece o relacionamento$\}$. Seja $\mathcal{V}$ o conjunto dos valores da chave primária de todos os tuplos existentes, assim definido: $\mathcal{V} = \{v(t,\mathcal{S}) \mid t \in r_1\}$. Existe uma *violação de integridade referencial* entre as relações r_1 e r_2 *sse*: $\exists\ t \in r_2 : v(t,\mathcal{T}) \notin \mathcal{V}$.

- **Tuplos duplicados –** Sejam a_1 e a_2 os nomes de dois atributos representativos da mesma propriedade do mundo real. Seja $\mathcal{S}$ o conjunto de nomes de atributos assim definido: $\mathcal{S} = \{(a_1, a_2) \mid a_1 \in \mathcal{R}_1(\mathcal{A}_1) \wedge a_2 \in \mathcal{R}_2(\mathcal{A}_2) \wedge a_1, a_2$ são usados na identificação de duplicados$\}$. Seja $similarity_{(a_1,a_2)}$ a função que determina a semelhança entre dois valores do par de atributos $(a_1, a_2) \in S$, devolvendo-a sob a forma de um número real entre 0 e 1. Seja $\theta_{(a_1,a_2)}$ um número real compreendido entre 0 e 1, representativo do limiar a partir do qual se considera que os valores do par de atributos $(a_1, a_2) \in S$ são equivalentes. Existem *tuplos duplicados* entre as relações r_1 e r_2 *sse*: $\exists\ t_1 \in r_1, t_2 \in r_2\ \forall\ (a_1, a_2) \in \mathcal{S} : v(t_1,a_1) = v(t_2,a_2) \vee similarity_{(a_1,a_2)}(v(t_1,a_1),v(t_2,a_2)) \geq \theta_{(a_1,a_2)}$.

  A classificação de dois tuplos t_1 ($t_1 \in r_1$) e t_2 ($t_2 \in r_2$) como sendo duplicados pode ainda ser mais detalhada, de acordo com o que a seguir se apresenta. Sejam $\mathcal{U}$ e $\mathcal{V}$ os conjuntos de nomes de atributos assim definidos: $\mathcal{U} = \{a \mid a \in \mathcal{R}_1(\mathcal{A}_1) \wedge a$ não faz parte da chave primária$\}$ e $\mathcal{V} = \{a \mid a \in \mathcal{R}_2(\mathcal{A}_2) \wedge a$ não faz parte da chave primária$\}$.

  - **Duplicados iguais** – Diz-se que os tuplos são *duplicados iguais sse*: $\forall\ a_1 \in \mathcal{U}\ \exists\ a_2 \in \mathcal{V}$: $v(t_1,a_1) = v(t_2,a_2)$.
  - **Duplicados aproximados** – Caso não sejam iguais, diz-se que são *duplicados aproximados sse*: $\forall\ a_1 \in \mathcal{U}\ \exists\ a_2 \in \mathcal{V}$: $v(t_1,a_1) = v(t_2,a_2) \vee similarity_{(a_1,a_2)}(v(t_1,a_1),v(t_2,a_2)) \geq \theta_{(a_1,a_2)}$.

- **Duplicados inconsistentes** – Caso as anteriores situações não se verifiquem, diz-se que os tuplos são *duplicados inconsistentes* uma vez que: $\exists\ a_1 \in \mathcal{U},\ a_2 \in \mathcal{V}$: $similarity_{(a_1,a_2)}(v(t_1,a_1),v(t_2,a_2)) < \theta_{(a_1,a_2)}$.

  Nota: Relembre-se que a_1 e a_2 representam nomes de dois atributos referentes à mesma propriedade do mundo real.

- **Violação de restrição de integridade** – Seja *check* uma função que recebe os valores dos atributos dos tuplos das relações r_1 e r_2, verifica se uma dada restrição de integridade x é respeitada e devolve um valor lógico. Sejam $\mathcal{S}$ e $\mathcal{T}$ conjuntos de nomes de atributos assim definidos: $\mathcal{S} = \{a \mid a \in \mathcal{R}_1(\mathcal{A}_1) \wedge a$ é usado na formulação de $x\}$ e $\mathcal{T} = \{a \mid a \in \mathcal{R}_2(\mathcal{A}_2) \wedge a$ é usado na formulação de $x\}$. Sejam $\mathcal{U}$ e $\mathcal{V}$ conjuntos que contêm os valores dos atributos de todos os tuplos das relações r_1 e r_2, assim definidos: $\mathcal{U} = \{v(t_1,\mathcal{S}) \mid t_1 \in r_1\}$ e $\mathcal{V} = \{v(t_2,\mathcal{T}) \mid t_2 \in r_2\}$. Existe uma *violação de restrição de integridade* entre as relações r_1 e r_2 *sse*: $check(\mathcal{U}, \mathcal{V}) = falso$.

4.5 Cobertura das Ferramentas de Limpeza de Dados

Tal como referido na introdução deste capítulo, um dos motivos pelo qual a taxionomia é importante advém desta permitir a avaliação da cobertura das ferramentas de qualidade dos dados. Uma vez que o trabalho efectuado no âmbito desta dissertação se centra na área da LD, nesta secção é efectuada uma análise à cobertura que as soluções existentes dão aos PDQ. O objectivo é aferir até que ponto as actuais soluções suportam os PQD que fazem parte da taxionomia, a nível de DC.

Fundamentalmente, as soluções existentes subdividem-se em dois grandes grupos: protótipos de investigação e ferramentas comerciais. Os protótipos de investigação são originários da comunidade científica/académica, enquanto que as ferramentas comerciais são produzidas por organizações comerciais/empresariais que desenvolvem software nesta área. A cobertura dada pelas soluções de maior representatividade de cada um dos grupos, a nível de DC, é apresentada nas duas secções seguintes. Em cada secção, uma tabela efectua o cruzamento entre os PQD e as soluções de LD consideradas. Os significados dos acrónimos a usar nas tabelas são: OBS – Operação Baseada em *Structured Query Language* (SQL); FDU – Função Definida pelo Utilizador; OP – Operação Predefinida; e, NS – Não Suportado. O significado de cada um destes tipos diferentes de cobertura de detecção/correcção é a seguir clarificado:

- **Operação baseada em SQL** – O utilizador necessita de especificar uma operação que envolve a escrita de código SQL.
- **Função definida pelo utilizador** – O utilizador tem de implementar uma função que suporte o problema.
- **Operação predefinida –** O problema é suportado de base ou raiz. Nalguns casos é necessário apenas que o utilizador forneça os parâmetros.

4.5.1 Protótipos de Investigação

Na Tabela 4.3 apresenta-se a cobertura dada aos PQD pelos cinco protótipos de investigação apresentados na Secção 3.5.1 do Capítulo 3, cujas designações são: *Ajax* [Galhardas *et al.*, 2001]; *Arktos II* [Vassiliadis *et al.*, 2003]; *IntelliClean* [Low *et al.*, 2001]; *FraQL* [Sattler e Schallehn, 2001]; e, *Potter's Wheel* [Raman e Hellerstein, 2001].

Tabela 4.3 – Cobertura dada pelos protótipos de investigação de LD aos PQD

PQD	Tipo de Operação	*Ajax*	*Arktos II*	*Intelli Clean*	*FraQL*	*Potter's Wheel*
Valor em falta	**Detecção**	OBS	OP	NS	OBS	OP
	Correcção	OBS/FDU	FDU	FDU	OBS	NS
Violação de sintaxe	**Detecção**	OBS/FDU	OP	NS	NS	OP
	Correcção	FDU	OP	FDU	OP/ FDU	OP
Erro ortográfico	**Detecção**	FDU	FDU	NS	NS	OP
	Correcção	FDU	FDU	FDU	NS	NS
Violação de domínio	**Detecção**	OBS	OP	NS	NS	FDU
	Correcção	OBS/FDU	FDU	FDU	OP/ FDU	NS
Violação de unicidade	**Detecção**	OBS	OP	NS	NS	FDU
	Correcção	OBS/FDU	FDU	NS	NS	NS
Existência de sinónimos	**Detecção**	FDU	FDU	NS	NS	FDU
	Correcção	FDU	FDU	FDU	NS	NS
Violação de dependência funcional	**Detecção**	OBS	FDU	OP	NS	FDU
	Correcção	FDU	FDU	NS	NS	NS
Circularidade entre tuplos num auto-relac.	**Detecção**	FDU	FDU	NS	NS	NS
	Correcção	FDU	FDU	NS	NS	NS

Tabela 4.3 – Cobertura dada pelos protótipos de investigação de LD aos PQD (continuação)

PQD	Tipo de Operação	*Ajax*	*Arktos II*	*Intelli Clean*	*FraQL*	*Potter's Wheel*
Tuplos duplicados	**Detecção**	OBS/FDU	FDU	OP/ FDU	OBS/ FDU	NS
	Correcção	OBS/FDU	FDU	OP/ FDU	OBS/ FDU	NS
Existência de homónimos	**Detecção**	OBS	FDU	NS	NS	NS
	Correcção	FDU	FDU	NS	NS	NS
Dif. granularidades de representação	**Detecção**	OBS/FDU	FDU	NS	NS	FDU
	Correcção	OBS/FDU	FDU	FDU	NS	NS
Violação de integridade referencial	**Detecção**	OBS	OP	NS	NS	NS
	Correcção	FDU	FDU	NS	NS	NS
Heterogeneidade de sintaxes	**Detecção**	FDU	FDU	NS	NS	NS
	Correcção	FDU	FDU	NS	NS	NS
Heterogeneidade de unidades de medida	**Detecção**	FDU	FDU	NS	NS	NS
	Correcção	FDU	FDU	NS	NS	NS

A análise da tabela permite alcançar as seguintes conclusões:

- A cobertura dada aos PQD pelos protótipos de investigação com base em operações predefinidas é reduzida. De acordo com este critério, o protótipo que dá maior cobertura é o *Arktos II*. Mesmo assim, são somente cinco os PQD que se encontram cobertos, a generalidade dos quais apenas ao nível da detecção. Constata-se que nenhum dos protótipos suporta de base a DC de um número apreciável de problemas.
- Em três dos protótipos, designadamente no *IntelliClean*, no *FraQL* e no *Potter's Wheel* há um conjunto considerável de problemas que não possui sequer suporte, nem a nível de detecção nem a nível de correcção. Apesar de serem ferramentas de LD de carácter genérico, uma vez que o seu âmbito não se encontra limitado a um só problema, a sua abrangência acaba por ser limitada.
- Todos os protótipos se socorrem das suas potencialidades de extensibilidade para cobrirem um maior leque de problemas, suportando funções definidas pelo utilizador. Os dois protótipos que mais seguem esta abordagem são o *Ajax* e o *Arktos II*. No caso específico do *Ajax* nem sequer são disponibilizadas operações predefinidas. A DC dos PQD são efectuadas maioritariamente por funções definidas pelo utilizador. Alguns problemas podem ser detectados e corrigidos por intermédio de operações baseadas em SQL que o utilizador também tem de especificar. Apesar do caracter genérico e flexível,

uma solução que centrada nas funções definidas pelo utilizador apresenta inconvenientes, designadamente:

- Obriga a que seja efectuado um esforço na sua implementação, o que significa dispêndio de valores monetários.
- As funções têm que ser desenvolvidas por utilizadores altamente especializados (*i.e.*, programadores), impedindo que a LD possa ser realizada por pessoal menos especializado.
- O desenvolvimento de certas funções não é trivial (*e.g.*: detecção da existência de circularidade entre tuplos num auto-relacionamento), podendo não ser adoptada a melhor forma de o fazer, o que se reflecte negativamente no desempenho da LD.

4.5.2 Ferramentas Comerciais

Na Tabela 4.4 apresenta-se a cobertura dada aos PQD que fazem parte da taxionomia pelas cinco ferramentas comerciais apresentados na Secção 3.5.2 do Capítulo 3, cujas designações são: *WinPure Clean and Match 2007* [WinPure, 2007]; *ETI Data Cleanser* [ETI, 2007a]; *Trillium Quality* [Trillium Software, 2007a]; *dfPower Quality* [DataFlux, 2007a]; e, *HIQuality* [Human Inference, 2007a]. Esta última ferramenta é altamente modular, tendo sido considerados os seguintes módulos na análise efectuada: *Cleanse* (*Name* e *Address*); *Identify*; e, *Merge*. A identificação da cobertura ou não dos PQD pelas ferramentas foi efectuada com base na informação técnica disponível nos *sites* das empresas proprietárias e, nos casos em que foi possível, na própria experimentação das versões de demonstração.

Tabela 4.4 – Cobertura dada por ferramentas comerciais de LD aos PQD

PQD	Tipo de Operação	*WinPure*	*ETI Data Cleanser*	*Trillium Quality*	*dfPower Quality*	*HI Quality*
Valor em falta	**Detecção**	OP	NS	OP[1]	NS	NS
	Correcção	NS	NS	OP[1]	NS	NS
Violação de sintaxe	**Detecção**	OP[2]	NS	NS	NS	OP
	Correcção	OP[2]	OP	OP	OP	OP
Erro ortográfico	**Detecção**	NS	NS	NS	NS	NS
	Correcção	NS	NS	NS	NS	NS
Violação de domínio	**Detecção**	OP[3]	OP[4,5]	OP[4,6,7]	OP[4]	OP[4,5,8,9]
	Correcção	OP[3]	OP[10,11]	OP[7,10]	OP[10]	OP[4,5,10,11]
Violação de unicidade	**Detecção**	OP	NS	NS	NS	NS
	Correcção	NS	NS	NS	NS	NS

Tabela 4.4 – Cobertura dada por ferramentas comerciais de LD aos PQD (continuação)

PQD	Tipo de Operação	*WinPure*	*ETI Data Cleanser*	*Trillium Quality*	*dfPower Quality*	*HI Quality*
Existência de sinónimos	**Detecção**	NS	NS	NS	NS	NS
	Correcção	OP	OP	OP	OP	OP
Violação de restrição de integridade	**Detecção**	NS	NS	NS	NS	NS
	Correcção	NS	NS	NS	NS	NS
Violação de dependência funcional	**Detecção**	NS	NS	NS	NS	NS
	Correcção	NS	NS	NS	NS	NS
Circularidade entre tuplos num auto-relac.	**Detecção**	NS	NS	NS	NS	NS
	Correcção	NS	NS	NS	NS	NS
Tuplos duplicados	**Detecção**	OP[12]	OP	OP	OP	OP[12]
	Correcção	OP[12]	OP	OP	OP	OP[12]
Existência de homónimos	**Detecção**	NS	NS	NS	NS	NS
	Correcção	NS	NS	NS	NS	NS
Dif. granularidades de representação	**Detecção**	NS	NS	NS	NS	NS
	Correcção	NS	NS	NS	NS	NS
Violação de integridade referencial	**Detecção**	NS	NS	NS	NS	NS
	Correcção	NS	NS	NS	NS	NS
Heterogeneidade de sintaxes	**Detecção**	NS	NS	NS	NS	NS
	Correcção	NS	NS	NS	NS	NS
Heterogeneidade de unidades de medida	**Detecção**	NS	NS	NS	NS	NS
	Correcção	NS	NS	NS	NS	NS

Notas:

1. Apenas em atributos que armazenam elementos referentes a um endereço postal (*e.g.*: nome da rua; código postal).
2. Apenas algumas operações sintácticas simples, especialmente em atributos que armazenam endereços de *e-mail*.
3. Apenas em atributos que armazenam endereços de *e-mail*.
4. Inclui potencialidade de detecção de valores sobrecarregados.
5. Inclui potencialidade de detecção de valores incompletos.
6. Com especial ênfase em atributos que armazenam nomes de pessoas.

7. Com especial ênfase em atributos que armazenam endereços postais.
8. Apenas atributos que armazenam nomes de pessoas.
9. Apenas atributos que armazenam endereços postais.
10. Inclui potencialidades de correcção de valores sobrecarregados.
11. Inclui potencialidades de correcção de valores incompletos.
12. Apenas permite efectuar a manipulação de duplicados ao nível de uma só relação.

A análise da tabela anterior permite a obtenção das seguintes conclusões:

- O leque de problemas da taxionomia coberto pelas ferramentas comerciais analisadas é reduzido. Claramente, o número de problemas que não possui suporte é bem maior do que o número de problemas que possui.
- Essencialmente, os problemas cobertos pelas ferramentas analisadas são os mesmos. Certamente isto resulta da concorrência que existe entre as empresas que desenvolvem soluções nesta área, nomeadamente em fornecerem nas suas soluções as mesmas potencialidades do que as existentes nas soluções concorrentes. Esta constatação permite inferir que soluções de outras empresas darão aproximadamente a mesma cobertura aos PQD da taxionomia.
- Nalguns PQD o suporte que é dado a nível de DC encontra-se limitado a dados de um determinado tipo (*e.g.*: nomes de pessoas; endereços postais; endereços de *e-mail*) pelo que se pode considerar que este é apenas parcial.
- Nestas ferramentas não existem potencialidades de extensibilidade, *i.e.*, de incorporarem funções definidas pelo utilizador para a detecção ou correcção de PQD específicos. Nem sequer é possível ao utilizador a execução de instruções SQL. As ferramentas analisadas estão muito orientadas para o utilizador não perito.
- O suporte dado à correcção dos PQD é maior do que o dado à detecção. A generalidade das ferramentas analisadas (a excepção é a ferramenta *WinPure*) faz parte de um pacote que integra outras ferramentas (*e.g.*: *data profiling*; enriquecimento de dados) que, em conjunto, visam a melhoria da qualidade dos dados. As ferramentas de *data profiling* geram automaticamente um conjunto de resultados que permitem a detecção de vários PQD. É por este motivo que a detecção de certos problemas, mesmo triviais (*e.g.*: valor em falta), não é suportada nas ferramentas analisadas. Importa então conhecer a cobertura que as ferramentas comerciais de *data profiling* dão à detecção dos PQD que fazem parte da taxionomia. Esta cobertura é apresentada na secção seguinte.

4.6 Cobertura das Ferramentas de Data Profiling

Na Tabela 4.5 apresentam-se os resultados de cinco ferramentas comerciais, designadamente: *Datiris Profiler* [Datiris, 2007]; *ETI Data Profiler* [ETI, 2007b]; *Trillium Discovery* [Trillium Software, 2007b]; *dfPower Profile* [DataFlux, 2007b]; e, *HIQuality Inspect* [Human Inference, 2007b]. Na tabela, estas ferramentas surgem representadas, respectivamente, pelas letras A; B; C; D; e E. À excepção da primeira, todas as outras ferramentas de *data profiling* pertencem, respectivamente, aos mesmos pacotes das ferramentas de LD: *ETI Data Cleanser*, *Trillium Quality*, *dfPower Quality* e *HIQuality*. A tabela foi elaborada com base na informação técnica sobre as ferramentas, disponível nos *sites* das empresas proprietárias. Nos casos em que foi possível, a experimentação das versões de demonstração também contribuiu para a elaboração da tabela.

Tabela 4.5 – Cobertura dada por ferramentas comerciais de *data profiling* aos PQD

PQD	NG				
	Atributo		Tuplo	Relação	Múltiplas relações/ FD
	Valor individual	Multi-valor			
Valor em falta	A B C D E				
Violação de sintaxe	A B C D E				
Erro ortográfico	×				
Violação de domínio	A B C D E				
Violação de unicidade		A B C D E			
Existência de sinónimos		×			×
Violação de restrição de integridade		B[1] C[1] D[3]	B[1] C[1] D[3]	B[1] C[1] D[3]	B[1,2] C[1] D[3]
Violação de dependência funcional				B C	
Tuplos duplicados				×	×
Circularidade entre tuplos num auto-relacio.				×	

Tabela 4.5 – Cobertura dada por ferramentas comerciais de *data profiling* aos PQD (continuação)

PQD	NG				
	Atributo		Tuplo	Relação	Múltiplas relações/ FD
	Valor individual	Multi-valor			
Existência de homónimos					×
Violação de integridade referencial					B[4] C D E[4]
Dif. granularidades de representação					×
Heterogeneidade de sintaxes					×
Heterogeneidade de unidades de medida					×

Notas:

1. Compete ao utilizador definir a restrição de integridade, sendo identificadas as violações que a esta ocorrem.
2. Não suporta a detecção de violações de restrições de integridade que envolvam múltiplas FD.
3. A ferramenta disponibiliza de base um conjunto simples de restrições de integridade, competindo ao utilizador definir as regras de maior complexidade. As violações às restrições de integridade são assinaladas.
4. Não suporta a detecção de violações de integridade referencial que envolvam relações em múltiplas FD.

A análise da tabela anterior suscipta as seguintes considerações:

- Na generalidade, as potencialidades de detecção de PQD das ferramentas de *data profiling* analisadas são aproximadamente as mesmas. A concorrência faz com que as empresas procurem oferecer as mesmas potencialidades que as suas rivais oferecem. A única excepção é a ferramenta *Datiris Profiler* que possui mais limitações.
- Há diversos PQD que as ferramentas de *data profiling*, fruto da sua natureza intrínseca (*e.g.*: produzir meta-dados com base nos dados existentes), não têm capacidade de detectar (*e.g.*: existência de sinónimos). Na realidade, são mais os problemas que não têm cobertura do que aqueles que têm. Os problemas que não têm cobertura manifestam-se especialmente nos níveis multi-relação de uma ou múltiplas FD.

- Alguns PQD encontram-se cobertos pelas ferramentas de *data profiling* de forma indirecta. A determinação automática dos meta-dados com base nos dados existentes contribui para a identificação dos problemas, mas não os detecta por si só. Por exemplo, a determinação automática do valor máximo e mínimo dos atributos permite ao utilizador identificar violações de domínio. No entanto, localizar os tuplos onde estas de facto ocorrem é da responsabilidade do utilizador. Em certos PQD, as potencialidades de navegação até aos tuplos (*i.e.*, de *drill-down*) oferecidas pelas ferramentas de *data profiling* simplificam esta tarefa.

4.7 Conclusão

Neste capítulo foram identificados exaustivamente os diversos problemas de qualidade que afectam dados relacionais. A identificação foi efectuada por NG onde estes ocorrem, designadamente: atributo; tuplo; relação; múltiplas relações; e, múltiplas FD. A análise realizada seguiu uma aproximação ascendente (*bottom-up*), desde o NG mais elementar (*i.e.*, atributo) até ao de maior complexidade (*i.e.*, múltiplas FD), de que resultou uma taxionomia de PQD. Os critérios que conduziram à identificação dos PQD em cada NG foram devidamente apresentados. A comparação com trabalho relacionado evidencia que a taxionomia desenvolvida cobre um conjunto maior de problemas. No âmbito deste trabalho de doutoramento, o facto de ser mais completa assume especial importância, uma vez que se pretende desenvolver um modelo de LD que suporte a DC automáticas do maior número possível de PQD. A aproximação usada na identificação dos problemas foi rigorosa e sistemática, o que sustenta a convicção de que não está a faltar qualquer PQD.

Os PDQ que fazem parte da taxionomia foram também especificados através de definições matemáticas. Esta característica diferencia a taxionomia das anteriormente propostas, uma vez que nestas apenas são usadas definições textuais. A definição formal dos PQD revela-se importante pelos seguintes motivos: (*i*) é a única forma de garantir uma definição clara e precisa do significado de cada PQD; (*ii*) é útil porque define tudo aquilo que é necessário para detectar automaticamente o problema, *i.e.*: (*a*) os meta-dados necessários; (*b*) a expressão matemática que define o PQD e que pode ser entendida como a regra que permite automatizar a sua detecção; e, (*c*) uma eventual função necessária à detecção. Todos estes elementos encontram-se explícitos na definição de cada PQD.

A taxionomia foi usada para avaliar a cobertura que as actuais soluções de LD dão à DC dos PQD. Dois grupos distintos de soluções foram analisados: protótipos de investigação e

ferramentas comerciais. Em cada um dos grupos foram seleccionadas as soluções de maior representatividade. Concluiu-se que a cobertura dada de raíz pelos protótipos de investigação ao conjunto de PQD que fazem parte da taxionomia é reduzida. Alguns destes protótipos baseiam-se na inclusão de funções definidas pelo utilizador para cobrirem aquilo que não suportam de base. Ainda que em certos problemas não haja alternativa (*e.g.*: detecção de violação de restrição de integridade), a maioria dos problemas poderia ser suportada de base pelos protótipos, pelo menos a nível de detecção. Nestes casos, a obrigatoriedade do desenvolvimento de funções não é justificável, especialmente pelas dificuldades e custos que lhes estão associados. Concluiu-se igualmente que a cobertura dada à DC dos PQD pelas ferramentas comerciais é diminuta, *i.e.*, não é dado o adequado suporte à limpeza dos problemas incluídos na taxionomia. Como estão muito orientadas para o utilizador não perito, nem sequer permitem a incorporação de funções definidas pelo utilizador. Isto significa que não conseguem cobrir outros problemas para além daqueles que suportam de raiz.

Comercialmente, uma boa parte da responsabilidade de detectar os PQD é colocada nas ferramentas de *data profiling*. No entanto, fruto da natureza destas ferramentas de identificar os PQD com base nos metadados gerados, o leque de problemas coberto acaba por ser reduzido. Quer isto dizer que, mesmo recorrendo a ferramentas de *data profiling*, as potencialidades de detecção dos PQD incluídos na taxionomia são escassas. Como comentário final salienta-se o facto das actuais soluções evidenciarem uma cobertura deficiente, a nível de DC, face ao vasto leque de problemas de qualidade que afectam os dados. No âmbito deste trabalho procura-se explorar e suprir as deficiências de cobertura identificadas.

FORMALIZAÇÃO DAS OPERAÇÕES DE DETECÇÃO 5

Neste capítulo apresenta-se a formalização sintáctica e semântica das operações que conduzem à detecção dos Problemas de Qualidade dos Dados (PQD) que compõem a taxionomia apresentada no capítulo anterior. Assim, começa-se por expor as definições de carácter geral, a nível sintáctico e semântico, necessárias à generalidade das formalizações apresentadas nas secções seguintes. A formalização das Operações de Detecção (OD) é efectuada seguindo a forma de organização dos PQD estabelecida pela taxionomia. Como tal, começa-se pela formalização das OD dos PQD que ocorrem ao nível do atributo. A seguir, formalizam-se as OD ao nível do tuplo e, após estas, as OD ao nível da relação. Por último, são apresentadas as formalizações das OD dos PQD que ocorrem ao nível de múltiplas relações, pertencentes à mesma ou a bases de dados diferentes.

5.1 Introdução

No seguimento do capítulo anterior, no qual se apresentou uma taxionomia de PQD organizada por nível de granularidade, neste capítulo apresenta-se a formalização das operações que permitem a detecção desses problemas. As operações são formalizadas a dois níveis: *sintáctico*, *i.e.*, da linguagem que permite especificar as OD dos diferentes PQD; e, *semântico*, *i.e.*, o modo de operação que conduz à detecção de cada tipo de PQD (*e.g.*: valor em falta; violação de sintaxe; violação de domínio). A formalização das operações que conduzem à detecção dos PQD constitui, precisamente, um dos objectivos deste trabalho, de acordo com o enunciado no Capítulo 1 – Secção 1.3. Na apresentação da formalização semântica de cada OD é adoptada a abordagem que a seguir se descreve. Em primeiro lugar, é efectuada uma descrição textual sucinta e informal da filosofia subjacente à OD. Após esta, é que se apresenta de modo formal a semântica da operação.

As formalizações materializam a filosofia adoptada neste trabalho de fornecer, de raiz, suporte à detecção de todos os PQD em que tal é possível à partida. Naturalmente, as formalizações constituem o necessário suporte para uma posterior implementação e consequente automatização da execução das OD. Assim, a detecção dos PQD não fica excessivamente dependente do utilizador e das funções que este se vê obrigado a definir, em virtude da detecção do PQD em

causa não ser suportada de base. Na grande maioria dos PQD não há justificação para que isto aconteça. Este foi um dos problemas identificados nas actuais soluções vocacionadas para a melhoria da qualidade dos dados (ver Capítulo 1 – Secção 1.2) e que neste trabalho se visa suprir. Estas formalizações foram efectuadas com esse objectivo em mente, *i.e.*, apresentar soluções que suportem a detecção dos PQD que compõem a taxionomia.

Na formalização semântica das OD recorre-se a um misto de *álgebra relacional* [13] e *linguagem algorítmica*. A escolha da álgebra relacional na representação da semântica das operações surgiu com naturalidade, uma vez que estas são executadas num contexto de base de dados relacional. Na especificação das operações, tudo o que envolve acessos a base de dados é representado sob a forma de álgebra relacional. Contudo, na grande maioria das operações a álgebra relacional, por si só, não é suficiente para representar a semântica inerente à detecção do PQD. No geral, a definição da semântica obriga a que esta também seja especificada sob a forma procedimental. Na sua representação adoptou-se linguagem algorítmica (*i.e.*, pseudo-código), uma vez que é simples de compreender e facilmente se traduz numa qualquer linguagem de programação. O objectivo destas formalizações é, precisamente, o de constituir o suporte para uma subsequente implementação das OD.

5.2 Definições Gerais

Nesta secção enunciam-se as definições de carácter geral, a nível sintáctico e semântico, comuns à generalidade das formalizações apresentadas nas secções seguintes para cada PQD que faz parte da taxionomia. Por este motivo, esta secção é de consulta obrigatória ao longo da leitura do capítulo.

5.2.1 Sintaxe da Linguagem de Especificação

A linguagem declarativa que suporta a especificação das OD dos PQD é aqui definida na sua globalidade, tendo sido inicialmente proposta em [Oliveira *et al.*, 2006c]. É com base nesta linguagem que o utilizador define as OD que pretende efectuar. A linguagem segue o espírito da *Structured Query Language* (SQL), sendo definida através da seguinte gramática em BNF estendido:

[13] No Apêndice B desta dissertação é apresentado um breve resumo sobre as operações de álgebra relacional utilizadas nas formalizações apresentadas neste capítulo.

OperaçãoDetecção → DETECT ProblemaQualidadeDados NívelOperação NasColunas
DasTabelas DasBasesDados UsandoColunas UsandoChavesOrdenação
UsandoLimiar UsandoOperações DependentesDe DasTabelasReferência
DasBasesDadosReferência UsandoDomínio UsandoCondição
AgupandoPor UsandoDicionário UsandoMétrica

ProblemaQualidadeDados → NívelValorIndividual | NívelColuna | NívelTuplo | NívelRelação |
NívelMúltiplasRelações

NívelValorIndividual → MISSING-VALUE | SYNTAX-VIOLATION |
DOMAIN-VIOLATION | INCOMPLETE-VALUE |
OVERLOADED-VALUE | MISSPELLING-ERRORS |

NívelColuna → EXISTENCE-OF-SYNONYMS | UNIQUENESS-VIOLATION |
INTEGRITY-CONSTRAINT-VIOLATION

NívelTuplo → INTEGRITY-CONSTRAINT-VIOLATION

NívelRelação → FUNCTIONAL-DEPENDENCY-VIOLATION |
EXISTENCE-OF-CIRCULARITY | DUPLICATE-TUPLES |
INTEGRITY-CONSTRAINT-VIOLATION

NívelMúltiplasRelações → HETEROGENEITY-OF-SYNTAXES |
HETEROGENEITY-OF-MEASURE-UNITS |
EXISTENCE-OF-SYNONYMS |
EXISTENCE-OF-HOMONYMS |
DIFFERENT-GRANULARITY-OF-REPRESENTATION |
REFERENTIAL-INTEGRITY-VIOLATION |
DUPLICATE-TUPLES |
INTEGRITY-CONSTRAINT-VIOLATION

NívelOperação → ε | AT LEVEL OF Nível

Nível → COLUMN | ROW | RELATION | MULTIPLE RELATIONS

NasColunas → ε | ColunasAlvo

ColunasAlvo → ON Coluna RestantesColunas

Coluna → AliasTabelaDaColuna NomeColuna

AliasTabelaDaColuna → ε | identificador "."

NomeColuna → identificador

RestantesColunas → ε | ColunaAdicional^{+}

ColunaAdicional → "," Coluna

DasTabelas → FROM Tabela RestantesTabelas

Tabela → NomeTabela AliasTabela

NomeTabela → identificador

AliasTabela → ε | identificador

RestantesTabelas → ε | TabelaAdicional^{+}

TabelaAdicional → "," Tabela

DasBasesDados → OF BaseDados RestantesBasesDados

BaseDados → identificador

RestantesBasesDados → ε | BaseDadosAdicional^{+}

BaseDadosAdicional → "," BaseDados

UsandoColunas → ε | DefiniçãoColunas

DefiniçãoColunas → USING Coluna RestantesColunas

UsandoChavesOrdenação → ε | ChavesOrdenação

ChavesOrdenação → ChaveOrdenação^{+} TamanhoJanelaComparações

ChaveOrdenação → SORT KEY NumeroInteiro: CaracteresColuna RestantesCaracteresColuna

CaracteresColuna → CHARS NumeroInteiro RestantesPosCaracter OF ATRIBUTE NomeColuna

RestantesPosCaracter → ε | PosiçãoCaracterAdicional^{+}

PosiçãoCaracterAdicional → "," NumeroInteiro

RestantesCaracteresColuna → ε | CaracteresColunaAdicional^{+}

CaracteresColunaAdicional → "+" CaracteresColuna

TamanhoJanelaComparações → WINDOW SIZE: NumeroInteiro

UsandoLimiar → ε | DefiniçãoLimiar

DefiniçãoLimiar → USING THRESHOLD ValorLimiar

UsandoOperações → UsandoAtribuições UsandoFunçãoVoid

UsandoAtribuições → ε | DefiniçãoAtribuições

DefiniçãoAtribuições → SET Atribuição RestantesAtribuições

Atribuição → Variável = LadoDireitoAtribuição

Variável → identificador

LadoDireitoAtribuição → Variável | Constante | ExpressãoAritmética | NomeColuna
FunçãoDefUtilizador | inquérito-sql

Constante → string | valor-numérico

ExpressãoAritmética → ParêntesisEsq Operando Operador Operador ParêntesisDir

ParêntesisEsq → ε | "("

Operando → valor-numérico | Variável | ExpressãoAritmética | NomeColuna

Operador → operação-aritmética

ParêntesisDir → ε | ")"

FunçãoDefUtilizador → NomeFunção(ArgumentoFunção RestantesArgumentosFunção)

ArgumentoFunção → NomeColuna | Variável | Constante

RestantesArgumentosFunção → ε | ArgumentoAdicionalFunção^{+}

ArgumentoAdicionalFunção → "," ArgumentoFunção

RestantesAtribuições → ε | AtribuiçãoAdicional^{+}

AtribuiçãoAdicional → "," Atribuição

UsandoFunçãoVoid → ε | USING FUNCTION FunçãoDefUtilizador

DependentesDe → ε | DefiniçãoColunasDependentes

DefiniçãoColunasDependentes → DEPENDENT ON Coluna RestantesColunas

DasTabelasReferência → ε | DefiniçãoTabelasReferência

DefiniçãoTabelasReferência → DasTabelas

DasBasesDadosReferência → ε | DefiniçãoBasesDadosReferência

DefiniçãoBasesDadosReferência → DasBasesDados

UsandoDomínio → ε | DefiniçãoDomínio

DefiniçãoDomínio → USING DOMAIN NomeDomínio

NomeDomínio → identificador

UsandoCondição → ε | DefiniçãoCondição

DefiniçãoCondição → WHERE Condição

Condição → expressão-where-sql

AgupandoPor → ε | DefiniçãoAgrupamento

DefiniçãoAgrupamento → GROUP BY CritérioAgrupamento FiltrandoPor

CritérioAgrupamento → expressão-group-by-sql

FiltrandoPor → ε | DefiniçãoFiltro

DefiniçãoFiltro → HAVING CritérioFiltragem

CritérioFiltragem → expressão-having-sql

UsandoDicionário → ε | DefiniçãoDicionário

DefiniçãoDicionário → USING DICTIONARY NomeDicionário

NomeDicionário → identificador

UsandoMétrica → ε | DefiniçãoMétrica

DefiniçãoMétrica → USING METRIC NomeMétrica

NomeMétrica → identificador

A linguagem de especificação das OD que acabou de ser definida na sua globalidade será novamente apresentada na Secção 5.3 e seguintes, devidamente instanciada a cada OD em concreto.

5.2.2 Semântica das Operações de Detecção

As definições de carácter específico necessárias à formalização semântica das OD são apresentadas imediatamente antes da formalização do problema a que dizem respeito. Nos parágrafos seguintes apresentam-se apenas as definições de carácter geral usadas em diversas formalizações.

seja $\mathcal{R}$ uma relação com o seguinte esquema: $\mathcal{R}(a_1,\ldots,a_m,a_{m+1},\ldots,a_{m+n})$, em que $a_1,\ldots,a_m$ representam os atributos que formam a sua chave primária.

seja atr_1 um atributo que não pertence à chave primária de $\mathcal{R}$, *i.e.*, $atr_1 \in \{a_{m+1},\ldots,a_{m+n}\}$.

seja $\mathcal{S}$ uma relação com o seguinte esquema: $\mathcal{S}(b_1,\ldots,b_r,b_{r+1},\ldots,b_{r+s})$, em que $b_1,\ldots,b_r$ representam os atributos que formam a sua chave primária.

seja atr_2 um atributo que não pertence à chave primária de $\mathcal{S}$, *i.e.*, $atr_2 \in \{b_{r+1},\ldots,b_{r+s}\}$.

seja p o número de OD que incidem sobre atr_1 e que na *sequência de OD*[14] antecedem uma dada operação em atr_1.

seja $\mathcal{Z}' = \{\mathcal{Z}1_1,\ldots,\mathcal{Z}1_p\}$ o conjunto de p relações que contêm os PQD detectados em atr_1 pelas operações da sequência de detecção que antecedem uma dada operação sobre atr_1.

seja q o número de OD que incidem sobre atr_2 e que na sequência de OD antecedem uma dada operação em atr_2.

seja $\mathcal{Z}'' = \{\mathcal{Z}2_1,\ldots,\mathcal{Z}2_q\}$ o conjunto de q relações que contêm os PQD já detectados em atr_2 pelas operações da sequência de detecção que antecedem uma dada operação sobre atr_2.

seja *existe_relação*(*nome_relação*) uma função que recebe o nome de uma relação, verifica se esta existe fisicamente ou não na base de dados e devolve como resultado um valor lógico.

seja *cond* a condição de selecção opcionalmente existente na OD com o objectivo de delimitar o seu âmbito de aplicação.

---

[14] De acordo com o modelo proposto, na detecção dos PQD as operações especificadas pelo utilizador devem ser executadas segundo uma determinada ordem. Esta ordem é designada de *sequência de detecção*. O modelo defendido para a limpeza de dados será apresentado detalhadamente no Capítulo 7. Por agora, é suficiente ter-se a noção de que as OD são executadas obedecendo a uma sequência preestabelecida.

seja *le_tuplo*($\mathcal{R}$) uma função que lê sequencialmente e devolve o próximo tuplo de $\mathcal{R}$. Quando o final de $\mathcal{R}$ é alcançado (*i.e.*, não existem mais tuplos), a função devolve o valor *null*.

seja *existe_operação_detecção_violação_domínio*(*atr*) uma função que recebe um atributo *atr* e verifica se na sequência de OD existe uma OD de violação de domínio sobre *atr*, devolvendo um valor booleano como resultado.

seja *dimensão*(*vec*[ ]) uma função que recebe um vector *vec*, determina o número de elementos que o compõem e devolve este valor como resultado.

seja *numlinhas*(*mat*[ ][ ]) uma função que recebe uma matriz *mat*, determina e devolve o número de linhas que a compõem.

seja *numcolunas*(*mat*[ ][ ]) uma função que recebe uma matriz *mat*, determina e devolve o número de colunas que a compõem.

seja $fdu(p_1,p_2,\ldots,p_y)$ uma Função Definida pelo Utilizador (FDU) que recebe um conjunto de parâmetros $p_1,p_2,\ldots,p_y$. Cada um destes representa uma constante, o valor de um atributo, o conjunto de valores de um atributo ou uma variável que contém o resultado de uma função anterior. Como resultado, a função retorna um valor do tipo atómico ou estruturado (*i.e.*, um vector ou uma matriz).

seja $avalia\_condição(p_1,p_2,\ldots,p_z)$ uma função que recebe um conjunto de parâmetros $p_1,p_2,\ldots,p_z$ necessários à avaliação da condição que permite detectar o PQD em questão. Estes parâmetros representam uma constante, o valor de um atributo, o conjunto de valores de um atributo ou uma variável que contém o resultado de uma função anterior. Um destes parâmetros é necessariamente a variável que contém o resultado devolvido pela última função da sequência de FDU. A função efectua a avaliação lógica da condição e retorna como resultado o valor booleano correspondente.

seja "o" o operador de concatenação de strings.

seja "like" o operador que verifica se uma dada string obedece a um padrão especificado pelo utilizador, tal como vulgarmente utilizado em SQL.

Para não tornar as formalizações expostas nas secções seguintes excessivamente extensas, optou-se por efectuar a sua apresentação considerando que a condição de selecção *cond* foi especificada pelo utilizador na OD. Na situação de tal não acontecer (uma vez que é opcional), em cada formalização apresentada deve ser removido *cond* das expressões em álgebra relacional e caso este represente a única condição existente deve também ser removido o respectivo operador de selecção (*i.e.*, σ). Estas alterações são suficientes para que as formalizações passem a reflectir a semântica inerente à situação mencionada.

Quando se especificam OD ao nível do atributo, a linguagem proposta suporta que se defina numa única operação (*e.g.*: detecção de valor em falta) múltiplos atributos de uma dada relação (*e.g.*: nome; morada; localidade). Isto significa que a OD deve ser executada em cada um destes atributos. Assim, ainda que seja especificada uma só OD, esta corresponde a múltiplas operações, uma vez que a sua execução é efectuada individualmente em cada atributo. As formalizações apresentadas na secção seguinte reflectem a semântica subjacente à execução da OD num atributo individual.

5.3 Problemas ao Nível do Atributo

Nesta secção apresenta-se a formalização das OD dos PQD que ocorrem ao nível do atributo. Seguindo a organização dos PQD proposta na taxionomia, começa-se por formalizar as OD dos problemas que ocorrem no contexto do valor individual do atributo. De seguida, são apresentadas as formalizações das OD dos PQD que se manifestam no contexto dos vários valores de um atributo.

5.3.1 Contexto do Valor Individual

Começa-se, então, por apresentar a formalização das OD dos PQD que ocorrem ao nível do valor individual do atributo.

5.3.1.1 Valor em falta

Formalização Sintáctica

DETECT MISSING-VALUE ColunasAlvo DasTabelas DasBasesDados UsandoCondição

Formalização Semântica

A detecção deste PQD envolve identificar as chaves primárias dos tuplos onde o atributo em causa (*i.e.*, atr_1) não possui valor (*i.e.*, encontra-se a *null*).

Algoritmo Detecção_Valor_em_Falta $\stackrel{\text{def}}{=}$

seja $\mathcal{Z}1$ a relação que armazena os valores das chaves primárias dos tuplos de $\mathcal{R}$ onde existem valores em falta em atr_1, com o seguinte esquema: $\mathcal{Z}1(a_1,\ldots,a_m)$.

início

se *existe_relação*($\mathcal{Z}1$) = *falso* **então**

$$\mathcal{Z}1 \leftarrow \pi_{a_1,\ldots,a_m}\left(\sigma_{atr_1=null \wedge cond}\left(\mathcal{R} \rhd\lhd \left(\pi_{a_1,\ldots,a_m}(\mathcal{R}) - \left(\bigcup_{i=1}^{p} \pi_{a_1,\ldots,a_m}(\mathcal{Z}1_i)\right)\right)\right)\right)$$

senão

$$\mathcal{Z}1 \leftarrow \pi_{a_1,\ldots,a_m}\left(\sigma_{atr_1=null \wedge cond}(\mathcal{R} \rhd\lhd \mathcal{Z}1)\right)$$

fim se

fim

Nota: Nos problemas que ocorrem ao nível do contexto do valor individual do atributo e no que ocorre ao nível do tuplo, surge uma estrutura condicional do género: se *existe_relação*($\mathcal{Z}1$) = *falso* então… senão… A finalidade desta estrutura condicional é a

seguir explicada. No caso da relação que armazena os PQD (*i.e.*, $\mathcal{Z}1$) não existir, isso significa que se está perante a primeira execução da OD. Nesta primeira execução consideram-se todos os valores do atributo, à excepção daqueles que tenham sido explicitamente excluídos na especificação da operação (pelo utilizador) e daqueles onde já tenham sido identificados PQD resultantes da execução de operações anteriores da sequência de OD, em relação à operação corrente (nas formalizações, a exclusão destes valores é representada pela sub-expressão: $(\mathcal{R} \triangleright\triangleleft (\pi_{a_1,\ldots,a_m}(\mathcal{R})...(\mathcal{Z}1_i)))$). No caso de existir a relação que armazena os PQD, isso significa que já se verificou uma execução anterior da OD. Nesta situação, apenas são sujeitos à execução da operação de detecção os valores do atributo onde foram anteriormente identificados PQD. Todos os restantes valores são ignorados, uma vez que já foram previamente verificados e não se detectou o PQD em questão.

5.3.1.2 Violação de sintaxe / domínio

Formalização Sintáctica

DETECT PQD ColunasAlvo DasTabelas DasBasesDados UsandoAtribuições BaseadoEm

PQD → SYNTAX-VIOLATION | DOMAIN-VIOLATION
BaseadoEm → DefiniçãoDomínio | DefiniçãoCondição

Formalização Semântica

No caso de se tratar da detecção de violação de sintaxe/domínio com base numa condição susceptível de ser expressa em SQL, a operação envolve identificar as chaves primárias dos tuplos e os respectivos valores de atr_1 que obedecem à condição especificada (*i.e.*, que representam uma violação de sintaxe ou domínio). Estes resultados são armazenados na relação que resulta da OD (*i.e.*, $\mathcal{Z}1$). A semântica que se encontra subjacente à operação é a seguir apresentada.

Algoritmo Detecção_Violação_Sintaxe/Domínio_Baseada_em_SQL $\stackrel{\text{def}}{=}$

seja $\mathcal{Z}1$ a relação que armazena os valores das chaves primárias dos tuplos de $\mathcal{R}$ onde existem violações de sintaxe/domínio em atr_1, com o seguinte esquema: $\mathcal{Z}1(a_1,\ldots,a_m, atr_1)$.

início

se *existe_relação*($\mathcal{Z}1$) = *falso* **então**

$$\mathcal{Z}1 \leftarrow \pi_{a_1,\ldots,a_m,atr_1}\left(\sigma_{cond}\left(\mathcal{R} \triangleright\triangleleft \left(\pi_{a_1,\ldots,a_m}(\mathcal{R}) - \left(\bigcup_{i=1}^{p} \pi_{a_1,\ldots,a_m}(\mathcal{Z}1_i)\right)\right)\right)\right)$$

senão

$$\mathcal{Z}1 \leftarrow \pi_{a_1,\ldots,a_m,atr_1}\left(\sigma_{cond}\left(\mathcal{R} \triangleright\triangleleft \mathcal{Z}1\right)\right)$$

fim se

fim

Nota: Na formalização anterior, *cond*, além de representar uma eventual condição de selecção que delimita a execução da OD, também representa a condição de que resulta a detecção da violação de sintaxe/domínio. O conteúdo de *cond* resulta do que se encontra especificado na cláusula *where* da OD.

Fruto da sua complexidade, a detecção de violação de sintaxe/domínio pode implicar o recurso a FDU. Nesta situação, o valor do atributo atr_1 pode constituir um parâmetro de uma ou mais funções que façam parte da sequência de atribuições especificadas pelo utilizador na OD. Estas funções também podem receber outros parâmetros como variáveis e constantes. Como resultado, cada função fdu_x retorna um valor atómico. A formalização subjacente à OD de violação de sintaxe/domínio envolvendo FDU é, a seguir, exposta.

Algoritmo Detecção_Violação_Sintaxe/Domínio_Baseada_em_FDU $\stackrel{\text{def}}{=}$

início

se *existe_relação*($\mathcal{Z}1$) = *falso* **então**

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \left(\pi_{a_1,\ldots,a_m}(\mathcal{R}) - \left(\bigcup_{i=1}^{p} \pi_{a_1,\ldots,a_m}(\mathcal{Z}1_i)\right)\right)\right)\right)$$

senão

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \mathcal{Z}1\right)\right)$$

fim se

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

repetir enquanto $t \neq$ *null*

$$v_1 \leftarrow fdu_1\left(p_{1,1}, p_{1,2}, \ldots, p_{1,x}\right)$$

$$v_2 \leftarrow fdu_2\left(p_{2,1}, p_{2,2}, \ldots, p_{2,y}\right)$$

$$\vdots$$

$$v_j \leftarrow fdu_j\left(p_{j,1}, p_{j,2}, \ldots, p_{j,w}\right)$$

se $avalia\_condição\left(p_{k,1}, p_{k,2}, \ldots, p_{k,z}\right) = verdadeiro$ **então**

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(t.a_1, \ldots, t.a_m, t.atr_1)\}$

fim se

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

fim repetir

fim

Nota: O esquema de *Z1* é igual ao que se encontra apresentado na formalização da detecção de violação de sintaxe/domínio baseada em SQL.

No caso de se tratar da detecção de violação de domínio com base num domínio de referência armazenado numa relação *X*, a operação envolve identificar as chaves primárias dos tuplos e os respectivos valores de atr_1 que não se encontram nesse domínio. Ambos os resultados são armazenados na relação que resulta da OD (*i.e.*, *Z1*). Neste caso, a formalização é a que a seguir se apresenta.

Algoritmo Detecção_Violação_Domínio_Baseada_Domínio_Referência $\stackrel{def}{=}$

seja *X* a relação que armazena o domínio de valores válidos com o seguinte esquema: *X*(*id*, *dom*), em que *id* constitui a chave primária e *dom* o atributo onde se encontram os valores considerados válidos para atr_1.

início

se *existe_relação*(*Z1*) = *falso* **então**

$$Z1 \leftarrow \pi_{a_1,\ldots,a_m,atr_1}(\sigma_{cond}(\mathcal{R} \rhd\lhd (\pi_{a_1,\ldots,a_m}(\mathcal{R})-(\bigcup_{i=1}^{p}\pi_{a_1,\ldots,a_m}(Z1_i))) \rhd\lhd (\pi_{atr_1}(\mathcal{R})-\pi_{dom}(X))))$$

senão

$$Z1 \leftarrow \pi_{a_1,\ldots,a_m,atr_1}(\sigma_{cond}(\mathcal{R} \rhd\lhd Z1 \rhd\lhd (\pi_{atr_1}(\mathcal{R})-\pi_{dom}(X))))$$

fim se

fim

Nota: O esquema de *Z1* é igual ao que se encontra apresentado na formalização da detecção de violação de sintaxe/domínio baseada em SQL.

5.3.1.2.1 Valor incompleto

Formalização Sintáctica

DETECT INCOMPLETE-VALUE ColunasAlvo DasTabelas DasBasesDados DefiniçãoDomínio UsandoCondição

Formalização Semântica

A detecção de valor incompleto envolve verificar se cada valor do atributo em causa (*i.e.*, atr_1) que não faz parte do domínio respectivo (eventualmente já confirmado por uma operação prévia de detecção de violação de domínio) constitui uma *substring* de um ou mais valores válidos do domínio. Caso esta situação se verifique, o valor é classificado como estando incompleto. Na relação que resulta desta OD (*i.e.*, *Z1*), além dos valores da chave primária dos tuplos onde se encontram os valores identificados como incompletos, os próprios valores incompletos são

também armazenados, assim como os diversos valores que, ao completá-los, constituem alternativas válidas (*i.e.*, pertencem ao domínio de valores válidos do atributo).

Algoritmo Detecção_Valor_Incompleto $\stackrel{\text{def}}{=}$

seja $\mathcal{Z}1_{\text{x}}$ a relação que armazena as violações de domínio detectadas em atr_1 com base num domínio de referência.

seja $\mathcal{X}$ a relação que armazena o domínio de valores válidos com o seguinte esquema: $\mathcal{X}(id, dom)$, em que *id* constitui a chave primária e *dom* o atributo onde se encontram os valores considerados válidos para atr_1.

seja $\mathcal{Z}1$ a relação que armazena os valores das chaves primárias dos tuplos de $\mathcal{R}$ onde existem valores incompletos em atr_1, os respectivos valores incompletos e as diversas alternativas válidas que os completam. O esquema de $\mathcal{Z}1$ é: $\mathcal{Z}1(a_1,\ldots,a_{\text{m}}, atr_1, dom)$.

início

se *existe_operação_detecção_violação_domínio*(atr_1) = *verdadeiro* **então**

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_{\text{m}},atr_1}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \left(\pi_{a_1,\ldots,a_{\text{m}}}\left(\mathcal{Z}1_{\text{x}}\right)\right)\right)\right)$$

senão se *existe_relação*($\mathcal{T}$)= *falso* **então**

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_{\text{m}},atr_1}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \left(\pi_{a_1,\ldots,a_{\text{m}}}\left(\mathcal{R}\right) - \left(\bigcup_{i=1}^{p} \pi_{a_1,\ldots,a_{\text{m}}}\left(\mathcal{Z}1_i\right)\right)\right)\right)\right)$$

senão

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_{\text{m}},atr_1}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \mathcal{T}\right)\right)$$

fim se

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

repetir enquanto $t \neq$ *null*

se *existe_operação_detecção_violação_domínio*(atr_1) **ou** $\sigma_{dom = t.atr_1}(X) = \varnothing$ **então**

$$\mathcal{U} \leftarrow \pi_{dom}\left(\sigma_{dom \text{ like } '\%t.atr_1\%' \vee\ dom \text{ like } 't.atr_1\%' \vee\ dom \text{ like } '\%t.atr_1'}(X)\right)$$

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

repetir enquanto $u \neq$ *null*

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(t.a_1,\ldots,t.a_{\text{m}},t.atr_1,u.dom)\}$

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

fim repetir

fim se

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

fim repetir

fim

5.3.1.2.2 Valor sobrecarregado

Formalização Sintáctica

DETECT OVERLOADED-VALUE ColunasAlvo DasTabelas DasBasesDados
DefiniçãoDomínio UsandoCondição

Formalização Semântica

A detecção deste PQD consiste em verificar se cada valor do domínio inicia ou termina da mesma forma do que um valor do atributo em causa (*i.e.*, atr_1) que não faz parte do domínio respectivo (eventualmente já confirmado por uma operação prévia de detecção de violação de domínio) ou que é composto por um número de caracteres inferiores a este valor, constitui uma *substring* deste. Caso esta situação se verifique, o valor é considerado *sobrecarregado*. Na relação que resulta desta OD (*i.e.*, $Z1$), além dos valores da chave primária dos tuplos onde se encontram os valores identificados como sobrecarregados, os próprios valores sobrecarregados são também armazenados, assim como os diversos valores que constituem alternativas válidas (*i.e.*, pertencem ao domínio de valores válidos do atributo). Caso exista uma operação prévia de detecção de valor incompleto sobre atr_1, os valores já identificados como tal são excluídos desta OD de valor sobrecarregado.

Algoritmo Detecção_Valor_Sobrecarregado $\stackrel{\text{def}}{=}$

seja *existe_operação_detecção_valor_incompleto*(*atr*) uma função que recebe um atributo *atr* e verifica se na sequência de OD existe uma OD de valor incompleto sobre *atr*, devolvendo um valor booleano como resultado.

seja $Z1_y$ a relação que armazena os valores incompletos detectados em atr_1.

seja *nr_tokens*(*str*) uma função que recebe uma string *str* e devolve o número de *tokens* que a compõem.

seja *extrai_primeiro_token*(*str*) uma função que recebe uma string *str* e devolve o primeiro *token* dessa string.

seja *extrai_ultimo_token*(*str*) uma função que recebe uma string *str* e devolve o último *token* dessa string.

seja *length*(*str*) uma função que recebe uma string *str* e devolve o número de caracteres que a compõem.

início

se *existe_operação_detecção_violação_domínio*(atr_1) = *verdadeiro* **então**

se *existe_operação_detecção_valor_incompleto*(atr_1) = *verdadeiro* **então**

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \left(\pi_{a_1,\ldots,a_m}\left(Z1_x\right)\right) \rhd\lhd \left(\pi_{a_1,\ldots,a_m}\left(\mathcal{R}\right) - \pi_{a_1,\ldots,a_m}\left(Z1_y\right)\right)\right)\right)$$

senão

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \left(\pi_{a_1,\ldots,a_m}\left(\mathcal{Z}1_x\right)\right)\right)\right)$$

fim se

senão se *existe_relação*($\mathcal{T}$)= *falso* **então**

se *existe_operação_detecção_valor_incompleto*(atr_1) = *verdadeiro* **então**

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}(\sigma_{cond}(\mathcal{R} \rhd\lhd (\pi_{a_1,\ldots,a_m}(\mathcal{R})-(\bigcup_{i=1}^{p}\pi_{a_1,\ldots,a_m}(\mathcal{Z}1_i))) \rhd\lhd (\pi_{a_1,\ldots,a_m}(\mathcal{R})- -\pi_{a_1,\ldots,a_m}(\mathcal{Z}1_y))))$$

senão

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}(\sigma_{cond}(\mathcal{R} \rhd\lhd (\pi_{a_1,\ldots,a_m}(\mathcal{R})-(\bigcup_{i=1}^{p}\pi_{a_1,\ldots,a_m}(\mathcal{Z}1_i)))))$$

fim se

senão

se *existe_operação_detecção_valor_incompleto*(atr_1) = *verdadeiro* **então**

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \mathcal{T} \rhd\lhd \left(\pi_{a_1,\ldots,a_m}\left(\mathcal{R}\right)-\pi_{a_1,\ldots,a_m}\left(\mathcal{Z}1_y\right)\right)\right)\right)$$

senão

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \mathcal{T}\right)\right)$$

fim se

fim se

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

repetir enquanto $t \neq$ *null*

se *nr_tokens*($t.atr_1$) ≥ 1 **e** (*existe_operação_detecção_violação_domínio*(atr_1) **ou**

$$\sigma_{dom = t.atr_1}(X) = \varnothing)$$ **então**

palavra $\leftarrow$ *extrai_primeiro_token*($t.atr_1$)

$$\mathcal{U} \leftarrow \pi_{dom}\left(\sigma_{dom \text{ like } 'palavra\%' \wedge length(dom)<t.atr_1}(X)\right)$$

determina_valores_validos($\mathcal{U}$,t)

palavra $\leftarrow$ *extrai_ultimo_token*($t.atr_1$)

$$\mathcal{U} \leftarrow \pi_{dom}\left(\sigma_{dom \text{ like } '\%palavra' \wedge length(dom)<t.atr_1}(X)\right)$$

determina_valores_validos($\mathcal{U}$,t)

$$\mathcal{U} \leftarrow \pi_{dom}\left(\sigma_{length(dom)<t.atr_1}(X)\right)$$

determina_valores_validos($\mathcal{U}$,t)

fim se

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

fim repetir

fim

Notas: Os significados de $\mathcal{Z}1_x$ e $\mathcal{X}$ usados nesta formalização encontram-se enunciados nas definições da formalização da detecção de valor incompleto.

A finalidade de $\mathcal{Z}1_y$ é excluir da detecção de valor sobrecarregado todos os valores que já foram identificados como estando incompletos. Na sequência de execução de OD, a detecção de valor incompleto antecede a detecção de valor sobrecarregado. Um valor não pode estar ao mesmo tempo incompleto e sobrecarregado.

Procedimento determina_valores_validos($\mathcal{U}$,t**)** $\stackrel{\text{def}}{=}$

seja *substring*(str_1,str_2) uma função que recebe duas strings, str_1 e str_2, verifica se a primeira constitui uma substring da segunda e devolve como resultado um valor booleano.

seja $\mathcal{Z}1$ a relação que armazena os valores das chaves primárias dos tuplos de $\mathcal{R}$ onde existem valores sobrecarregados em atr_1, os respectivos valores sobrecarregados e as diversas alternativas válidas que solucionam o problema. O esquema de $\mathcal{Z}1$ é: $\mathcal{Z}1(a_1,\ldots,a_m, atr_1,dom)$.

início

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

repetir enquanto $u \neq null$

se *substring*($u.dom$, $t.atr_1$) = *verdadeiro* **então**

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(t.a_1,\ldots,t.a_m,t.atr_1,u.dom)\}$

fim se

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

fim repetir

fim

5.3.1.3 Erro ortográfico

Formalização Sintáctica

DETECT MISSPELLING-ERRORS ColunasAlvo DasTabelas DasBasesDados
DefiniçãoDicionário DefiniçãoMétrica UsandoCondição

Formalização Semântica

A detecção de erro ortográfico envolve verificar se cada palavra ou *token* que compõe cada valor do atributo em questão (*i.e.*, atr_1) existe num dado dicionário definido pelo utilizador. Caso não exista, está-se perante um erro ortográfico. Quando isto acontece, procura-se identificar as palavras existentes no dicionário lexicalmente similares ao erro ortográfico, com base numa métrica de semelhança. Na situação de serem identificadas palavras semelhantes (uma ou mais), estas são acompanhadas do respectivo grau de semelhança gerado pela métrica. Além dos valores

da chave primária dos tuplos onde foram encontrados erros ortográficos no atributo, a relação que resulta desta OD (*i.e.*, *Z1*) armazena esses erros, assim como as respectivas palavras similares e os graus de semelhança associados. Naturalmente, estes últimos apenas são armazenados caso a identificação de palavras semelhantes tenha tido sucesso.

Algoritmo Detecção_Erro_Ortográfico $\stackrel{\text{def}}{=}$

seja *tokenize_string*(*str*) uma função que recebe uma string *str* e devolve um vector com os diversos *tokens* que a compõem.

seja *existe_no_dicionário*(*str*, *dic*) uma função que recebe uma string *str* e uma relação *dic* que armazena um dicionário, verifica a existência de *str* em *dic* e, como resultado, devolve um valor booleano.

seja *identifica_palavras_semelhantes*(*str*, *dic*, *metr*) uma função que recebe uma string *str*, uma relação *dic* que armazena um dicionário e uma string *metr* com a métrica de semelhança a usar. Esta função identifica as diversas palavras existentes no dicionário *dic* lexicalmente similares a *str*, com base na métrica de semelhança *metr* e devolve uma estrutura de dados bidimensional (*i.e.*, uma matriz). Na primeira coluna desta estrutura encontra-se a palavra semelhante, enquanto que na segunda se encontra o respectivo grau de semelhança gerado pela métrica utilizada. No caso de não serem identificadas palavras semelhantes, a função retorna o valor *null*.

seja *Z1* a relação que armazena os valores das chaves primárias dos tuplos de $\mathcal{R}$ onde existem erros ortográficos em atr_1, a correspondente palavra errada (*i.e.*, com erro ortográfico), bem como as diversas palavras semelhantes a esta encontradas no dicionário e os respectivos graus de semelhança. O esquema de *Z1* é: $Z1(a_1,\ldots,a_m, pal\_err, pal\_semelh, grau\_semelh)$.

início

se *existe_relação*($\mathcal{T}$)= *falso* **então**

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \left(\pi_{a_1,\ldots,a_m}(\mathcal{R}) - \left(\bigcup_{i=1}^{p} \pi_{a_1,\ldots,a_m}(Z1_i)\right)\right)\right)\right)$$

senão

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \mathcal{T}\right)\right)$$

fim se

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

repetir enquanto $t \neq$ *null*

tokens $\leftarrow$ *tokenize_string*($t.atr_1$)

repetir para $i \leftarrow 1$ **até** *dimensão*(*tokens*)

palavra $\leftarrow$ *tokens*[*i*]

se *existe_no_dicionário*(*palavra*, *dic*) = *falso* **então**

plvs_semelhs $\leftarrow$ *identifica_palavras_semelhantes*(*palavra*, *dic*, *metr*)

se *plvs_semelhs* = *null* **então**

$Z1 \leftarrow Z1 \cup \{(t.a_1,\ldots,t.a_m,palavra)\}$

senão

repetir para $j \leftarrow 1$ **até** *numlinhas*(*plvs_semelhs*)

$Z1 \leftarrow Z1 \cup \{(t.a_1,\ldots,t.a_m,palavra,plvs\_semelhs[j][1],plvs\_semelhs[j][2])\}$

fim repetir

fim se

fim se

fim repetir

$t \leftarrow le\_tuplo(T)$

fim repetir

fim

5.3.2 Contexto Multi-Valor

Nesta secção apresenta-se a formalização das OD dos PQD que se manifestam ao nível dos vários valores do atributo.

5.3.2.1 Existência de sinónimos

Formalização Sintáctica

DETECT EXISTENCE-OF-SYNONYMS ColunasAlvo DasTabelas DasBasesDados
DefiniçãoDicionário UsandoCondição

Formalização Semântica

A detecção da existência de sinónimos envolve verificar se os valores que constituem os pares resultantes de cada *cluster* de sinónimos que constam de um determinado dicionário definido pelo utilizador existem, simultaneamente, no atributo em questão (*i.e.*, atr_1). No caso de se detectar a sua existência, está-se perante um problema de sinónimos. Na relação que resulta desta OD (*i.e.*, *Z1*), além das chaves primárias dos tuplos onde existem sinónimos no atributo, os respectivos valores também são armazenados, assim como o *cluster* de sinónimos a que pertencem. Nesta relação, um dado valor apenas é armazenado uma única vez, apesar das múltiplas situações em que possa ser identificado como sinónimo de outros valores.

Algoritmo Detecção_Existência_Sinónimos $\stackrel{def}{=}$

seja *X* a relação onde se encontra armazenado o dicionário de sinónimos com esquema: *X*(*idsin*, *idcluster*, *sin*), em que *idsin* constitui a chave primária, *idcluster* o *cluster* a que pertence o sinónimo e *sin* o termo sinónimo.

seja *Z1* a relação que armazena os valores das chaves primárias dos tuplos de $\mathcal{R}$ onde foi detectada a existência de sinónimos em atr_1, o correspondente sinónimo, bem como o

respectivo *cluster* de sinónimos a que este pertence. O esquema de $Z1$ é: $Z1(a_1,\ldots,a_m,atr_1,idcluster)$.

início

$\mathcal{T} \leftarrow \pi_{idcluster}\left(\sigma_{cond}\left(X\right)\right)$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$\mathcal{U} \leftarrow \pi_{sin,sin2}(X \rhd\lhd_{(idcluster=t.idcluster \wedge idcluster2=t.idcluster \wedge idsin2>idsin)} (\rho_{X2(idsin2 \leftarrow idsin, idcluster2 \leftarrow idcluster, sin2 \leftarrow sin)}(X)))$

$u \leftarrow le\_tuplo(\mathcal{U})$

repetir enquanto $u \neq null$

$\mathcal{V} \leftarrow \pi_{a_1,\ldots,a_m,atr_1,b_1,\ldots,b_m,atr_2}(\sigma_{((atr_1=u.sin \vee atr_1 \text{ like } 'u.sin\%' \vee atr_1 \text{ like } '\%u.sin\%' \vee atr_1 \text{ like } '\%u.sin') \wedge (atr_2=u.sin2 \vee atr_2 \text{ like } 'u.sin2\%' \vee atr_2 \text{ like } '\%u.sin2\%' \vee atr_2 \text{ like } '\%u.sin2'))}(\mathcal{R} \rhd\lhd_{(a_1=b_1 \wedge \ldots \wedge a_m=b_m)}(\rho_{\mathcal{R}2(b_1 \leftarrow a_1,\ldots,b_m \leftarrow a_m,atr_2 \leftarrow atr_1)}(\mathcal{R}))))$

$v \leftarrow le\_tuplo(\mathcal{V})$

repetir enquanto $v \neq null$

se $\sigma_{a_1=v.a_1 \wedge \ldots \wedge a_m=v.a_m}(Z1) = \varnothing$ **então**

$Z1 \leftarrow Z1 \cup \{(v.a_1,\ldots,v.a_m,v.atr_1,t.idcluster)\}$

fim se

se $\sigma_{b_1=v.b_1 \wedge \ldots \wedge b_m=v.b_m}(Z1) = \varnothing$ **então**

$Z1 \leftarrow Z1 \cup \{(v.b_1,\ldots,v.b_m,v.atr_2,t.idcluster)\}$

fim se

$v \leftarrow le\_tuplo(\mathcal{V})$

fim repetir

$u \leftarrow le\_tuplo(\mathcal{U})$

fim repetir

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

5.3.2.2 Violação de unicidade

Formalização Sintáctica

DETECT UNIQUENESS-VIOLATION ColunasAlvo DasTabelas DasBasesDados

UsandoCondição

Formalização Semântica

A detecção de violação de unicidade envolve identificar os valores das chaves primárias dos tuplos que possuem igual valor no atributo em causa (*i.e.*, atr_1). Além dos valores das chaves primárias, na relação que resulta desta OD (*i.e.*, $\mathcal{Z}1$) são também armazenados os respectivos valores que constituem violações à unicidade do atributo.

Algoritmo Detecção_Violação_Unicidade $\stackrel{\text{def}}{=}$

seja $\mathcal{Z}1$ a relação que armazena os valores das chaves primárias dos tuplos de $\mathcal{R}$ e os respectivos valores que constituem violações de unicidade em atr_1, com o seguinte esquema: $\mathcal{Z}1(a_1,\ldots,a_m,atr_1)$.

seja $\mathcal{Z}1_\chi$ a relação que armazena os valores das chaves primárias dos tuplos de $\mathcal{R}$ e os respectivos valores que constituem sinónimos em atr_1, resultantes da OD prévia (caso esta exista).

início

$$\mathcal{Z}1 \leftarrow \pi_{a_1,\ldots,a_m,atr_1}(\sigma_{cond}(\mathcal{R} \rhd\lhd (\pi_{a_1,\ldots,a_m}(\mathcal{R})\text{-}(\pi_{a_1,\ldots,a_m}(\mathcal{Z}1_\chi))) \rhd\lhd (\pi_{atr_1}(\rho_{\mathcal{R}1}(\mathcal{R}) \rhd\lhd \rhd\lhd_{(\mathcal{R}1.a_1 \neq \mathcal{R}2.a_1 \wedge \ldots \wedge \mathcal{R}1.a_m \neq \mathcal{R}2.a_m \wedge \mathcal{R}1.atr_1 = \mathcal{R}2.atr_1)} (\rho_{\mathcal{R}2}(\mathcal{R})))))$$

fim

5.3.2.3 Violação de restrição de integridade

Formalização Sintáctica

DETECT INTEGRITY-CONSTRAINT-VIOLATION AT LEVEL OF COLUMN
ColunasAlvo DasTabelas
DasBasesDados DefiniçãoAtribuições
DefiniçãoCondição AgupandoPor

Formalização Semântica

No caso da detecção de violação de restrição de integridade ser efectuada com base numa operação susceptível de ser traduzida num inquérito SQL, a operação envolve uma das seguintes situações: identificar as chaves primárias dos tuplos e os respectivos valores do atributo que constituem uma violação de restrição de integridade; ou, identificar apenas o valor ou valores agregados que materializam a existência de uma violação de restrição de integridade. A semântica inerente a esta operação é a seguir apresentada.

Algoritmo Detecção_Violação_Restrição_Integridade_Baseada_em_SQL $\stackrel{\text{def}}{=}$

seja *func_agrega* uma função de agregação existente em SQL (*e.g.*: *count*; *sum*; *avg*).

seja $\mathcal{Z}1$ a relação que armazena o valor que constitui violação à restrição de integridade, com o seguinte esquema: $\mathcal{Z}1(a_1,\ldots,a_m,atr_1)$.

início

$$\mathcal{Z}1 \leftarrow \pi_{a_1,\ldots,a_m,atr_1}\left(\sigma_{cond}\left({}_{atr_1}\gamma_{func\_agrega(atr_1)}(\mathcal{R})\right)\right)$$

fim

Notas: A formalização apresentada é o mais genérica possível, *i.e.*, abarca todas as situações possíveis de serem expressas numa OD deste tipo. No entanto, dependendo da OD em concreto, algumas das partes que constituem a expressão em álgebra relacional podem não se aplicar. Assim, o operador de agregação (*i.e.*, γ) pode não existir. Caso exista, pode não existir a função de agregação (*i.e.*, *func_agrega*), assim como pode também não existir o operador de projecção (*i.e.*, π). No caso deste existir, os atributos que constituem a chave primária de $\mathcal{R}$ (*i.e.*, $a_1,\ldots,a_m$) podem não constar no operador de projecção. Naturalmente, nesta situação também não constam de $\mathcal{Z}1$.

Na formalização, *cond*, além de representar uma eventual condição de selecção que delimita a execução da OD, também representa a condição de que resulta a detecção da violação da restrição de integridade. O conteúdo de *cond* resulta do que se encontra especificado na cláusula *where* da OD.

Na detecção de violação de restrição de integridade ao nível da coluna pode ser necessário recorrer a FDU (*e.g.*: devido à complexidade das operações envolvidas na detecção). Nestas situações, o conjunto de valores do atributo atr_1 pode ser passado como parâmetro, de acordo com os requisitos de cada FDU presente na sequência de atribuições especificada na OD. As funções também podem receber outros parâmetros, como variáveis e constantes. À excepção da última função (*i.e.*, fdu_j) que obrigatoriamente retorna um valor atómico representativo do problema detectado, todas as outras podem retornar um valor atómico ou uma estrutura de dados (*e.g.*: um vector; uma matriz). A formalização da detecção do PQD em causa, baseada em FDU, é seguidamente apresentada.

Algoritmo Detecção_Violação_Restrição_Integridade_Baseada_em_FDU $\stackrel{\text{def}}{=}$

seja $\mathcal{Z}1$ a relação que armazena o valor que constitui violação à restrição de integridade, com o seguinte esquema: $\mathcal{Z}1$(*valor*). O tipo de dados de *valor* é o mesmo do que o retornado pela última função (no caso, fdu_j) da sequência de FDU na OD.

início

$$\mathcal{T} \leftarrow \pi_{atr_1}\left(\sigma_{cond}(\mathcal{R})\right)$$

$$v_1 \leftarrow fdu_1\left(p_{1,1},p_{1,2},\ldots,p_{1,x}\right)$$

$$v_2 \leftarrow fdu_2\left(p_{2,1},p_{2,2},\ldots,p_{2,y}\right)$$

$$\vdots$$

$$v_j \leftarrow fdu_j\left(p_{j,1},p_{j,2},\ldots,p_{j,w}\right)$$

se $avalia\_condição\left(p_{k,1},p_{k,2},\ldots,p_{k,z}\right) = verdadeiro$ **então**

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(v_j)\}$

fim se

fim

No caso da última função da sequência de atribuições (*i.e.*, fdu_j) retornar uma estrutura de dados bidimensional (*i.e.*, uma matriz), a formalização anterior sofre as alterações que a seguir se apresentam. O que permanece inalterado relativamente à formalização anterior encontra-se representado sob a forma de reticências.

Algoritmo Detecção_Violação_Restrição_Integridade $\stackrel{\text{def}}{=}$

seja $\mathcal{Z}1$ a relação que armazena os valores dos atributos dos tuplos que constituem violações à restrição de integridade, com o seguinte esquema: $\mathcal{Z}1(a_x,\ldots,a_z)$.

início

⋮

repetir para $h \leftarrow 1$ **até** $numlinhas(v_j)$

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(v_j[h][1], \ldots, v_j[h][numcolunas])\}$

fim repetir

fim

Nota: Entre os atributos de $\mathcal{Z}1$ podem encontrar-se os que constituem a chave primária de $\mathcal{R}$.

5.4 Problemas ao Nível do Tuplo

Nesta secção apresenta-se a formalização da OD do único PQD que se manifesta ao nível do tuplo.

5.4.1 Violação de restrição de integridade

Formalização Sintáctica

DETECT INTEGRITY-CONSTRAINT-VIOLATION AT LEVEL OF ROW ColunasAlvo
DasTabelas DasBasesDados
UsandoAtribuições DefiniçãoCondição

Formalização Semântica

No caso da detecção de violação de restrição de integridade com base numa condição susceptível de ser expressa em SQL, a operação envolve identificar as chaves primárias dos tuplos e os respectivos valores dos atributos envolvidos na restrição de integridade que respeitam a condição

especificada (e que assim constituem uma violação de restrição de integridade). Todos estes valores são armazenados na relação que resulta da OD (*i.e.*, $\mathcal{Z}1$). A semântica que se encontra subjacente a esta operação é de seguida apresentada.

Algoritmo Detecção_Violação_Restrição_Integridade_Baseada_em_SQL $\stackrel{\text{def}}{=}$

seja $\mathcal{A} = \{a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos de $\mathcal{R}$ que não fazem parte da sua chave primária.

seja $\mathcal{C}$ o conjunto de atributos envolvidos na restrição de integridade, assim definido: $\mathcal{C} = \{c \mid c$ é usado na formulação da restrição de integridade $\wedge\ c \in \mathcal{A}\}$, *i.e.*, $\mathcal{C} \subseteq \mathcal{A}$.

seja o a cardinalidade de $\mathcal{C}$.

seja $\mathcal{Z}' = \{\mathcal{Z}1_1,\ldots,\mathcal{Z}1_o\}$ o conjunto de o relações que contêm os PQD já detectados nos atributos de $\mathcal{C}$ pelas operações anteriores que compõem a sequência de detecção.

seja $\mathcal{Z}1$ a relação que armazena os valores das chaves primárias dos tuplos de $\mathcal{R}$ e os respectivos valores que constituem violações à restrição de integridade, com o seguinte esquema: $\mathcal{Z}1(a_1,\ldots,a_m,c_1,\ldots,c_o)$.

início

se *existe_relação*($\mathcal{Z}1$) = *falso* **então**

$$\mathcal{Z}1 \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_o}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \left(\pi_{a_1,\ldots,a_m}(\mathcal{R}) - \left(\bigcup_{i=1}^{p} \pi_{a_1,\ldots,a_m}(\mathcal{Z}1_i)\right)\right)\right)\right)$$

senão

$$\mathcal{Z}1 \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_o}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \mathcal{Z}1\right)\right)$$

fim se

fim

Nota: Na formalização anterior, *cond*, além de representar uma eventual condição de selecção que delimita a execução da OD, também representa a condição de que resulta a detecção da violação da restrição de integridade. O conteúdo de *cond* resulta do que se encontra especificado na cláusula *where* da OD.

A detecção de violação de restrição de integridade ao nível do tuplo pode envolver a utilização de FDU. Nos casos em que isto acontece, os valores dos atributos envolvidos na formulação da restrição de integridade podem ser passados como parâmetros, obedecendo aos requisitos de cada FDU que se encontra na sequência de atribuições especificada pelo utilizador na OD. Variáveis e constantes também podem ser parâmetros das funções. Como resultado, cada função fdu_χ retorna um valor atómico. De seguida, expõe-se a formalização desta OD baseada em FDU.

Algoritmo Detecção_Violação_Restrição_Integridade_Baseada_em_FDU $\stackrel{\text{def}}{=}$

início

se *existe_relação*($\mathcal{Z}1$) = *falso* **então**

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_o}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \left(\pi_{a_1,\ldots,a_m}(\mathcal{R}) - \left(\bigcup_{i=1}^{p} \pi_{a_1,\ldots,a_m}(\mathcal{Z}1_i)\right)\right)\right)\right)$$

senão

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_o}\left(\sigma_{cond}(\mathcal{R} \rhd\lhd \mathcal{Z}1)\right)$$

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$$v_1 \leftarrow fdu_1\left(p_{1,1}, p_{1,2}, \ldots, p_{1,x}\right)$$

$$v_2 \leftarrow fdu_2\left(p_{2,1}, p_{2,2}, \ldots, p_{2,y}\right)$$

$$\vdots$$

$$v_j \leftarrow fdu_j\left(p_{j,1}, p_{j,2}, \ldots, p_{j,w}\right)$$

se $avalia\_condição\left(p_{k,1}, p_{k,2}, \ldots, p_{k,z}\right) = verdadeiro$ **então**

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(t.a_1, \ldots, t.a_m, t.c_1, \ldots, t.c_o)\}$

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: Os significados de $\mathcal{Z}1$, C e $\mathcal{Z}1_i$ utilizados nesta formalização encontram-se enunciados nas definições da formalização anterior.

5.5 Problemas ao Nível da Relação

Nesta secção apresenta-se a formalização das OD dos PQD que ocorrem ao nível da relação.

5.5.1 Violação de dependência funcional

Formalização Sintáctica

DETECT FUNCTIONAL-DEPENDENCY-VIOLATION ColunasAlvo DasTabelas
DasBasesDados
DefiniçãoColunasDependentes
UsandoCondição

Formalização Semântica

A detecção deste PQD envolve identificar os valores das chaves primárias dos tuplos nos quais os valores dos atributos que formam o conjunto de que atr_1 depende a nível funcional são iguais, mas nos quais o valor de atr_1 é diferente. Além dos valores das chaves primárias, a relação que resulta desta OD (*i.e.*, $\mathcal{Z}1$) armazena os valores dos atributos que representam a violação de dependência funcional (*i.e.*, dos atributos independentes e do atributo dependente).

Algoritmo Detecção_Violação_Dependência_Funcional $\triangleq^{def}$

seja $\mathcal{A} = \{a_1,\ldots,a_m,a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos de $\mathcal{R}$.

seja $\mathcal{C}$ o conjunto de atributos de que atr_1 depende funcionalmente, assim definido: $\mathcal{C} = \{c \mid atr_1$ depende funcionalmente de $c \wedge c \in \mathcal{A}\}$, *i.e.*, $\mathcal{C} \subset \mathcal{A}$.

seja o a cardinalidade de $\mathcal{C}$.

seja $\mathcal{Z}1$ a relação que armazena os valores das chaves primárias dos tuplos de $\mathcal{R}$ e os respectivos valores que constituem violações à dependência funcional existente, com o seguinte esquema: $\mathcal{Z}1(a_1,\ldots,a_m,c_1,\ldots,c_o, atr_1)$.

início

$$\mathcal{Z}1 \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_o,atr_1}(\sigma_{cond}(\mathcal{R} \rhd\lhd (\pi_{c_1,\ldots,c_o}(\rho_{\mathcal{R}1}(\mathcal{R}) \rhd\lhd_{(\mathcal{R}1.c_1=\mathcal{R}2.c_1 \wedge \ldots\ldots \wedge \mathcal{R}1.c_o=\mathcal{R}2.c_o \wedge \mathcal{R}1.atr_1 \neq \mathcal{R}2.atr_1)} (\rho_{\mathcal{R}2}(\mathcal{R}))))))$$

fim

5.5.2. Circularidade entre tuplos num auto-relacionamento

Formalização Sintáctica

DETECT EXISTENCE-OF-CIRCULARITY ColunasAlvo DasTabelas
DasBasesDados DefiniçãoColunas
UsandoCondição

Formalização Semântica

A detecção deste PQD envolve desenvolver todas as sequências possíveis de tuplos que resultam do auto-relacionamento. Assim, começa-se por identificar os tuplos que constituem o ponto de partida das sequências resultantes dos auto-relacionamento. Para cada um destes tuplos, expande-se a sequência de auto-relacionamento o máximo possível. Para o efeito utilizam-se os valores da chave primária e da chave estrangeira de cada tuplo, à custa dos quais se estabelece o auto-relacionamento para sucessivamente ir desenvolvendo essa sequência. Este desenvolvimento é interrompido quando: se alcançou um tuplo cuja chave estrangeira não estabelece um

relacionamento com qualquer outro tuplo, *i.e.*, o seu valor encontra-se a *null*; ou, é incluído um tuplo na sequência que já existia nesta, o que significa que se está perante uma situação de circularidade. No desenvolvimento das sequências é utilizada uma abordagem em profundidade, *i.e.*, todas as sequências possíveis de se estabelecerem a partir do tuplo que constituiu o ponto de partida são geradas na procura da existência de circularidades. Apenas quando já não é possível proceder à expansão de mais nenhuma sequência, é que se passa à expansão das sequências relativas a outro tuplo que constitua ponto de partida. À medida que as sequências vão sendo desenvolvidas são armazenadas numa *string*, ficando as chaves primárias/estrangeiras dos tuplos que as constituem separadas por ponto e vírgula. As sequências circulares identificadas são armazenadas na relação respectiva (*i.e.*, *Z1*) também neste formato.

Algoritmo Detecção_Existência_Circularidade $\stackrel{\text{def}}{=}$

seja $\mathcal{A} = \{a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos de $\mathcal{R}$ que não fazem parte da sua chave primária.

seja C o conjunto de atributos que formam a chave estrangeira que define o auto-relacionamento em $\mathcal{R}$, assim definido: $C = \{c \mid c$ faz parte da chave estrangeira que define o auto-relacionamento $\wedge\ c \in \mathcal{A}\}$, *i.e.*, $C \subseteq \mathcal{A}$.

seja *Z1* a relação que armazena as sequências de tuplos nas quais se detectou circularidade, com o seguinte esquema: *Z1*(***seq_circular***). O tipo de dados do atributo ***seq_circular*** é textual.

início

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_m}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \left(\pi_{a_1,\ldots,a_m}(\mathcal{R}) - \pi_{c_1,\ldots,c_m}(\mathcal{R})\right)\right)\right)$$

vec_total_seqs_circulares ← *gerar_sequencias*(*T*, *vec_total_seqs_circulares*)

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_m}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd_{(\mathcal{R}.a_1=\mathcal{R}.2.c_1\ \wedge\ldots\wedge\ \mathcal{R}.a_m=\mathcal{R}.2.c_m)}\left(\rho_{\mathcal{R}2}\left(\pi_{c_1,\ldots,c_m}(\mathcal{R})\right)\right)\right)\right)$$

vec_total_seqs_circulares ← *gerar_sequencias*(*T*, *vec_total_seqs_circulares*)

repetir para *i* ← 1 **até** *dimensão*(*vec_total_seqs_circulares*)

Z1 ← *Z1* ∪ {(*vec_total_seqs_circulares* [*i*])}

fim repetir

fim

Função *gerar_sequências*(*T*, *vec_total_seqs_circulares*) $\stackrel{\text{def}}{=}$

seja *existe_valor_em_vector_ordenado*(***elem***, ***vec***[ ]) uma função que recebe um dado elemento ***elem*** e um vector ordenado ***vec*** e verifica a existência de ***elem*** em ***vec***. Como resultado desta verificação a função devolve um valor booleano.

seja *insere_valor_em_vector_ordenado*(***elem***, ***vec***[ ]) uma função que recebe um dado elemento ***elem*** e um vector ordenado ***vec*** e procede à inserção ordenada de ***elem*** em ***vec***, devolvendo este como resultado.

seja *insere_valor_em_vector*(***elem***, ***vec***[ ]) uma função que recebe um dado elemento ***elem*** e um vector ***vec*** e procede à inserção de ***elem*** em ***vec***, devolvendo este como resultado.

início

t ← *le_tuplo*(*T*)

repetir enquanto *t* ≠ *null*

seq_inicial ← $t.a_1$ o "," o … o "," o $t.a_m$ o ";" o $t.c_1$ o "," o … o "," o $t.c_m$

se *existe_valor_em_vector_ordenado*(*seq_inicial*, *vec_seqs_iniciais*) = *falso* **então**

vec_seqs_iniciais ← *insere_valor_em_vector_ordenado*(*seq_inicial*, *vec_seqs_iniciais*)

vec_seqs_circulares ← *identifica_seqs_circulares*(*seq_inicial*)

repetir para *i* ← 1 **até** *dimensão*(*vec_seqs_circulares*)

vec_total_seqs_circulares ← *insere_valor_em_vector*(*vec_seqs_circulares* [*i*],

,*vec_total_seqs_circulares*)

fim repetir

fim se

t ← *le_tuplo*(*T*)

fim repetir

retorna *vec_total_seqs_circulares*

fim

Função *identifica_seqs_circulares*(*seq_inicial*) $\stackrel{\text{def}}{=}$

seja *vectores_iguais*(*vec1*[ ], *vec2*[ ]) uma função que recebe os vectores *vec1* e *vec2*, verifica se estes são iguais e, como resultado, devolve o valor booleano correspondente.

seja *cria_vector_a_partir_de_string*(*string*, *token*) uma função que recebe uma dada *string* e um *token*, efectuado a separação da *string* nos diversos *tokens* que a compõem com base no *token* fornecido. Cada *token* é colocado num vector, sendo este devolvido como resultado da função.

seja *valores_duplicados_no_vector*(*vec*[ ]) uma função que recebe um vector, verifica se este contém valores duplicados e devolve o valor booleano correspondente.

seja *remove_elemento_do_vector*(*elem*, *vec*[ ]) uma função que recebe um dado elemento *elem* e um vector *vec* e procede à remoção de *elem* em *vec*, devolvendo este como resultado.

início

vec_seqs_geradas ← *insere_valor_em_vector*(*seq_inicial*, *vec_seqs_geradas*)

possível_expandir ← *verdadeiro*

repetir enquanto *possível_expandir* = *verdadeiro*

vec_novas_seqs_geradas ← *expandir_seqs_mais_um_tuplo*(*vec_seqs_geradas*)

se *vectores_iguais*(*vec_seqs_geradas*, *vec_novas_seqs_geradas*) = *verdadeiro* **então**

possível_expandir ← *falso*

senão

repetir para *i* ← 1 **até** *dimensão*(*vec_novas_seqs_geradas*)

vec_chaves_seq ← *cria_vector_a_partir_de_string*(*vec_novas_seqs_geradas*[*i*], ";")

se *valores_duplicados_no_vector*(*vec_chaves_seq*) = *verdadeiro* **então**

vec_seqs_circulares ← *insere_valor_em_vector*(*vec_novas_seqs_geradas*[*i*], ,*vec_seqs_circulares*)

vec_novas_seqs_geradas ← *remove_elemento_do_vector*(*vec_novas_seqs_geradas*[*i*], ,*vec_novas_seqs_geradas*)

fim se

fim repetir

vec_seqs_geradas ← *vec_novas_seqs_geradas*

fim se

fim repetir

retorna *vec_seqs_circulares*

fim

Nota: A finalidade da função *insere_valor_em_vector* encontra-se definida na função *gerar_sequências*.

Função *expandir_seqs_mais_um_tuplo*(*vec_seqs_geradas*) $\stackrel{\text{def}}{=}$

seja *extrai_valores_ultima_chave_seq*(***sequência***) uma função que recebe uma *string* com a sequência de valores da chave primária/estrangeira resultantes do auto-relacionamento existente. Esta função retorna os valores da última chave primária/estrangeira desta sequência.

início

repetir para $i \leftarrow 1$ **até** *dimensão*(*vec_seqs_geradas*)

vec_valores_chave ← *extrai_valores_ultima_chave_seq*(*vec_seqs_geradas*[*i*])

$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_m}\left(\sigma_{a_1=vec\_valores\_chave[i]\wedge\ldots\wedge a_m=vec\_valores\_chave[m]}(\mathcal{R})\right)$

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

repetir enquanto $t \neq$ *null*

vec_novas_seqs_geradas ←*insere_valor_em_vector*(*vec_seqs_geradas*[*i*] o ";"o $t.c_1$ o ","o … o "," o $t.c_m$, *vec_novas_seqs_geradas*)

seq_actual ← $t.a_1$ o "," o … o "," o $t.a_m$ o ";" o $t.c_1$ o "," o … o "," o $t.c_m$

se *existe_valor_em_vector_ordenado*(*seq_actual*, *vec_seq_iniciais*) = *falso* **então**

vec_seq_iniciais ← *insere_valor_em_vector_ordenado*(*seq_actual*, *vec_seq_iniciais*)

fim se

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

fim repetir

vec_novas_seqs_geradas ← *insere_valor_em_vector*(*vec_seqs_geradas*[*i*], *vec_novas_seqs_geradas*)

fim repetir

retorna *vec_novas_seqs_geradas*

fim

Nota: A finalidade das funções *existe_valor_em_vector_ordenado*, *insere_valor_em_vector_ordenado* e *insere_valor_em_vector* encontra-se definida na função *gerar_sequências*.

5.5.3 Tuplos duplicados

Formalização Sintáctica

DETECT DUPLICATE-TUPLES DasTabelas DasBasesDados DefiniçãoColunas
UsandoChavesOrdenação UsandoAtribuições
DefiniçãoCondição

Formalização Semântica

No caso da detecção de tuplos duplicados com base numa condição susceptível de ser expressa em SQL, é efectuado um *produto cartesiano parcial* (*i.e.*, o mesmo par de tuplos apenas é comparado uma única vez) sobre os tuplos da relação em questão (*i.e.*, $\mathcal{R}$) para identificar os que respeitam a condição definida pelo utilizador e que assim são considerados duplicados. A relação que resulta desta OD (*i.e.*, $\mathcal{Z}1$) armazena as chaves primárias dos tuplos identificados como duplicados, assim como o *cluster* que estes formam. Nesta relação, um dado tuplo apenas é armazenado uma única vez, apesar das múltiplas situações em que possa ser identificado como duplicado de outros tuplos. Todos estes tuplos pertencem ao mesmo *cluster* de duplicados. A semântica subjacente a esta operação é a seguir apresentada:

Algoritmo Detecção_Tuplos_Duplicados_Baseada_em_SQL $\stackrel{\text{def}}{=}$

seja $\mathcal{Z}1$ a relação que armazena as chaves primárias dos tuplos de $\mathcal{R}$ identificados como sendo duplicados, bem como o respectivo *cluster* de duplicados a que pertencem. O esquema de $\mathcal{Z}1$ é o seguinte: $\mathcal{Z}1(a_1,\ldots,a_m,cluster\_duplicados)$.

início

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_m}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd_{((c_1,\ldots,c_m)>(a_1,\ldots,a_m))}\left(\rho_{\mathcal{R}2(c_1\leftarrow a_1,\ldots,c_m\leftarrow a_m)}(\mathcal{R})\right)\right)\right)$$

$idcluster = 1$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$$\mathcal{U} \leftarrow \pi_{idcluster}\left(\sigma_{a_1=t.a_1\wedge\ldots\wedge a_m=t.a_m}(\mathcal{Z}1)\right)$$

$$\mathcal{V} \leftarrow \pi_{idcluster}\left(\sigma_{a_1=t.c_1\wedge\ldots\wedge a_m=t.c_m}(\mathcal{Z}1)\right)$$

$u \leftarrow le\_tuplo(\mathcal{U})$

$v \leftarrow le\_tuplo(\mathcal{V})$

se $u = null$ **e** $v = null$ **então**

idcluster ← *idcluster* + 1

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(t.a_1,\ldots,t.a_m,idcluster)\}$

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(t.c_1,\ldots,t.c_m,idcluster)\}$

senão se $u \neq null$ **e** $v = null$ **então**

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(t.c_1,\ldots,t.c_m,u.idcluster)\}$

senão se $u = null$ **e** $v \neq null$ **então**

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(t.a_1,\ldots,t.a_m,v.idcluster)\}$

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: Na formalização anterior, *cond*, além de representar uma eventual condição de selecção que delimita a execução da OD, também representa a condição de que resulta a detecção dos tuplos duplicados. O conteúdo de *cond* resulta do que se encontra especificado na cláusula *where* da OD.

No caso da detecção de tuplos duplicados pode ser necessário recorrer a FDU (*e.g.*: em virtude do critério de detecção de duplicados ser demasiado complexo para ser expresso numa condição SQL). Nestes casos, os valores dos atributos que constituem a chave primária de cada tuplo da relação (*i.e.*, $\mathcal{R}$) e os valores dos atributos envolvidos na detecção dos duplicados (definidos pelo utilizador) de cada tuplo podem ser passados como parâmetros, de acordo com os requisitos de cada FDU que consta da sequência de atribuições especificadas na OD. As funções também podem receber outros parâmetros como variáveis e constantes. No caso da detecção de duplicados ser efectuada com base em FDU, a formalização anterior é alvo das seguintes alterações/acrescentos. Tudo o resto (representado sob a forma de reticências) permanece inalterado.

Algoritmo Detecção_Tuplos_Duplicados_Baseada_em_FDU $\stackrel{def}{=}$

seja $\mathcal{A} = \{a_1,\ldots,a_m,a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos que fazem parte de $\mathcal{R}$.

seja C o conjunto de atributos envolvidos na detecção de duplicados, assim definido: $C = \{c \mid c$ é usado na detecção dos tuplos duplicados de $\mathcal{R} \wedge c \in \mathcal{A}\}$, *i.e.*, $C \subset \mathcal{A}$.

seja o a cardinalidade de C.

início

$$\mathcal{T} \leftarrow \mathcal{R}.a_1,\ldots,\mathcal{R}.a_m,\mathcal{R}2.b_1,\ldots\mathcal{R}2.b_m,\mathcal{R}.c_1,\ldots,\mathcal{R}.c_o,\mathcal{R}2.c_1,\ldots,\mathcal{R}2.c_o(\sigma_{cond}(\mathcal{R} \triangleright\!\triangleleft_{((\mathcal{R}_2.b_1,\ldots,\mathcal{R}_2.b_m)> > (\mathcal{R}.a_1,\ldots,\mathcal{R}.a_m))} (\rho_{\mathcal{R}2(b_1 \leftarrow a_1,\ldots,b_m \leftarrow a_m)}(\mathcal{R}))))$$

idcluster = 1

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$v_1 \leftarrow fdu_1\left(p_{1,1},p_{1,2},...,p_{1,\chi}\right)$

$v_2 \leftarrow fdu_2\left(p_{2,1},p_{2,2},...,p_{2,y}\right)$

$\vdots$

$v_j \leftarrow fdu_j\left(p_{j,1},p_{j,2},...,p_{j,w}\right)$

se $avalia\_condição\left(p_{k,1},p_{k,2},...,p_{k,z}\right) = verdadeiro$ **então**

$\mathcal{U} \leftarrow \pi_{idcluster}\left(\sigma_{a_1=t.a_1 \wedge ... \wedge a_m=t.a_m}(\mathcal{Z}1)\right)$

$\mathcal{V} \leftarrow \pi_{idcluster}\left(\sigma_{a_1=t.b_1 \wedge ... \wedge a_m=t.b_m}(\mathcal{Z}1)\right)$

$\vdots$

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: Como resultado, cada função fdu_χ retorna um valor atómico.

A detecção de tuplos duplicados pode ser efectuada com base no Método da Vizinhança Ordenada com Multi-Passagem (MVO-MP) [Hernández e Stolfo, 1995] [Hernández e Stolfo, 1998]. Este método encontra-se descrito no Capítulo 3 – Secção 3.4.1.3. Seguidamente, fornece-se a semântica que possibilita uma possível implementação do método.

Algoritmo Detecção_Tuplos_Duplicados_Baseada_em_MVO-MP $\stackrel{def}{=}$

seja *extrai_caracteres*(*pos_carac, str*) uma função que recebe uma string *str* e uma outra string *pos_carac* que contém as posições dos caracteres a extrair de *str*, separados por vírgula. A função procede à extracção dos caracteres de *str* que se encontram nas posições *pos_carac*, devolvendo a substring resultante.

seja *dimjanela* a dimensão ou tamanho da "janela deslizante" usada nas comparações entre tuplos vizinhos, típica do método da vizinhança ordenada.

seja *mat_chvord* uma matriz em que cada linha armazena uma chave de ordenação. Nas colunas impares da matriz encontram-se as posições dos caracteres a extrair e nas colunas pares o nome respectivo do atributo.

seja $\mathcal{Z}1$ a relação que armazena as chaves primárias dos tuplos de $\mathcal{R}$ identificados como sendo duplicados, bem como o respectivo *cluster* de duplicados a que pertencem. O esquema de $\mathcal{Z}1$ é o seguinte: $\mathcal{Z}1(a_1,\ldots,a_m,cluster\_duplicados)$.

seja $\mathcal{Z}_{temp1}$ a relação auxiliar que armazena a chave de ordenação para cada tuplo de $\mathcal{R}$, com o seguinte esquema: $\mathcal{Z}_{temp1}(a_1,\ldots,a_m,chaveord)$

seja $\mathcal{Z}_{temp2}$ a relação auxiliar usada na detecção dos tuplos duplicados, possuindo um esquema igual a $\mathcal{Z}1$.

início

idcluster ← 1

repetir para *i* ← 1 **até** *numlinhas*(*mat_chvord*)

$Z_{temp1} \leftarrow \emptyset$

$Z_{temp2} \leftarrow \emptyset$

$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,mat\_chvord[i][2],mat\_chvord[i][4],\ldots,mat\_chvord[i][numcolunas(mat\_chvord)]}(\sigma_{cond}(\mathcal{R}))$

t ← *le_tuplo*($\mathcal{T}$)

repetir enquanto *t* ≠ *null*

chaveord ← “”

repetir para *j* ← 1 **até** *numcolunas*(*mat_chvord*) **incremento** 2

chaveord ← *chaveord* o *extrai_caracteres*(*mat_chvord*[*i*][*j*], *mat_chvord*[*i*][*j*+1])

fim repetir

$Z_{temp1} \leftarrow Z_{temp1} \cup \{(t.a_1,\ldots,t.a_m,v.chaveord)\}$

t ← *le_tuplo*($\mathcal{T}$)

fim repetir

$\mathcal{U} \leftarrow \tau_{chaveord}(Z_{temp1})$

nrtuplos ← 0

u ← *le_tuplo*($\mathcal{U}$)

repetir enquanto *u* ≠ *null* **e** *nrtuplos* < *dimjanela*

nrtuplos ← *nrtuplos* + 1

$\mathcal{V} \leftarrow \sigma_{a_1=u.a_1 \wedge \ldots \wedge a_n=u.a_n}(\mathcal{T})$

v ← *le_tuplo*(v)

repetir para *j* ← 1 **até** *n*

mat_valores[*nrtuplos*][*j*] ← $v.a_j$

fim repetir

ind ← *n*

repetir para *j* ← 2 **até** *numcolunas*(*mat_chvord*) **incremento** 2

ind ← *ind* + 1

mat_valores[*nrtuplos*][*ind*] ← *v.mat_chvord*[*i*][*j*]

fim repetir

u ← *le_tuplo*($\mathcal{U}$)

fim repetir

repetir para $j \leftarrow 1$ **até** $dimjanela - 1$

repetir para $l \leftarrow j + 1$ **até** $nrtuplos$

$$v_1 = fdu_1\left(mat\_valores[j][x], mat\_valores[l][x], p_{1,1}, p_{1,2}, ..., p_{1,f}\right)$$

$$v_2 = fdu_2\left(mat\_valores[j][y], mat\_valores[l][y], p_{2,1}, p_{2,2}, ..., p_{2,g}\right)$$

$$\vdots$$

$$v_o = fdu_o\left(mat\_valores[j][w], mat\_valores[l][w], p_{o,1}, p_{o,2}, ..., p_{o,h}\right)$$

se $avalia\_condição\left(p_{r,1}, p_{r,2}, ..., p_{r,z}\right) = verdadeiro$ **então**

$idcluster \leftarrow insere\_duplicados\_relação\_output(mat\_valores, j, l, idcluster, ind)$

fim se

fim repetir

fim repetir

$u \leftarrow le\_tuplo(\mathcal{U})$

repetir enquanto $u \neq null$ **e** $nrtuplos < dimjanela$

repetir para $j \leftarrow 1$ **até** $dimjanela - 1$

repetir para $l \leftarrow 1$ **até** ind

$mat\_valores[j][l] \leftarrow v.mat\_chvord[j+1][l]$

fim repetir

fim repetir

$\mathcal{V} \leftarrow \sigma_{a_1 = u.a_1 \wedge ... \wedge a_n = u.a_n}(\mathcal{T})$

$v \leftarrow le\_tuplo(v)$

repetir para $j \leftarrow 1$ **até** n

$mat\_valores[dimjanela][j] \leftarrow v.a_j$

fim repetir

$ind \leftarrow n$

repetir para $j \leftarrow 2$ **até** $numcolunas(mat\_chvord)$ **incremento** 2

$ind \leftarrow ind + 1$

$mat\_valores[dimjanela][ind] \leftarrow v.mat\_chvord[i][j]$

fim repetir

repetir para $j \leftarrow 1$ **até** $dimjanela - 1$

$$v_1 = fdu_1\left(mat\_valores[j][x], mat\_valores[dimjanela][x], p_{1,1}, p_{1,2}, ..., p_{1,f}\right)$$

$$v_2 = fdu_2\left(mat\_valores[j][y], mat\_valores[dimjanela][y], p_{2,1}, p_{2,2}, ..., p_{2,g}\right)$$

$$\vdots$$

$v_o = fdu_o\left(mat\_valores[j][w], mat\_valores[dimjanela][w], p_{o,1}, p_{o,2}, ..., p_{o,h}\right)$

se $avalia\_condição\left(p_{r,1}, p_{r,2}, ..., p_{r,z}\right) = verdadeiro$ **então**

$idcluster \leftarrow insere\_duplicados\_relação\_output(mat\_valores, j, dimjanela, idcluster, ind)$

fim se

fim repetir

$u \leftarrow le\_tuplo(\mathcal{U})$

fim repetir

$$\mathcal{X} \leftarrow \pi_{idcluster\,1, a_1, ..., a_m, b_1, ..., b_r, idcluster\,2, c_1, ..., c_m, d_1, ..., d_r}(\mathcal{Z}_{temp2} \rhd\lhd ((c_1, ..., c_m) > (a_1, ..., a_m) \wedge \wedge\ idcluster\,2 > idcluster\,1)(\rho_{\mathcal{Z}_{temp2\_aux}(c_1 \leftarrow a_1, ..., c_m \leftarrow a_m, d_1 \leftarrow b_1, ..., d_r \leftarrow b_r)}(\mathcal{Z}_{temp2\_aux})))$$

$x \leftarrow le\_tuplo(\mathcal{X})$

repetir enquanto $x \neq null$ **e** $nrtuplos < dimjanela$

$v_1 \leftarrow fdu_1\left(x.b_1, x.d_1, p_{1,1}, p_{1,2}, ..., p_{1,f}\right)$

$v_2 \leftarrow fdu_2\left(x.b_2, x.d_2, p_{2,1}, p_{2,2}, ..., p_{2,g}\right)$

$\vdots$

$v_o \leftarrow fdu_o\left(x.b_r, x.d_r, p_{o,1}, p_{o,2}, ..., p_{o,h}\right)$

se $avalia\_condição\left(p_{s,1}, p_{s,2}, ..., p_{s,z}\right) = verdadeiro$ **então**

$\mathcal{Z}_{temp2} \leftarrow \mathcal{Z}_{temp2} - \left(\sigma_{a_1 = x.a_1 \wedge ... \wedge a_m = x.a_m}\left(\mathcal{Z}_{temp2}\right)\right)$

$\mathcal{Z}_{temp2} \leftarrow \mathcal{Z}_{temp2} \cup \{(x.a_1, \ldots, x.a_m, x.b_1, \ldots, x.b_r, x.idcluster2)\}$

fim se

$x \leftarrow le\_tuplo(\mathcal{X})$

fim repetir

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \mathcal{Z}_{temp2}$

fim repetir

fim

Notas: Em cada função fdu_x dois dos parâmetros são obrigatoriamente os que contêm os valores sujeitos a comparação e que se encontram na estrutura de dados *mat_valores*.

Como resultado, cada função fdu_x retorna um valor atómico.

Função *insere_duplicados_relação_output*(*mat_valores*, *linha1*, *linha2*, *idcluster*, *ind*) $\stackrel{\text{def}}{=}$

início

$\mathcal{T} \leftarrow \pi_{idcluster}\left(\sigma_{a_1=mat\_valores[linha1][1] \wedge \ldots \wedge a_n=mat\_valores[linha1][n]}\left(\mathcal{Z}_{temp2}\right)\right)$

$\mathcal{U} \leftarrow \pi_{idcluster}\left(\sigma_{a_1=mat\_valores[linha2][1] \wedge \ldots \wedge a_n=mat\_valores[linha2][n]}\left(\mathcal{Z}_{temp2}\right)\right)$

$t \leftarrow le\_tuplo(\mathcal{T})$

$u \leftarrow le\_tuplo(\mathcal{U})$

se $t = null$ **e** $u = null$ **então**

$idcluster \leftarrow idcluster + 1$

$\mathcal{Z}_{temp2} \leftarrow \mathcal{Z}_{temp2} \cup \{(mat\_valores[linha1][1],\ldots,mat\_valores[linha1][n],\ldots,$
$mat\_valores[linha1][ind], idcluster)\}$

$\mathcal{Z}_{temp2} \leftarrow \mathcal{Z}_{temp2} \cup \{(mat\_valores[linha2][1],\ldots,mat\_valores[linha2][n],\ldots,$
$mat\_valores[linha2][ind] , idcluster)\}$

senão se $t \neq null$ **e** $u = null$ **então**

$\mathcal{Z}_{temp2} \leftarrow \mathcal{Z}_{temp2} \cup \{(mat\_valores[linha2][1],\ldots,mat\_valores[linha2][n],\ldots,$
$mat\_valores[linha2][ind], t.idcluster,)\}$

senão se $t = null$ **e** $u \neq null$ **então**

$\mathcal{Z}_{temp2} \leftarrow \mathcal{Z}_{temp2} \cup \{(mat\_valores[linha1][1],\ldots,mat\_valores[linha1][n],\ldots,$
$mat\_valores[linha1][ind], u.idcluster,)\}$

fim se

retorna *idcluster*

fim

5.5.4 Violação de restrição de integridade

Formalização Sintáctica

DETECT INTEGRITY-CONSTRAINT-VIOLATION AT LEVEL OF RELATION
ColunasAlvo DasTabelas
DasBasesDados DefiniçãoAtribuições
DefiniçãoCondição AgupandoPor

Formalização Semântica

No caso da detecção de violação de restrição de integridade ser efectuada com base numa operação passível de ser convertida num inquérito SQL, a operação envolve uma das seguintes situações: identificar as chaves primárias dos tuplos e os respectivos valores dos atributos da

relação que constituem uma violação de restrição de integridade; ou, identificar apenas o valor ou valores agregados que materializam a existência de uma violação de restrição de integridade. A semântica subjacente à operação é apresentada de seguida.

Algoritmo Detecção_Violação_Restrição_Integridade_Baseada_em_SQL $\stackrel{\text{def}}{=}$

seja $\mathcal{A} = \{a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos de $\mathcal{R}$ que não fazem parte da sua chave primária.

seja C o conjunto de atributos envolvidos na restrição de integridade, assim definido: $C = \{c \mid c$ é usado na formulação da restrição de integridade $\wedge\ c \in \mathcal{A}\}$, *i.e.*, $C \subseteq \mathcal{A}$.

seja o a cardinalidade de C.

seja *func_agrega* uma função de agregação existente em SQL (*e.g.*: *count*; *sum*; *avg*).

seja $\mathcal{Z}1$ a relação que armazena o valor que constitui violação à restrição de integridade, com o seguinte esquema: $\mathcal{Z}1(a_1,\ldots,a_m,c_1,\ldots,c_o)$.

início

$$\mathcal{Z}1 \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_o}\left(\sigma_{cond}\left({}_{c_1,\ldots,c_o}\gamma_{func\_agrega(c_i)}\left(\mathcal{R} \rhd\lhd_{\substack{(\mathcal{R}.a_1,\ldots,\mathcal{R}.a_m)>\\(\mathcal{R}2.a_1,\ldots,\mathcal{R}2.a_m)}} \left(\rho_{\mathcal{R}2}(\mathcal{R})\right)\right)\right)\right)$$

fim

Notas: Nesta formalização aplicam-se as mesmas notas do que as apresentadas no âmbito da formalização da OD de violação de restrição de integridade baseada em SQL, no contexto dos múltiplos valores do atributo (Secção 5.3.2.3).

No caso da OD não implicar a realização de um produto cartesiano sobre $\mathcal{R}$ (o que constitui a situação mais usual), este (*i.e.*, $\mathcal{R} \rhd\lhd_{(\ldots)} \left(\rho_{\mathcal{R}2}(\mathcal{R})\right)$) é substituído apenas por $\mathcal{R}$.

Na detecção de violação de restrição de integridade ao nível da relação pode ser necessário recorrer a FDU (*e.g.*: em virtude da complexidade inerente à OD). Nestes casos, o conjunto de valores de cada atributo da relação que se encontra envolvido na formulação da restrição de integridade pode ser fornecido como parâmetro, respeitando os requisitos de cada FDU presente na sequência de atribuições especificadas na OD. Variáveis e constantes também podem ser fornecidos como parâmetros das funções. Exceptuando a última função (*i.e.*, fdu_j) que obrigatoriamente retorna um valor atómico representativo do problema detectado, todas as outras podem retornar um valor atómico ou uma estrutura de dados (*e.g.*: um vector; uma matriz). A formalização da detecção deste PQD envolvendo FDU é, a seguir, exposta.

Algoritmo Detecção_Violação_Restrição_Integridade_Baseada_em_FDU $\stackrel{\text{def}}{=}$

seja $Z1$ a relação que armazena o valor que constitui violação à restrição de integridade, com o seguinte esquema: $Z1(valor)$. O tipo de dados de *valor* é o mesmo do que o retornado pela última função (no caso, fdu_j) da sequência de FDU na OD.

início

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_o}\left(\sigma_{cond}\left(\mathcal{R}\right)\right)$$

$$v_1 \leftarrow fdu_1\left(p_{1,1},p_{1,2},\ldots,p_{1,x}\right)$$

$$v_2 \leftarrow fdu_2\left(p_{2,1},p_{2,2},\ldots,p_{2,y}\right)$$

$$\vdots$$

$$v_j \leftarrow fdu_j\left(p_{j,1},p_{j,2},\ldots,p_{j,w}\right)$$

se $avalia\_condição\left(p_{k,1},p_{k,2},\ldots,p_{k,z}\right) = verdadeiro$ **então**

$Z1 \leftarrow Z1 \cup \{(v_j)\}$

fim se

fim

Nota: O conjunto de todos os valores dos atributos envolvidos (no caso, $\mathcal{T}$) constitui um parâmetro de uma ou mais funções fdu_x

No caso da última função da sequência de FDU (*i.e.*, fdu_j) retornar uma estrutura de dados do tipo unidimensional (*i.e.*, um vector), a formalização anterior é alvo das alterações/acrescentos que a seguir se apresentam. O que permanece inalterado encontra-se representado sob a forma de reticências.

Algoritmo Detecção_Violação_Restrição_Integridade $\stackrel{\text{def}}{=}$

seja $Z1$ a relação que armazena os valores que constituem violações à restrição de integridade, com o seguinte esquema: $Z1(valor)$. O tipo de dados de *valor* é o mesmo do que o retornado pela última função (no caso, fdu_j) da sequência de FDU na OD.

início

$$\vdots$$

$$v_j \leftarrow fdu_j\left(p_{j,1},p_{j,2},\ldots,p_{j,w}\right)$$

repetir para $i \leftarrow 1$ **até** $dimensão(v_j)$

$Z1 \leftarrow Z1 \cup \{(v_j[i])\}$

fim repetir

fim

No caso da última função da sequência (*i.e.*, fdu_j) retornar uma estrutura de dados bidimensional (*i.e.*, uma matriz), a formalização inicial desta operação sofre as seguintes alterações/acrescentos.

Algoritmo Detecção_Violação_Restrição_Integridade $\stackrel{\text{def}}{=}$

seja $\mathcal{Z}1$ a relação que armazena os valores dos atributos dos tuplos que constituem violações à restrição de integridade, com o seguinte esquema: $\mathcal{Z}1(a_x,\ldots,a_z)$.

início

$\vdots$

repetir para $h \leftarrow 1$ **até** $numlinhas(v_j)$

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(v_j[h][1], \ldots, v_j[h][numcolunas])\}$

fim repetir

fim

Nota: Entre os atributos de $\mathcal{Z}1$ podem encontrar-se os que constituem a chave primária de $\mathcal{R}$.

5.6 Problemas ao Nível de Múltiplas Relações/Fontes de Dados

Nesta secção apresenta-se a formalização das OD dos PQD que ocorrem ao nível de múltiplas relações, independentemente destas pertencerem a uma só ou a múltiplas bases de dados.

5.6.1 Heterogeneidade de sintaxes

Formalização Sintáctica

DETECT HETEROGENEITY-OF-SYNTAXES ColunasAlvo DasTabelas DasBasesDados
UsandoCondição

Formalização Semântica

A detecção deste PQD envolve a invocação de uma função responsável por inferir a sintaxe que se encontra subjacente aos valores de cada um dos atributos em causa (*i.e.*, atr_1 e atr_2) pertencentes, respectivamente, a cada uma das relações (*i.e.*, $\mathcal{R}$ e $\mathcal{S}$). No caso de não ser possível inferir a sintaxe subjacente, face à inexistência de um padrão comum, a função retorna o valor *null* como resultado. As duas relações que eventualmente resultam desta OD (*i.e.*, $\mathcal{Z}1$ e $\mathcal{Z}2$) armazenam, respectivamente, as sintaxes diferentes identificadas em atr_1 e atr_2 ou reflectem a impossibilidade de se inferir a sintaxe nalgum dos atributos ou até em ambos. A inexistência das relações $\mathcal{Z}1$ e $\mathcal{Z}2$ após a execução desta OD significa que não há heterogeneidade de sintaxes.

Algoritmo Detecção_Heterogeneidade_Sintaxes $\stackrel{def}{=}$

seja *infere_sintaxe*(*vec*[ ]) uma função que recebe num vector *vec*[ ] o conjunto de valores de um atributo, infere a sintaxe que se lhes encontra subjacente e devolve-a como resultado. No caso de não ser possível inferir uma sintaxe (*i.e.*, devido à inexistência de um padrão), a função devolve como resultado o valor *null*.

sejam $\mathcal{Z}1$ e $\mathcal{Z}2$ duas relações onde são armazenadas, respectivamente, as sintaxes heterogéneas identificadas em atr_1 e atr_2, com o seguinte esquema: $\mathcal{Z}1$(*sint*) e $\mathcal{Z}2$(*sint*), em que *sint* é o atributo que armazena a sintaxe inferida.

início

$\mathcal{T} \leftarrow \pi_{atr_1}(\sigma_{cond}(\mathcal{R}))$

$\mathcal{U} \leftarrow \pi_{atr_2}(\sigma_{cond}(\mathcal{S}))$

$sint_1 \leftarrow infere\_sintaxe(\mathcal{T})$

$sint_2 \leftarrow infere\_sintaxe(\mathcal{U})$

se $sint_1 \neq sint_2$ **ou** ($sint_1 = null$ **e** $sint_2 = null$) **então**

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(sint_1)\}$

$\mathcal{Z}2 \leftarrow \mathcal{Z}2 \cup \{(sint_2)\}$

fim se

fim

5.6.2 Heterogeneidade de unidades de medida

Formalização Sintáctica

DETECT HETEROGENEITY-OF-MEASURE-UNITS ColunasAlvo DasTabelas
DasBasesDados DefiniçãoLimiar
UsandoAtribuições DefiniçãoCondição

Formalização Semântica

No caso da detecção da existência de heterogeneidade de unidades de medida com base numa condição susceptível de ser representada em SQL, a operação envolve verificar para cada par de tuplos identificado como duplicados pela OD respectiva (previamente executada), se a condição que traduz a existência de heterogeneidade entre atr_1 e atr_2 é respeitada ou não. No caso de existir heterogeneidade de unidades de medida entre os valores, estes são armazenados nas relações respectivas (*i.e.*, $\mathcal{Z}1$ e $\mathcal{Z}2$), juntamente com os valores das chaves primárias dos tuplos a que pertencem. No caso da não existência da heterogeneidade entre os valores, estes são armazenados noutras duas relações (no caso, $\mathcal{Z}1_{\text{não heterog}}$ e $\mathcal{Z}2_{\text{não heterog}}$), assim como os valores das chaves primárias dos tuplos onde se encontram. Quando o quociente entre o número de

situações em que não há heterogeneidade e o número de duplicados existentes excede um determinado limiar definido pelo utilizador, conclui-se que não há heterogeneidade de unidades de medida. Sempre que isto acontece, não se justifica a continuação da execução da OD, pelo que esta é interrompida. A semântica subjacente a esta operação é a seguir apresentada.

Algoritmo Detecção_Heterogeneidade_Unidades_Medida_Baseada_SQL $\stackrel{\text{def}}{=}$

seja *limiar* um valor definido pelo utilizador cuja finalidade é servir de referencial na verificação da existência de heterogeneidade de unidades de medida entre atr_1 e atr_2. Se este valor for superado isso significa que não existe heterogeneidade e vice-versa.

seja $\mathcal{Z}1_i$ e $\mathcal{Z}2_i$ as relações onde se encontram armazenadas as chaves primárias dos tuplos, respectivamente, de $\mathcal{R}$ e $\mathcal{S}$ identificados como sendo duplicados e o respectivo *cluster* que formam. Os esquemas de $\mathcal{Z}1_i$ e $\mathcal{Z}2_i$ são: $\mathcal{Z}1_i(a_1,\ldots,a_m,idcluster)$ e $\mathcal{Z}2_i(b_1,\ldots,b_r,idcluster)$

seja $\mathcal{Z}1$ e $\mathcal{Z}2$ as relações que armazenam as chaves primárias dos tuplos, respectivamente, de $\mathcal{R}$ e $\mathcal{S}$ onde foi detectada a existência de heterogeneidade de unidades de medida entre atr_1 e atr_2, assim como os respectivos valores destes atributos. Os esquemas de $\mathcal{Z}1$ e $\mathcal{Z}2$ são: $\mathcal{Z}1(nrseq,a_1,\ldots,a_m,atr_1)$ e $\mathcal{Z}2(nrseq,b_1,\ldots,b_r,atr_2)$. O atributo *nrseq* permite estabelecer uma associação entre cada heterogeneidade de unidades de medida identificada entre atr_1 e atr_2.

seja $\mathcal{Z}1_{\text{não heterog}}$ e $\mathcal{Z}2_{\text{não heterog}}$ as relações que armazenam as chaves primárias dos tuplos, respectivamente, de $\mathcal{R}$ e $\mathcal{S}$ onde a heterogeneidade de unidades de medida entre atr_1 e atr_2 não se verifica, bem como os respectivos valores destes atributos. Os esquemas de $\mathcal{Z}1_{\text{não heterog}}$ e $\mathcal{Z}2_{\text{não heterog}}$ são: $\mathcal{Z}1_{\text{não heterog}}(nrseq,a_1,\ldots,a_m,atr_1)$ e $\mathcal{Z}2_{\text{não heterog}}(nrseq,b_1,\ldots,b_r,atr_2)$. A finalidade do atributo *nrseq* é a mesma do que a enunciada no ponto anterior.

início

$nrduplicados \leftarrow \gamma_{count(\mathcal{Z}1_i.idcluster)}\left(\mathcal{Z}1_i \rhd\lhd \mathcal{Z}2_i\right)$

$heterog \leftarrow 0$

$nãoheterog \leftarrow 0$

$\mathcal{T} \leftarrow \pi_{idcluster}\left(\mathcal{Z}1_i\right)$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$ **e** $nãoheterog/nrduplicados \leq limiar$

$\mathcal{U} \leftarrow \pi_{a_1,\ldots,a_m,b_1,\ldots,b_r}\left(\mathcal{Z}1_i \rhd\lhd_{(\mathcal{Z}1_i.idcluster=t.idcluster \wedge \mathcal{Z}2_i.idcluster=t.idcluster)} \mathcal{Z}2_i\right)$

$u \leftarrow le\_tuplo(\mathcal{U})$

repetir enquanto $u \neq null$ **e** $nãoheterog/nrduplicados \leq limiar$

$\mathcal{V} \leftarrow \pi_{a_1,\ldots,a_m,b_1,\ldots,b_r,atr_1,atr_2}\left(\mathcal{R} \rhd\lhd_{(a_1=u.a_1\wedge\ldots\wedge a_m=u.a_m\wedge b_1=u.b_1\wedge\ldots\wedge b_r=u.b_r\wedge cond)} \mathcal{S}\right)$

$v \leftarrow le\_tuplo(\mathcal{V})$

se $v \neq null$ **então**

$heterog \leftarrow heterog + 1$

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(heterog,v.a_1,\ldots,v.a_m,v.atr_1)\}$

$\mathcal{Z}2 \leftarrow \mathcal{Z}2 \cup \{(heterog,v.b_1,\ldots,v.b_r,v.atr_2)\}$

senão

$\mathcal{V} \leftarrow \pi_{a_1,\ldots,a_m,b_1,\ldots,b_r,atr_1,atr_2}\left(\mathcal{R} \rhd\lhd_{(a_1=u.a_1 \wedge \ldots \wedge a_m=u.a_m \wedge b_1=u.b_1 \wedge \ldots \wedge b_r=u.b_r \wedge \neg cond')} S\right)$

$v \leftarrow le\_tuplo(\mathcal{V})$

se $v \neq null$ **então**

$não heterog \leftarrow não heterog + 1$

$\mathcal{Z}1_{\text{não heterog}} \leftarrow \mathcal{Z}1_{\text{não heterog}} \cup \{(não heterog, v.a_1, \ldots, v.a_m, v.atr_1)\}$

$\mathcal{Z}1_{\text{não heterog}} \leftarrow \mathcal{Z}1_{\text{não heterog}} \cup \{(não heterog, v.b_1, \ldots, v.b_r, v.atr_2)\}$

fim se

fim se

$u \leftarrow le\_tuplo(\mathcal{U})$

fim repetir

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

se $nrduplicados/não heterog > 1 - limiar$ **então** (se não há heterogeneidade de unidades de medida, então há que eliminar os tuplos que, eventualmente, tenham sido colocados nas duas relações de output)

$\mathcal{Z}1 \leftarrow \emptyset$

$\mathcal{Z}2 \leftarrow \emptyset$

fim se

fim

Nota: Na detecção da existência de heterogeneidade de unidades de medida são considerados todos os tuplos de cada *cluster* de duplicados. Não é possível delimitar a execução da OD apenas a um subconjunto destes tuplos. A existir heterogeneidade de unidades de medida, esta deve verificar-se em todos os tuplos dos *clusters* ou, pelo menos, num número elevado de tuplos que supere o limiar estipulado pelo utilizador.

A detecção da existência de heterogeneidade de unidades de medida pode envolver a utilização de FDU. Neste caso, o valor do atributo atr_1 pode ser passado como parâmetro, de acordo com os requisitos específicos de cada FDU que consta da sequência de atribuições especificada na OD pelo utilizador. Outros parâmetros, como variáveis e constantes também podem ser passados para as funções. Como resultado, cada função fdu_χ retorna um valor atómico. A última função da sequência é responsável por retornar o valor que será objecto de comparação com o valor do atributo atr_2. Caso os valores sejam iguais, isso significa que a transformação operada no valor de atr_1 confirma a existência de heterogeneidade de unidades de medida com o valor de atr_2 e vice-versa. A formalização anterior sofre as seguintes alterações/acrescentos para reflectir a utilização de FDU. O que permanece inalterado encontra-se representado sob a forma de reticências.

Algoritmo Detecção_Heterogeneidade_Unidades_Medida_Baseada_FDU

⋮

início

$\vdots$

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

repetir enquanto $t \neq$ *null* **e** *nrduplicados*/ *nãoheterog* ≤ 1 - *limiar*

$\mathcal{V} \leftarrow \pi_{a_1,\ldots,a_m,b_1,\ldots,b_r,atr_1,atr_2}\left(\mathcal{R} \rhd\lhd_{(a_1=u.a_1 \wedge \ldots \wedge a_m=u.a_m \wedge b_1=u.b_1 \wedge \ldots \wedge b_r=u.b_r)} \mathcal{S}\right)$

$v \leftarrow$ *le_tuplo*($\mathcal{V}$)

$v_1 \leftarrow fdu_1\left(p_{1,1},p_{1,2},\ldots,p_{1,x}\right)$

$v_2 \leftarrow fdu_2\left(p_{2,1},p_{2,2},\ldots,p_{2,y}\right)$

$\vdots$

$v_o \leftarrow fdu_o\left(p_{o,1},p_{o,2},\ldots,p_{o,z}\right)$

se $v_o = v.atr_2$ **então**

heterog $\leftarrow$ *heterog* + 1

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(heterog, v.a_1,\ldots,v.a_m,v.atr_1)\}$

$\mathcal{Z}2 \leftarrow \mathcal{Z}2 \cup \{(heterog, v.b_1,\ldots,v.b_r,v.atr_2)\}$

senão

nãoheterog $\leftarrow$ *nãoheterog* + 1

$\mathcal{Z}1_{\text{nãoheterog}} \leftarrow \mathcal{Z}1_{\text{nãoheterog}} \cup \{(nãoheterog, v.a_1,\ldots,v.a_m,v.atr_1)\}$

$\mathcal{Z}2_{\text{nãoheterog}} \leftarrow \mathcal{Z}2_{\text{nãoheterog}} \cup \{(nãoheterog, v.b_1,\ldots,v.b_r,v.atr_2)\}$

fim se

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

fim repetir

$\vdots$

fim

Nota: O valor do atributo atr_1 constitui um parâmetro de uma ou mais funções fdu_x.

5.6.3 Existência de sinónimos

Formalização Sintáctica

DETECT EXISTENCE-OF-SYNONYMS ColunasAlvo DasTabelas DasBasesDados
DefiniçãoDicionário UsandoCondição

Formalização Semântica

A detecção deste PQD envolve verificar se os valores que compõem os pares resultantes de cada *cluster* de sinónimos que constam num determinado dicionário definido pelo utilizador existem simultaneamente nos dois atributos em questão (*i.e.*, atr_1 e atr_2) pertencentes, respectivamente, a cada uma das relações (*i.e.*, $\mathcal{R}$ e $\mathcal{S}$). Caso se confirme a sua existência, está-se perante um problema de sinónimos. Nas duas relações que eventualmente podem resultar da execução desta OD (*i.e.*, $\mathcal{Z}1$ e $\mathcal{Z}2$), além dos valores que constituem sinónimos entre atr_1 e atr_2 são também armazenadas as chaves primárias, respectivamente, dos tuplos de $\mathcal{R}$ e $\mathcal{S}$ onde estes se encontram, assim como o *cluster* de sinónimos a que pertencem. Em ambas as relações, a informação relativa a um sinónimo detectado é armazenada apenas uma única vez, independentemente do número de situações em que este seja identificado como sinónimo de outros valores.

Algoritmo Detecção_Existência_Sinónimos $\stackrel{\text{def}}{=}$

seja $\mathcal{T}$ a relação onde se encontra armazenado o dicionário de sinónimos, com esquema: $\mathcal{T}$(*idsin*, *idcluster*, *sin*), em que *idsin* constitui a chave primária, *idcluster* o *cluster* a que pertence o sinónimo e *sin* o termo sinónimo.

sejam $\mathcal{Z}1$ e $\mathcal{Z}2$ as relações que armazenam os valores das chaves primárias dos tuplos de $\mathcal{R}$ e $\mathcal{S}$ onde foi detectada a existência de sinónimos entre atr_1 e atr_2, os correspondentes sinónimos, bem como o respectivo *cluster* de sinónimos a que estes pertencem. Os esquemas de $\mathcal{Z}1$ e $\mathcal{Z}2$, respectivamente, são: $\mathcal{Z}1(a_1,\ldots,a_m,atr_1,idcluster)$ e $\mathcal{Z}2(b_1,\ldots,b_r,atr_2,idcluster)$.

início

$\mathcal{U} \leftarrow \pi_{idcluster}\left(\sigma_{cond}\left(\mathcal{T}\right)\right)$

$u \leftarrow le\_tuplo(\mathcal{U})$

repetir enquanto $u \neq null$

$$\mathcal{V} \leftarrow \pi_{sin,\,sin\,2}(\mathcal{T} \rhd\lhd_{(idcluster=u.idcluster \,\wedge\, idcluster\,2=u.idcluster \,\wedge\, idsin\,2>idsin)} (\rho_{\mathcal{S}2(idsin\,2\leftarrow idsin,\ ,idcluster\,2\leftarrow idcluster,\,sin\,2\leftarrow sin)}(\mathcal{T})))$$

$v \leftarrow le\_tuplo(\mathcal{V})$

repetir enquanto $v \neq null$

$$\mathcal{X} \leftarrow \pi_{a_1,\ldots,a_m,b_1,\ldots,b_r,atr_1,atr_2}(\mathcal{R} \rhd\lhd \rhd\lhd_{\begin{array}{r}(((atr_1=v.sin \,\vee\, atr_1 \text{ like } 'v.sin\%' \,\vee\, atr_1 \text{ like } '\%v.sin\%' \,\vee\, atr_1 \text{ like } '\%v.sin') \,\wedge \\ \wedge\, (atr_2=v.sin\,2 \,\vee\, atr_2 \text{ like } 'v.sin\,2\%' \,\vee\, atr_2 \text{ like } '\%v.sin\,2\%' \,\vee\, atr_2 \text{ like } '\%v.sin\,2')) \,\vee \\ \vee\, ((atr_1=v.sin\,2 \,\vee\, atr_1 \text{ like } 'v.sin\,2\%' \,\vee\, atr_1 \text{ like } '\%v.sin\,2\%' \,\vee\, atr_1 \text{ like } '\%v.sin\,2') \,\wedge \\ \wedge\, (atr_2=v.sin \,\vee\, atr_2 \text{ like } 'v.sin\%' \,\vee\, atr_2 \text{ like } '\%v.sin\%' \,\vee\, atr_2 \text{ like } '\%v.sin')))\ \mathcal{S})\end{array}}$$

$x \leftarrow le\_tuplo(\mathcal{X})$

repetir enquanto $x \neq null$

se $\sigma_{a_1=x.a_1 \wedge \ldots \wedge a_m=x.a_m}(\mathcal{Z}1) = \varnothing$ **então**

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(x.a_1,\ldots,x.a_m,x.atr_1,u.idcluster)\}$

fim se

se $\sigma_{b_1=x.b_1 \wedge \ldots \wedge br=x.b_r}(\mathcal{Z}2) = \varnothing$ **então**

$\mathcal{Z}2 \leftarrow \mathcal{Z}2 \cup \{(x.b_1, \ldots, x.b_r, x.atr_2, u.idcluster)\}$

fim se

$x \leftarrow le\_tuplo(X)$

fim repetir

$v \leftarrow le\_tuplo(\mathcal{V})$

fim repetir

$u \leftarrow le\_tuplo(\mathcal{U})$

fim repetir

fim

5.6.4 Existência de homónimos

Formalização Sintáctica

DETECT EXISTENCE-OF-HOMONYMS ColunasAlvo DasTabela DasBasesDados
UsandoCondição

Formalização Semântica

A detecção deste PQD envolve verificar se existem valores iguais entre os dois atributos (*i.e.*, atr_1 e atr_2) pertencentes, respectivamente, a cada uma das relações (*i.e.*, $\mathcal{R}$ e $\mathcal{S}$). Na situação de serem detectados valores iguais está-se perante a existência de homónimos. Da execução desta OD podem resultar duas relações (*i.e.*, $\mathcal{Z}1$ e $\mathcal{Z}2$) que armazenam os valores iguais detectados entre atr_1 e atr_2, bem como as respectivas chaves primárias dos tuplos de $\mathcal{R}$ e $\mathcal{S}$ onde estes se encontram. De referir que na execução desta OD são excluídos os pares de valores que tenham sido identificados previamente como sinónimos pela operação correspondente (caso esta exista).

Algoritmo Detecção_Existência_Homónimos $\stackrel{\text{def}}{=}$

seja $\mathcal{Z}1_x$ e $\mathcal{Z}2_y$ as relações que armazenam, respectivamente, os valores de $\mathcal{R}$ e $\mathcal{S}$ que constituem sinónimos, resultantes da execução da OD antecedente correspondente.

sejam $\mathcal{Z}1$ e $\mathcal{Z}2$ as relações que armazenam, respectivamente, os valores das chaves primárias dos tuplos de $\mathcal{R}$ e $\mathcal{S}$, bem como os respectivos valores que traduzem a existência de homónimos, com os seguintes esquemas: $\mathcal{Z}1(a_1,\ldots,a_m,atr_1)$ e $\mathcal{Z}2(b_1,\ldots,b_r,atr_2)$.

início

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,b_1,\ldots,b_r,atr_1}(\sigma_{cond}((\mathcal{R} \rhd\lhd (\pi_{a_1,\ldots,a_m}(\mathcal{R})\text{-}(\pi_{a_1,\ldots,a_m}(\mathcal{Z}1_x)))) \rhd\lhd_{(atr_1=atr_2)} (\mathcal{S} \rhd\lhd \rhd\lhd (\pi_{b_1,\ldots,b_r}(\mathcal{S})\text{-}(\pi_{b_1,\ldots,b_r}(\mathcal{Z}2_y)))))) \quad (*)$$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

se $\sigma_{a_1=t.a_1 \wedge ... \wedge a_m=t.a_m}(\mathcal{Z}1) = \varnothing$ **então**

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(t.a_1,\ldots,t.a_m,t.atr_1)\}$

fim se

se $\sigma_{b_1=t.b_1 \wedge ... \wedge br=t.b_r}(\mathcal{Z}2) = \varnothing$ **então**

$\mathcal{Z}2 \leftarrow \mathcal{Z}2 \cup \{(t.b_1,\ldots,t.b_r,t.atr_1)\}$

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: Na formalização anterior, no caso de não existir uma OD de sinónimos prévia, a expressão em álgebra relacional assinalada com um asterisco é substituída pela seguinte: $\mathcal{T} \leftarrow \pi_{a_1,...,a_m,b_1,...,b_r,atr_1}(\sigma_{cond}(\mathcal{R} \rhd\lhd_{(atr_1=atr_2)} \mathcal{S}))$

5.6.5 Diferentes granularidades de representação

Formalização Sintáctica

DETECT DIFFERENT-GRANULARITY-OF-REPRESENTATION ColunasAlvo DasTabelas
DasBasesDados
UsandoCondição

Formalização Semântica

A detecção deste PQD envolve identificar os valores de atr_1 (da relação $\mathcal{R}$) que não existem em atr_2 (da relação $\mathcal{S}$) e vice-versa. No caso de serem identificadas diferenças, os valores são armazenados respectivamente nas relações $\mathcal{Z}1$ e $\mathcal{Z}2$. Além destes, são também armazenadas as chaves primárias dos tuplos onde se encontram os valores diferentes.

Algoritmo Detecção_Diferentes_Granularidades_Representação $\stackrel{def}{=}$

sejam $\mathcal{Z}1$ e $\mathcal{Z}2$ as relações que armazenam, respectivamente, os valores das chaves primárias dos tuplos de $\mathcal{R}$ e $\mathcal{S}$, bem como os respectivos valores que apresentam diferenças ao nível da granularidade de representação usada entre atr_1 e atr_2, com os seguintes esquemas: $\mathcal{Z}1(a_1,\ldots,a_m,atr_1)$ e $\mathcal{Z}2(b_1,\ldots,b_r,atr_2)$.

início

$$\mathcal{Z}1 \leftarrow \pi_{a_1,...,a_m,atr_1}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \left(\pi_{atr_1}(\mathcal{R}) - \pi_{atr_2}(\mathcal{S})\right)\right)\right)$$

$$\mathcal{Z}2 \leftarrow \pi_{b_1,...,b_p,atr_2}\left(\sigma_{cond}\left(\mathcal{S} \rhd\lhd \left(\pi_{atr_2}(\mathcal{S}) - \pi_{atr_1}(\mathcal{R})\right)\right)\right)$$

fim

5.6.6 Violação de integridade referencial

Formalização Sintáctica

DETECT REFERENTIAL-INTEGRITY-VIOLATION ColunasAlvo DasTabelas
DasBasesDados
DefiniçãoColunasDependentes
DefiniçãoTabelasReferência
DefiniçãoBasesDadosReferência
UsandoCondição

Formalização Semântica

A detecção deste PQD envolve identificar os tuplos nos quais os atributos que formam a chave estrangeira em questão possuem uma combinação de valores inexistente como chave primária na relação associada. A relação resultante desta OD (*i.e.*, $\mathcal{Z}1$) armazena os valores das chaves primárias dos tuplos em que tal acontece, assim como os respectivos valores dos atributos que constituem a chave estrangeira.

Algoritmo Detecção_Violação_Integridade_Referencial $\stackrel{\text{def}}{=}$

seja $\mathcal{A} = \{a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos de $\mathcal{R}$ que não fazem parte da sua chave primária.

seja $\mathcal{C}$ o conjunto de atributos que formam a chave estrangeira de $\mathcal{R}$, assim definido: $\mathcal{C} = \{c \mid c$ faz parte da chave estrangeira de $\mathcal{R} \wedge c \in \mathcal{A}\}$, *i.e.*, $\mathcal{C} \subseteq \mathcal{A}$

seja $\mathcal{Z}1$ a relação que armazena os valores das chaves primárias dos tuplos de $\mathcal{R}$ e os respectivos valores que constituem violações à integridade referencial, com o seguinte esquema: $\mathcal{Z}1(a_1,\ldots,a_m,c_1,\ldots,c_r)$.

início

se *existe_relação*($\mathcal{Z}1$) = *falso* **então**

$$\mathcal{Z}1 \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_r}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \left(\pi_{c_1,\ldots,c_r}(\mathcal{R}) - \pi_{b_1,\ldots,b_r}(S)\right)\right)\right)$$

senão

$$\mathcal{Z}1 \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_r}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \mathcal{Z}1 \rhd\lhd \left(\pi_{c_1,\ldots,c_r}(\mathcal{R}) - \pi_{b_1,\ldots,b_r}(S)\right)\right)\right)$$

fim se

fim

5.6.7 Tuplos duplicados

Formalização Sintáctica

DETECT DUPLICATE-TUPLES DasTabelas DasBasesDados DefiniçãoColunas
UsandoChavesOrdenação UsandoAtribuições
DefiniçãoCondição

Formalização Semântica

No caso da detecção de tuplos duplicados com base numa condição susceptível de ser expressa em SQL, é efectuado um *produto Cartesiano* que envolve os tuplos das duas relações em questão (*i.e.*, $\mathcal{R}$ e $\mathcal{S}$) para identificar os que respeitam a condição definida pelo utilizador e que assim constituem duplicados. As relações que eventualmente possam resultar desta OD (*i.e.*, $\mathcal{Z}1$ e $\mathcal{Z}2$) armazenam, respectivamente, as chaves primárias dos tuplos de $\mathcal{R}$ e $\mathcal{S}$ que foram identificados como duplicados entre si, bem como o *cluster* que formam. Em ambas as relações, um dado tuplo apenas é armazenado uma única vez, apesar das múltiplas situações em que possa ser identificado como duplicado de outros tuplos (da outra relação). Todos estes tuplos formam o mesmo *cluster* de duplicados. A semântica subjacente a esta OD é apresentada de seguida:

Algoritmo Detecção_Tuplos_Duplicados_Baseada_em_SQL $\stackrel{\text{def}}{=}$

sejam $\mathcal{Z}1$ e $\mathcal{Z}2$ as relações que armazenam, respectivamente, as chaves primárias dos tuplos de $\mathcal{R}$ e $\mathcal{S}$ identificados como sendo duplicados, bem como o respectivo *cluster* de duplicados que formam. Os esquemas de $\mathcal{Z}1$ e $\mathcal{Z}2$ são: $\mathcal{Z}1(a_1,\ldots,a_m,cluster\_duplicados)$ e $\mathcal{Z}2(b_1,\ldots,b_r,cluster\_duplicados)$

início

$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,b_1,\ldots,b_r}\left(\sigma_{cond}\left(\mathcal{R}\times\mathcal{S}\right)\right)$

$idcluster = 1$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$\mathcal{U} \leftarrow \pi_{idcluster}\left(\sigma_{a_1=t.a_1\wedge\ldots\wedge a_m=t.a_m}\left(\mathcal{Z}1\right)\right)$

$\mathcal{V} \leftarrow \pi_{idcluster}\left(\sigma_{b_1=t.b_1\wedge\ldots\wedge b_r=t.b_r}\left(\mathcal{Z}2\right)\right)$

$u \leftarrow le\_tuplo(\mathcal{U})$

$v \leftarrow le\_tuplo(\mathcal{V})$

se $u = null$ **e** $v = null$ **então**

$idcluster \leftarrow idcluster + 1$

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(t.a_1,\ldots,t.a_m,idcluster)\}$

$\mathcal{Z}2 \leftarrow \mathcal{Z}2 \cup \{(t.b_1,\ldots,t.b_r,idcluster)\}$

senão se $u \neq null$ **e** $v = null$ **então**

$\mathcal{Z}2 \leftarrow \mathcal{Z}2 \cup \{(t.b_1,\ldots,t.b_r,u.idcluster)\}$

senão se $u = null$ **e** $v \neq null$ **então**

$\mathcal{Z}1 \leftarrow \mathcal{Z}1 \cup \{(t.a_1,\ldots,t.a_m,v.idcluster)\}$

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: Na formalização anterior, *cond*, além de representar uma eventual condição de selecção que delimita a execução da OD, também representa a condição de que resulta a detecção dos tuplos duplicados. O conteúdo de *cond* resulta do que se encontra especificado na cláusula *where* da OD.

A detecção de tuplos duplicados pode implicar o recurso a FDU (*e.g.*: como consequência da complexidade do critério de detecção de duplicados não ser exprimível numa condição SQL). Neste caso, os valores dos atributos que constituem a chave primária de cada tuplo das duas relações (*i.e.*, $\mathcal{R}$ e $\mathcal{S}$) e os valores dos atributos envolvidos na detecção de duplicados (definidos pelo utilizador) referentes a cada tuplo podem ser passados como parâmetros, de acordo com os requisitos de cada FDU que consta da sequência de atribuições especificadas na OD. Além destes, as funções também podem receber outros parâmetros, como variáveis e constantes. Caso a detecção de duplicados seja efectuada com base em FDU, a formalização anterior sofre as seguintes alterações/acrescentos. O que permanece inalterado é representado sob a forma de reticências.

Algoritmo Detecção_Tuplos_Duplicados_Baseada_em_FDU $\stackrel{def}{=}$

seja $\mathcal{A} = \{a_1,\ldots,a_m,a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos de $\mathcal{R}$.

seja $\mathcal{C}$ o conjunto de atributos de $\mathcal{R}$ envolvidos na detecção de duplicados, assim definido: $\mathcal{C} = \{c \mid c \text{ é usado na detecção dos tuplos duplicados entre } \mathcal{R} \text{ e } \mathcal{S} \wedge c \in \mathcal{A}\}$, *i.e.*, $\mathcal{C} \subset \mathcal{A}$.

seja o a cardinalidade de $\mathcal{C}$.

seja $\mathcal{B} = \{b_1,\ldots,b_r,b_{r+1},\ldots,b_{r+s}\}$ o conjunto de atributos de $\mathcal{S}$.

seja $\mathcal{D}$ o conjunto de atributos de $\mathcal{S}$ envolvidos na detecção de duplicados, assim definido: $\mathcal{D} = \{d \mid d \text{ é usado na detecção dos tuplos duplicados entre } \mathcal{R} \text{ e } \mathcal{S} \wedge d \in \mathcal{B}\}$, *i.e.*, $\mathcal{D} \subset \mathcal{B}$.

seja k a cardinalidade de $\mathcal{D}$.

início

$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_o,b_1,\ldots,b_r,d_1,\ldots,d_k}\left(\sigma_{cond}\left(\mathcal{R}\times\mathcal{S}\right)\right)$

$idcluster = 1$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$v_1 \leftarrow fdu_1\left(p_{1,1}, p_{1,2}, \ldots, p_{1,x}\right)$

$v_2 \leftarrow fdu_2\left(p_{2,1}, p_{2,2}, \ldots, p_{2,y}\right)$

$\vdots$

$v_i \leftarrow fdu_i\left(p_{i,1}, p_{i,2}, \ldots, p_{i,w}\right)$

se $avalia\_condição\left(p_{j,1}, p_{j,2}, \ldots, p_{j,z}\right) = verdadeiro$ **então**

$\mathcal{U} \leftarrow \pi_{idcluster}\left(\sigma_{a_1=t.a_1 \wedge \ldots \wedge a_m=t.a_m}(\mathcal{Z}1)\right)$

$\mathcal{V} \leftarrow \pi_{idcluster}\left(\sigma_{b_1=t.b_1 \wedge \ldots \wedge b_r=t.b_r}(\mathcal{Z}2)\right)$

$\vdots$

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Notas: Os valores dos atributos envolvidos ($a_1,\ldots,a_m$; $b_1,\ldots,b_r$; $c_1,\ldots,c_o$; $d_1,\ldots,d_k$) constituem parâmetros de uma ou mais funções fdu_x.

Como resultado, cada função fdu_x retorna um valor atómico.

5.6.8 Violação de restrição de integridade

Formalização Sintáctica

DETECT INTEGRITY-CONSTRAINT-VIOLATION AT LEVEL OF MULTIPLE
RELATIONS ColunasAlvo
DasTabelas DasBaseDados
DefiniçãoAtribuições
DefiniçãoCondição AgupandoPor

Formalização Semântica

No caso da detecção de violação de restrição de integridade ser efectuada com base numa operação susceptível de ser traduzida num inquérito SQL, a operação envolve uma das seguintes situações: identificar as chaves primárias dos tuplos e os respectivos valores dos atributos das várias relações que constituem uma violação de restrição de integridade; ou, identificar apenas o

valor ou valores agregados que consubstanciam a existência de uma violação de restrição de integridade. A semântica subjacente à operação é a que a seguir se apresenta.

Algoritmo Detecção_Violação_Restrição_Integridade_Baseada_em_SQL $\stackrel{def}{=}$

seja $\mathcal{A} = \{a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos de $\mathcal{R}$ que não fazem parte da sua chave primária.

seja $\mathcal{C}$ o conjunto de atributos de $\mathcal{R}$ envolvidos na restrição de integridade, assim definido: $\mathcal{C} = \{c \mid c$ é usado na formulação da restrição de integridade $\wedge\ c \in \mathcal{A}\}$, *i.e.*, $\mathcal{C} \subseteq \mathcal{A}$.

seja g a cardinalidade de $\mathcal{C}$.

seja $\mathcal{B} = \{b_{r+1},\ldots, b_{r+s}\}$ o conjunto de atributos de $\mathcal{S}$ que não fazem parte da sua chave primária.

seja $\mathcal{D}$ o conjunto de atributos de $\mathcal{S}$ envolvidos na restrição de integridade, assim definido: $\mathcal{D} = \{d \mid d$ é usado na formulação da restrição de integridade $\wedge\ d \in \mathcal{B}\}$, *i.e.*, $\mathcal{D} \subseteq \mathcal{B}$.

seja h a cardinalidade de $\mathcal{D}$.

seja *func_agrega* uma função de agregação existente em SQL (*e.g.*: *count*; *sum*; *avg*).

seja e um atributo pertencente a $\mathcal{C}$ ou $\mathcal{D}$, *i.e.*, $e \in \mathcal{C} \vee e \in \mathcal{D}$.

seja $\mathcal{Z}$ a relação que armazena o valor que constitui violação à restrição de integridade, com o seguinte esquema: $\mathcal{Z}1(a_1,\ldots,a_m,c_1,\ldots,c_g,\ldots,b_1,\ldots,b_r,d_1,\ldots,d_h)$.

início

$$\mathcal{Z} \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_g,\ldots,b_1,\ldots,b_r,d_1,\ldots,d_h}\left(\sigma_{cond}\left(_{c_1,\ldots,c_g,\ldots,d_1,\ldots,d_h}\gamma_{func\_agrega(e)}\left(\mathcal{R} \rhd\lhd \ldots \rhd\lhd \mathcal{S}\right)\right)\right)$$

fim

Notas: A formalização apresentada é o mais genérica possível, *i.e.*, abarca todas as situações possíveis de serem expressas numa OD deste tipo. No entanto, dependendo da OD em concreto, algumas das partes que constituem a expressão em álgebra relacional podem não se aplicar. Assim, o operador de agregação (*i.e.*, γ) pode não existir. Caso exista, pode não existir a função de agregação (*i.e.*, *func_agrega*), assim como pode também não existir o operador de projecção (*i.e.*, π). No caso deste existir, os atributos que constituem a chave primária de $\mathcal{R}$, …, $\mathcal{S}$ (*i.e.*, $a_1,\ldots,a_m,\ldots,b_1,\ldots,b_r$) podem não constar no operador de projecção. Naturalmente, nesta situação também não constam de $\mathcal{Z}$.

Na formalização, *cond*, além de representar uma eventual condição de selecção que delimita a execução da OD, também representa a condição de que resulta a detecção da violação da restrição de integridade. O conteúdo de *cond* resulta do que se encontra especificado na cláusula *where* da OD.

A detecção de violação de restrição de integridade ao nível de múltiplas relações pode implicar a utilização de FDU. Nestes casos, o conjunto de valores de cada atributo das relações envolvidas na formulação da restrição de integridade pode constituir um parâmetro das FDU, de acordo com os seus requisitos específicos definidos na sequência de atribuições que consta da OD. Variáveis e constantes também podem constituir parâmetros destas funções. À excepção da

última função (*i.e.*, fdu_j) que obrigatoriamente retorna um valor atómico representativo do problema detectado, todas as outras podem retornar um valor atómico ou uma estrutura de dados (*e.g.*: um vector; uma matriz). A formalização desta OD envolvendo FDU é a seguir apresentada.

Algoritmo Detecção_Violação_Restrição_Integridade_Baseada_em_FDU $\stackrel{\text{def}}{=}$

seja $\mathcal{Z}$ a relação que armazena o valor que constitui violação à restrição de integridade, com o seguinte esquema: $\mathcal{Z}(valor)$. O tipo de dados de *valor* é o mesmo do que o retornado pela última função (no caso, fdu_j) da sequência de FDU na OD.

início

$$\mathcal{T} \leftarrow \left(\sigma_{cond}\left(\pi_{a_1,\ldots,a_m,c_1,\ldots,c_g}(\mathcal{R})\right) \rhd\lhd \ldots \rhd\lhd \left(\pi_{b_1,\ldots,b_r,d_1,\ldots,d_h}(\mathcal{S})\right)\right)$$

$$v_1 \leftarrow fdu_1\left(p_{1,1}, p_{1,2}, \ldots, p_{1,x}\right)$$

$$v_2 \leftarrow fdu_2\left(p_{2,1}, p_{2,2}, \ldots, p_{2,y}\right)$$

$$\vdots$$

$$v_j \leftarrow fdu_j\left(p_{j,1}, p_{j,2}, \ldots, p_{j,w}\right)$$

se $avalia\_condição\left(p_{k,1}, p_{k,2}, \ldots, p_{k,z}\right) = verdadeiro$ **então**

$$\mathcal{Z} \leftarrow \mathcal{Z} \cup \{(v_j)\}$$

fim se

fim

Notas: Os valores dos diversos atributos envolvidos (*e.g.*, $a_1,\ldots,a_m$; $c_1,\ldots,c_g$; $b_1,\ldots,b_r$) constituem parâmetros de uma ou mais funções fdu_x.

No caso da última função da sequência de FDU (*i.e.*, fdu_j) retornar uma estrutura de dados do tipo unidimensional (*i.e.*, um vector), a formalização anterior é alvo das alterações/acrescentos que a seguir se apresentam. O que permanece inalterado encontra-se representado sob a forma de reticências.

Algoritmo Detecção_Violação_Restrição_Integridade $\stackrel{\text{def}}{=}$

seja $\mathcal{Z}$ a relação que armazena os valores que constituem violações à restrição de integridade, com o seguinte esquema: $\mathcal{Z}(valor)$. O tipo de dados de *valor* é o mesmo do que o retornado pela última função (no caso, fdu_j) da sequência de FDU na OD.

início

$$\vdots$$

$$v_j \leftarrow fdu_j\left(p_{j,1}, p_{j,2}, \ldots, p_{j,w}\right)$$

repetir para $i \leftarrow 1$ **até** $dimensão(v_j)$

$\mathcal{Z} \leftarrow \mathcal{Z} \cup \{(v_j[i])\}$

fim repetir

fim

No caso da última função da sequência (*i.e.*, fdu_j) retornar uma estrutura de dados bidimensional (*i.e.*, uma matriz), a formalização inicial desta operação sofre as seguintes alterações/acrescentos.

Algoritmo Detecção_Violação_Restrição_Integridade $\stackrel{\text{def}}{=}$

seja $\mathcal{Z}$ a relação que armazena os valores dos atributos que constituem violações à restrição de integridade, com o seguinte esquema: $\mathcal{Z}(a_x,\ldots,a_z,\ldots,b_w,\ldots,b_y)$.

início

$\vdots$

repetir para $h \leftarrow 1$ **até** *numlinhas*(v_j)

$\mathcal{Z} \leftarrow \mathcal{Z} \cup \{(v_j[h][1], \ldots, v_j[h][numcolunas])\}$

fim repetir

fim

Nota: Entre os atributos de $\mathcal{Z}$ podem encontrar-se os que constituem as chaves primárias de algumas ou de todas as relações envolvidas na restrição de integridade (*i.e.*, $\mathcal{R}$; $\mathcal{S}$; etc.).

5.7 Conclusão

Neste capítulo foram apresentadas as formalizações, sintácticas e semânticas, das OD de todos os PQD incluídos na taxionomia apresentada no capítulo anterior. A *nível sintáctico*, formalizou-se uma linguagem que permite a especificação de qualquer OD automática dos PQD que fazem parte da taxionomia. A *nível semântico*, formalizaram-se todas as operações possíveis de serem disponibilizadas de base para a detecção dos PQD.

Fruto da sua diversidade e dependência do domínio subjacente, não é possível representar explicitamente a semântica inerente a todas as OD susceptíveis de virem a ser executadas num cenário de limpeza de dados. Este aspecto é particularmente notório na detecção de violações a restrições de integridade. Nestes casos e nalguns outros, não há mesmo outra alternativa que não seja a implementação da lógica que conduz à detecção do problema numa função definida pelo próprio utilizador. Quando se iniciou a realização deste trabalho já havia a perfeita noção de que não seria possível fornecer suporte de raiz à detecção de todo e qualquer PQD. No entanto, existia a convicção de que poderia ser concedido esse suporte para a grande maioria dos PQD, o que não acontecia na generalidade das soluções de limpeza de dados até então existentes. Esta convicção tornou-se uma realidade neste capítulo com a formalização, na totalidade e de forma

explícita, da semântica inerente à generalidade das OD. Para os restantes problemas, *i.e.*, aqueles cuja detecção obriga ao uso de funções definidas pelo próprio utilizador, a semântica subjacente não pode ser totalmente especificada *a priori*, uma vez que não é conhecida na íntegra. Ainda assim, nestes casos foi identificado e formalizado o tronco comum a qualquer OD do tipo em causa (*e.g.*: violação de restrição de integridade). A especificidade de cada operação reside na FDU, não podendo ser representada explicitamente nas formalizações produzidas, em virtude da sua diversidade. De referir que a especificação de operações com base em FDU permite virtualmente detectar qualquer PQD. Mesmo aqueles para os quais se propôs e formalizou uma OD neste capítulo podem ser detectados por uma outra via, devidamente implementada na FDU.

No geral, as formalizações semânticas apresentadas neste capítulo foram desenvolvidas tendo como preocupação as questões relacionadas com a sua execução eficiente. No entanto, não foram levadas ao limite, *i.e.*, haverão certamente optimizações que podem ser introduzidas na execução da detecção de alguns PQD. Uma destas situações é a que diz respeito à detecção de duplicados quando esta é efectuada com base no produto Cartesiano. A não introdução de optimizações pode fazer com que a execução deste tipo de OD, em certas situações, se torne computacionalmente muito "pesada". Naturalmente, isto depende do número de tuplos envolvidos e da capacidade de processamento do computador onde a operação é executada. A optimização de algumas OD será objecto de trabalho num futuro próximo.

FORMALIZAÇÃO DAS OPERAÇÕES DE CORRECÇÃO 6

Neste capítulo apresenta-se a formalização sintáctica e semântica das operações que permitem solucionar os Problemas de Qualidade dos Dados (PQD). Dependendo do PQD, a correcção pode implicar a alteração aos valores de um ou mais atributos ou a remoção de um ou mais tuplos de uma dada relação. À semelhança do capítulo anterior, começa-se por apresentar as definições de carácter geral, a nível sintáctico e semântico, comuns à generalidade das formalizações expostas nas secções seguintes. Igualmente, a apresentação da formalização das operações é efectuada seguindo a organização dos PQD por nível de granularidade, decorrente da taxionomia apresentada no Capítulo 4. Assim, começa-se pela formalização das operações que incidem sobre os problemas que ocorrem ao nível do atributo. A seguir, formaliza-se a Operação de Correcção (OC) do único problema que ocorre ao nível do tuplo e, após esta, formalizam-se as operações relativas aos problemas que se manifestam ao nível da relação. Por último, apresenta-se a formalização das operações referentes à correcção dos PQD que ocorrem ao nível de múltiplas relações, de uma ou mais Bases de Dados (BD).

6.1 Introdução

No capítulo anterior apresentou-se a formalização das Operações de Detecção (OD). Neste capítulo, apresenta-se a formalização do outro tipo de operações envolvido na limpeza de dados e que permitem a correcção dos PQD detectados. Para a generalidade dos problemas que fazem parte da taxionomia é proposta uma OC automática. Na grande maioria das operações, a correcção envolve "meramente" a alteração aos valores de um ou mais atributos. Num conjunto restrito de operações, a correcção implica a remoção ou eliminação de um ou mais tuplos da relação. As OC baseiam-se nos resultados produzidos pelas respectivas OD. As operações propostas suportam as correcções que, de uma forma racional, se consideraram úteis e que à partida se podem disponibilizar de base ou raiz. Assim, minimiza-se a necessidade do utilizador ter de recorrer a funções especialmente desenvolvidas para o efeito. Relembre-se que nas actuais soluções informáticas vocacionadas para a limpeza de dados, a correcção dos PQD se encontra excessivamente dependente da implementação de funções por parte do utilizador (ver Capítulo 1 – Secção 1.2). Esta limitação é suprida neste trabalho através do fornecimento de um conjunto

genérico de OC. Naturalmente, há a consciência de que, por opção do utilizador ou por não ser viável correcção automática, muitas correcções terão de ser efectuadas manualmente, envolvendo uma intervenção individualizada (*i.e.*, caso a caso). No entanto, muitas outras correcções poderão ser efectuadas de forma totalmente automática. É com base neste argumento que se propõem as OC apresentadas neste capítulo.

Tal como as OD, as OC são formalizadas a dois níveis: sintáctico e semântico. A nível *sintáctico*, é definida uma linguagem que permite especificar as OC dos diferentes PQD que fazem parte da taxionomia. Por questões de uniformização e facilidade de aprendizagem, esta linguagem segue o mesmo espírito da linguagem de especificação das OD. O nível *semântico* representa a lógica inerente a cada operação que conduz à correcção do PQD. Na apresentação da formalização semântica de cada OC é adoptada a abordagem que a seguir se descreve. Em primeiro lugar, é efectuada uma descrição textual sucinta e informal da filosofia subjacente à OC. Após esta, é que se apresenta de modo formal a semântica da operação. De referir que a este propósito se adoptou a mesma abordagem usada na formalização das OD, *i.e.*, um misto de *álgebra relacional*[15] e *linguagem algorítmica*. A justificação para esta abordagem é a mesma do que a exposta na Secção 5.1 do capítulo anterior. À semelhança das OD, a formalização das OC constitui um dos objectivos deste trabalho, tal como enunciado no Capítulo 1 – Secção 1.3. A formalização das OC é importante, uma vez que constitui a base para a sua implementação computacional.

6.2 Definições Gerais

Nesta secção enunciam-se as definições de carácter geral, a nível sintáctico e semântico, utilizadas na generalidade das formalizações das OC dos PQD. Como tal, esta secção tenderá a ser consultada inúmeras vezes ao longo da leitura deste capítulo.

6.2.1 Sintaxe da Linguagem de Especificação

Nesta secção define-se a linguagem declarativa que permite ao utilizador especificar as OC a efectuar para solucionar, automaticamente, os PQD detectados. Tal como a linguagem de especificação das OD, esta linguagem segue o espírito da *Structured Query Language* (SQL). A

[15] No Apêndice B desta dissertação é apresentado um breve resumo sobre as operações de álgebra relacional utilizadas nas formalizações apresentadas neste capítulo.

definição da linguagem é efectuada através da gramática em BNF estendido que a seguir se apresenta.

Operação → TipoOperação ProblemaQualidadeDados NívelOperação NasColunas DasTabelas DasBasesDados UsandoTécnica UsandoColunas UsandoOperações UsandoTabelas UsandoTransformações UsandoValorPróximo UsandoTermosUniformização DependentesDe FonteTermosUniformização UsandoCondição

TipoOperação → CORRECT | REMOVE

ProblemaQualidadeDados → NívelValorIndividual | NívelColuna | NívelTuplo | NívelRelação | NívelMúltiplasRelações

NívelValorIndividual → MISSING-VALUE | SYNTAX-VIOLATION | DOMAIN-VIOLATION | INCOMPLETE-VALUE | OVERLOADED-VALUE | MISSPELLING-ERRORS |

NívelColuna → EXISTENCE-OF-SYNONYMS | UNIQUENESS-VIOLATION | INTEGRITY-CONSTRAINT-VIOLATION

NívelTuplo → INTEGRITY-CONSTRAINT-VIOLATION

NívelRelação → FUNCTIONAL-DEPENDENCY-VIOLATION | DUPLICATE-TUPLES |INTEGRITY-CONSTRAINT-VIOLATION

NívelMúltiplasRelações → HETEROGENEITY-OF-SYNTAXES | HETEROGENEITY-OF-MEASURE-UNITS | EXISTENCE-OF-SYNONYMS | DUPLICATE-TUPLES | INTEGRITY-CONSTRAINT-VIOLATION

NívelOperação → ε | AT LEVEL OF Nível

Nível → COLUMN | ROW | RELATION | MULTIPLE RELATIONS

NasColunas → ε | ColunasAlvo

ColunasAlvo → ON Coluna RestantesColunas

Coluna → AliasTabelaDaColuna NomeColuna

AliasTabelaDaColuna → ε | identificador "."

NomeColuna → identificador

RestantesColunas → ε | ColunaAdicional^{+}

ColunaAdicional → "," Coluna

DasTabelas → FROM Tabela RestantesTabelas

Tabela → NomeTabela AliasTabela

NomeTabela → identificador

AliasTabela → ε | identificador

RestantesTabelas → ε | TabelaAdicional^{+}

TabelaAdicional → "," Tabela

NaBaseDados → ε | DasBasesDados

DasBasesDados → OF BaseDados RestantesBasesDados

BaseDados → identificador

RestantesBasesDados → ε | BaseDadosAdicional^{+}

BaseDadosAdicional → "," BaseDados

UsandoTécnica → ε | TécnicaExistenteBase

TécnicaExistenteBase → USING TécnicaPreenchimento

TécnicaPreenchimento → AVERAGE | MODE | MEDIAN | PROBABILISTIC
DISTRIBUTION

UsandoColunas → ε | DefiniçãoColunas

DefiniçãoColunas → USING Coluna RestantesColunas

UsandoOperações → UsandoAtribuições UsandoFunçãoVoid

UsandoAtribuições → ε | DefiniçãoAtribuições

DefiniçãoAtribuições → SET Atribuição RestantesAtribuições

Atribuição → LadoEsquerdoAtribuição = LadoDireitoAtribuição

LadoEsquerdoAtribuição → NomeColuna | Variável

Variável → identificador

LadoDireitoAtribuição → Variável | Constante | ExpressãoAritmética | NomeColuna
FunçãoDefUtilizador | inquérito-sql

Constante → string | valor-numérico

ExpressãoAritmética → ParêntesisEsq Operando Operador Operador ParêntesisDir

ParêntesisEsq → ε | "("

Operando → valor-numérico | Variável | ExpressãoAritmética

Operador → operação-aritmética

ParêntesisDir → ε | ")"

FunçãoDefUtilizador → NomeFunção(ArgumentoFunção RestantesArgumentosFunção)

ArgumentoFunção → NomeColuna | Variável | Constante

RestantesArgumentosFunção → ε | ArgumentoAdicionalFunção^{+}

ArgumentoAdicionalFunção → "," ArgumentoFunção

RestantesAtribuições → ε | AtribuiçãoAdicional^{+}

AtribuiçãoAdicional → "," Atribuição

UsandoFunçãoVoid → ε | USING FUNCTION FunçãoDefUtilizador

UsandoTransformações → ε | DefiniçãoTransformações

DefiniçãoTransformações → BY TRANSFORMING Transformação RestantesTransformações

Transformação → FormatoOrigem INTO FormatoDestino

FormatoOrigem → identificador

FormatoDestino → identificador

RestantesTransformações → ε | TransformaçãoAdicional^{+}

TransformaçãoAdicional → "," Transformação

UsandoValorPróximo → ε | USING CLOSEST VALUE

UsandoTermosUniformização → ε | DefiniçãoTermosUniformização

DefiniçãoTermosUniformização → OF VALUES Termo RestantesTermos

Termo → identificador

RestantesTermos → ε | TermoAdicional^{+}

TermoAdicional → "," Termo

DependentesDe → ε | DefiniçãoColunasDependentes

DefiniçãoColunasDependentes → DEPENDENT ON Coluna RestantesColunas

FonteTermosUniformização → ε | DefiniçãoFonteTermos

DefiniçãoFonteTermos → USING VALUES OF NomeColuna DasTabelas NaBaseDados

UsandoTabelas → ε | FROM Tabelas

Tabelas → expressão-from-sql

UsandoCondição → ε | WHERE Condição

Condição → condição-where-sql

A linguagem de especificação das OC que acabou de ser definida na sua globalidade será, novamente, apresentada na Secção 6.3 e seguintes, devidamente instanciada a cada operação que se propõe.

6.2.2 Semântica das Operações de Correcção

As definições semânticas de carácter específico necessárias à formalização de algumas OC são apresentadas na formalização a que dizem respeito. As definições que a seguir se apresentam possuem um carácter geral, uma vez que são utilizadas em diversas formalizações.

seja $\mathcal{R}$ uma relação com o seguinte esquema: $\mathcal{R}(a_1,\ldots,a_m,a_{m+1},\ldots,a_{m+n})$, em que $a_1,\ldots,a_m$ representam os atributos que formam a sua chave primária.

seja atr_1 um atributo que não pertence à chave primária de $\mathcal{R}$, *i.e.*, $atr_1 \in \{a_{m+1},\ldots,a_{m+n}\}$, cujo valor será objecto de correcção do PQD detectado.

seja $\mathcal{S}$ uma relação com o seguinte esquema: $\mathcal{S}(b_1,\ldots,b_r,b_{r+1},\ldots,b_{r+s})$, em que $b_1,\ldots,b_r$ representam os atributos que formam a sua chave primária.

seja atr_2 um atributo que não pertence à chave primária de $\mathcal{S}$, *i.e.*, $atr_2 \in \{b_{r+1},\ldots,b_{r+s}\}$, cujo valor será objecto de correcção do PQD detectado.

seja *Z1* a relação que contém informação sobre os tuplos de $\mathcal{R}$ onde foi detectado o PQD em causa.

seja *Z2* a relação que contém informação sobre os tuplos de $\mathcal{S}$ onde foi detectado o PQD em causa.

seja *cond* a condição opcional especificada na OC, permitindo delimitar o seu âmbito de aplicação.

seja *le_tuplo*($\mathcal{R}$) uma função que lê sequencialmente e devolve o próximo tuplo de $\mathcal{R}$. Quando o final de $\mathcal{R}$ é alcançado (*i.e.*, não existem mais tuplos), a função devolve o valor *null*.

seja *dimensão*(*vec*[ ]) uma função que recebe um vector, determina o número de elementos que o compõem e devolve este valor como resultado.

seja *fdu*($p_1,p_2,\ldots,p_k$) uma Função Definida pelo Utilizador (FDU) que faz parte da sequência de funções especificadas na OC. A função recebe um conjunto de k parâmetros $p_1,p_2,\ldots,p_k$, em que cada um destes representa uma constante, o valor de um atributo, o conjunto de valores de um atributo ou uma variável que armazena o resultado de uma função anterior. Como resultado, a função retorna um valor atómico ou uma estrutura de dados de um dado tipo (*e.g.*: um vector; uma matriz).

seja *atrib* uma atribuição da sequência de atribuições especificadas na OC.

seja *reinicializa_leitura_tuplos*($\mathcal{R}$) um procedimento que recebe uma relação $\mathcal{R}$ e re-inicializa a leitura sequencial dos tuplos em $\mathcal{R}$, *i.e.*, faz com que o próximo tuplo a ser lido seja novamente o primeiro.

seja *count*(*nome_atr*) a função usual de SQL que recebe o nome de um atributo e devolve o número de valores (não nulos) que este contém.

seja *length*(*str*) uma função que recebe uma string e devolve o número de caracteres que a compõem.

seja *substitui_substring*(*str*, *substr1*, *substr2*) uma função que recebe uma string (*str*) e duas substrings (*substr1*; *substr2*), substituindo *substr1* por *substr2* em *str*. Como resultado, a função devolve a

nova string resultante. No caso de *substr1* não existir em *str*, a função devolve esta última sem qualquer alteração.

seja *vec_atribs_chave_prim_*$\mathcal{R}$ um vector que contém os nomes dos atributos que constituem a chave primária de $\mathcal{R}$, *i.e.*, $a_1,\ldots,a_m$.

seja *existem_atribs_na_relação*(*vec_atribs*[ ], $\mathcal{R}$) uma função que recebe um vector com nomes de atributos (*vec_atribs*) e uma relação ($\mathcal{R}$), verificando se todos os atributos existem na relação. Como resultado da verificação, a função devolve um valor booleano.

seja *pqd* o problema de qualidade de dados a corrigir de acordo com o especificado na OC.

seja *flag_actualiza_valores_relação_*$\mathcal{R}$ uma variável booleana que indica se o PQD em questão deve ser solucionado em $\mathcal{R}$ ou em $\mathcal{S}$. O valor desta variável resulta da especificação da OC.

seja *mat_alterações* uma matriz onde se encontram as alterações de formato a realizar (uma por linha), resultante da especificação da OC. A matriz é composta por duas colunas, representando a primeira a expressão regular, e a segunda o correspondente novo formato sintáctico.

seja *efectua_correcção_sintáctica*(*valor*, *mat_alterações*) uma função que recebe um valor de um atributo e uma matriz de alterações de formato e caso haja concordância entre o valor e uma expressão regular, coloca o valor de acordo com o formato sintáctico associado. Como resultado, a função devolve o valor no seu novo formato. Caso não haja concordância, a função não efectua qualquer modificação, retornando o valor inalterado.

Para não tornar as formalizações expostas nas secções seguintes excessivamente extensas, optou-se por efectuar a sua apresentação, considerando que a condição de selecção *cond* foi especificada pelo utilizador na OC. Na situação de tal não acontecer (uma vez que esta é opcional), basta remover *cond* das expressões em álgebra relacional, bem como o respectivo operador de selecção (*i.e.*, σ), em cada formalização apresentada. Estas alterações fazem com que as formalizações passem a reflectir a semântica inerente à situação mencionada.

À semelhança do que sucede nas OD, quando se especificam OC ao nível do atributo, a linguagem proposta permite que se defina numa única operação (*e.g.*: correcção de erro ortográfico) múltiplos atributos de uma determinada relação (*e.g.*: nome; morada; localidade). Isto significa que a OC deve ser executada em cada um destes atributos. Assim, ainda que seja especificada uma só OC, esta corresponde a múltiplas operações, uma vez que a sua execução é efectuada individualmente para cada atributo. As formalizações apresentadas na secção seguinte reflectem a semântica subjacente à execução da OC num atributo individual.

6.3 Problemas ao Nível do Atributo

Nesta secção apresentam-se as formalizações das OC dos PQD que ocorrem ao nível do atributo. Preservando a forma de organização dos PQD por nível de granularidade preconizada na taxionomia, começa-se por apresentar as formalizações das OC que ocorrem no contexto do

valor individual do atributo. De seguida, apresentam-se as formalizações das OC dos PQD que ocorrem no contexto dos vários valores do atributo.

6.3.1 Contexto do Valor Individual

Nas subsecções seguintes apresentam-se as formalizações das OC dos PQD que ocorrem ao nível do valor individual do atributo.

6.3.1.1 Valor em falta / Violação de sintaxe / Violação de domínio

Formalização Sintáctica

CORRECT PQD ColunasAlvo DasTabelas DasBasesDados UsandoTécnica UsandoAtribuições UsandoTransformações UsandoCondição

PQD → MISSING-VALUE | SYNTAX-VIOLATION | DOMAIN-VIOLATION

Formalização Semântica

As abordagens usadas na correcção dos PQD *valor em falta* e *violação de domínio* são as mesmas. Assim, a formalização da correcção destes problemas é apresentada conjuntamente. Em função de se tratar de *valor em falta* ou *violação de domínio*, a correcção do PQD no atributo em causa envolve, respectivamente: a colocação de valores nos tuplos onde se detectou a sua inexistência; ou, a substituição dos valores nos tuplos onde se detectou que estes violam o domínio. As técnicas de preenchimento/substituição consideradas na correcção destes problemas são: média (limitada a atributos do tipo numérico); moda (*i.e.*, o valor que ocorre mais frequentemente); mediana (*i.e.*, o valor que ocupa a posição intermédia num conjunto ordenado de valores numéricos); valor probabilístico (*i.e.*, valor gerado a partir de uma distribuição probabilística); e, valor/expressão de preenchimento/substituição definida pelo utilizador. Neste último caso, a expressão envolve exclusivamente atributos, variáveis e/ou constantes. As quatro primeiras técnicas quando utilizadas na correcção de valores em falta, apenas consideram os valores dos atributos que se encontram preenchidos. Por outro lado, as mesmas técnicas quando usadas na correcção de violações de domínio, apenas têm em conta os valores válidos do atributo. A aplicação da técnica *moda* implica que um dos valores possua um número de ocorrências superior a todos os outros. Caso isto não aconteça, não é efectuada qualquer correcção. Quando o número de valores envolvidos na aplicação da técnica *mediana* não é impar, é considerada a média resultante dos valores que se encontram nas duas posições intermédias.

Naturalmente, o preenchimento de um valor inexistente ou a substituição de um valor inválido, baseado nas técnicas apresentadas, é sempre uma solução de recurso a que se recorre quando não há alternativa. Ainda que o PQD em causa seja solucionado, normalmente o novo valor não traduz a realidade (*e.g.*: do preenchimento da altura de um indivíduo com a média das alturas dos outros indivíduos, muito dificilmente resulta a sua altura real). Apesar das limitações, estas são as técnicas que mais vulgarmente surgem na literatura como solução para o problema de valores em falta. Neste trabalho, as mesmas técnicas são empregues na substituição dos valores que violam o domínio do atributo.

Algoritmo Correcção_Valor_em_Falta/Violação_Domínio $\stackrel{\text{def}}{=}$

seja *técnica_correcção* a técnica de correcção do valor em falta/violação de domínio especificada na OC.

seja *avg*(*nome_atr*) a função usual de SQL que recebe o nome de um atributo e devolve a média dos seus valores.

seja *int*(*valor*) uma função que recebe um valor numérico do tipo real e devolve a sua parte inteira.

seja *gera_nr_aleatorio*() uma função que efectua a geração de um número real compreendido no intervalo [0; 1], de forma aleatória.

início

$$\mathcal{T} \leftarrow \pi_{\mathcal{R}.a_1,\ldots,\mathcal{R}.a_m,\ldots,\mathcal{R}.a_{m+n}}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd_{(a_1=b_1\wedge\ldots\wedge a_m=b_m)} \rho_{\mathcal{Z}1(b_1\leftarrow a_1,\ldots,b_m\leftarrow a_m)}(\mathcal{Z}1)\right)\right)$$

se *técnica_correcção* = "média" **então**

se *pqd* = "valor em falta" **então**

$$u \leftarrow \gamma_{avg(atr_1)}(\mathcal{R})$$

senão

$$u \leftarrow \gamma_{avg(atr_1)}\left(\mathcal{R} \rhd\lhd\left(\pi_{a_1,\ldots,a_m}(\mathcal{R}) - \pi_{a_1,\ldots,a_m}(\mathcal{Z}1)\right)\right)$$

fim se

novo_valor ← *u.avg*(atr_1)

senão se *técnica_correcção* = "moda" **então**

se *pqd* = "valor em falta" **então**

$$\mathcal{U} \leftarrow \tau_{count(atr_1)}\left({}_{atr_1}\gamma_{count(atr_1)}(\mathcal{R})\right)^{(*)}$$

senão

$$\mathcal{U} \leftarrow \tau_{count(atr_1)}\left({}_{atr_1}\gamma_{count(atr_1)}\left(\mathcal{R} \rhd\lhd\left(\pi_{a_1,\ldots,a_m}(\mathcal{R}) - \pi_{a_1,\ldots,a_m}(\mathcal{Z}1)\right)\right)\right)^{(*)}$$

fim se

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

prim_nr_ocorr ← *u.count*(atr_1)

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

seg_nr_ocorr $\leftarrow$ *u.count*(atr_1)

se *prim_nr_ocorr* > *seg_nr_ocorr* **então**

novo_valor $\leftarrow$ *prim_nr_ocorr*

senão

novo_valor $\leftarrow$ *null*

fim se

senão se *técnica_correcção* = "mediana" **então**

se *pqd* = "valor em falta" **então**

$$\mathcal{U} \leftarrow \gamma_{count(atr_1)}\left(\sigma_{atr_1 \neq null}\left(\mathcal{R}\right)\right)$$

$$\mathcal{V} \leftarrow \tau_{atr_1}\left(\pi_{atr_1}\left(\sigma_{atr_1 \neq null}\left(\mathcal{R}\right)\right)\right)$$

senão

$$\mathcal{U} \leftarrow \gamma_{count(atr_1)}\left(\sigma_{atr_1 \neq null}\left(\mathcal{R} \triangleright\!\triangleleft \left(\pi_{a_1,\ldots,a_m}\left(\mathcal{R}\right) - \pi_{a_1,\ldots,a_m}\left(\mathcal{Z}1\right)\right)\right)\right)$$

$$\mathcal{V} \leftarrow \tau_{atr_1}\left(\pi_{atr_1}\left(\mathcal{R} \triangleright\!\triangleleft \left(\pi_{a_1,\ldots,a_m}\left(\mathcal{R}\right) - \pi_{a_1,\ldots,a_m}\left(\mathcal{Z}1\right)\right)\right)\right)$$

fim se

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

posição_mediana $\leftarrow$ *int*(*u.count*(atr_1)/2+0.5)

repetir para $i \leftarrow 1$ **até** *posição_mediana*

$v \leftarrow$ *le_tuplo*($\mathcal{V}$)

fim repetir

se *u.count*(atr_1)/2 $\neq$ *posição_mediana* **então**

novo_valor $\leftarrow$ $v.atr_1$

senão

primeiro_valor $\leftarrow$ $v.atr_1$

$v \leftarrow$ *le_tuplo*($\mathcal{V}$)

segundo_valor $\leftarrow$ $v.atr_1$

novo_valor $\leftarrow$ (*primeiro_valor* + *segundo_valor*)/2

fim se

senão se *técnica_correcção* = "valor probabilístico" **então**

se *pqd* = "valor em falta" **então**

$$v \leftarrow \gamma_{count(atr_1)}\left(\mathcal{R}\right)$$

$$\mathcal{U} \leftarrow {}_{atr_1}\gamma_{count(atr_1)/v.count(atr_1)}\left(\sigma_{atr_1 \neq null}\left(\mathcal{R}\right)\right)$$

senão

$v \leftarrow \gamma_{count(atr_1)}\left(\mathcal{R} \rhd\lhd \left(\pi_{a_1,\ldots,a_m}(\mathcal{R}) - \pi_{a_1,\ldots,a_m}(\mathcal{Z}1)\right)\right)$

$\mathcal{U} \leftarrow {}_{atr_1}\gamma_{count(atr_1)/v.count(atr_1)}\left(\mathcal{R} \rhd\lhd \left(\pi_{a_1,\ldots,a_m}(\mathcal{R}) - \pi_{a_1,\ldots,a_m}(\mathcal{Z}1)\right)\right)$

fim se

senão

$v_1 \leftarrow atrib_1$

$v_2 \leftarrow atrib_2$

$\vdots$

$novo\_valor \leftarrow atrib_k$

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

se *técnica_correcção* = "valor probabilístico" **então**

$nr\_aleatorio \leftarrow gera\_nr\_aleatorio()$

$soma\_probabilidades \leftarrow 0$

repetir enquanto $soma\_probabilidades <= nr\_aleatório$

$u \leftarrow le\_tuplo(\mathcal{U})$

$soma\_probabilidades \leftarrow soma\_probabilidades + u.count(atr_1)/v.count(atr_1)$

fim repetir

$novo\_valor \leftarrow u.atr_1$

$reinicializa\_leitura\_tuplos(\mathcal{U})$

fim se

se $novo\_valor \neq null$ **então**

$\mathcal{R} \leftarrow \mathcal{R} - t$

$t.atr_1 \leftarrow novo\_valor$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(t.a_1,\ldots,t.a_m,t.a_{m+1},\ldots,t.a_{m+n})\}$

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: Nas expressões em álgebra relacional assinaladas com um asterisco, a ordenação é efectuada sob a forma descendente.

No caso da violação de sintaxe, a correcção envolve efectuar alterações sintácticas aos valores do atributo nos quais o PQD foi detectado. O objectivo consiste em colocar os valores segundo a sintaxe pretendida. Nesta operação há uma função (no caso, *efectua_correcção_sintáctica*) que

suporta, de raiz, a realização das alterações mais comuns ao formato do valor de um atributo (*e.g.*: alterar o formato de *dd/mm/aaaa* para *aaaa/mm/dd*), com base em expressões regulares. Na especificação da OC, o utilizador define um conjunto de expressões regulares e os respectivos formatos a que o valor deve passar a obedecer. A função verifica a concordância entre o valor do atributo e cada expressão regular que faz parte do conjunto definido. Caso se verifique concordância numa expressão regular, o valor é alterado para o formato correspondente. No caso contrário, não é efectuada qualquer alteração ao valor do atributo.

Algoritmo Correcção_Violação_Sintaxe $\stackrel{\text{def}}{=}$

início

$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}(\sigma_{cond}(\mathcal{Z}1))$

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

repetir enquanto $t \neq$ *null*

novo_valor $\leftarrow$ *efectua_correcção_sintáctica*($t.atr_1$, *mat_transf_sintácticas*)

se *novo_valor* $\neq t.atr_1$ **então**

$u \leftarrow \sigma_{a_1=t.a_1 \wedge \ldots \wedge a_m=t.a_m}(\mathcal{R})$

$\mathcal{R} \leftarrow \mathcal{R} - u$

$u.atr_1 \leftarrow$ *novo_valor*

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(u.a_1,\ldots,u.a_m,u.a_{m+1},\ldots,u.a_{m+n})\}$

fim se

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

fim repetir

fim

Na correcção dos PQD valor em falta, violação de sintaxe ou violação de domínio pode ser necessário recorrer a funções definidas/implementadas pelo utilizador. No caso da correcção de valores em falta ou de violações de domínio pode querer utilizar-se uma outra técnica de preenchimento/substituição que não as existentes de base. No caso da violação de sintaxe, a complexidade das operações envolvidas na correcção pode não ser suportada pela função existente de raiz (*i.e.*, *efectua_correcção_sintáctica*). Estas situações obrigam o utilizador a implementar novas funcionalidades em funções especialmente definidas para o efeito. De acordo com os requisitos de cada função existente na sequência de atribuições da OC, o valor do atributo de cada tuplo no qual foi detectado o PQD em causa pode ser passado como parâmetro. Eventualmente, podem existir outros parâmetros, como variáveis e constantes. Como resultado, cada FDU retorna um valor atómico. A última função da sequência de atribuições (na

formalização seguinte representada por fdu_k) retorna o valor representativo da correcção ao valor em falta, à violação de sintaxe ou à violação de domínio, sendo usado na actualização do valor do atributo.

Algoritmo Correcção_Valor_em_Falta/Violação_Sintaxe/Domínio_Baseada_FDU $\stackrel{\text{def}}{=}$

início

$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}\left(\sigma_{cond}\left(\mathcal{Z}1\right)\right)$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$v_1 \leftarrow fdu_1\left(p_{1,1},p_{1,2},\ldots,p_{1,i}\right)$

$v_2 \leftarrow fdu_2\left(p_{2,1},p_{2,2},\ldots,p_{2,j}\right)$

$\vdots$

$v_k \leftarrow fdu_k\left(p_{k,1},p_{k,2},\ldots,p_{k,l}\right)$

$u \leftarrow \sigma_{a_1=t.a_1\wedge\ldots\wedge a_m=t.a_m}\left(\mathcal{R}\right)$

$\mathcal{R} \leftarrow \mathcal{R} - u$

$u.atr_1 \leftarrow v_k$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(u.a_1,\ldots,u.a_m,u.a_{m+1},\ldots,u.a_{m+n})\}$

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

6.3.1.1.1 Valor incompleto / Valor sobrecarregado

Formalização Sintáctica

CORRECT PQD ColunasAlvo DasTabelas DasBasesDados UsandoValorPróximo
UsandoCondição

PQD → INCOMPLETE-VALUE | OVERLOADED-VALUE

Formalização Semântica

A relação que armazena os PQD (*i.e.*, valores incompletos ou sobrecarregados) detectados (*i.e.*, *Z1*) contém, para cada um destes, as diversas alternativas que os transformam em valores válidos (*i.e.*, que não violam o domínio). No caso de existir apenas uma única alternativa válida, esta é utilizada na correcção do PQD. Na situação oposta (*i.e.*, mais do que uma alternativa), em função

da opção do utilizador (especificada na OC), a correcção ou não se realiza (o que significa que será objecto de correcção manual) ou então é utilizada a alternativa válida que apresenta menor diferença com o valor inválido em causa. A diferença é analisada ao nível do caracter entre ambos os valores. No caso da menor diferença corresponder a mais do que uma alternativa válida, não é efectuada qualquer correcção. Nestes casos é necessária a intervenção manual do utilizador.

Algoritmo Correcção_Valor_Incompleto/Valor_Sobrecarregado $\stackrel{\text{def}}{=}$

seja *dom* o atributo de *Z1* que, dependendo do PQD em causa, contém as diversas alternativas válidas: que completam os valores incompletos detectados em atr_1; para os valores sobrecarregados detectados em atr_1.

seja *flag_valor_proximo* uma variável booleana que indica se deve ou não ser usado o valor de *dom* mais próximo de atr_1, quando há mais do que uma alternativa de correcção. Para o efeito, considera-se que o valor mais próximo é aquele que difere no menor número de caracteres. O valor desta variável resulta do especificado na OC.

início

$\mathcal{T} \leftarrow {}_{a_1,\ldots,a_m,atr_1}\gamma_{dom,count(atr_1)}\left(\sigma_{cond}\left(Z1\right)\right)$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

se $t.count(atr_1) = 1$ **então**

actualiza_valor(*t*, *t.dom*)

senão se *flag_valor_proximo* = *verdadeiro* **então**

$\mathcal{U} \leftarrow \tau_{length(dom)}\left(\pi_{dom}\left(\sigma_{a_1=t.a_1\wedge\ldots\wedge a_m=t.a_m}\left(Z1\right)\right)\right)$ (*)

$u \leftarrow le\_tuplo(\mathcal{U})$

primeiro_valor ← *u.dom*

$u \leftarrow le\_tuplo(\mathcal{U})$

segundo_valor ← *u.dom*

se *pqd* = "valor incompleto" **então**

se *length*(*primeiro_valor*) < *length*(*segundo_valor*) **então**

actualiza_valor(*t*, *primeiro_valor*)

fim se

senão

se *length*(*primeiro_valor*) > *length*(*segundo_valor*) **então**

actualiza_valor(*t*, *primeiro_valor*)

fim se

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: No caso da correcção de valor sobrecarregado, na expressão em álgebra relacional assinalada com um asterisco, a ordenação é efectuada sob a forma descendente.

Procedimento actualiza_valor(*t*, *novo_valor***)** $\stackrel{\text{def}}{=}$

início

$v \leftarrow \sigma_{a_1=t.a_1 \wedge \ldots \wedge a_m=t.a_m}(\mathcal{R})$

$\mathcal{R} \leftarrow \mathcal{R} - v$

$v.atr_1 \leftarrow novo\_valor$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(v.a_1,\ldots,v.a_m,v.a_{m+1},\ldots,v.a_{m+n})\}$

fim

6.3.1.2 Erro ortográfico

Formalização Sintáctica

TipoOperação MISSPELLING-ERRORS ColunasAlvo DasTabelas DasBasesDados
UsandoCondição

Formalização Semântica

A relação que armazena os erros ortográficos detectados (*i.e.*, *Z1*) contém, para cada um destes (sempre que tal foi possível), as palavras correctas semelhantes identificadas no dicionário. Em cada erro ortográfico, as palavras similares identificadas possuem um grau de semelhança associado resultante da métrica utilizada. A correcção automática envolve substituir o erro ortográfico pela palavra correcta que apresenta o maior grau de semelhança. Apesar de solucionar o PQD, nem sempre a substituição de um erro ortográfico pela palavra semelhante de maior proximidade produz o resultado pretendido. Naturalmente, este tipo de correcção deve ser efectuado com alguma prudência. Compete ao utilizador decidir se o utiliza ou não. No caso de existir mais do que uma palavra com igual grau de semelhança, não é efectuada qualquer correcção. Nestes casos, terá de ser o utilizador a solucionar individualmente cada erro ortográfico. Uma outra situação que também impede a realização de correcção automática e obriga à intervenção do utilizador, advém da não identificação de palavras semelhantes a um dado erro ortográfico detectado. Isto ocorre quando não há qualquer semelhança entre o erro ortográfico e as palavras que constam do dicionário.

Algoritmo Correcção_Erro_Ortográfico $\stackrel{\text{def}}{=}$

seja *pal_err* o atributo de *Z1* onde se encontram as palavras nas quais foram detectados erros ortográficos.

seja *pal_semelh* o atributo de *Z1* onde se encontram as palavras lexicalmente similares às que se encontram no atributo *pal_err*, mas que não contêm erros ortográficos.

seja *grau_semelh* o atributo de *Z1* que contém o grau de semelhança existente entre *pal_err* e *pal_semelh*. Este valor resulta da OD, tendo sido gerado pela métrica de avaliação de semelhança utilizada.

início

$\mathcal{T} \leftarrow {}_{a_1,\ldots,a_m,pal\_err}\gamma_{count(pal\_err)}\left(\sigma_{cond}\left(\mathcal{Z}1\right)\right)$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$\mathcal{U} \leftarrow \tau_{grau\_semelh}\left(\pi_{pal\_semelh,grau\_semelh}\left(\sigma_{a_1=t.a_1\wedge\ldots\wedge a_m=t.a_m\wedge pal\_err=t.pal\_err}\left(\mathcal{Z}1\right)\right)\right)^{(*)}$

$u \leftarrow le\_tuplo(\mathcal{U})$

se $u.pal\_semelh \neq null$ **então**

$prim\_grau\_semelh \leftarrow u.grau\_semelh$

$prim\_pal\_semelh \leftarrow u.pal\_semelh$

$u \leftarrow le\_tuplo(\mathcal{U})$

se $u = null$ **ou** $u.grau\_semelh < prim\_grau\_semelh$ **então**

$v \leftarrow \sigma_{a_1=t.a_1\wedge\ldots\wedge a_m=t.a_m}\left(\mathcal{R}\right)$

$\mathcal{R} \leftarrow \mathcal{R} - v$

$v.atr_1 \leftarrow substitui\_substring(v.atr_1, t.pal\_err, prim\_pal\_semelh)$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(v.a_1,\ldots,v.a_m,v.a_{m+1},\ldots,v.a_{m+n})\}$

fim se

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: Na expressão em álgebra relacional assinalada com um asterisco, a ordenação é efectuada sob a forma descendente.

Em alternativa à correcção dos erros ortográficos pelas palavras de maior proximidade, é possível proceder, pura e simplesmente, à sua eliminação. Cada erro ortográfico detectado é removido do valor do atributo. A remoção apenas é susceptível de ser efectuada quando o valor do atributo é composto por mais do que uma palavra. No caso de existir uma única palavra, a remoção desta implicaria que o valor do atributo ficasse a nulo. De facto, esta operação encontra-se orientada

para aquelas situações em que se considera importante a inexistência de erros ortográficos, nem que para isso seja necessário a eliminação da respectiva palavra do valor do atributo e que desta não resultam consequências de maior a nível de perda de informação (*e.g.*: em muitas situações, a eliminação de uma das palavras do nome de um indivíduo não é relevante).

Algoritmo Remoção_Erro_Ortográfico $\stackrel{def}{=}$

seja "o" o operador de concatenação de strings.

início

$\mathcal{T} \leftarrow {}_{a_1,\ldots,a_m,pal\_err}\gamma\left(\sigma_{cond}\left(\mathcal{Z}1\right)\right)$

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

repetir enquanto $t \neq null$

$u \leftarrow \sigma_{a_1=t.a_1 \wedge \ldots \wedge a_m=t.a_m}\left(\mathcal{R}\right)$

$\mathcal{R} \leftarrow \mathcal{R} - u$

novo_valor ← *substitui_substring*($u.atr_1$, "□" o *t.pal_err* o "□", "□")

se *novo_valor* = $u.atr_1$ **então**

novo_valor ← *substitui_substring*($u.atr_1$, "□" o *t.pal_err*, "")

se *novo_valor* = $u.atr_1$ **então**

novo_valor ← *substitui_substring*($u.atr_1$, *t.pal_err* o "□", "")

fim se

fim se

$u.atr_1 \leftarrow$ *novo_valor*

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(u.a_1,\ldots,u.a_m,u.a_{m+1},\ldots,u.a_{m+n})\}$

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

fim repetir

fim

Notas: O símbolo "□" representa o caracter espaço.

O significado do atributo *pal_err* e da função *substitui_substring*(*str*, *substr1*, *substr2*) encontra-se definido na formalização anterior (correcção de erro ortográfico).

6.3.2 Contexto Multi-Valor

Nas subsecções seguintes apresentam-se as formalizações das OC dos PQD que ocorrem ao nível dos vários valores do atributo.

6.3.2.1 Existência de sinónimos

Formalização Sintáctica

TipoOperação EXISTENCE-OF-SYNONYMS ColunasAlvo DasTabelas DasBasesDados DefiniçãoTermosUniformização

Formalização Semântica

A relação que armazena os sinónimos detectados no atributo (*i.e.*, *Z1*) permite agrupá-los por *clusters* de sinónimos. O utilizador indica entre os sinónimos que formam cada *cluster* qual será utilizado como termo de uniformização dos demais. A correcção automática deste PQD envolve a actualização dos valores do atributo que contêm os sinónimos, com o termo de uniformização seleccionado em cada *cluster*. De referir que a actualização do valor do atributo corresponde a um novo valor que apenas difere do anterior no termo de uniformização que substitui o sinónimo, mantendo-se os restantes termos inalterados (*e.g.*: considerando os termos *canalizador* e *picheleiro* como sinónimos e adoptando este último como termo de uniformização, o valor *técnico canalizador* é actualizado para *técnico picheleiro*).

Algoritmo Correcção_Existência_Sinónimos $\stackrel{\text{def}}{=}$

seja *idcluster* o atributo de *Z1* que identifica o *cluster* de sinónimos a que cada termo sinónimo detectado pertence.

seja *vec_sinonimos* um vector que contém os termos de uniformização utilizados em substituição de todos os outros seus sinónimos. Este vector resulta da especificação da OC.

seja $\mathcal{T}$ a relação onde se encontra armazenado o dicionário de sinónimos, com esquema: $\mathcal{T}$(*idsin*, *idcluster*, *sin*), em que *idsin* constitui a chave primária, *idcluster* o *cluster* a que pertence o sinónimo e *sin* o termo sinónimo.

início

repetir para $i \leftarrow 1$ **até** *dimensão*(*vec_sinonimos*)

$$u \leftarrow \pi_{idcluster}(\sigma_{atr_1=vec\_sinonimos[i] \vee atr_1 \text{ like } '\%vec\_sinonimos[i]\%' \vee atr_1 \text{ like } 'vec\_sinonimos[i]\%' \vee \vee atr_1 \text{ like } '\%vec\_sinonimos[i]'}(\mathcal{Z}1))$$

$$\mathcal{V} \leftarrow \pi_{sin}\left(\sigma_{idcluster=u.idcluster \wedge sin \neq vec\_sinonimos[i]}(\mathcal{T})\right)$$

$$\mathcal{X} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}(\sigma_{idcluster=u.idcluster \wedge atr_1 \neq vec\_sinonimos[i] \wedge atr_1 \text{ not like } '\%vec\_sinonimos[i]\%' \wedge \wedge atr_1 \text{ not like } 'vec\_sinonimos[i]\%' \wedge atr_1 \text{ not like } '\%vec\_sinonimos[i]'}(\mathcal{Z}1))$$

$x \leftarrow$ *le_tuplo*($\mathcal{X}$)

repetir enquanto $x \neq$ *null*

$v \leftarrow$ *le_tuplo*($\mathcal{V}$)

novo_valor $\leftarrow x.atr_1$

repetir enquanto $v \neq null$ **e** $novo\_valor = x.atr_1$

$novo\_valor \leftarrow substitui\_substring(x.atr_1, v.atr_1, vec\_sinonimos[i])$

se $novo\_valor = x.atr_1$ **então**

$v \leftarrow le\_tuplo(\mathcal{V})$

fim se

fim repetir

$y \leftarrow \sigma_{a_1=x.a_1 \wedge \ldots \wedge a_m=x.a_m}(\mathcal{R})$

$\mathcal{R} \leftarrow \mathcal{R} - y$

$y.atr_1 \leftarrow novo\_valor$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(y.a_1,\ldots,y.a_m,y.a_{m+1},\ldots,y.a_{m+n})\}$

$reinicializa\_leitura\_tuplos(\mathcal{V})$

$x \leftarrow le\_tuplo(X)$

fim repetir

fim repetir

fim

Em alternativa à correcção dos sinónimos baseada na estratégia de uniformização apresentada, é possível proceder à eliminação dos tuplos que, no atributo em questão, contêm sinónimos de um determinado conjunto de termos definidos pelo utilizador. Assim, o utilizador começa por indicar entre os sinónimos que formam cada *cluster*, qual o termo que prevalece sobre os restantes. A execução automática desta operação envolve a eliminação de todos os tuplos onde constam os sinónimos dos termos eleitos pelo utilizador como "sobreviventes" em cada *cluster*. A eliminação dos tuplos faz sentido em todas aquelas situações em que a execução da operação de correcção anterior introduziria redundância (*i.e.*, tuplos duplicados). Tipicamente, esta situação ocorre em tabelas auxiliares, vulgarmente existentes nas BD (*e.g.*: tabela de profissões). Estas tabelas caracterizam-se por possuírem dois atributos: um identificador/código cuja finalidade é a de ser chave primária e um descritivo onde se encontram os valores que a tabela armazena. Neste tipo de tabelas, a uniformização de dois termos sinónimos (*e.g.*: *picheleiro* e *canalizador*) num só, resultaria em dois tuplos duplicados. Como tal, a solução passa por manter apenas um dos tuplos eliminando os restantes. Na eliminação dos tuplos é levada em consideração a não violação da integridade referencial da BD. Antes da eliminação de um tuplo, os valores das chaves estrangeiras dos tuplos pertencentes às tabelas com as quais há um relacionamento e se referem à chave primária do tuplo em questão (a eliminar), são actualizados para o valor da chave primária do tuplo "sobrevivente". A manutenção automática da integridade referencial encontra-se dependente da existência das respectivas restrições na BD.

Algoritmo Remoção_Existência_Sinónimos $\stackrel{\text{def}}{=}$

seja w o número de relações $\mathcal{S}$ com as quais $\mathcal{R}$ se encontra relacionada.

seja $\mathcal{B} = \{b_{r+1},\ldots,b_{r+s}\}$ o conjunto de atributos de $\mathcal{S}$ que não fazem parte da sua chave primária.

seja C o conjunto de atributos que formam a chave estrangeira de $\mathcal{S}$, assim definido: $C = \{c \mid c$ faz parte da chave estrangeira de $\mathcal{S} \wedge c \in \mathcal{B}\}$, *i.e.*, $C \subseteq \mathcal{B}$

início

repetir para $i \leftarrow 1$ **até** *dimensão*(*vec_sinonimos*)

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,idcluster}(\sigma_{atr_1=vec\_sin\,onimos[i] \vee atr_1 \text{ like '\%}vec\_sin\,onimos[i]\text{\%' } \vee \vee atr_1 \text{ like '}vec\_sin\,onimos[i]\text{\%' } \vee atr_1 \text{ like '\%}vec\_sin\,onimos[i]\text{'}}(\mathcal{Z}1))$$

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

se $t \neq$ *null* **então**

$$\mathcal{U} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}(\sigma_{idcluster=t.idcluster \wedge atr_1 \neq vec\_sin\,onimos[i] \wedge atr_1 \text{ not like '\%}vec\_sin\,onimos[i]\text{\%' } \wedge \wedge atr_1 \text{ not like '}vec\_sin\,onimos[i]\text{\%' } \wedge atr_1 \text{ not like '\%}vec\_sin\,onimos[i]\text{'}}(\mathcal{Z}1))$$

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

repetir enquanto $u \neq$ *null*

repetir para $i \leftarrow 1$ **até** w

$\mathcal{V} \leftarrow \sigma_{c_1=u.a_1 \wedge \ldots \wedge c_m=u.a_m}(S_i)$

$S_i \leftarrow S_i - \mathcal{V}$

$v \leftarrow$ *le_tuplo*($\mathcal{V}$)

repetir enquanto $v \neq$ *null*

$v.c_1 \leftarrow t.a_1$

$\vdots$

$v.c_m \leftarrow t.a_m$

$S_i \leftarrow S_i \cup \{(v.b_1,\ldots,v.b_r,v.b_{r+1},\ldots,v.b_{r+s})\}$

$v \leftarrow$ *le_tuplo*($\mathcal{V}$)

fim repetir

fim repetir

$\mathcal{R} \leftarrow \mathcal{R} - \left(\sigma_{a_1=u.a_1 \wedge \ldots \wedge a_m=u.a_m}(\mathcal{R})\right)$

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

fim repetir

fim se

fim repetir

fim

Nota: O significado dos atributos *idcluster* e *sin* encontra-se definido na formalização anterior (correcção da existência de sinónimos).

6.3.2.2 Violação de unicidade

Formalização Sintáctica

REMOVE UNIQUENESS-VIOLATION ColunasAlvo DasTabelas DasBasesDados
UsandoCondição

Formalização Semântica

A relação resultante da respectiva OD (*i.e.*, *Z1*) permite agrupar os tuplos em função dos valores das violações de unicidade que ocorrem no atributo. A operação aqui proposta efectua esse agrupamento e em cada conjunto de tuplos resultante, à excepção de um, elimina todos os outros. Esta eliminação pressupõe que os tuplos onde se verifica a violação de unicidade constituem duplicados entre si. A existência de violações de unicidade que, no fundo, traduzem a existência de tuplos redundantes é comum em tabelas auxiliares (*e.g.*: tabela de países). Este tipo de tabelas, composto por apenas dois campos (código e descritivo), é vulgarmente utilizado em BD. Frequentemente, existe mais do que um tuplo com a mesma informação, o que constitui uma violação à unicidade do descritivo (*e.g.*: o país *Portugal* encontra-se em dois tuplos com códigos diferentes). Nestes casos, pode manter-se apenas um dos tuplos, removendo todos os outros. Quando se procede à remoção dos tuplos, a não violação da integridade referencial da BD é tida em consideração por esta operação (o significado desta afirmação encontra-se explicado na parte final da descrição da operação anterior de remoção de sinónimos). Noutros enquadramentos (que não do tipo concreto referido) em que se verifiquem violações de unicidade num atributo de uma determinada tabela, só a intervenção manual do utilizador as pode solucionar.

Algoritmo Remoção_Violação_Unicidade $\stackrel{\text{def}}{=}$

início

$\mathcal{T} \leftarrow \tau_{atr_1}\left(\pi_{a_1,\ldots,a_m,atr_1}\left(\sigma_{cond}\left(Z1\right)\right)\right)$

$t \leftarrow le\_tuplo(\mathcal{T})$

$flag\_eof \leftarrow falso$

repetir enquanto $flag\_eof = falso$

$primeiro\_valor \leftarrow t.atr_1$

$proximo\_valor \leftarrow t.atr_1$

$u \leftarrow t$

repetir enquanto $flag\_eof = falso$ **e** $proximo\_valor = primeiro\_valor$

$t \leftarrow le\_tuplo(\mathcal{T})$

se $t = null$ **então**

$flag\_eof \leftarrow verdadeiro$

senão

$proximo\_valor \leftarrow t.atr_1$

se $proximo\_valor = primeiro\_valor$ **então**

repetir para $i \leftarrow 1$ **até** w

$\mathcal{V} \leftarrow \sigma_{c_1=t.a_1 \wedge \ldots \wedge c_m=t.a_m}(S_i)$

$S_i \leftarrow S_i - \mathcal{V}$

$v \leftarrow le\_tuplo(\mathcal{V})$

repetir enquanto $v \neq null$

$v.c_1 \leftarrow u.a_1$

$\vdots$

$v.c_m \leftarrow u.a_m$

$S_i \leftarrow S_i \cup \{(v.b_1,\ldots,v.b_r,v.b_{r+1},\ldots,v.b_{r+s})\}$

$v \leftarrow le\_tuplo(\mathcal{V})$

fim repetir

fim repetir

$\mathcal{R} \leftarrow \mathcal{R} - \left(\sigma_{a_1=t.a_1 \wedge \ldots \wedge a_m=t.a_m}(\mathcal{R})\right)$

fim se

fim se

fim repetir

fim repetir

fim

Nota: O significado da variável w e dos atributos $c_1,\ldots,c_m$ encontra-se definido na formalização anterior.

6.3.2.3 Violação de restrição de integridade

Formalização Sintáctica

CORRECT INTEGRITY-CONSTRAINT-VIOLATION AT LEVEL OF COLUMN
ColunasAlvo DasTabelas
DasBasesDados DefiniçãoAtribuições
UsandoTabelas UsandoCondição

Formalização Semântica

A relação resultante da OD respectiva (*i.e.*, $\mathcal{Z}1$), além de conter os valores que constituem violações à restrição de integridade ao nível da coluna, pode conter também informação sobre os tuplos onde estas ocorrem. No caso desta informação não existir, isso significa que a violação da restrição de integridade resulta dos valores do atributo na globalidade de tuplos existentes (*e.g.*: o somatório de uma distribuição percentual armazenada num atributo não é igual a 100%). A OC aqui proposta assenta na alteração aos valores do atributo nos tuplos especificados pelo utilizador, entre os que se encontram envolvidos no PQD (*i.e.*, todos os tuplos de $\mathcal{R}$ ou apenas aqueles que constam de $\mathcal{Z}1$). Assim, para cada um desses tuplos e com base num conjunto de atribuições especificadas pelo utilizador, a OC actualiza os valores do atributo necessários à resolução do PQD, em função dos valores existentes noutros atributos. Cada atribuição é representada em função de uma expressão que envolve exclusivamente atributos, variáveis (*e.g.*: contendo o resultado de uma atribuição anterior) e/ou constantes (*e.g.*: um determinado valor numérico).

Algoritmo Correcção_Violação_Restrição_Integridade $\stackrel{\text{def}}{=}$

seja $\mathcal{A} = \{a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos de $\mathcal{R}$ que não fazem parte da sua chave primária.

seja $\mathcal{C}$ o conjunto de atributos assim definido: $\mathcal{C} = \{c \mid c$ é usado na correcção da violação da restrição de integridade em $atr_1 \wedge c \in \mathcal{A}\}$, *i.e.*, $\mathcal{C} \subseteq \mathcal{A}$. Este conjunto resulta da especificação da OC.

seja e a cardinalidade de $\mathcal{C}$.

início

se *existem_atribs_na_relação*(*vec_atribs_chave_prim_*$\mathcal{R}$, $\mathcal{Z}1$) = *verdadeiro* **então**

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_e}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \mathcal{Z}1\right)\right)$$

senão

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_e}\left(\sigma_{cond}\left(\mathcal{R}\right)\right)$$

fim se

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

repetir enquanto $t \neq$ *null*

$v_1 \leftarrow atrib_1$

$v_2 \leftarrow atrib_2$

$\vdots$

$v_k \leftarrow atrib_k$

$$u \leftarrow \sigma_{a_1=t.a_1 \wedge \ldots \wedge a_m=t.a_m}\left(\mathcal{R}\right)$$

$\mathcal{R} \leftarrow \mathcal{R} - u$

$u.atr_1 \leftarrow v_k$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(u.a_1,\ldots,u.a_m,u.a_{m+1},\ldots,u.a_{m+n})\}$

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Notas: As atribuições $atrib_i$ são expressas em função dos atributos de C.

A última atribuição (no caso, $atrib_k$) define o novo valor de atr_1.

Na correcção de violação de restrição de integridade ao nível da coluna pode ser necessário recorrer a funções definidas/implementadas pelo utilizador (*e.g.*: devido à complexidade das operações envolvidas na correcção). Nesta situação, o conjunto de valores de cada atributo necessário à correcção do PQD pode ser passado como parâmetro, de acordo com os requisitos de cada FDU que consta da sequência de atribuições especificada na OC. O conjunto de valores de cada atributo provém dos tuplos envolvidos no PQD detectado (*i.e.*, todos os tuplos de $\mathcal{R}$ ou apenas os que constam de $\mathcal{Z}1$). As funções também podem receber outros parâmetros, como variáveis e constantes. Como resultado da sua execução, cada função retorna um valor atómico ou uma estrutura de dados (*e.g.*: um vector; uma matriz). A última função da sequência de atribuições (na formalização seguinte representada por fdu_k), obrigatoriamente retorna uma estrutura de dados unidimensional (*i.e.*, um vector) de um dado tipo (*e.g.*: inteiro; real). Cada elemento do vector contém um valor que actualiza o valor do atributo nos diferentes tuplos em questão. A ordem destes valores necessita de ser a mesma da considerada na extracção dos valores do atributo. Este é um requisito a que o vector tem de obedecer, para que o valor do atributo nos tuplos seja correctamente actualizado. A formalização anterior sofre as alterações a seguir apresentadas, de modo a reflectir a correcção da violação de restrição de integridade baseada em FDU. O que permanece inalterado relativamente à formalização anterior encontra-se representado sob a forma de reticências.

Algoritmo Correcção_Violação_Restrição_Integridade_Baseada_em_FDU $\stackrel{\text{def}}{=}$

início

$\vdots$

fim se

$v_1 \leftarrow fdu_1\left(p_{1,1},p_{1,2},\ldots,p_{1,i}\right)$

$v_2 \leftarrow fdu_2\left(p_{2,1},p_{2,2},\ldots,p_{2,j}\right)$

$\vdots$

$v_{k} \leftarrow fdu_{k}\left(p_{k,1}, p_{k,2}, \ldots, p_{k,l}\right)$

$h \leftarrow 0$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$h \leftarrow h + 1$

se $v_{k}[h] \neq t.atr_1$ **então**

$u \leftarrow \sigma_{a_1 = t.a_1 \wedge \ldots \wedge a_m = t.a_m}(\mathcal{R})$

$\mathcal{R} \leftarrow \mathcal{R} - u$

$u.atr_1 \leftarrow v_{k}[h]$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(u.a_1, \ldots, u.a_m, u.a_{m+1}, \ldots, u.a_{m+n})\}$

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: Os valores dos atributos de C podem constituir parâmetros de uma ou mais funções fdu_i.

6.4 Problemas ao Nível do Tuplo

Nesta secção apresenta-se a formalização da OC do único PQD que ocorre ao nível do tuplo.

6.4.1 Violação de restrição de integridade

Formalização Sintáctica

CORRECT INTEGRITY-CONSTRAINT-VIOLATION AT LEVEL OF ROW ColunasAlvo
DasTabelas DasBasesDados
DefiniçãoAtribuições UsandoTabelas
UsandoCondição

Formalização Semântica

A relação resultante da OD (*i.e.*, *Z1*) contém informação sobre os tuplos nos quais foram identificadas violações à restrição de integridade. Para cada um destes tuplos e com base num conjunto de atribuições especificadas pelo utilizador, esta operação actualiza os valores dos atributos necessários à correcção do PQD, em função dos valores existentes noutros atributos do mesmo tuplo. O utilizador tem possibilidade de limitar quais os tuplos objecto de correcção

mediante uma condição. Cada atribuição é representada em função de uma expressão que envolve exclusivamente atributos, variáveis e/ou constantes.

Algoritmo Correcção_Violação_Restrição_Integridade $\stackrel{\text{def}}{=}$

seja $\mathcal{A} = \{a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos de $\mathcal{R}$ que não fazem parte da sua chave primária.

seja C o conjunto de atributos assim definido: $C = \{c \mid c$ é usado na correcção da violação da restrição de integridade $\wedge\ c \in \mathcal{A}\}$, *i.e.*, $C \subseteq \mathcal{A}$. Este conjunto resulta da especificação da OC.

seja e a cardinalidade de C.

seja k o número de atribuições existentes na sequência de atribuições da OC.

sejam x, y e z números inteiros compreendidos no intervalo [1; e].

sejam o, p e q números inteiros compreendidos no intervalo [1; k].

início

$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_e}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \mathcal{Z}1\right)\right)$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$v_1 \leftarrow atrib_1$

$v_2 \leftarrow atrib_2$

$\vdots$

$v_k \leftarrow atrib_k$

$u \leftarrow \sigma_{a_1=t.a_1\wedge\ldots\wedge a_m=t.a_m}\left(\mathcal{R}\right)$

$\mathcal{R} \leftarrow \mathcal{R} - u$

$u.c_x \leftarrow v_o$

$u.c_y \leftarrow v_p$

$\vdots$

$u.c_z \leftarrow v_q$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(u.a_1,\ldots,u.a_m,u.a_{m+1},\ldots,u.a_{m+n})\}$

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: As atribuições $atrib_i$ são expressas em função dos atributos de C.

Na correcção de violação de restrição de integridade ao nível do tuplo também pode ser necessário recorrer a FDU. Neste caso, os valores dos atributos de cada tuplo no qual foi detectado este PQD podem ser passados como parâmetros, de acordo com os requisitos de cada

FDU que consta da sequência de atribuições. As funções também podem receber outros parâmetros, como variáveis e constantes. Como resultado, as funções retornam um valor atómico. A formalização anterior é objecto das alterações apresentadas de seguida, de modo a reflectir a correcção da violação de restrição de integridade baseada em FDU. As reticências representam aquilo que permanece inalterado relativamente à formalização anterior.

Algoritmo Correcção_Violação_Restrição_Integridade_Baseada_em_FDU $\stackrel{\text{def}}{=}$

início

$\vdots$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$v_1 \leftarrow fdu_1\left(p_{1,1}, p_{1,2}, \ldots, p_{1,i}\right)$

$v_2 \leftarrow fdu_2\left(p_{2,1}, p_{2,2}, \ldots, p_{2,j}\right)$

$\vdots$

$v_k \leftarrow fdu_k\left(p_{k,1}, p_{k,2}, \ldots, p_{k,l}\right)$

$u \leftarrow \sigma_{a_1 = t.a_1 \wedge \ldots \wedge a_m = t.a_m}(\mathcal{R})$

$\mathcal{R} \leftarrow \mathcal{R} - u$

$u.c_x \leftarrow v_o$

$u.c_y \leftarrow v_p$

$\vdots$

$u.c_z \leftarrow v_q$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(u.a_1, \ldots, u.a_m, u.a_{m+1}, \ldots, u.a_{m+n})\}$

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: Os valores dos atributos de C podem constituir parâmetros de uma ou mais funções fdu_i.

6.5 Problemas ao Nível da Relação

Nesta secção apresentam-se as formalizações das OC dos PQD que ocorrem ao nível da relação.

6.5.1 Violação de dependência funcional

Formalização Sintáctica

CORRECT FUNCTIONAL-DEPENDENCY-VIOLATION ColunasAlvo DasTabelas
DasBasesDados
DefiniçãoColunasDependentes
UsandoCondição

Formalização Semântica

A relação resultante da respectiva OD permite agrupar e determinar o número de ocorrências associado a cada conjunto de valores envolvido em cada violação de dependência funcional. A OC proposta assenta no *princípio da maioria*. Todos os valores do atributo, de cada conjunto de valores envolvidos numa violação de dependência funcional cujo número de ocorrências não seja máximo, são substituídos pelo valor do atributo do conjunto de valores cujo número de ocorrências é máximo. No caso de não existir um conjunto de valores que possua um número máximo de ocorrências, não é efectuada qualquer correcção, uma vez que o *princípio da maioria* não é aplicável. A resolução destas violações de dependência funcional fica a cargo do utilizador. De modo a clarificar o modo de funcionamento desta OC, suponha-se que foi detectada a seguinte violação de dependência funcional entre o código postal (na sua forma abreviada) e a localidade: 4000 → Porto; 4000 → Porto; 4000 → Porto; 4000 → Maia. Uma vez que o número de ocorrências de "4000 → Porto" é igual a três e o número de ocorrências de "4000 → Maia" é apenas de um, por aplicação do princípio da maioria este último é transformado também em "4000 → Porto".

Algoritmo Correcção_Violação_Dependência_Funcional $\stackrel{\text{def}}{=}$

seja $C = \{c_1,\ldots,c_o\}$ o conjunto de atributos de $\mathcal{Z}1$ onde se encontram os valores que constituem violação de dependência funcional, juntamente com os valores de atr_1. Todos os atributos são originários de $\mathcal{R}$.

início

$\mathcal{T} \leftarrow \pi_{c_1,\ldots,c_o}\left(\sigma_{cond}\left(\mathcal{Z}1\right)\right)$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$\mathcal{U} \leftarrow \tau_{count(atr_1)}\left({}_{c_1,\ldots,c_o,atr_1}\gamma_{count(atr_1)}\left(\sigma_{c_1=t.c_1\wedge\ldots\wedge c_o=t.c_o}\left(\mathcal{Z}1\right)\right)\right)$ (*)

$u \leftarrow le\_tuplo(\mathcal{U})$

$valor\_actualiz \leftarrow u.atr_1$

$max\_ocorr \leftarrow u.count(atr_1)$

$u \leftarrow le\_tuplo(\mathcal{U})$

se $u.count(atr_1) < max\_ocorr$ **então**

repetir enquanto $u \neq null$

$v \leftarrow \sigma_{c_1=u.c_1 \wedge \ldots \wedge c_o=u.c_o \wedge atr_1=u.atr_1}(\mathcal{R})$

$\mathcal{R} \leftarrow \mathcal{R} - v$

$v.atr_1 \leftarrow valor\_actualiz$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(v.a_1,\ldots,v.a_m,v.a_{m+1},\ldots,v.a_{m+n})\}$

$u \leftarrow le\_tuplo(\mathcal{U})$

fim repetir

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: Na expressão em álgebra relacional assinalada com um asterisco, a ordenação é efectuada sob a forma descendente.

6.5.2 Tuplos duplicados

Formalização Sintáctica

REMOVE DUPLICATE-TUPLES DasTabelas DasBasesDados DefiniçãoColunas
UsandoAtribuições UsandoCondição

Formalização Semântica

A relação que resulta da respectiva OD permite agrupar os tuplos em função do *cluster* de duplicados a que estes pertencem. À semelhança da OC de violação de unicidade, esta OC agrupa os tuplos por *cluster* de duplicados e em cada um destes, à excepção de um tuplo, elimina todos os outros. Esta eliminação pressupõe que cada *cluster* é composto por duplicados rigorosamente iguais entre si (*i.e.*, duplicados exactos). Na eliminação dos tuplos é levada em consideração a não violação da integridade referencial da BD. Antes da eliminação de um tuplo, os valores das chaves estrangeiras dos tuplos pertencentes às tabelas com as quais há um relacionamento e se referem à chave primária do tuplo em questão (a eliminar), são actualizados para o valor da chave primária do tuplo "sobrevivente". A manutenção automática da integridade referencial encontra-se dependente da existência das respectivas restrições na BD.

Algoritmo Remoção_Tuplos_Duplicados $\stackrel{def}{=}$

seja *idcluster* o atributo de $\mathcal{Z}1$ que identifica o *cluster* de duplicados a que cada tuplo detectado como duplicado pertence.

seja w o número de relações $\mathcal{S}$ com as quais $\mathcal{R}$ se encontra relacionada.

seja $\mathcal{B} = \{b_{r+1},\ldots,b_{r+s}\}$ o conjunto de atributos de $\mathcal{S}$ que não fazem parte da sua chave primária.

seja $\mathcal{C}$ o conjunto de atributos que formam a chave estrangeira de $\mathcal{S}$, assim definido: $\mathcal{C} = \{c \mid c$ faz parte da chave estrangeira de $\mathcal{S} \wedge c \in \mathcal{B}\}$, *i.e.*, $\mathcal{C} \subseteq \mathcal{B}$.

incio

$\mathcal{T} \leftarrow \pi_{idcluster}\left(\sigma_{cond}\left(\mathcal{Z}1\right)\right)$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$\mathcal{U} \leftarrow \pi_{\mathcal{R}.a_1,\ldots,\mathcal{R}.a_m,\mathcal{R}.a_{m+1},\ldots,\mathcal{R}.a_{m+n}}\left(\sigma_{idcluster=t.idcluster}\left(\mathcal{R} \rhd\lhd \mathcal{Z}1\right)\right)$

$u \leftarrow le\_tuplo(\mathcal{U})$

$v \leftarrow u$

$u \leftarrow le\_tuplo(\mathcal{U})$

repetir enquanto $u \neq null$

repetir para $i \leftarrow 1$ **até** w

$\mathcal{X} \leftarrow \sigma_{c_1=u.a_1\wedge\ldots\wedge c_m=u.a_m}\left(S_i\right)$

$S_i \leftarrow S_i - \mathcal{X}$

$x \leftarrow le\_tuplo(\mathcal{X})$

repetir enquanto $x \neq null$

$x.c_1 \leftarrow v.a_1$

$\vdots$

$x.c_m \leftarrow v.a_m$

$S_i \leftarrow S_i \cup \{(x.b_1,\ldots,x.b_r,x.b_{r+1},\ldots,x.b_{r+s})\}$

$x \leftarrow le\_tuplo(\mathcal{X})$

fim repetir

fim repetir

$\mathcal{R} \leftarrow \mathcal{R} - \left(\sigma_{a_1=u.a_1\wedge\ldots\wedge a_m=u.a_m}\left(\mathcal{R}\right)\right)$

$u \leftarrow le\_tuplo(\mathcal{U})$

fim repetir

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

No caso de cada *cluster* de duplicados não ser constituído por duplicados rigorosamente iguais (o que na realidade corresponde à situação mais comum), a operação de remoção anterior não é susceptível de ser aplicada. Nestes casos é necessário identificar o tuplo que "sobrevive" e, por consequência, quais os que serão eliminados. Muitas vezes, o tuplo "sobrevivente" resulta de uma combinação dos valores dos atributos dos diversos duplicados existentes. Daqui resulta um novo tuplo que substitui os duplicados que estiveram na sua génese. Nestas situações, o utilizador necessita de definir o critério de sobrevivência dos tuplos ou dos valores dos atributos (*e.g.*: o valor mais completo), em cada *cluster* de duplicados. Estes critérios necessitam de estar implementados em FDU. Cada função, dependendo dos seus requisitos específicos, pode receber como parâmetros os valores de um ou mais atributos, resultantes de cada *cluster* de duplicados identificado. Além dos valores dos atributos, as funções podem receber outros parâmetros, como variáveis (*e.g.*: contendo o resultado da execução de uma outra função) e constantes. Como resultado da sua execução, cada função retorna um valor atómico ou uma estrutura de dados (*e.g.*: um vector; uma matriz). O resultado de cada função responsável por gerar um valor "sobrevivente" é obrigatoriamente um valor atómico. Na formalização a seguir apresentada, estes correspondem aos valores que se encontram nas atribuições que envolvem os atributos da relação $\mathcal{R}$ (*i.e.*, a_x, a_y, …, a_z). De referir que à semelhança da anterior, esta operação também tem em conta a não violação da integridade referencial. Para tal, é necessário que as respectivas restrições tenham sido definidas na BD em questão.

Algoritmo Remoção_Tuplos_Duplicados_Baseada_em_FDU $\stackrel{\text{def}}{=}$

seja $\mathcal{A} = \{a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos de $\mathcal{R}$ que não fazem parte da sua chave primária.

seja k o número de atribuições existentes na sequência de atribuições da OC.

sejam x, y e z números inteiros compreendidos no intervalo $[m + 1; m + n]$.

sejam o, p e q números inteiros compreendidos no intervalo $[1; k]$.

início

$\mathcal{T} \leftarrow \pi_{idcluster}\left(\sigma_{cond}\left(\mathcal{Z}1\right)\right)$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$\mathcal{U} \leftarrow \pi_{\mathcal{R}.a_1,\ldots,\mathcal{R}.a_m,\mathcal{R}.a_{m+1},\ldots,\mathcal{R}.a_{m+n}}\left(\sigma_{idcluster=t.idcluster}\left(\mathcal{R} \rhd\lhd \mathcal{Z}1\right)\right)$

$v_1 \leftarrow fdu_1\left(p_{1,1},p_{1,2},\ldots,p_{1,i}\right)$

$v_2 \leftarrow fdu_2\left(p_{2,1},p_{2,2},\ldots,p_{2,j}\right)$

$\vdots$

$v_{k} \leftarrow fdu_{k}\left(p_{k,1}, p_{k,2}, \ldots, p_{k,l}\right)$

$u \leftarrow le\_tuplo(\mathcal{U})$

$v \leftarrow \sigma_{a_1=u.a_1 \wedge \ldots \wedge a_m=u.a_m}(\mathcal{R})$

$\mathcal{R} \leftarrow \mathcal{R} - v$

$v.a_x \leftarrow v_o$

$v.a_y \leftarrow v_p$

$\vdots$

$v.a_z \leftarrow v_q$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(v.a_1, \ldots, v.a_m, v.a_{m+1}, \ldots, v.a_{m+n})\}$

$u \leftarrow le\_tuplo(\mathcal{U})$

repetir enquanto $u \neq null$

repetir para $i \leftarrow 1$ **até** w

$X \leftarrow \sigma_{c_1=u.a_1 \wedge \ldots \wedge c_m=u.a_m}(S_i)$

$S_i \leftarrow S_i - X$

$x \leftarrow le\_tuplo(X)$

repetir enquanto $x \neq null$

$x.c_1 \leftarrow v.a_1$

$\vdots$

$x.c_m \leftarrow v.a_m$

$S_i \leftarrow S_i \cup \{(x.b_1, \ldots, x.b_r, x.b_{r+1}, \ldots, x.b_{r+s})\}$

$x \leftarrow le\_tuplo(X)$

fim repetir

fim repetir

$\mathcal{R} \leftarrow \mathcal{R} - \left(\sigma_{a_1=u.a_1 \wedge \ldots \wedge a_m=u.a_m}(\mathcal{R})\right)$

$u \leftarrow le\_tuplo(\mathcal{U})$

fim repetir

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: O significado de w e dos atributos *idcluster*, $c_1, \ldots, c_m$ encontra-se definido na formalização anterior.

6.5.3 Violação de restrição de integridade

Formalização Sintáctica

CORRECT INTEGRITY-CONSTRAINT-VIOLATION AT LEVEL OF RELATION
ColunasAlvo DasTabelas
DasBasesDados DefiniçãoAtribuições
UsandoTabelas UsandoCondição

Formalização Semântica

A relação que resulta da OD respectiva (*i.e.*, $\mathcal{Z}1$), além de conter os valores que representam violações à restrição de integridade ao nível da relação, também pode conter informação sobre os tuplos que as provocam. No caso desta informação não existir, isso significa que a violação da restrição de integridade resulta dos valores dos atributos envolvidos, quando se considera a globalidade dos tuplos existentes (*e.g.*: o valor total dos artigos em stock resultantes do somatório dos valores que resultam da multiplicação da quantidade pelo preço unitário de cada artigo, tem de ser igual a um determinado valor fornecido pelo utilizador). A OC proposta baseia-se na alteração aos valores dos atributos nos tuplos especificados pelo utilizador, entre os que se encontram envolvidos no PQD (*i.e.*, todos os tuplos de $\mathcal{R}$ ou apenas aqueles que constam de $\mathcal{Z}1$). Assim, para cada um desses tuplos e com base no conjunto de atribuições especificadas pelo utilizador, a OC actualiza os valores dos atributos necessários à resolução do PQD, em função dos valores existentes noutros atributos. Cada atribuição é representada em função de uma expressão que envolve exclusivamente atributos, variáveis e/ou constantes.

Algoritmo Correcção_Violação_Restrição_Integridade $\stackrel{\text{def}}{=}$

seja $\mathcal{A} = \{a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos de $\mathcal{R}$ que não fazem parte da sua chave primária.

seja $\mathcal{C}$ o conjunto de atributos assim definido: $\mathcal{C} = \{c \mid c$ é usado na correcção da violação da restrição de integridade $\wedge\ c \in \mathcal{A}\}$, *i.e.*, $\mathcal{C} \subseteq \mathcal{A}$. Este conjunto resulta da especificação da OC.

seja e a cardinalidade de $\mathcal{C}$.

seja k o número de atribuições existentes na sequência de atribuições da OC.

sejam x, y e z números inteiros compreendidos no intervalo [1; e].

sejam o, p e q números inteiros compreendidos no intervalo [1; k].

início

se *existem_atribs_na_relação*(*vec_atribs_chave_prim_*$\mathcal{R}$, $\mathcal{Z}1$) = *verdadeiro* **então**

$$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_e}\left(\sigma_{cond}\left(\mathcal{R} \rhd\lhd \mathcal{Z}1\right)\right)$$

senão

$\mathcal{T} \leftarrow \pi_{a_1,\ldots,a_m,c_1,\ldots,c_e}(\sigma_{cond}(\mathcal{R}))$

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$v_1 \leftarrow atrib_1$

$v_2 \leftarrow atrib_2$

$\vdots$

$v_k \leftarrow atrib_k$

$u \leftarrow \sigma_{a_1=t.a_1 \wedge \ldots \wedge a_m=t.a_m}(\mathcal{R})$

$\mathcal{R} \leftarrow \mathcal{R} - u$

$u.c_x \leftarrow v_o$

$u.c_y \leftarrow v_p$

$\vdots$

$u.c_z \leftarrow v_q$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(u.a_1,\ldots,u.a_m,u.a_{m+1},\ldots,u.a_{m+n})\}$

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: As atribuições $atrib_i$ são expressas em função dos atributos de C.

Na correcção de violação de restrição de integridade ao nível da relação, também pode ser necessário recorrer a FDU. Nesta situação, o conjunto de valores de cada atributo necessário à correcção do PQD pode ser passado como parâmetro, de acordo com os requisitos específicos de cada FDU que consta da sequência de atribuições. O conjunto de valores de cada atributo provém dos tuplos envolvidos no PQD detectado (*i.e.*, todos os tuplos de $\mathcal{R}$ ou apenas os que constam de $\mathcal{Z}1$). As funções também podem receber outros parâmetros, como variáveis e constantes. Como resultado da sua execução, cada função retorna um valor atómico ou uma estrutura de dados (*e.g.*: um vector; uma matriz). A última função da sequência de atribuições (na formalização seguinte representada por fdu_k), obrigatoriamente retorna uma estrutura de dados bidimensional (*i.e.*, uma matriz). Cada coluna da matriz contém valores que actualizam um dos atributos envolvidos na correcção da violação da restrição de integridade (na formalização seguinte representados por: c_x, c_y,…, c_z). A ordem dos valores nas linhas da matriz necessita de ser a mesma da considerada na extracção dos valores dos atributos. Este é um requisito a que a

matriz tem de obedecer para que os valores dos atributos nos tuplos sejam correctamente actualizados. A formalização anterior é alvo das alterações que a seguir se apresentam, de modo a reflectir a correcção da violação de restrição de integridade com base em FDU. O que permanece inalterado relativamente à formalização anterior encontra-se representado sob a forma de reticências.

Algoritmo Correcção_Violação_Restrição_Integridade_Baseada_em_FDU $\stackrel{\text{def}}{=}$

início

$\vdots$

fim se

$v_1 \leftarrow fdu_1\left(p_{1,1},p_{1,2},\ldots,p_{1,i}\right)$

$v_2 \leftarrow fdu_2\left(p_{2,1},p_{2,2},\ldots,p_{2,j}\right)$

$\vdots$

$v_k \leftarrow fdu_k\left(p_{k,1},p_{k,2},\ldots,p_{k,l}\right)$

$g \leftarrow 0$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$g \leftarrow g + 1$

se $v_k[g][1] \neq t.c_x$ **ou** $v_k[g][2] \neq t.c_y$ **ou** … **ou** $v_k[g][h] \neq t.c_z$ **então**

$u \leftarrow \sigma_{a_1=t.a_1 \wedge \ldots \wedge a_m=t.a_m}(\mathcal{R})$

$\mathcal{R} \leftarrow \mathcal{R} - u$

$u.c_x \leftarrow v_k[g][1]$

$u.c_y \leftarrow v_k[g][2]$

$\vdots$

$u.c_z \leftarrow v_k[g][h]$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(u.a_1,\ldots,u.a_m,u.a_{m+1},\ldots,u.a_{m+n})\}$

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

Nota: Os valores dos atributos de C podem constituir parâmetros de uma ou mais funções fdu_i.

6.6 Problemas ao Nível de Múltiplas Relações/Fontes de Dados

Nesta secção apresentam-se as formalizações das OC dos PQD que ocorrem ao nível multi-relação, independentemente destas pertencerem a uma só ou a múltiplas BD.

6.6.1 Heterogeneidade de sintaxes

Formalização Sintáctica

CORRECT HETEROGENEITY-OF-SYNTAXES ColunasAlvo DasTabelas DasBasesDados DefiniçãoAtribuições UsandoCondição

Formalização Semântica

A OC de heterogeneidade de sintaxes envolve efectuar alterações à sintaxe dos valores do atributo de uma das relações, de modo a colocá-los de acordo com a sintaxe do atributo da outra relação. Os valores do atributo sujeitos a alteração dependem do especificado pelo utilizador. Nesta OC recorre-se a uma função (no caso, *efectua_correcção_sintáctica*) que de raiz suporta a realização das alterações mais comuns de formato, com base em expressões regulares. Na especificação da OC, o utilizador define as expressões regulares e os respectivos novos formatos dos valores. A função verifica a concordância entre o valor do atributo e cada expressão regular. Caso seja encontrada uma concordância, o valor do atributo é alterado para o formato respectivo. Caso contrário, não é efectuada qualquer alteração ao valor do atributo.

Algoritmo Correcção_Heterogeneidade_Sintaxes $\stackrel{\text{def}}{=}$

início

se *flag_actualiza_valores_relação_*$\mathcal{R}$ = *verdadeiro* **então**

$\mathcal{T} \leftarrow \sigma_{cond}(\mathcal{R})$

senão

$\mathcal{T} \leftarrow \sigma_{cond}(S)$

fim se

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

repetir enquanto $t \neq null$

novo_valor $\leftarrow$ *efectua_correcção_sintáctica*($t.atr_1$, *mat_transf_sintácticas*)

se *novo_valor* $\neq t.atr_1$ **então**

se *flag_actualiza_valores_relação_*$\mathcal{R}$ = *verdadeiro* **então**

$\mathcal{R} \leftarrow \mathcal{R} - t$

$t.atr_1 \leftarrow novo\_valor$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(t.a_1,\ldots,t.a_m,t.a_{m+1},\ldots,t.a_{m+n})\}$

senão

$\mathcal{S} \leftarrow \mathcal{S} - t$

$t.atr_1 \leftarrow novo\_valor$

$\mathcal{S} \leftarrow \mathcal{S} \cup \{(t.b_1,\ldots,t.b_r,t.b_{r+1},\ldots,t.b_{r+s})\}$

fim se

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

No caso das alterações necessárias à correcção da heterogeneidade sintáctica detectada não serem suportadas pela função existente de raiz (*i.e.*, *efectua_correcção_sintáctica*), é necessário recorrer a FDU. De acordo com os requisitos de cada função existente na sequência de atribuições da OC, o valor do atributo de cada tuplo da relação alvo de alteração sintáctica pode ser passado como parâmetro. Além deste, podem existir outros parâmetros na função, como variáveis e constantes. Como resultado, cada FDU retorna um valor atómico. A última função da sequência de atribuições (na formalização seguinte representada por fdu_k) retorna o valor que homogeneíza a sintaxe do valor do atributo, com a sintaxe utilizada no outro atributo. A formalização anterior sofre as alterações a seguir apresentadas, de modo a reflectir a correcção da heterogeneidade de sintaxes baseada em FDU. As reticências representam aquilo que permanece inalterado em relação à formalização anterior.

Algoritmo Correcção_Heterogeneidade_Sintaxes_Baseada_em_FDU $\stackrel{def}{=}$

início

$\vdots$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

$v_1 \leftarrow fdu_1\left(p_{1,1},p_{1,2},\ldots,p_{1,i}\right)$

$v_2 \leftarrow fdu_2\left(p_{2,1},p_{2,2},\ldots,p_{2,j}\right)$

$\vdots$

$v_k \leftarrow fdu_k\left(p_{k,1},p_{k,2},\ldots,p_{k,l}\right)$

se *flag_actualiza_valores_relação_*$\mathcal{R}$ = *verdadeiro* **então**

$\mathcal{R} \leftarrow \mathcal{R} - t$

$t.atr_1 \leftarrow v_k$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(t.a_1,\ldots,t.a_m,t.a_{m+1},\ldots,t.a_{m+n})\}$

senão

$\mathcal{S} \leftarrow \mathcal{S} - t$

$t.atr_1 \leftarrow v_k$

$\mathcal{S} \leftarrow \mathcal{S} \cup \{(t.b_1,\ldots,t.b_r,t.b_{r+1},\ldots,t.b_{r+s})\}$

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

fim

6.6.2 Heterogeneidade de unidades de medida

Formalização Sintáctica

CORRECT HETEROGENEITY-OF-MEASURE-UNITS ColunasAlvo DasTabelas
DasBasesDados
FonteTermosUniformização
UsandoCondição

Formalização Semântica

Da respectiva OD resultam duas relações (*i.e.*, $\mathcal{Z}1$ e $\mathcal{Z}2$) onde se encontram os tuplos de $\mathcal{R}$ e $\mathcal{S}$ nos quais há heterogeneidade de unidades de medida entre os atributos em causa. O utilizador especifica qual a relação ($\mathcal{R}$ ou $\mathcal{S}$) cujos valores serão utilizados na homogeneização das unidades de medida usadas em ambas as relações. A correcção automática deste PQD envolve unicamente, a actualização dos valores do atributo da relação seleccionada, com os valores que se encontram no atributo da outra relação. Dependendo de ser $\mathcal{R}$ ou $\mathcal{S}$ a relação de homogeneização, são usados os valores de $\mathcal{Z}1$ ou $\mathcal{Z}2$. Além destas, a OD também pode ter originado as relações $\mathcal{Z}1_{não heterog}$ e $\mathcal{Z}2_{não heterog}$ onde se encontra informação sobre os tuplos de $\mathcal{R}$ e $\mathcal{S}$ nos quais a heterogeneidade de unidades de medida não é respeitada. Previamente à execução desta OC, estas situações devem ser objecto de análise por parte do utilizador que procede às correcções que entende necessárias. Estas correcções são importantes uma vez que a relação $\mathcal{Z}1_{não heterog}$ ou $\mathcal{Z}2_{não heterog}$ (dependendo de ser $\mathcal{R}$ ou $\mathcal{S}$ a relação de homogeneização) também é utilizada na actualização dos valores do atributo da relação seleccionada, com os valores que se encontram no atributo da outra relação.

Algoritmo Correcção_Heterogeneidade_Unidades_Medida $\stackrel{def}{=}$

seja *nrseq* o atributo de $\mathcal{Z}1$ e $\mathcal{Z}2$ que permite associar cada heterogeneidade de unidades de medida detectada entre os valores de atr_1 e atr_2.

seja $\mathcal{Z}1_{não heterog}$ e $\mathcal{Z}2_{não heterog}$ as relações que armazenam as chaves primárias dos tuplos, respectivamente, de $\mathcal{R}$ e S onde não se verifica a heterogeneidade de unidades de medida entre atr_1 e atr_2, bem como os respectivos valores destes atributos.

início

$\mathcal{T} \leftarrow \pi_{a_1,...,a_m,atr_1,b_1,...,b_r,atr_2}\left(\sigma_{cond}\left(\mathcal{Z}1 \rhd\lhd_{(\mathcal{Z}1.nrseq=\mathcal{Z}2.nrseq)} \mathcal{Z}2\right)\right)$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

se *flag_actualiza_valores_relação_*$\mathcal{R}$ $=$ *verdadeiro* **então**

*actualiza_tuplo_relação_*R(t)

senão

*actualiza_tuplo_relação_*S(t)

fim se

$t \leftarrow le\_tuplo(\mathcal{T})$

fim repetir

se $\mathcal{Z}1_{não heterog} \neq \emptyset$ **então**

$\mathcal{U} \leftarrow \pi_{a_1,...,a_m,atr_1,b_1,...,b_r,atr_2}\left(\mathcal{Z}1_{não heterog} \rhd\lhd_{\left(\mathcal{Z}1_{não heterog}.nrseq=\mathcal{Z}2_{não heterog}.nrseq\right)} \mathcal{Z}2_{não heterog}\right)$

$u \leftarrow le\_tuplo(\mathcal{U})$

repetir enquanto $u \neq null$

se *flag_actualiza_valores_relação_*$\mathcal{R}$ $=$ *verdadeiro* **então**

*actualiza_tuplo_relação_*R(u)

senão

*actualiza_tuplo_relação_*S(u)

fim se

$u \leftarrow le\_tuplo(\mathcal{U})$

fim repetir

fim se

fim

Procedimento actualiza_tuplo_relação_R(x) $\stackrel{def}{=}$

início

$v \leftarrow \sigma_{a_1=x.a_1 \wedge ... \wedge a_m=x.a_m}(\mathcal{R})$

$\mathcal{R} \leftarrow \mathcal{R} - v$

$v.atr_1 \leftarrow x.atr_2$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(v.a_1,\ldots,v.a_m,v.a_{m+1},\ldots,v.a_{m+n})\}$

fim

Procedimento actualiza_tuplo_relação_S(x) $\stackrel{def}{=}$

início

$v \leftarrow \sigma_{b_1=x.b_1 \wedge \ldots \wedge b_r=x.b_r}(S)$

$S \leftarrow S - v$

$v.atr_2 \leftarrow x.atr_1$

$S \leftarrow S \cup \{(v.b_1,\ldots,v.b_r,v.b_{r+1},\ldots,v.b_{r+s})\}$

fim

6.6.3 Existência de sinónimos

Formalização Sintáctica

CORRECT EXISTENCE-OF-SYNONYMS ColunasAlvo DasTabelas DasBasesDados
FonteTermosUniformização

Formalização Semântica

As relações que resultam da respectiva OD permitem agrupar os sinónimos em *clusters*. O utilizador especifica qual a relação cujos termos serão utilizados na uniformização dos termos sinónimos existentes na outra relação. A correcção automática deste PQD envolve, para cada *cluster*, a actualização do valor que contém o sinónimo no atributo de uma das relações, com o termo de uniformização utilizado no atributo da outra relação. Da actualização do valor do atributo resulta um novo valor que apenas difere do anterior no termo de uniformização que substitui o sinónimo, mantendo-se tudo o resto inalterado (*e.g.*: considerando os termos *canalizador* e *picheleiro* como sinónimos e sendo este último o termo de uniformização, o valor *técnico canalizador* é actualizado para *técnico picheleiro*).

Algoritmo Correcção_Existência_Sinónimos $\stackrel{def}{=}$

seja *idcluster* um atributo de *Z2* que identifica o *cluster* de sinónimos a que cada termo sinónimo detectado pertence.

seja $\mathcal{T}$ a relação onde se encontra armazenado o dicionário de sinónimos, com esquema: $\mathcal{T}$(*idsin*, *idcluster*, *sin*), em que *idsin* constitui a chave primária, *idcluster* o *cluster* a que pertence o sinónimo e *sin* o termo sinónimo.

início

se *flag_actualiza_valores_relação_*$\mathcal{R}$ = *verdadeiro* **então**

$\mathcal{U} \leftarrow \pi_{idcluster,atr_2}(\mathcal{Z}2)$

senão

$\mathcal{U} \leftarrow \pi_{idcluster,atr_1}(\mathcal{Z}1)$

fim se

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

repetir enquanto $u \neq$ *null*

se *flag_actualiza_valores_relação_*$\mathcal{R}$ = *verdadeiro* **então**

$\mathcal{V} \leftarrow \pi_{sin}(\sigma_{idcluster=u.idcluster \wedge sin \neq u.atr_2 \wedge sin \text{ not like '\%}u.atr_2\text{\%'} \wedge sin \text{ not like '}u.atr_2\text{\%'} \wedge \wedge sin \text{ not like '\%}u.atr_2\text{'}}(\mathcal{T}))$

$\mathcal{X} \leftarrow \pi_{a_1,\ldots,a_m,atr_1}(\sigma_{idcluster=u.idcluster}(\mathcal{Z}1))$

senão

$\mathcal{V} \leftarrow \pi_{sin}(\sigma_{idcluster=u.idcluster \wedge sin \neq u.atr_1 \wedge sin \text{ not like '\%}u.atr_1\text{\%'} \wedge sin \text{ not like '}u.atr_1\text{\%'} \wedge \wedge sin \text{ not like '\%}u.atr_1\text{'}}(\mathcal{T}))$

$\mathcal{X} \leftarrow \pi_{a_1,\ldots,a_m,atr_2}(\sigma_{idcluster=u.idcluster}(\mathcal{Z}2))$

fim se

$x \leftarrow$ *le_tuplo*($\mathcal{X}$)

repetir enquanto $x \neq$ *null*

novo_valor $\leftarrow x.atr_1$

$v \leftarrow$ *le_tuplo*($\mathcal{V}$)

repetir enquanto $v \neq$ *null* **e** *novo_valor* = $x.atr_1$

novo_valor $\leftarrow$ *substitui_substring*($x.atr_1$, $v.sin$, $u.atr_2$)

se $x.atr_1$ = *novo_valor* **então**

$v \leftarrow$ *le_tuplo*($\mathcal{V}$)

fim se

fim repetir

se *flag_actualiza_valores_relação_*$\mathcal{R}$ = *verdadeiro* **então**

$y \leftarrow \sigma_{a_1=x.a_1 \wedge \ldots \wedge a_m=x.a_m}(\mathcal{R})$

$\mathcal{R} \leftarrow \mathcal{R} - y$

$y.atr_1 \leftarrow$ *novo_valor*

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(y.a_1,\ldots,y.a_m,y.a_{m+1},\ldots,y.a_{m+n})\}$

senão

$y \leftarrow \sigma_{a_1=x.a_1 \wedge \ldots \wedge a_m=x.a_m}(\mathcal{S})$

$\mathcal{S} \leftarrow \mathcal{S} - y$

$y.atr_2 \leftarrow$ *novo_valor*

$\mathcal{S} \leftarrow \mathcal{S} \cup \{(y.b_1,\ldots,y.b_r,y.b_{r+1},\ldots,y.b_{r+s})\}$

fim se

reinicializa_leitura_tuplos($\mathcal{V}$)

$x \leftarrow$ *le_tuplo*(X)

fim repetir

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

fim repetir

fim

6.6.4 Tuplos duplicados

Formalização Sintáctica

REMOVE DUPLICATE-TUPLES DasTabelas DasBasesDados DefiniçãoColunas UsandoAtribuições UsandoCondição

Formalização Semântica

As relações que resultam da respectiva OD (*i.e.*, $\mathcal{Z}1$ e $\mathcal{Z}2$) permitem agrupar os tuplos em função do *cluster* de duplicados que estes formam. Em função do especificado pelo utilizador, esta operação remove todos os tuplos duplicados existentes na relação $\mathcal{R}$ ou $\mathcal{S}$. Esta eliminação pressupõe que cada *cluster* é composto por duplicados rigorosamente iguais entre si (*i.e.*, duplicados exactos) ou que os valores existentes nos duplicados de uma das relações apresentam um grau de confiança menor, pelo que os tuplos respectivos podem ser eliminados. A eliminação de cada tuplo duplicado só ocorre caso desta não resulte uma violação à integridade referencial da BD. Naturalmente, a capacidade de assegurar automaticamente a integridade referencial, encontra-se dependente da existência das respectivas restrições na BD.

Algoritmo Remoção_Tuplos_Duplicados $\stackrel{\text{def}}{=}$

seja *idcluster* um atributo de $\mathcal{Z}1$ e $\mathcal{Z}2$ que identifica o *cluster* de duplicados a que os tuplo de $\mathcal{R}$ e $\mathcal{S}$ identificados como duplicados pertencem.

início

$\mathcal{T} \leftarrow \pi_{idcluster}\left(\sigma_{cond}\left(\mathcal{Z}1 \rhd\lhd_{(\mathcal{Z}1.idcluster=\mathcal{Z}2.idcluster)} \mathcal{Z}2\right)\right)$

$t \leftarrow$ *le_tuplo*($\mathcal{T}$)

repetir enquanto $t \neq$ *null*

se *flag_actualiza_valores_relação_*$\mathcal{R}$ = *verdadeiro* **então**

$\mathcal{V} \leftarrow \pi_{\mathcal{R}.a_1,\ldots,\mathcal{R}.a_m,\ldots,\mathcal{R}.a_{m+n}}\left(\sigma_{\mathcal{Z}1.idcluster=t.idcluster}\left(\mathcal{R} \rhd\lhd_{(\mathcal{R}.a_1=\mathcal{Z}1.a_1\wedge\ldots\wedge\mathcal{R}.a_m=\mathcal{Z}1.a_m)} \mathcal{Z}1\right)\right)$

senão

$\mathcal{V} \leftarrow \pi_{S.b_1,\ldots,S.b_r,\ldots,S.b_{r+s}} \left(\sigma_{\mathcal{Z}2.idcluster=t.idcluster} \left(S \rhd\lhd_{(S.b_1=\mathcal{Z}2.b_1 \wedge \ldots \wedge S.b_r=\mathcal{Z}2.b_r)} \mathcal{Z}2 \right) \right)$

fim se

$v \leftarrow$ *le_tuplo*$(\mathcal{V})$

repetir enquanto $v \neq$ *null*

se *viola_integridade_referencial*(v) = *falso* **então**

se *flag_actualiza_valores_relação_*$\mathcal{R}$ = *verdadeiro* **então**

$\mathcal{R} \leftarrow \mathcal{R} - v$

senão

$S \leftarrow S - v$

fim se

fim se

$v \leftarrow$ *le_tuplo*$(\mathcal{V})$

fim repetir

$t \leftarrow$ *le_tuplo*$(\mathcal{T})$

fim repetir

fim

Função Viola_Integridade_Referencial(t**)** $\stackrel{\text{def}}{=}$

seja w o número de relações S_i com as quais $\mathcal{R}$ ou S se encontra relacionada.

seja $\mathcal{A} = \{a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos de $\mathcal{R}$ que não fazem parte da sua chave primária.

seja C o conjunto de atributos assim definido: $C = \{c \mid c$ faz parte da chave estrangeira de $S_i \wedge c \in \mathcal{A}\}$, *i.e.*, $C \subseteq \mathcal{A}$.

seja $\mathcal{B} = \{b_{r+1},\ldots,b_{r+s}\}$ o conjunto de atributos de S que não fazem parte da sua chave primária.

seja $\mathcal{D}$ o conjunto de atributos assim definido: $\mathcal{D} = \{d \mid d$ faz parte da chave estrangeira de $S_i \wedge d \in \mathcal{B}\}$, *i.e.*, $\mathcal{D} \subseteq \mathcal{B}$.

início

$i \leftarrow 1$

flag_encontrado $\leftarrow$ *falso*

repetir enquanto $i \leq w$ **e** *flag_encontrado* = *falso*

se *flag_actualiza_valores_relação_*$\mathcal{R}$ = *verdadeiro* **então**

$\mathcal{X} \leftarrow \sigma_{c_1=v.a_1 \wedge \ldots \wedge c_m=v.a_m} (S_i)$

senão

$\mathcal{X} \leftarrow \sigma_{d_1=v.b_1 \wedge \ldots \wedge d_m=v.b_r} (S_i)$

fim se

```
        x ← le_tuplo(X)
        se x ≠ null então
            flag_encontrado ← verdadeiro
        senão
            i ← i + 1
        fim se
    fim repetir
    retorna flag_encontrado
fim
```

No caso de cada *cluster* de duplicados não ser composto por duplicados exactamente iguais (trata-se de uma situação comum), dificilmente faz sentido a aplicação da operação anterior. Nestes casos, frequentemente pretende-se consolidar (*e.g.*: completar) os valores dos atributos dos tuplos que "sobrevivem", com os valores dos atributos dos tuplos a eliminar. A forma como a consolidação dos valores é efectuada necessita de ser implementada numa FDU. Cada função, dependendo dos seus requisitos específicos, pode receber como parâmetros os valores de dois atributos (um de $\mathcal{R}$ e o outro de $\mathcal{S}$), resultantes de cada *cluster* de duplicados identificado. Além destes, as funções podem receber outros parâmetros, como variáveis e constantes. Como resultado, cada função retorna um valor atómico ou uma estrutura de dados (*e.g.*: um vector; uma matriz). O resultado de cada função responsável por produzir um valor consolidado é obrigatoriamente um valor atómico. Na formalização a seguir apresentada, estes correspondem aos valores presentes nas atribuições que envolvem os atributos das relações $\mathcal{R}(i.e., a_x, a_y, \ldots, a_z)$ e $\mathcal{S}(i.e., b_x, b_y, \ldots, b_z)$. Note-se que à semelhança da anterior, esta operação também garante a não violação da integridade referencial. Para tal, basta que as respectivas restrições se encontrem definidas na BD em questão.

Algoritmo Remoção_Tuplos_Duplicados_Baseada_em_FDU $\stackrel{\text{def}}{=}$

seja $\mathcal{A} = \{a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos de $\mathcal{R}$ que não fazem parte da sua chave primária.

seja k o número de atribuições existentes na sequência de atribuições da OC.

sejam x, y e z números inteiros compreendidos no intervalo $[m + 1; m + n]$.

sejam o, p e q números inteiros compreendidos no intervalo $[1; k]$.

início

$\mathcal{T} \leftarrow \pi_{idcluster}\left(\sigma_{cond}\left(\mathcal{Z}1 \rhd\lhd_{(\mathcal{Z}1.idcluster = \mathcal{Z}2.idcluster)} \mathcal{Z}2\right)\right)$

$t \leftarrow le\_tuplo(\mathcal{T})$

repetir enquanto $t \neq null$

se *flag_actualiza_valores_relação_*$\mathcal{R}$= *verdadeiro* **então**

$$\mathcal{U} \leftarrow \pi_{\mathcal{R}.a_1,\ldots,\mathcal{R}.a_m,\ldots,\mathcal{R}.a_{m+n}} \left(\sigma_{\mathcal{Z}1.idcluster=t.idcluster} \left(\mathcal{R} \rhd\lhd_{(\mathcal{R}.a_1=\mathcal{Z}1.a_1 \wedge \ldots \wedge \mathcal{R}.a_m=\mathcal{Z}1.a_m)} \mathcal{Z}1 \right) \right)$$

$$\mathcal{V} \leftarrow \pi_{\mathcal{S}.b_1,\ldots,\mathcal{S}.b_r,\ldots,\mathcal{S}.b_{r+s}} \left(\sigma_{\mathcal{Z}2.idcluster=t.idcluster} \left(\mathcal{S} \rhd\lhd_{(\mathcal{S}.b_1=\mathcal{Z}2.b_1 \wedge \ldots \wedge \mathcal{S}.b_r=\mathcal{Z}2.b_r)} \mathcal{Z}2 \right) \right)$$

senão

$$\mathcal{U} \leftarrow \pi_{\mathcal{S}.b_1,\ldots,\mathcal{S}.b_r,\ldots,\mathcal{S}.b_{r+s}} \left(\sigma_{\mathcal{Z}2.idcluster=t.idcluster} \left(\mathcal{S} \rhd\lhd_{(\mathcal{S}.b_1=\mathcal{Z}2.b_1 \wedge \ldots \wedge \mathcal{S}.b_r=\mathcal{Z}2.b_r)} \mathcal{Z}2 \right) \right)$$

$$\mathcal{V} \leftarrow \pi_{\mathcal{R}.a_1,\ldots,\mathcal{R}.a_m,\ldots,\mathcal{R}.a_{m+n}} \left(\sigma_{\mathcal{Z}1.idcluster=t.idcluster} \left(\mathcal{R} \rhd\lhd_{(\mathcal{R}.a_1=\mathcal{Z}1.a_1 \wedge \ldots \wedge \mathcal{R}.a_m=\mathcal{Z}1.a_m)} \mathcal{Z}1 \right) \right)$$

fim se

$$v_1 \leftarrow fdu_1 \left(p_{1,1}, p_{1,2}, \ldots, p_{1,i} \right)$$

$$v_2 \leftarrow fdu_2 \left(p_{2,1}, p_{2,2}, \ldots, p_{2,j} \right)$$

$$\vdots$$

$$v_k \leftarrow fdu_k \left(p_{k,1}, p_{k,2}, \ldots, p_{k,l} \right)$$

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

repetir enquanto $u \neq$ *null*

se *flag_actualiza_valores_relação_*$\mathcal{R}$= *verdadeiro* **então**

$\mathcal{R} \leftarrow \mathcal{R} - u$

$u.a_x \leftarrow v_o$

$u.a_y \leftarrow v_p$

$\vdots$

$u.a_z \leftarrow v_q$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(u.a_1, \ldots, u.a_m, u.a_{m+1}, \ldots, u.a_{m+n})\}$

senão

$\mathcal{S} \leftarrow \mathcal{S} - u$

$u.b_x \leftarrow v_o$

$u.b_y \leftarrow v_p$

$\vdots$

$u.b_z \leftarrow v_q$

$\mathcal{S} \leftarrow \mathcal{S} \cup \{(u.b_1, \ldots, u.b_r, u.b_{r+1}, \ldots, u.b_{r+s})\}$

fim se

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

fim repetir

$v \leftarrow$ *le_tuplo*($\mathcal{V}$)

repetir enquanto $v \neq$ *null*

```
            se viola_integridade_referencial(v) = falso então
                se flag_actualiza_valores_relação_ℛ = verdadeiro então
                    ℛ ← ℛ – v
                senão
                    𝒮 ← 𝒮 – v
                fim se
            fim se
            v ← le_tuplo(𝒱)
        fim repetir
        t ← le_tuplo(𝒯)
    fim repetir
fim
```

Nota: O significado do atributo *idcluster* e a formalização da função *viola_integridade_referencial* encontram-se na formalização anterior.

6.6.5 Violação de restrição de integridade

Formalização Sintáctica

CORRECT INTEGRITY-CONSTRAINT-VIOLATION AT LEVEL OF MULTIPLE RELATIONS ColunasAlvo DasTabelas DasBasesDados DefiniçãoAtribuições UsandoTabelas UsandoCondição

Formalização Semântica

A relação que resulta da OD respectiva (*i.e.*, $\mathcal{Z}$), além de conter os valores que constituem violações à restrição de integridade ao nível multi-relação, também pode conter informação sobre os tuplos das relações envolvidas que as provocam. No caso desta informação não existir, isso significa que a violação à restrição de integridade resulta dos valores dos atributos envolvidos, quando se considera a globalidade dos tuplos existentes nas diferentes relações. A OC proposta baseia-se na alteração, em cada relação, dos valores dos atributos nos tuplos especificados pelo utilizador, entre os que se encontram envolvidos no PQD (*i.e.*, todos os tuplos de $\mathcal{R}$, …, $\mathcal{S}$ ou apenas aqueles que constam de $\mathcal{Z}$). Assim, para cada um desses tuplos de cada relação presente no conjunto de atribuições definidas pelo utilizador, a OC actualiza os valores dos atributos necessários à resolução do PQD, em função dos valores resultantes das operações de atribuição.

Cada atribuição é representada em função de uma expressão que envolve exclusivamente atributos, variáveis e/ou constantes.

Algoritmo Correcção_Violação_Restrição_Integridade $\stackrel{def}{=}$

seja *vec_atribs_chave_prim_S* um vector que contém os nomes dos atributos que constituem a chave primária de $\mathcal{S}$, *i.e.*, $b_1,\ldots,b_r$.

seja $\mathcal{A} = \{a_{m+1},\ldots,a_{m+n}\}$ o conjunto de atributos de $\mathcal{R}$ que não fazem parte da sua chave primária.

seja $\mathcal{C}$ o conjunto de atributos assim definido: $\mathcal{C} = \{c \mid c$ é usado na correcção da violação da restrição de integridade $\wedge\ c \in \mathcal{A}\}$, *i.e.*, $\mathcal{C} \subseteq \mathcal{A}$. Este conjunto resulta da especificação da OC.

seja *e* a cardinalidade de $\mathcal{C}$.

seja *k1* o número de atribuições existentes na sequência de atribuições da OC referentes aos atributos da relação $\mathcal{R}$.

sejam *x1*, *y1* e *z1* números inteiros compreendidos no intervalo [1; *e*].

sejam *o1*, *p1* e *q1* números inteiros compreendidos no intervalo [1; *k1*].

seja $\mathcal{B} = \{b_{r+1},\ldots,a_{r+s}\}$ o conjunto de atributos de $\mathcal{S}$ que não fazem parte da sua chave primária.

seja $\mathcal{D}$ o conjunto de atributos assim definido: $\mathcal{D} = \{d \mid d$ é usado na correcção da violação da restrição de integridade $\wedge\ d \in \mathcal{B}\}$, *i.e.*, $\mathcal{D} \subseteq \mathcal{B}$. Este conjunto resulta da especificação da OC.

seja *f* a cardinalidade de $\mathcal{D}$.

seja *k2* o número de atribuições existentes na sequência de atribuições da OC referentes aos atributos da relação $\mathcal{S}$.

sejam *x2*, *y2* e *z2* números inteiros compreendidos no intervalo [1; *f*].

sejam *o2*, *p2* e *q2* números inteiros compreendidos no intervalo [1; *k2*].

seja $\mathcal{Z}$ a relação que armazena as violações identificadas à restrição de integridade.

início

se *existem_atribs_na_relação*(*vec_atribs_chave_prim_R*, $\mathcal{Z}$) = *verdadeiro* **ou**
ou *existem_atribs_na_relação*(*vec_atribs_chave_prim_S*, $\mathcal{Z}$) = *verdadeiro* **então**

$\mathcal{T} \leftarrow \sigma_{cond} (\mathcal{R} \rhd\lhd \ldots \rhd\lhd \mathcal{S} \rhd\lhd \mathcal{Z})$

senão

$\mathcal{T} \leftarrow \sigma_{cond} (\mathcal{R} \rhd\lhd \ldots \rhd\lhd \mathcal{S})$

fim se

$\mathcal{U} \leftarrow \pi_{a_1,\ldots,a_m,a_{m+1},\ldots,a_{m+n}} (\mathcal{T})$

$u \leftarrow$ *le_tuplo*($\mathcal{U}$)

repetir enquanto $u \neq$ *null*

$\mathcal{R} \leftarrow \mathcal{R} - u$

$v_1 \leftarrow atrib_1$

$v_2 \leftarrow atrib_2$

$\vdots$

$v_{k1} \leftarrow atrib_{k1}$

$u.c_{x1} \leftarrow v_{o1}$

$u.c_{y1} \leftarrow v_{p1}$

$\vdots$

$u.c_{z1} \leftarrow v_{q1}$

$\mathcal{R} \leftarrow \mathcal{R} \cup \{(u.a_1,\ldots,u.a_m,u.a_{m+1},\ldots,u.a_{m+n})\}$

$u \leftarrow le\_tuplo(\mathcal{U})$

fim repetir

$\vdots$

$\mathcal{U} \leftarrow \pi_{b_1,\ldots,b_r,b_{r+1},\ldots,b_{r+s}}(\mathcal{T})$

$u \leftarrow le\_tuplo(\mathcal{U})$

repetir enquanto $u \neq null$

$\mathcal{S} \leftarrow \mathcal{S} - u$

$v_1 \leftarrow atrib_1$

$v_2 \leftarrow atrib_2$

$\vdots$

$v_{k2} \leftarrow atrib_{k2}$

$u.d_{x2} \leftarrow v_{o2}$

$u.d_{y2} \leftarrow v_{p2}$

$\vdots$

$u.d_{z2} \leftarrow v_{q2}$

$\mathcal{S} \leftarrow \mathcal{S} \cup \{(u.b_1,\ldots,u.b_r,u.b_{r+1},\ldots,u.b_{r+s})\}$

$u \leftarrow le\_tuplo(\mathcal{U})$

fim repetir

fim

6.7 Conclusão

Neste capítulo apresentou-se a formalização sintáctica e semântica das operações que suportam a correcção dos PQD. A *nível sintáctico*, formalizou-se uma linguagem que permite a especificação de qualquer OC automática dos PQD que fazem parte da taxionomia exposta no Capítulo 4 e cujo modelo de limpeza de dados a apresentar no capítulo seguinte concede o necessário suporte. A *nível semântico*, formalizou-se as operações consideradas úteis e importantes na correcção dos problemas e que são susceptíveis de serem disponibilizadas de base numa implementação computacional do referido modelo. Estas operações cobrem a generalidade dos problemas que

fazem parte da taxionomia. Nalguns problemas (*e.g.*: erro ortográfico; existência de sinónimos) foram até formalizadas duas OC distintas. Comparativamente à taxionomia, apenas em quatro problemas não são apresentadas as respectivas formalizações que suportam a sua correcção automática. Os PQD em causa são: (*i*) circularidade entre tuplos num auto-relacionamento; (*ii*) existência de homónimos; (*iii*) diferentes granularidades de representação; e, (*iv*) violação de integridade referencial. Nestes problemas não foi possível perspectivar qualquer OC racional que possa ser fornecida à partida e cuja lógica subjacente mereça ser formalizada. Na realidade, a sua correcção parece condenada a uma intervenção manual por parte do utilizador. Se não for o caso, é sempre possível especificar-se a correcção através de uma operação baseada em SQL (na clausula *Set* da OC) ou especificar-se uma operação que recorre a uma função externa definida pelo utilizador. Seguindo esta última via, é possível efectuar-se a correcção automática de qualquer PQD. Mesmo aqueles para os quais se propôs e formalizou uma OC neste capítulo, podem ser solucionados por uma outra via, de acordo com o implementado na FDU.

A formalização semântica das diversas OC foi efectuada tendo em conta os aspectos relacionados com a sua eficiência. No entanto, estas preocupações não foram levadas ao limite, pelo que poderá haver margem para a introdução de optimizações. A introdução destas eventuais optimizações será objecto de análise num futuro próximo. De qualquer forma, cada formalização semântica não representa literalmente a forma como a OC será implementada. A título de exemplo, considere-se o caso da actualização do valor de um atributo. Nas formalizações, esta é representada pela eliminação do tuplo respectivo e pela sua re-inserção, já com o valor do atributo devidamente actualizado. A adopção desta abordagem prende-se com a inexistência de um operador de actualização em álgebra relacional. A implementação da actualização ao valor do atributo corresponde simplesmente a uma instrução "*Update*" de SQL.

Apesar de se formalizar um conjunto apreciável de operações, há a perfeita noção de que nem sempre estas corresponderão às necessidades de correcção. Isto acontece em virtude de ser impossível prever todas as necessidades possíveis e imagináveis de correcção e apresentar a respectiva formalização. Mesmo que o fosse, isso também não seria exequível. Como tal, por vezes será necessário proceder ao desenvolvimento de funções externas que executem as correcções pretendidas. As restrições de integridade constituem um bom exemplo, face à sua diversidade e dependência do domínio subjacente. Nestes casos, a solução passa por recorrer a uma função externa definida pelo utilizador, na qual se encontra implementada a lógica que conduz à correcção do PQD em causa. Mesmo assim, nas OC de violação de restrição de integridade que ocorrem em cada nível de granularidade da taxionomia (*i.e.*, atributo; tuplo;

relação; e, multi-relação de uma ou diferentes BD), foi possível identificar um "tronco" comum, tendo este sido devidamente formalizado. A especificidade de cada OC fica para a FDU. Por outro lado, nem sempre será viável a realização de correcções automáticas, *i.e.*, será necessária intervenção humana para, caso a caso, solucionar os problemas detectados. Num determinado PQD, a necessidade de intervenção manual poderá ser parcial (*i.e.*, apenas alguns casos entre os que foram detectados, sendo os restantes solucionados por via automática) ou total (*i.e.*, todos os casos detectados). A resolução de uma circularidade entre tuplos muito dificilmente poderá ser solucionada por outra via que não a manual.

Apesar das considerações tecidas no parágrafo anterior, as formalizações apresentadas materializam um conjunto de operações que, de raiz, fornecem um suporte que se considera útil e adequado à correcção de um leque alargado de PQD. Embora as OC propostas não constituam uma solução universal para todos os PQD, constituem uma mais-valia ao que actualmente é fornecido pelas aplicações informáticas de limpeza de dados (*i.e.*, protótipos de investigação e ferramentas comerciais).

MODELO PROPOSTO PARA A LIMPEZA DE DADOS 7

Neste capítulo apresenta-se, em detalhe, o modelo que se propõe para Detecção e Correcção (DC) dos Problemas de Qualidade dos Dados (PQD), *i.e.*, para a Limpeza de Dados (LD). O capítulo inicia-se com a apresentação da arquitectura que materializa o modelo proposto. Esta apresentação é acompanhada de uma descrição dos diversos componentes que integram a arquitectura, assim como da forma como estes interactuam entre si. Nas secções seguintes do capítulo apresentam-se, detalhadamente, os principais módulos da arquitectura. Estes módulos constituem a essência do modelo proposto, daí que se justifique plenamente uma atenção especial. Assim, começa-se por apresentar o módulo sequenciador de Operações de Detecção (OD). A seguir, apresenta-se o módulo que constitui o motor de execução incremental das OD. A apresentação dos principais módulos termina com o que constitui o motor de execução das Operações de Correcção[16] (OC). A descrição do modelo proposto finda com a apresentação do diagrama de sequência respectivo. Por fim, é apresentada uma descrição sucinta do protótipo desenvolvido, baptizado com a designação *SmartClean*. Este protótipo constitui uma implementação fiel da arquitectura apresentada nas secções anteriores e, consequentemente, do modelo que se defende para a LD.

7.1 Introdução

Neste capítulo descreve-se, pormenorizadamente, o modelo que se propõe e defende para a LD. Este capítulo surge na sequência natural dos dois anteriores, nos quais foram apresentadas as formalizações sintácticas e semânticas das operações de LD, *i.e.*, OD e OC. O modelo que se propõe para a LD articula a execução integrada de ambos os tipos de operações (*i.e.*, DC) que nos dois capítulos anteriores surgiram separadamente. À luz do modelo, os dois tipos de operações encontram-se intimamente relacionados, sendo executados em alternância. De facto, sempre que um problema é identificado, este é de imediato solucionado, uma vez que pode ocultar ou redundar na detecção de outros PQD. Para cada tipo de PQD, o modelo estabelece qual a sua

[16] O termo *operação de correcção* possui o seguinte significado: operação especificada pelo utilizador com o propósito de solucionar PQD de um determinado tipo. Cada correcção pode envolver não só alterações aos valores de um ou mais atributos num ou mais tuplos, como também a remoção/eliminação de um ou mais tuplos.

ordem de execução. Neste aspecto, o modelo assenta na manipulação (*i.e.*, DC) dos PQD por Nível de Granularidade (NG), resultante da taxionomia elaborada no Capítulo 4. Na manipulação dos PQD adoptou-se uma aproximação ascendente (*i.e.*, *bottom-up*) por NG. Começa-se pelos problemas que se manifestam no nível mais elementar (*i.e.*, atributo) e termina-se nos que ocorrem no nível de maior complexidade (*i.e.*, múltiplas relações de diferentes fontes de dados). Em cada NG, os PQD são também manipulados seguindo uma determinada sequência preestabelecida. Aliás, desta sequência de manipulação resultou a tese que se defende no âmbito desta dissertação de doutoramento.

Ao identificarem-se dependências na manipulação dos PQD, isso significa que a sequênciação da execução da generalidade das operações pode ser automatizada. Assim, em função das operações especificadas pelo utilizador, a ordem pela qual são executadas é determinada automaticamente. Esta característica constitui uma mais-valia comparativamente às soluções actualmente existentes destinadas à melhoria da qualidade dos dados (ver Capítulo 1 – Secção 1.2). Nos protótipos que resultaram de trabalhos de investigação, fica a cargo do utilizador a definição das dependências de execução entre as operações. No caso das ferramentas comerciais, ainda que não seja necessário definir a sequência de execução das operações, o leque de PQD coberto é reduzido quando comparado com os problemas que o modelo proposto suporta. Por outro lado, a sequência de manipulação dos PQD adoptada por estas, não parece adequada às diferentes situações de LD que podem ocorrer (ver Capítulo 1 – Secção 1.2). O estabelecimento de uma ordem de execução das operações de LD permite a implementação de um mecanismo de execução incremental. A detecção de um PQD não implica a interrupção da execução de certas OD para que este seja solucionado. Todas as OD que não dependam do PQD detectado continuam a sua execução. Numa iteração subsequente das OD, o âmbito de execução destas fica restrito aos problemas anteriormente detectados. Todos os outros atributos e tuplos em que se verificou não existirem problemas, aquando da iteração anterior de execução das OD, não são considerados na nova iteração.

Além da potencialidade que acabou de ser referida, o modelo proposto supre outros problemas e limitações identificados nas actuais soluções informáticas vocacionadas para a LD (novamente, ver Capítulo 1 – Secção 1.2). Assim, o modelo foi concebido especificamente para a LD, permitindo que as operações sejam efectuadas directamente na própria fonte. A realização de LD é efectuada num contexto que não obriga à realização de transformação de dados. Como o modelo suporta a execução de todas as OD e OC formalizadas nos dois capítulos anteriores, a cobertura fornecida de base à DC dos PQD é bastante abrangente.

As principais características do modelo proposto de LD foram introduzidas, de forma sumária, nesta secção. Nas secções seguintes, estas são expostas com o necessário detalhe que merecem. O modelo proposto encontra-se materializado na arquitectura apresentada na secção seguinte.

7.2 Arquitectura

Na Figura 7.1 apresenta-se a arquitectura proposta para a DC dos PQD, *i.e.*, para a LD. Esta arquitectura foi inicialmente proposta em [Oliveira *et al.*, 2005b]. Os novos desenvolvimentos entretanto ocorridos foram apresentados em [Oliveira *et al.*, 2006b]. Desde então, a arquitectura sofreu mais alguns refinamentos. A Figura 7.1 reflecte a versão final a que se chegou.

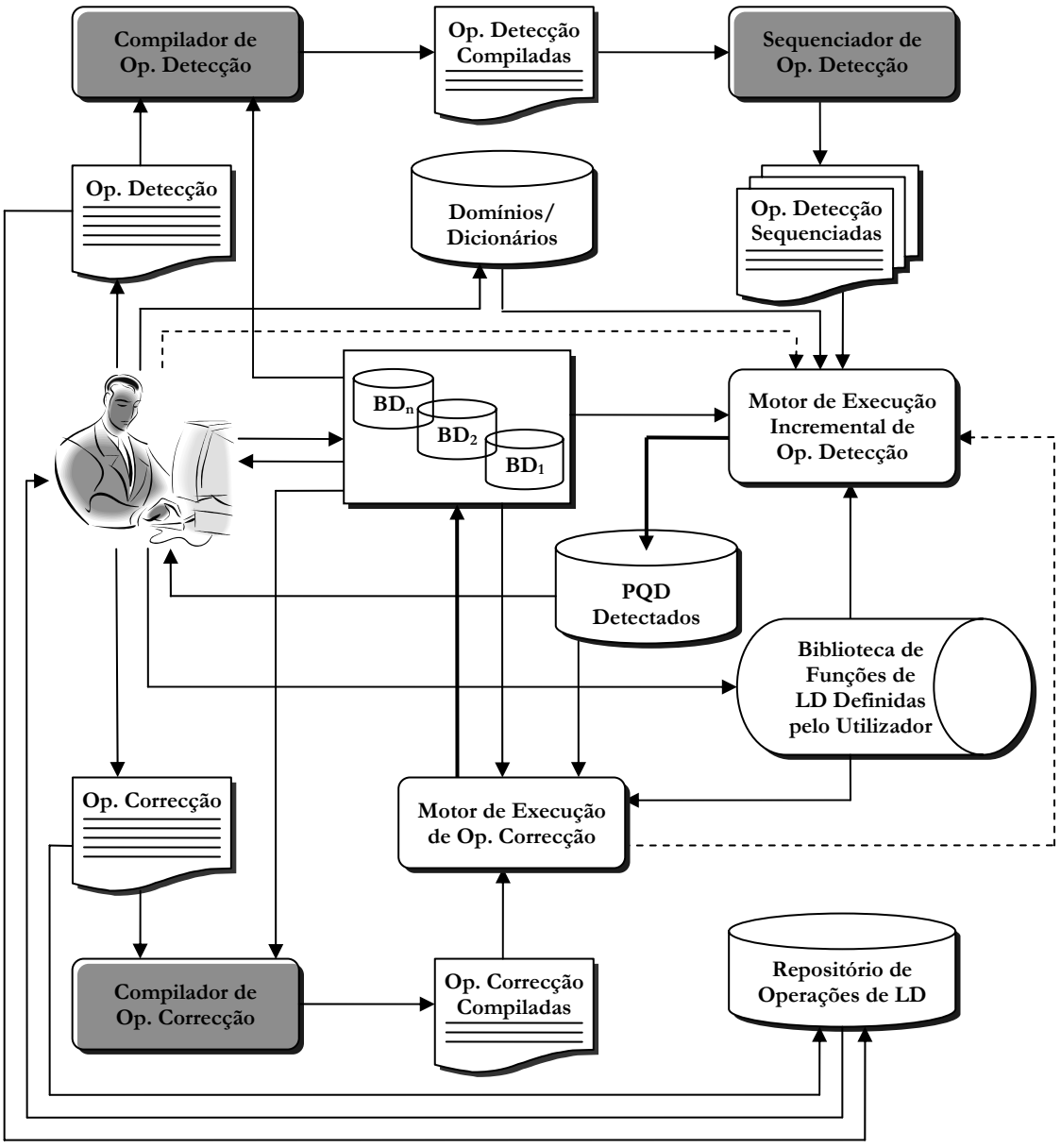


Figura 7.1 – Arquitectura proposta para a LD

No diagrama, os fluxos a traço contínuo representam um *fluxo de dados*, enquanto que os fluxos a tracejado representam um *fluxo de controlo*. Uma descrição dos diversos componentes que compõem a arquitectura e da forma como estes interactuam entre si é, a seguir, fornecida.

O *utilizador*[17] começa por especificar as OD que pretende executar. Estas operações são especificadas tendo por base a linguagem declarativa apresentada no Capítulo 5. Após a especificação destas operações, estas são submetidas ao *compilador de OD*. Este efectua o *parse* das OD e caso seja detectado algum erro sintáctico, este é reportado ao utilizador para que este proceda à sua correcção. No diagrama da arquitectura, este fluxo não foi intencionalmente representado, para não o sobrecarregar excessivamente. Na situação contrária (*i.e.*, não ser detectado qualquer problema sintáctico), as OD são convertidas para uma representação interna, utilizada nos módulos *sequenciador de OD* e *motor de execução incremental das OD*.

Na sua nova representação, as OD são sujeitas ao módulo *sequenciador de OD*. Este é um dos principais módulos da arquitectura, daí que vá ser objecto de apresentação detalhada na secção seguinte. Por agora, é suficiente dizer que este módulo é responsável por organizar as operações por NG (*e.g.*: atributo; tuplo; relação) e, em cada um destes, estabelecer a ordem pela qual serão executadas. Neste trabalho defende-se a detecção e, de imediato, a correcção dos PQD por NG, começando no mais elementar (*i.e.*, o atributo) e terminando no de maior complexidade (*i.e.*, múltiplas relações de diferentes fontes de dados). Além desta abordagem ascendente, também se advoga que em cada NG os PQD devem ser manipulados (*i.e.*, detectados e corrigidos) seguindo uma determinada sequência. Compete a este módulo, efectuar todo este trabalho de organização e sequenciação das operações.

Uma vez agrupadas por NG e sequenciadas, as operações são submetidas ao módulo *motor de execução incremental das OD*. À semelhança do anterior, pela sua importância na arquitectura, este módulo também merece uma atenção especial. A sua apresentação será efectuada, em pormenor, na Secção 7.4, sendo aqui fornecida apenas uma descrição sucinta. Como o próprio nome o indica, este módulo é responsável pela execução de cada OD. Naturalmente, estas operações são executadas sobre a Base de Dados[18] (BD) alvo de LD. Apesar da panóplia de OD que se disponibilizam de raiz (apresentadas e formalizadas no Capítulo 5) e que este módulo permite

[17] O termo *utilizador* é empregue com o sentido de alguém que possui os conhecimentos necessários para efectuar LD, *i.e.*, de especificar as OD adequadas e, caso sejam identificados PQD, especificar as OC apropriadas ou corrigir manualmente esses PQD.

[18] Ainda que o termo surja sempre no singular (*i.e.*, base de dados), o processo de LD pode envolver simultaneamente várias BD. Aliás, esta perspectiva é ilustrada graficamente na figura que representa a arquitectura.

executar, o utilizador pode especificar operações que envolvem a invocação de funções definidas por si. A necessidade do utilizador recorrer a estas funções, advém de requisitos específicos de detecção de PQD (*e.g.*: detecção de violação de uma dada restrição de integridade). Uma particularidade deste módulo é que certas OD (no caso, as referentes aos NG do atributo e do tuplo) são executadas incrementalmente. Assim, além da execução das OD ser efectuada por NG, em cada NG, a detecção de um determinado PQD faz com que se pare nesse ponto. Todas as restantes operações da sequência de detecção que envolvam esse valor ou valores ficam com a execução pendente até que o problema seja solucionado. Todas as outras OD da sequência cuja execução não dependa da correcção do PQD são executadas. Quando todas as OD tiverem sido executadas é que o utilizador toma conhecimento dos problemas entretanto identificados. Após a correcção dos PQD, a execução da sequência de OD reinicia-se, mas apenas nos locais onde tinha parado anteriormente, *i.e.*, onde tinham sido detectados os problemas. Os valores já anteriormente analisados e nos quais não foram detectados PQD não são novamente considerados. Até todos os problemas estarem solucionados, podem ser necessárias múltiplas iterações detecção – correcção. Em cada iteração, os PQD existentes vão sendo progressivamente detectados e, de seguida, corrigidos, tendo como ponto de reinício da detecção os locais onde esta tinha sido interrompida na iteração anterior. É por este motivo que se considera a execução das OD incremental. Da execução de cada OD resulta a identificação dos PQD existentes, sendo estes armazenados num repositório. Sempre que a execução das OD é interrompida, o utilizador acede a este repositório para analisar os PQD identificados (que problemas e em que locais).

Os PQD detectados podem ser solucionados por via manual, automática ou um misto das duas (*i.e.*, certos problemas são solucionados manualmente enquanto outros o são automaticamente). Pela primeira via (*i.e.*, manual), o utilizador interage directamente com a própria BD na correcção individual dos PQD identificados. Apesar dos vários inconvenientes desta solução (*e.g.*: morosa; trabalhosa; propícia a erro), em certos PQD não há outra hipótese que não seja o recurso a esta via. Alguns problemas (*e.g.*: resolução de uma circularidade entre tuplos) não parecem susceptíveis de resolução automática, uma vez que não há uma regra genérica que possa ser enunciada e aplicada. A sua resolução depende de uma análise individualizada a cada PQD detectado. Pela segunda via (*i.e.*, automática), as correcções são efectuadas automaticamente em função das operações definidas pelo utilizador. Contrariamente às OD que são especificadas ao mesmo tempo para os diversos NG, as OC são especificadas para solucionar os PQD detectados no nível em questão. As OC são especificadas com base na linguagem declarativa apresentada no capítulo anterior.

O processo a que as OC são sujeitas é em tudo idêntico ao das OD. Assim, após a sua especificação, são submetidas ao *compilador de OC*. O *parse* das operações é efectuado, sendo reportados ao utilizador os erros sintácticos detectados, para que este efectue a sua correcção. Caso não sejam detectados problemas sintácticos, as OC são convertidas para uma representação interna e submetidas ao módulo *motor de execução das OC*. Em virtude da sua importância na arquitectura, este módulo também será objecto de uma atenção especial, corporizada na Secção 7.5 deste capítulo. Por agora, é suficiente dizer que este módulo tem por objectivo a execução das OC especificadas. As OC que sejam de tipos diferente são executadas paralelamente, uma vez que não há qualquer tipo de dependência entre estas. No caso de existir mais do que uma operação de um dado tipo (*e.g.*: correcção de violação de sintaxe), incidindo sobre o mesmo atributo ou conjunto de atributos de uma ou mais relações, estas são executadas respeitando a ordem de especificação do utilizador. Na execução de uma operação, este módulo acede ao repositório onde se encontram os PQD detectados, uma vez que apenas estes são alvo de correcção. Da execução da operação resultam actualizações à BD, com o objectivo de solucionar os PQD detectados. No Capítulo 6, formalizou-se um conjunto alargado de OC que, de base, suportam a generalidade dos PQD que compõem a taxionomia. Este módulo é responsável por suportar a execução dessas OC. Apesar da sua existência, a especificidade de certas OC (*e.g.*: correcção de uma determinada violação de restrição de integridade) pode implicar que se recorram a funções definidas pelo próprio utilizador. Após a execução das OC é despoletado novamente a execução das sequências de OD, nos mesmos pontos onde estas tinham parado anteriormente (*i.e.*, aquando da iteração anterior). Desta forma, reinicia-se uma nova iteração de detecção e, na eventualidade de serem detectados outros PQD, também de correcção. Quando não for detectado qualquer PQD no NG em questão, avança-se para a detecção dos problemas no nível seguinte. Caso se trate do último NG, este processo iterativo de detecção – correcção dos PQD é dado como concluído.

A execução das OD ou OC pode envolver a invocação de funções implementadas pelo próprio utilizador. A finalidade da biblioteca de funções de LD que se encontra na arquitectura é, precisamente, a de permitir o armazenamento das funções definidas pelo utilizador. Dependendo do domínio, podem ser necessárias funções específicas que suportem a DC de certos PQD. O problema que mais frequentemente obriga ao desenvolvimento de funções é o das restrições de integridade. Isto acontece em virtude da sua diversidade e dependência ao domínio subjacente. Arquitectar, implementar e testar as funções são tarefas feitas pelo utilizador em separado. Além destas, a inclusão das funções na biblioteca é também uma tarefa a cargo do utilizador. Uma vez

incluída na biblioteca, uma função pode ser usada na situação actual de LD, mas também pode ser reutilizada em qualquer outra situação futura.

A execução da OD de erro ortográfico envolve o acesso a um dicionário de termos, armazenado sob a forma de uma tabela. A execução da OD de violação de domínio pode envolver o acesso ao domínio de valores válidos do atributo, também este armazenado sob a forma de uma tabela. A finalidade do repositório presente arquitectura intitulado de *Domínios/Dicionários* é, precisamente, a de suportar o seu armazenamento. A definição e manutenção destes constituem tarefas que ficam a cargo do utilizador.

Quer as OD quer as OC são armazenadas num repositório de operações de LD. O objectivo deste repositório é de funcionar como um histórico onde ficam registadas todas as operações efectuadas. O utilizador pode aceder a este repositório e, entre os diversos conjuntos de operações armazenados ao longo do tempo, seleccionar um destes para execução integral (*i.e.*, todas as operações) ou apenas parcial (*i.e.*, escolhendo apenas algumas operações para execução). O utilizador também pode definir as operações a executar, seleccionando-as a partir de diferentes conjuntos de operações armazenados ao longo do tempo.

Há um requisito a que todas as relações envolvidas em operações de LD têm de obedecer: ter uma chave primária definida. Esta chave primária é utilizada na detecção para assinalar os tuplos que se encontram afectados por PQD. A mesma chave primária é utilizada na correcção para solucionar os PQD nos tuplos anteriormente identificados. Assim, se não existir uma chave primária numa tabela, é acrescentado um atributo identificador numérico que cumpre essa finalidade. Esta tarefa não foi representada no diagrama da arquitectura para não o sobrecarregar em demasia.

Além dos fluxos de dados, no diagrama da arquitectura apresentado na Figura 7.1 encontram-se representados dois fluxos de controlo (a tracejado). Um dos fluxos tem início no módulo *motor de execução das OC* e fim no módulo *motor de execução incremental das OD*. Este fluxo representa o desencadear automático da execução das OD após a execução das OC. O outro fluxo tem origem no *utilizador* e termina no *motor de execução incremental das OD*. Este fluxo representa a capacidade que o utilizador tem de despoletar a execução incremental das OD. Em circunstâncias "normais", a execução das OD é desencadeada após a execução do módulo *sequenciador de OD* ou do *módulo motor de execução das OC*. No entanto, uma vez que o utilizador pode efectuar correcções manualmente na própria BD, este também tem a possibilidade de despoletar a re-execução das OD. Desta forma, o utilizador certifica-se que os problemas foram efectivamente solucionados e

dá continuidade à execução de outras eventuais OD que estivessem pendentes da resolução desses PQD.

Nas três secções seguintes descrevem-se, com o devido detalhe que merecem, os principais módulos da arquitectura que acabou de ser apresentada.

7.3 Sequenciador de Operações de Detecção

Este módulo da arquitectura é responsável por colocar as OD definidas pelo utilizador de acordo com a ordem de execução preconizada neste trabalho. Assim, as operações começam por ser organizadas por NG. As primeiras a serem executadas são as que envolvem o valor individual dos atributos. As últimas a executar são as que envolvem relações de diferentes fontes de dados. Em cada um dos NG, este módulo também é responsável por colocar as operações pela ordem que serão executadas. Há operações que necessariamente têm de ser executadas antes de outras. Estas duas tarefas são apresentadas em detalhe nas duas secções seguintes.

7.3.1 Sequenciação por Nível de Granularidade

Neste trabalho defende-se um modelo de LD em que os PQD são manipulados (*i.e.*, detectados e corrigidos) por NG. A existência de um problema num NG pode impedir a detecção de um ou mais problemas noutro ou noutros níveis (*e.g.*: a existência de um erro ortográfico (NG: atributo – contexto do valor individual) pode impedir a detecção da existência de sinónimos (NG: multi-relação – multi-fonte de dados)). Noutras situações, a existência de um problema num NG pode resultar na detecção de um ou mais problemas noutro ou noutros níveis (*e.g.*: a existência de sinónimos (NG: atributo – contexto multi-valor) pode conduzir à detecção de uma violação de dependência funcional (NG: relação)). Neste caso, um PQD num NG pode reflectir-se, em cascata, num conjunto de outros problemas noutros níveis. Ao corrigir-se o primeiro, todos os outros supostos problemas também são solucionados.

Face à interdependência entre problemas de diferentes NG, só se deve avançar para a detecção dos problemas num NG quando todos os problemas dos outros níveis mais elementares tiverem sido detectados e solucionados. Caso não se adopte esta abordagem gradual, podem não ser detectados certos problemas ou, redundantemente, serem reportados outros.

Na Figura 7.2 apresenta-se a sequência que se propõe para a DC dos PQD por NG. A sequência segue uma filosofia ascendente (*i.e.*, *bottom-up*). Inicia-se com os problemas que ocorrem no NG

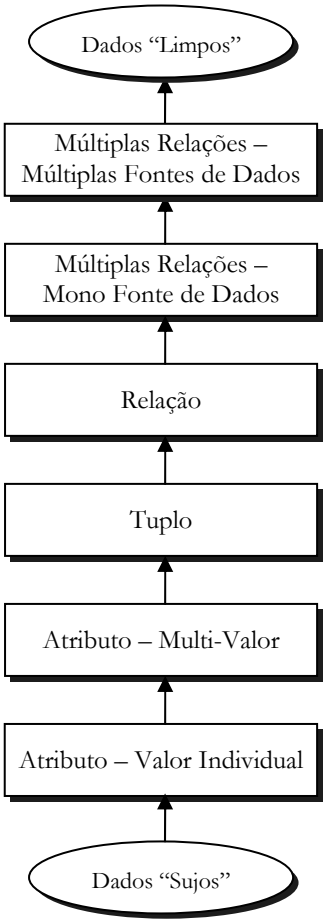


Figura 7.2 – Sequência de DC dos PQD por NG

mais elementar (*i.e.*, os que ocorrem ao nível do atributo) e termina nos que apresentam uma maior complexidade (*i.e.*, os que ocorrem ao nível de múltiplas relações de diferentes fontes de dados). Como resultado da aplicação desta sequência de manipulação a dados afectados por problemas de qualidade (na figura representados como *dados* "*sujos*"), obtêm-se dados em que esses problemas estão solucionados (na figura representados como *dados* "*limpos*").

Face à sequência apresentada, os primeiros problemas a serem manipulados (*i.e.*, detectados e, de seguida, corrigidos) são os que ocorrem no NG do atributo, em particular, os que se verificam no contexto do valor individual (*e.g.*: violação de sintaxe; erro ortográfico). O objectivo consiste em começar por manipular os problemas que envolvem um único valor e cuja resolução não depende de outros valores, sejam estes do mesmo ou de outros atributos. Por outro lado, a detecção dos PQD nos restantes NG depende da resolução dos problemas que ocorrem no contexto do valor individual do atributo. Por exemplo, a existência de um erro ortográfico pode: (*i*) impedir a detecção da existência de sinónimos (NG: atributo – contexto multi-valor); (*ii*) impedir a detecção da violação de uma determinada restrição de integridade (NG: tuplo); (*iii*) resultar na detecção de uma violação de dependência funcional (NG: relação); (*iv*) resultar na

detecção de uma violação de integridade referencial (NG: multi-relação – mono fonte de dados); e, (*v*) impedir a detecção da existência de tuplos duplicados (NG: multi-relação – multi-fonte de dados).

Os próximos problemas a serem manipulados continuam a ser os que se manifestam no NG do atributo, mas agora os que se verificam no contexto dos seus múltiplos valores quando analisados conjuntamente (*e.g.*: violação de unicidade; existência de sinónimos). O objectivo consiste em manipular os PQD que afectam cada atributo quando considerado isoladamente (*i.e.*, apenas uma coluna da relação) e cuja resolução depende apenas dos seus valores e não dos valores de outros atributos da relação. Nesta fase da sequência, os eventuais problemas existentes ao nível do valor individual do atributo já foram detectados e solucionados. A detecção dos PQD nos restantes NG está condicionada à resolução dos problemas que ocorrem no contexto dos múltiplos valores do atributo. Por exemplo, a existência de valores sinónimos num atributo pode: (*i*) impedir a detecção de uma determinada violação de restrição de integridade (NG: tuplo); (*ii*) conduzir à detecção de uma violação de dependência funcional (NG: relação); (*iii*) impedir a detecção de uma determinada violação de restrição de integridade (NG: multi-relação – mono fonte de dados); e, (*iv*) impedir a detecção da existência de tuplos duplicados (NG: multi-relação – multi-fonte de dados).

A seguir são manipulados os problemas que ocorrem no NG do tuplo (neste nível, há apenas um tipo de problemas: violação de restrição de integridade). O objectivo consiste em manipular os PQD que afectam cada tuplo quando considerado individualmente (*i.e.*, apenas uma linha da relação) e cuja resolução depende apenas dos valores dos seus atributos e não dos valores dos atributos de outros tuplos da relação. Note-se que nesta fase da sequência, os possíveis problemas existentes ao nível do atributo já foram detectados e solucionados. A detecção dos PQD nos demais NG encontra-se dependente da resolução dos problemas que ocorrem ao nível do tuplo. Por exemplo, a existência de uma dada violação de restrição de integridade ao nível do tuplo pode: (*i*) impedir a detecção de tuplos duplicados (NG: relação); (*ii*) resultar na detecção de uma violação de restrição de integridade (NG: multi-relação – mono fonte de dados); e, (*iii*) impedir a detecção da existência de tuplos duplicados (NG: multi-relação – multi-fonte de dados).

Os problemas que ocorrem no NG da relação (*e.g.*: violação de dependência funcional; tuplos duplicados) são os próximos a serem manipulados. O objectivo consiste em manipular os PQD que afectam cada relação quando considerada isoladamente (*i.e.*, múltiplas linhas e múltiplas colunas da relação) e cuja resolução depende somente dos valores dos seus atributos e não dos

valores dos atributos de outras relações. Recorde-se que nesta fase da sequência, os eventuais problemas existentes ao nível do atributo e do tuplo já foram detectados e solucionados. A detecção dos PQD nos restantes NG encontra-se condicionada à resolução dos problemas que ocorrem ao nível da relação. Por exemplo, a existência de tuplos duplicados ao nível da relação pode resultar na detecção de uma violação de restrição de integridade nos NG multi-relação de uma fonte de dados e multi-relação de diferentes fonte de dados.

Os próximos problemas a serem manipulados são os que ocorrem ao nível de múltiplas relações de uma fonte de dados (*e.g.*: violação de integridade referencial; violação de restrição de integridade). O objectivo passa por manipular os PQD que afectam as várias relações da fonte de dados quando consideradas conjuntamente e cuja resolução depende exclusivamente destas e não de relações existentes noutras fontes de dados. Nesta fase da sequência, os possíveis problemas existentes nos NG atributo, tuplo e relação individual já foram detectados e corrigidos. A detecção dos PQD existentes no NG múltiplas relações de múltiplas fontes de dados encontra-se dependente da resolução dos problemas que ocorrem ao nível multi-relação de cada fonte de dados individual. Por exemplo, a existência de sinónimos entre relações de uma fonte de dados pode impedir a detecção de uma violação de restrição de integridade ao nível de múltiplas relações de diferentes fontes de dados.

Por último, são manipulados os problemas que ocorrem ao nível de múltiplas relações de diferentes fontes de dados. O objectivo consiste em manipular os PQD que afectam várias relações de diferentes fontes de dados quando consideradas conjuntamente. Note-se que nesta última fase da sequência, os eventuais problemas existentes nos NG atributo, tuplo, relação e multi-relação de cada fonte de dados já foram detectados e corrigidos, daí que estejam reunidas as condições para a manipulação dos problemas deste NG.

Este módulo começa por separar as OD especificadas pelo utilizador, organizando-as por NG. A sua execução ficará a cargo do módulo respectivo (*i.e.*, motor de execução incremental das OD) e respeita a sequência ascendente por NG que acabou de ser descrita. Em primeiro lugar são executadas as OD que incidem sobre o atributo e, em último, as que envolvem relações de fontes de dados diferentes.

7.3.2 Sequenciação em cada de Nível de Granularidade

Da mesma forma que se defende a manipulação gradual dos PQD por NG, também se advoga em cada nível a DC progressiva dos PQD, de acordo com uma determinada sequência. A

existência de um PQD num NG pode impedir a detecção de um ou mais problemas no mesmo NG (*e.g.*: ao nível do atributo, a existência de uma violação de sintaxe pode impedir a detecção de uma violação de domínio). É necessário solucionar o PQD detectado para que seja possível identificar outros problemas. Por outro lado, a existência de um problema num NG pode resultar na detecção de um ou mais problemas no mesmo NG (*e.g.*: ao nível do valor individual do atributo, a existência de um erro ortográfico pode conduzir à detecção de uma violação de domínio). À semelhança do que acontece entre problemas de diferentes NG, um PQD de um NG também pode reflectir-se, em cascata, num conjunto de outros problemas no mesmo NG. Ao corrigir-se o primeiro, todos os outros supostos problemas são também solucionados. Face à interdependência entre os PQD em cada NG, só se deve avançar para a detecção de um problema quando todos os problemas de que este depende, tiverem sido detectados e corrigidos. Caso não se adopte esta abordagem gradual, podem não ser detectados certos problemas ou, de forma redundante, serem identificados outros. Nas secções seguintes, apresenta-se a sequência de manipulação proposta para cada NG de PQD.

Estando as OD organizadas por NG, este módulo estabelece qual a sua ordem de execução, originando sequências de dependências em cada um dos níveis, de acordo com o proposto nas secções seguintes. A execução das operações será efectuada pelo módulo respectivo (*i.e.*, motor de execução incremental das OD), respeitando essas sequências de dependências em cada NG.

7.3.2.1 Atributo – Contexto do Valor Individual

Na Figura 7.3 apresenta-se a sequência de execução das OD dos PQD que ocorrem ao nível do atributo, mais especificamente no contexto do seu valor individual.

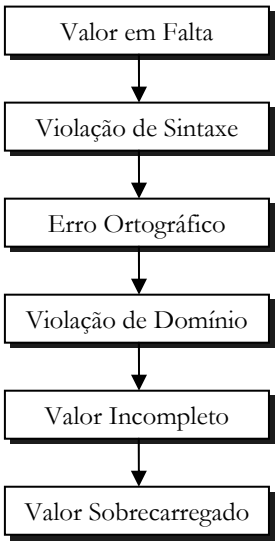


Figura 7.3 – Sequência de dependências das OD no contexto do valor individual do atributo

No caso do utilizador especificar OD para todos os problemas susceptíveis de afectarem o valor individual de um dado atributo, a execução dessas operações é efectuada de acordo com a sequência apresentada. No caso de sobre o atributo incidir apenas um subconjunto de OD (*e.g.*: violação de sintaxe; violação de domínio), a sua execução obedece na mesma à sequência apresentada (*i.e.*, em primeiro lugar a detecção de violação de sintaxe, seguida da detecção de violação de domínio).

A primeira operação a ser executada é a de detecção de valores em falta. A existência de um valor em falta num atributo de preenchimento obrigatório, obriga a que este problema seja identificado e imediatamente solucionado. Após ter sido fornecido o valor, é possível executar as restantes operações da sequência de OD. Desta forma, verifica-se se o valor fornecido possui qualquer outro PQD.

A OD de violação de sintaxe é a seguinte da sequência. A existência de uma violação de sintaxe pode influenciar negativamente a detecção de erros ortográficos e violações de domínio. Uma violação de sintaxe pode causar a detecção redundante de um erro ortográfico ou de uma violação de domínio. Como tal, a DC das violações de sintaxe tem de preceder a detecção de erros ortográficos e violações de domínio.

A próxima operação a ser executada envolve a detecção de erros ortográficos. A existência de um erro ortográfico origina a detecção redundante de uma violação de domínio. Para que tal não aconteça, é necessário começar por detectar e solucionar os erros ortográficos existentes.

A OD de violação de domínio é a seguinte da sequência. Estando solucionados os PQD que podem resultar na detecção de violações de domínio, que não são mais do que um mero reflexo desses problemas, estão reunidas as condições para a execução desta operação.

Além da OD de violação de domínio, também podem ser especificadas pelo utilizador OD de valor incompleto ou de valor sobrecarregado. Os valores incompletos ou sobrecarregados representam casos específicos de violação de domínio. Estas operações ainda que normalmente acompanhem a OD de violação de domínio, podem ser efectuadas independentemente desta existir ou não. Quando existe a OD de violação de domínio, qualquer uma das duas OD (de valor sobrecarregado ou incompleto) tira partido dos resultados já produzidos. Apenas constitui um possível valor incorrecto ou sobrecarregado, os que foram identificados como violações ao domínio. A não violação do domínio do atributo garante, desde logo, que o valor não é incompleto nem sobrecarregado. Quando o valor não viola o domínio, nem sequer se justifica a

execução das OD de valor incompleto ou sobrecarregado. Por este motivo, as OD de valor incompleto e valor sobrecarregado surgem na sequência após a OD de violação de domínio.

A operação seguinte da sequência é, então, a de detecção de valor incompleto. A ordem entre esta OD e a de detecção de valor sobrecarregado é absolutamente irrelevante. No caso, optou-se por colocar em primeiro a de detecção de valor incompleto, ficando como última operação da sequência a detecção de valor sobrecarregado. A execução sequencial destas duas operações é importante, uma vez que os valores identificados como estando incompletos, já não são verificados pela OD de valor sobrecarregado. Note-se que um valor não pode estar, ao mesmo tempo, incompleto e sobrecarregado.

No contexto do valor individual do atributo, o presente módulo é responsável pela criação das sequências de dependências das OD especificadas. Para cada atributo, de cada relação, de cada BD que se encontre envolvido em operações ao nível do valor individual do atributo é criada uma sequência de dependências de execução. As OD especificadas pelo utilizador começam por ser agrupadas por atributo. A cada OD relativa ao mesmo atributo é atribuído um número sequencial inteiro que corresponde à ordem que esta ocupa na sequência de execução proposta. Seguidamente, as OD que incidem sobre o atributo em questão são seriadas em função desse número sequencial, o que permite estabelecer a ordem pela qual são executadas.

7.3.2.2 Atributo – Contexto Multi-Valor

Na Figura 7.4 apresenta-se a sequência de execução das OD dos problemas que ocorrem ao nível do atributo, mas agora no contexto dos seus múltiplos valores (*i.e.*, coluna da relação).

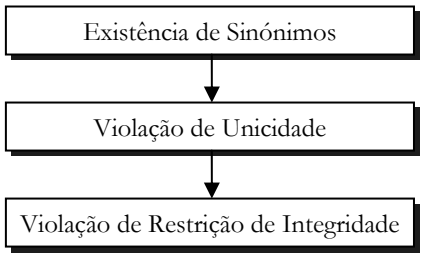


Figura 7.4 – Sequência de dependências das OD no contexto multi-valor do atributo

No caso do utilizador especificar OD para todos os PQD susceptíveis de se manifestarem no contexto dos múltiplos valores de um dado atributo, a execução dessas operações é efectuada de acordo com a sequência apresentada. No caso de sobre o atributo incidir apenas um subconjunto de OD, a sua execução respeita na mesma a sequência apresentada

A primeira operação da sequência consiste na OD de existência de sinónimos. Na dependência de execução desta operação encontram-se as OD de violação de unicidade e violação de restrição de integridade. No caso da violação de unicidade, a criação da dependência visa "optimizar" a execução desta operação. Os pares de valores identificados como sinónimos, naturalmente não são iguais, daí que possam ser excluídos da OD de violação de unicidade. Apenas por este motivo é que a execução destas duas operações não é realizada em simultâneo. De facto, a existência de sinónimos nos valores de um atributo não impede a detecção das violações de unicidade que possam existir e vice-versa. Igualmente, da existência de sinónimos não resulta a detecção de violações de unicidade que surgem como um mero reflexo daquele problema e vice-versa. No caso da violação de restrição de integridade, o cenário já não é o mesmo, uma vez que a existência de sinónimos pode impedir a sua detecção. Pelos motivos expostos, a detecção da existência de sinónimos surge como primeira operação da sequência.

A operação seguinte da sequência envolve a detecção de violações à unicidade do atributo. Este tipo de problema tem de ser identificado e solucionado antes da OD de violação de restrição de integridade. Se tal não acontecer, uma violação de unicidade pode traduzir-se numa violação de restrição de integridade. Esta última manifesta-se apenas como uma consequência da existência da primeira.

Pelos motivos já apresentados, a detecção de violação de restrição de integridade constitui a última operação da sequência. Para que seja possível detectar correctamente as violações de restrição de integridade existentes é necessário que o atributo não contenha sinónimos e que não existam violações à unicidade do atributo. No caso deste tipo específico de PQD, o utilizador pode especificar mais do que uma OD de violação de restrição de integridade que incida sobre o mesmo atributo. Quando tal acontece, também pode existir uma ordem na sua execução, *i.e.*, apenas ser executada a detecção de uma dada violação de restrição de integridade, quando uma outra tiver sido executada e os respectivos PQD identificados, devidamente solucionados. Assim, estas OD são executadas segundo a ordem pela qual foram especificadas pelo utilizador. Mediante a semântica subjacente a estas operações, é ao utilizador a quem compete colocá-las pela ordem de execução adequada.

O presente módulo é responsável pela criação das sequências de dependências das OD especificadas no contexto dos múltiplos valores do atributo. Para cada atributo, de cada relação, de cada BD que se encontre envolvido em operações ao nível dos múltiplos valores do atributo, é elaborada uma sequência de dependências. Na elaboração de cada sequência de dependências de

execução das OD, recorre-se à mesma abordagem descrita no último parágrafo da secção anterior (contexto do valor individual do atributo).

7.3.2.3 Tuplo

O NG do tuplo é composto apenas por um tipo de PQD: violação de restrição de integridade. Assim, neste nível, parece não fazer sentido falar-se em sequência de OD. No entanto, tal como no contexto multi-valor do atributo, o utilizador pode especificar múltiplas OD de violação de restrição de integridade. Estas operações podem incidir sobre atributos diferentes ou sobre os mesmos atributos. Na primeira situação, as OD são executadas em paralelo, uma vez que pertencem a sequências de dependências diferentes. Na segunda situação, em que há atributos comuns nas operações, estes podem-no ser na totalidade ou apenas parcialmente. Quando há atributos comuns, as operações podem ter de ser executadas segundo uma determinada ordem. Esta ordem de execução corresponde à ordem de especificação das operações por parte do utilizador, uma vez que é este quem conhece a semântica que lhes está subjacente.

Ainda que o utilizador especifique as OD em função da ordem de execução pretendida, a sua execução não é necessariamente sequencial. Em função dos atributos sobre os quais incidem as OD que fazem parte da sequência de dependências, certas operações são executadas em simultâneo. Nestes casos, a sequência de dependências, na prática, corresponde a um grafo dirigido de OD. Para o ilustrar, considere-se o exemplo que a seguir se apresenta. Suponha-se que o utilizador especificou três OD de violação de restrição de integridade ao nível do tuplo: a primeira incide sobre os atributos *a* e *b*; a segunda incide sobre os atributos *c* e *d*; e, a terceira incide sobre os atributos *b* e *c*. Ainda que a ordem de especificação das OD tenha sido a apresentada, a sua execução não é feita literalmente de forma sequencial. As OD em causa originam o grafo de dependências apresentado na Figura 7.5.

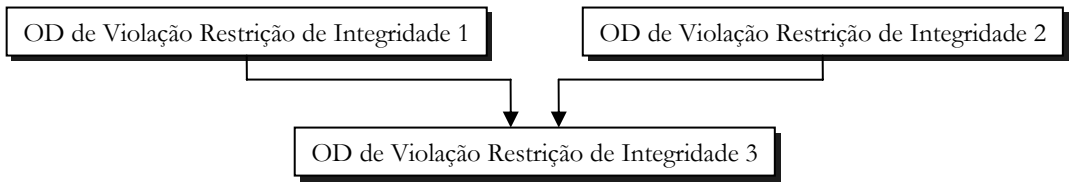


Figura 7.5 – Grafo dirigido de dependências correspondente às três OD consideradas no exemplo

Como se pode depreender do grafo de dependências, as duas primeiras OD de violação de restrição de integridade são executadas simultaneamente. Isto é possível em virtude das operações em causa incidirem sobre conjuntos de atributos disjuntos.

O módulo em questão é responsável pela criação das sequências de dependências das OD especificadas ao nível do tuplo. Para tal, as OD começam por ser agrupadas em função da existência de dependências entre estas, preservando-se a ordem pela qual foram especificadas pelo utilizador. As dependências traduzem-se na existência de atributos comuns entre as OD. Em cada sequência de dependências resultante, os atributos envolvidos em cada OD são analisados para identificar as situações em que é possível efectuar a sua execução em simultâneo. Caso haja operações nestas condições, isso origina um grafo dirigido de dependências de execução das OD. A informação relativa a cada sequência de dependências é armazenada numa tabela temporária auxiliar, cujo esquema contém os atributos: *NrOperação*, que corresponde ao número sequencial atribuído à OD em causa (cada OD especificada pelo utilizador é identificada por um número sequencial de 1 a *n*); e, *NrOperaçãoPrecedente*, que corresponde ao número sequencial atribuído à OD que a antecede. A título ilustrativo, o grafo de dependências apresentado na Figura 7.5 é armazenado de acordo com o que se encontra representado na Tabela 7.1.

Tabela 7.1 – Representação do grafo de dependências da Figura 7.5

NrOperação	NrOperaçãoPrecedente
3	1
3	2

Como as OD 1 e 2 não possuem operações precedentes, não são representadas na tabela. Como ambas se encontram na mesma situação, a sua execução é efectuada em simultâneo.

7.3.2.4 Relação

A Figura 7.6 ilustra graficamente as duas sequências de execução das OD dos PQD que ocorrem no NG da relação.

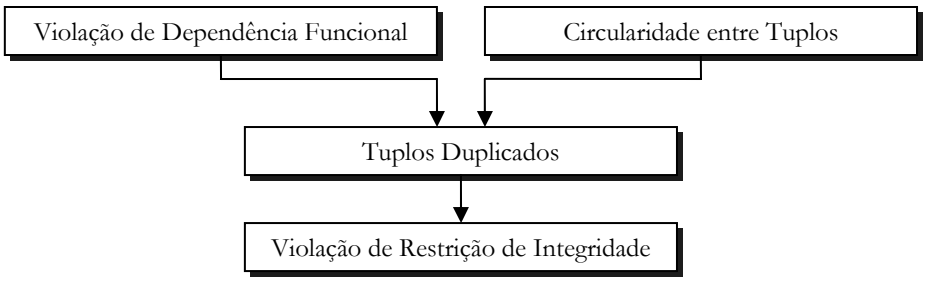


Figura 7.6 – Sequências de dependências das OD no NG da relação

Contrastando com o exposto até ao momento, neste NG há duas sequências de dependências, como consequência da possibilidade de execução alternativa das OD de violação de dependência funcional e circularidade entre tuplos num auto-relacionamento. Estas alternativas originam não uma, mas duas sequências de execução das OD: (*i*) violação de dependência funcional; tuplos duplicados; e, violação de restrição de integridade; ou, (*ii*) circularidade entre tuplos; tuplos duplicados; e, violação de restrição de integridade. Isto acontece em virtude das duas OD iniciais possuírem contextos de aplicação radicalmente diferentes. Assim, no contexto em que se justifica a execução de uma das operações, não se justifica a execução da outra. A execução de uma OD de violação de dependência funcional justifica-se no contexto de uma relação que não se encontra na terceira forma normal (*i.e.*, normalizada). Apenas nestes casos faz sentido identificar violações de dependência funcional. No caso da detecção de existência de circularidade entre tuplos, a execução desta operação justifica-se em relações que se encontram normalizadas. Só assim é possível implementar um mecanismo de auto-relacionamento. Em relações normalizadas não existem violações de dependência funcional.

Dependendo da situação, a primeira operação a executar consiste na detecção de violação de dependência funcional ou na detecção da existência de circularidade entre tuplos num auto-relacionamento. A existência de uma violação de dependência funcional ou de uma circularidade entre tuplos resulta de valores incorrectos nos atributos. Estes valores incorrectos podem impedir que um ou mais tuplos sejam identificados como duplicados de outros tuplos da relação. Por outro lado, da existência de uma violação de dependência funcional ou de uma circularidade pode resultar a detecção da violação a uma determinada restrição de integridade, que surge como um reflexo redundante da existência de qualquer um dos problemas. Por ambas as razões, é necessário começar por identificar e solucionar as violações de dependência funcional ou as circularidades entre tuplos existentes na relação.

A operação seguinte envolve a detecção dos tuplos duplicados. Este tipo de problemas tem de ser identificado e solucionado antes da detecção das violações de restrição de integridade. Caso tal não aconteça, podem ser reportadas supostas violações de restrição de integridade. A resolução prévia dos tuplos duplicados existentes faz com que estas supostas violações já não sejam detectadas e reportadas ao utilizador.

A última operação ao nível da relação envolve a detecção de violação de restrição de integridade. Para que a detecção deste tipo de problema produza resultados fiáveis é necessário que as violações de dependência funcional, as circularidades entre tuplos num auto-relacionamento e os tuplos duplicados tenham sido identificados e solucionados.

Em certos tipos de PQD (*e.g.*: violação de dependência funcional; violação de restrição de integridade) pode ser especificada mais do que uma OD. Quando isto acontece, caso as OD em causa não possuam atributos comuns, isso faz com que pertençam a sequências de dependências diferentes, pelo que a sua execução é efectuada paralelamente. Quando há atributos comuns, a execução das OD pode ter de obedecer a uma determinada ordem. Esta corresponde à ordem de especificação das OD pelo utilizador. Compete a este, em função da semântica das diferentes operações e das dependências que possam existir entre estas, sequenciá-las de modo a que a sua execução faça sentido. No entanto, ainda que o utilizador especifique as OD de acordo com a ordem de execução pretendida, isso não quer dizer que esta acabe por ser totalmente sequencial. Dependendo dos atributos que são alvo das OD que fazem parte da sequência de dependências, a execução de algumas destas acaba por ser efectuada simultaneamente. Nestes casos, a sequência de dependências corresponde a um grafo dirigido de OD. Com as devidas adaptações, esta situação é idêntica à exemplificada na secção anterior para o NG do tuplo. A Figura 7.7 ilustra-a graficamente.

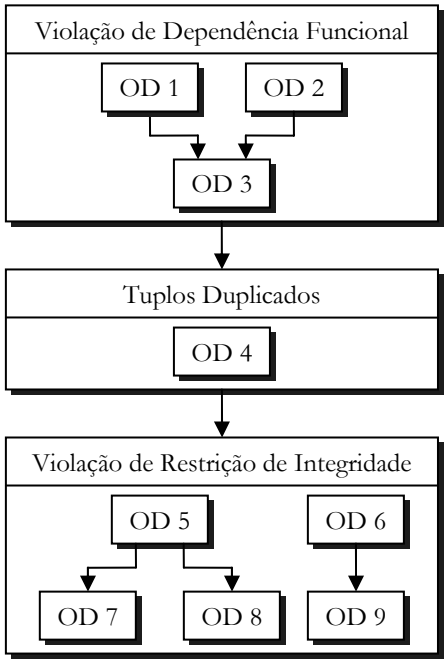


Figura 7.7 – Exemplo de um grafo de dependências de OD no NG da relação

Compete ao presente módulo a criação das sequências de dependências das OD especificadas ao nível da relação. As OD começam por ser agrupadas em função do tipo de PQD que visam, preservando-se a ordem pela qual foram especificadas pelo utilizador. Seguidamente, estes grupos de OD são colocados de acordo com a ordem que se defende para a sua execução. Em cada um dos grupos, as OD são novamente agrupadas, agora em função da existência de dependências

entre estas, mais uma vez mantendo-se a ordem de especificação do utilizador. Estas dependências manifestam-se através da existência de atributos comuns entre as OD. Os atributos envolvidos em cada OD pertencente a cada sequência de dependências resultante, permitem identificar as operações cuja execução pode ser efectuada em paralelo. Quando há operações nestas condições, daqui resulta um grafo dirigido de dependências de execução das OD. A informação referente a cada sequência de dependências é armazenada em tabelas temporárias auxiliares, cuja estrutura está exposta no penúltimo parágrafo da secção anterior.

7.3.2.5 Multi-Relação – Mono-Fonte / Multi-Fonte de Dados

A Figura 7.8 ilustra esquematicamente as três sequências de execução possíveis das OD dos PQD (correspondentes aos três caminhos possíveis) que envolvem múltiplas relações, independentemente destas pertencerem à mesma ou a diferentes fontes de dados. Uma vez que os problemas que se manifestam no contexto dos dois últimos NG (*i.e.*, fonte de dados e multi-fonte de dados) são comuns, a sequência de dependências das OD é também a mesma para ambos os casos. Sendo a mesma, justifica-se que a sua apresentação seja efectuada em conjunto.

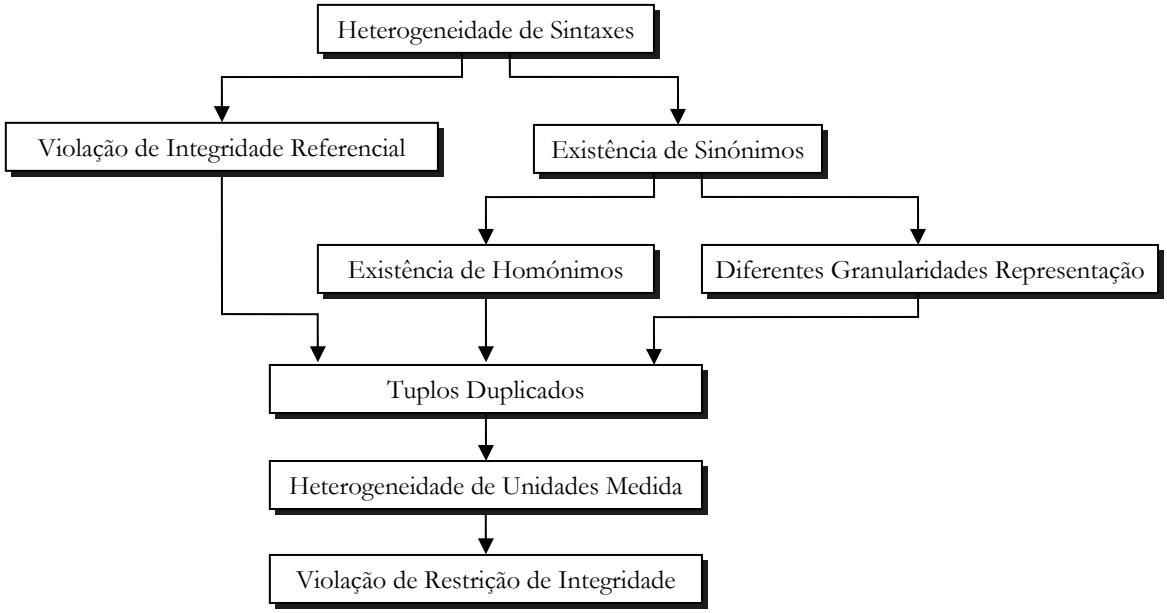


Figura 7.8 – Sequências de dependências das OD no NG multi-relação da fonte/multi-fonte de dados

A primeira operação que consta da sequência de dependências envolve a detecção de heterogeneidade de sintaxes. A existência deste PQD entre duas relações possui consequências nefastas, uma vez que afecta a detecção de todos os outros problemas. As diferenças sintácticas podem impedir a detecção de alguns problemas (*e.g.*: tuplos duplicados; heterogeneidade de unidades de medida) ou resultar na detecção de outros que apenas existem como um reflexo

destas (*e.g.*: violação de restrição de integridade). Como tal, a detecção dos restantes problemas encontra-se condicionada à DC das heterogeneidades sintácticas existentes entre os atributos das relações. Por esta razão, a execução desta OD encabeça a sequência de dependências apresentada.

As OD de violação de integridade referencial e de existência de sinónimos possuem contextos de aplicação radicalmente diferentes. Entre dois atributos nos quais se efectue uma OD de violação de integridade referencial, não faz qualquer sentido a execução de uma OD de sinónimos e vice-versa. Os atributos que estabelecem um relacionamento (entre duas tabelas), obviamente que não contêm valores sinónimos. Acresce ainda o facto dos tipos de dados dos atributos envolvidos em ambas as operações serem, normalmente, diferentes. Na detecção de sinónimos, o tipo de dados dos atributos é textual (*i.e.*, *string*). Na detecção de violações de integridade referencial, o tipo de dados dos atributos (chave primária e chave estrangeira), normalmente, é inteiro. O que acabou de se dizer também é valido entre a OD de violação de integridade referencial e as OD de existência de homónimos e de diferentes granularidades de representação. Assim, não faz sentido que estas operações existam simultaneamente na mesma sequência de execução de OD.

A existência de um valor num atributo de um tuplo que constitui uma violação de integridade referencial pode: (*i*) impedir que o tuplo seja identificado como duplicado de outro tuplo noutra relação (assumindo que o atributo é usado na identificação de tuplos duplicados); (*ii*) ao não permitir a identificação de tuplos duplicados, impedir a detecção de heterogeneidade de unidades de medida; e, (*iii*) causar a detecção de uma violação de restrição de integridade, em que esta surge como uma mera consequência do problema em causa.

A existência de sinónimos entre os valores de dois atributos: (*i*) resulta na detecção de diferentes granularidades de representação, sendo que este problema é apenas um mero reflexo do anterior; (*ii*) pode impedir que dois tuplos sejam identificados como duplicados; (*iii*) pode impedir a detecção de heterogeneidade de unidades de medida, em virtude de não terem sido identificados os duplicados existentes; e, (*iv*) pode impedir a detecção da violação de uma determinada restrição de integridade.

Na Figura 7.8, a execução da OD de homónimos depende da execução da OD de sinónimos. Esta dependência foi introduzida para "optimizar" a detecção de homónimos, uma vez que os pares de valores identificados como sinónimos podem ser excluídos da OD de homónimos. Esta dependência é, assim, diferente das mencionadas no parágrafo anterior. Na realidade, a existência de sinónimos: (*i*) não impede que sejam detectados homónimos; (*ii*) não resulta na identificação

de homónimos que surgem como um reflexo daquele problema. Se não tivesse sido introduzida a referida "optimização", estas duas operações seriam executadas em paralelo.

O contexto de execução das OD de existência de homónimos e de diferenças ao nível da granularidade de representação é totalmente diferente. Não faz qualquer sentido executar uma OD de homónimos aos valores de dois atributos e, simultaneamente, efectuar uma OD da existência de diferentes granularidades de representação. O objectivo da OD de homónimos consiste em identificar os valores dos atributos que são iguais. O objectivo da OD da existência de diferentes granularidades de representação envolve a identificação dos valores dos atributos que são diferentes. Como tal, os objectivos das duas operações são completamente antagónicos, daí que nunca envolvam os mesmos atributos. Ao não envolver os mesmos atributos, estas operações nunca podem existir, ao mesmo tempo, na mesma sequência de execução de OD.

A existência de homónimos entre os valores de dois atributos pode: (*i*) contribuir para a detecção de falsos tuplos duplicados, *i.e.*, tuplos identificados como duplicados mas que na realidade não o são; (*ii*) impedir a detecção de heterogeneidade de unidades de medida, em virtude de terem sido identificados falsos duplicados que afectam a correcta execução da operação; e, (*iii*) resultar na detecção da violação de uma determinada restrição de integridade que surge meramente como uma consequência da existência dos homónimos.

A existência de diferentes granularidades de representação pode: (*i*) impedir que dois tuplos sejam identificados como duplicados; (*ii*) ao não permitir a identificação de tuplos duplicados, impedir a detecção de heterogeneidade de unidades de medida; e, (*iii*) impedir a detecção da violação de uma determinada restrição de integridade.

O próximo PQD a ser detectado consiste na identificação dos tuplos duplicados existentes. Os resultados produzidos por esta operação são necessários na detecção de heterogeneidade de unidades de medida. Na realidade, nem sequer se consegue executar esta última se não existir uma operação prévia de detecção de tuplos duplicados. A detecção de heterogeneidade de unidades de medida entre os valores de dois atributos, implica que se saiba que os respectivos tuplos representam a mesma entidade do mundo real. Assim, a detecção de tuplos duplicados tem de preceder a detecção de heterogeneidade de unidades de medida. Naturalmente, na detecção de duplicados não podem ser usados os atributos que eventualmente possam estar afectados por heterogeneidade de unidades de medida. Regra geral, o tipo de atributos em causa (*e.g.*: representando valores monetários; temperaturas; distâncias) não é usado na detecção de tuplos duplicados. Este problema também tem de ser manipulado antes da detecção das violações

de restrição de integridade. A não identificação e resolução dos duplicados pode originar que sejam reportadas, de forma redundante, violações de restrição de integridade. Estas apenas surgem como consequência da existência de tuplos duplicados entre as relações.

A operação seguinte a ser executada envolve a detecção de heterogeneidade de unidades de medida. A existência deste PQD pode impedir a detecção da violação de uma dada restrição de integridade ou causar a detecção de uma suposta violação a uma restrição de integridade.

A última operação a ser executada é a de detecção de violação de restrição de integridade. Para que sejam detectadas as violações que efectivamente existem e não sejam reportadas supostas violações que, na prática, até não o são, é necessário que todos os outros anteriores problemas tenham sido identificados e solucionados.

Em cada tipo de PQD (*e.g.*: violação de integridade referencial; existência de sinónimos) pode ser especificada mais do que uma OD. Nesta situação, perante a inexistência de relações e atributos comuns entre as OD, a sua execução é efectuada em simultâneo, uma vez que pertencem a sequências de dependências de execução distintas. Quando há relações e atributos comuns nas OD, a sua execução poderá ter de ser efectuada seguindo uma determinada ordem. A ordem considerada é a que corresponde à sequência de especificação das operações por parte do utilizador. Em função da semântica inerente a cada operação e das precedências que possam existir entre estas, compete ao utilizador sequenciá-las de modo a que a sua execução possua um significado lógico.

Apesar do utilizador especificar as OD em função da ordem de execução desejada, tal não significa que estas sejam literalmente executadas de forma sequencial. Em função das relações e dos atributos sobre os quais incidem as OD que compõem a sequência de dependências, algumas destas operações acabam por ser executadas ao mesmo tempo. Nestes casos, a sequência de dependências corresponde a um grafo dirigido de OD. Esta situação é semelhante à exemplificada nas duas secções anteriores, respectivamente, para os NG do tuplo e da relação.

Este módulo tem por missão criar as sequências de dependências das OD especificadas ao nível multi-relação. Na criação destas sequências utiliza-se a mesma abordagem da que se encontra descrita no último parágrafo da secção anterior (referente ao NG da relação).

7.4 Motor de Execução Incremental das Operações de Detecção

Uma vez estabelecidas as sequências de dependências por NG das OD definidas pelo utilizador, é chegado o momento da sua execução. A execução das operações respeita estas sequências, iniciando-se com as que dizem respeito ao NG do atributo e terminado com as que se referem ao NG multi-relação de diferentes fontes de dados. Em cada NG podem existir diversas sequências de OD para execução. A existência destas sequências reflecte a independência de execução das operações que as compõem. Cada sequência pode ser composta por uma ou várias OD. Sempre que há sequências independentes, as OD que as integram são executadas em simultâneo.

Como se salientou na secção anterior, a existência de um determinado PQD pode impedir que outros problemas sejam detectados ou daí resultar a detecção redundante de outros supostos problemas. Estes problemas dificultam as tarefas de correcção do utilizador, uma vez que este tem de identificar os verdadeiros problemas e separá-los dos falsos problemas. Apenas os primeiros necessitam de ser solucionados, uma vez que os segundos, como consequência das correcções, também o são. Neste trabalho defende-se, portanto, que a detecção de um PQD deve ser seguida da sua imediata correcção. Adoptando este princípio, é necessário interromper constantemente a execução das OD e requerer a intervenção do utilizador para que este efectue as necessárias correcções. Como forma de minimizar este inconveniente, sempre que uma dada sequência de dependências o permite, procura-se identificar o máximo número possível de problemas em cada execução das OD que a compõem. Nestas situações, a execução de uma sequência de operações não é interrompida assim que se detecta o primeiro problema. Por exemplo, a identificação de um PQD num atributo de um tuplo impede que as demais OD que incidem sobre esse atributo desse tuplo possam ser executadas. No entanto, a existência deste problema não impede a execução das restantes OD sobre o atributo, em todos os outros tuplos nos quais o problema em questão não foi detectado. Assim, a execução das operações que fazem parte de uma sequência decorre até: (*i*) que não seja possível a execução de qualquer outra OD, em virtude dos problemas já detectados e das dependências existentes; ou, (*ii*) todas as operações terem sido executadas. Quando a execução das OD é interrompida (por um motivo ou outro) são reportados ao utilizador, de uma só vez, todos os PQD detectados dos diversos tipos. Por sua vez, este também pode, de uma só vez, executar as acções que conduzem à sua correcção. Desta forma, diminui-se ao número de intervenções que o utilizador necessita de efectuar durante o processo de LD (*i.e.*, de DC).

Após terem sido realizadas as correcções, a execução das OD reinicia-se nos mesmos locais onde anteriormente tinham sido detectados PQD. As operações que tinham sido responsáveis pela detecção dos problemas voltam a ser executadas, com o intuito de garantir que estes foram efectivamente solucionados. As OD da sequência que eventualmente tenham ficado pendentes vão agora ser executadas. No caso de serem detectados outros PQD, o procedimento descrito no parágrafo anterior volta a desenrolar-se. Note-se que nalguns NG (no caso, atributo – contexto do valor individual e tuplo), o reinício das OD apenas considera os valores nos quais, anteriormente, tinham sido detectados problemas. Todos os atributos/tuplos que já tinham sido analisados anteriormente e nos quais não se detectou a existência de problemas, ficam excluídos da execução das OD. As correcções efectuadas aos valores para solucionar os PQD detectados não têm qualquer influência nesses atributos/tuplos. Atendendo a que os problemas vão sendo gradualmente identificados à medida que outros são solucionados, daí dizer-se que a execução das OD é efectuada incrementalmente.

À semelhança do que sucedeu no módulo anterior, importa apresentar a forma como as OD são executadas quando se considera: (*i*) globalmente os vários NG; (*ii*) isoladamente cada NG. Contrastando com a secção anterior, nesta secção começa-se por expor esta última. Considera-se que esta inversão na ordem de apresentação facilita a exposição do motor de execução das OD.

7.4.1 Execução em cada Nível de Granularidade

Nas secções seguintes expõe-se a forma como são executadas as OD que compõem a sequência de dependências relativa a cada NG. Esta exposição inicia-se com o NG do atributo e termina no NG multi-relação.

7.4.1.1 Atributo – Contexto do Valor Individual

A identificação de um PQD no valor de um atributo de um tuplo, resultante de uma OD, não impede que as demais operações da sequência que incidem sobre esse atributo (no contexto do valor individual) sejam executadas. Considerando a sequência de dependências apresentada na Figura 7.3, a identificação de um PQD no valor de um atributo não implica a interrupção da execução das OD da sequência para que esse problema seja solucionado. A sequência de OD apenas é interrompida para o valor do atributo do tuplo em questão. Em todos os outros valores do atributo (referentes aos demais tuplos), a execução da sequência de OD prossegue com toda a normalidade. Só após a execução de todas as OD que fazem parte da sequência é que os PQD identificados por cada uma destas são reportados ao utilizador.

Após a realização das correcções pelo utilizador, independentemente de terem sido efectuadas por via manual ou automática (*i.e.*, com base numa OC), as OD especificadas voltam a ser executadas, de acordo com a sequência de dependências de execução. Na execução de cada operação que compõe a sequência são considerados somente os PQD anteriormente identificados. Desta forma, assegura-se que estes foram, de facto, solucionados pelas acções correctivas efectuadas. Cada OD da sequência considera, além destes, também os valores do atributo em que a execução das operações precedentes já não detectou a existência de PQD. O processo de execução repete-se, continuamente, até que das correcções efectuadas não resulte a identificação de qualquer PQD. Este processo de execução das OD que compõem a sequência encontra-se esquematicamente representado na Figura 7.9.

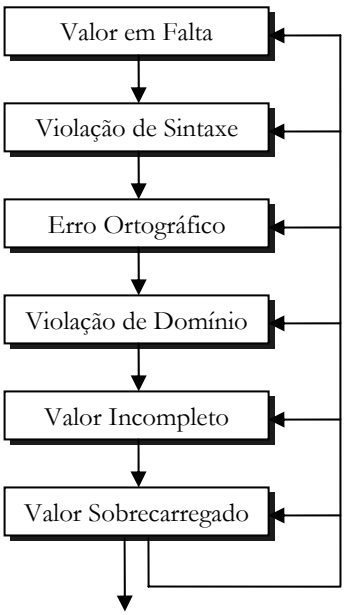


Figura 7.9 – Processo de execução das OD no contexto do valor individual do atributo

Para melhor ilustrar o processo atrás descrito, considere-se o exemplo seguinte que se inicia com a apresentação da relação que consta da Tabela 7.2. Suponha-se que o utilizador especificou três OD que incidem sobre o atributo *CodigoPostal*: valor em falta; violação de sintaxe; e, violação de domínio. A sequência de execução destas operações é precisamente esta (*i.e.*, em primeiro lugar valor em falta, depois violação de sintaxe e, por último, violação de domínio). A primeira operação a ser executada é a de detecção de valor em falta. Da execução desta operação resulta a identificação de um valor em falta no tuplo identificado com o *ID* 2. Como consequência, as restantes operações da sequência não são executadas neste valor até que o PQD seja solucionado. Nos restantes valores do atributo não é detectado qualquer valor em falta. Nestes, a sequência de execução das OD continua com toda a normalidade.

Tabela 7.2 – Relação para exemplificação da execução das OD no contexto do valor individual do atributo

ID	**atrib$_2$**	**CodigoPostal**	**…**	**atrib$_n$**
1	xxx	4000-123	…	xxx
2	xxx	*null*	…	xxx
3	xxx	4415-206	…	xxx
4	xxx	4415	…	xxx
5	xxx	4445-235	…	xxx
6	xxx	1000-111	…	xxx
7	xxx	3770-255	…	xxx

A operação seguinte a ser executada é a de detecção de violação de sintaxe. Esta operação considera todos os valores do atributo, à excepção do que se encontra no tuplo identificado com o *ID* 2. Da execução desta operação resulta a detecção de uma violação de sintaxe no tuplo identificado com o *ID* 4. A execução das restantes operações da sequência fica pendente para este valor até que o problema seja solucionado. Nos restantes valores do atributo não é detectada qualquer outra violação de sintaxe. Assim, nestes valores passa-se à execução da OD seguinte, *i.e.*, detecção de violação de domínio. Esta operação considera todos os valores do atributo, à excepção dos que foram identificados como estando em falta (tuplo com o *ID* 2) ou contendo uma violação à sintaxe (tuplo identificado com *ID* 4). Da execução desta última operação, resulta a detecção de uma violação de domínio no valor do atributo do tuplo identificado com o *ID* 6 (*i.e.*, 1000-111). Uma vez que esta é a última operação da sequência, conclui-se que todos os restantes valores (que constam dos tuplos com *ID* impar) não se encontram afectados pelos PQD em causa (*i.e.*, cuja detecção o utilizador especificou). Na próxima iteração das OD, estes valores não são considerados na execução destas, uma vez que já se confirmou que não estão afectados pelos problemas em questão. Ao utilizador são reportados os problemas encontrados nos três tuplos (*i.e.*, um valor em falta, uma violação de sintaxe e uma violação de domínio). Suponha-se que o utilizador procede à correcção manual de cada um destes problemas.

O reinício da execução das OD que compõem a sequência ocorre com a operação de valor em falta. A execução desta operação considera unicamente o valor do atributo *CodigoPostal* no tuplo identificado com o *ID* igual a 2. Constata-se que o problema foi de facto solucionado. A execução das operações da sequência prossegue com a detecção das violações de sintaxe. Esta

operação considera, não só o valor do atributo no qual anteriormente foi identificada a violação de sintaxe (*i.e.*, tuplo identificado com o *ID* 4), mas também o valor do atributo que já não está em falta (*i.e.*, tuplo identificado com o *ID* 2). Suponha-se que ambos os valores não constituem violações de sintaxe. Por fim, é executada a OD de violação de domínio. Esta operação considera não só o valor do atributo no qual anteriormente foi detectada a violação de domínio (*i.e.*, tuplo identificado com o *ID* 6), mas também os valores do atributo que não constituem violação de sintaxe (*i.e.*, tuplos identificados com os *ID* 2 e 4). Assim, como não foram detectados problemas, este processo de detecção – correcção dos PQD finda. No caso de ter sido detectado algum problema aquando desta segunda execução da sequência de OD, seria necessário efectuar as necessárias correcções e proceder a mais uma iteração de execução da sequência de OD.

No exemplo apresentado, aquando da segunda iteração, todas as OD que fazem parte da sequência foram novamente executadas. No entanto, nem sempre assim acontece. A sequência de OD pode ser executada parcialmente, *i.e.*, somente a partir de um determinado ponto em diante. Suponha-se que no exemplo anterior, aquando da primeira iteração das OD, não tinha sido detectada a existência de um valor em falta no atributo *CodigoPostal*. Assim, na segunda iteração das OD, a sequência seria executada a partir da violação de sintaxe (*i.e.*, detecção de violação de sintaxe e detecção de violação de domínio).

O exemplo apresentado ilustra apenas três OD. No entanto, todas as operações que visam a detecção de problemas deste NG obedecem ao processo de execução descrito. Recapitulando, a detecção de um PQD no valor de um atributo num tuplo, não impede a continuação da execução das OD que incidem sobre o mesmo atributo nos demais tuplos da relação. Por outro lado, apenas os valores afectados por PQD voltam a ser alvo da execução das OD aquando da iteração seguinte. Todos os outros valores do atributo em que não sejam identificados problemas, não voltam a ser objecto da execução das OD.

7.4.1.2 Atributo – Contexto Multi-Valor

No contexto multi-valor do atributo, uma sequência de dependências pode envolver OD de: existência de sinónimos; violação de unicidade; e, violação de restrição de integridade. Tal como exposto na Secção 7.3.2.2 estas OD são executadas, precisamente, segundo esta ordem. Neste NG, o processo de execução das OD que fazem parte da sequência envolve duas fases distintas. A primeira fase envolve a execução das duas primeiras OD da sequência (*i.e.*, existência de sinónimos e violação de unicidade). A segunda fase envolve a execução da OD que resta (*i.e.*, violação de restrição de integridade). Enquanto todos os PQD resultantes da execução das OD

pertencentes à primeira fase não estiverem solucionados, não se prossegue para a execução das OD referentes à segunda fase. Por sua vez, enquanto não forem solucionados todos os PQD que resultam da execução das OD de violação de restrição de integridade, o processo não é dado como concluído. A Figura 7.10 ilustra graficamente este processo de execução das OD. A linha a tracejado representa a separação entre as duas fases de execução. As iterações que podem ocorrer em cada fase, também se encontram representadas na figura.

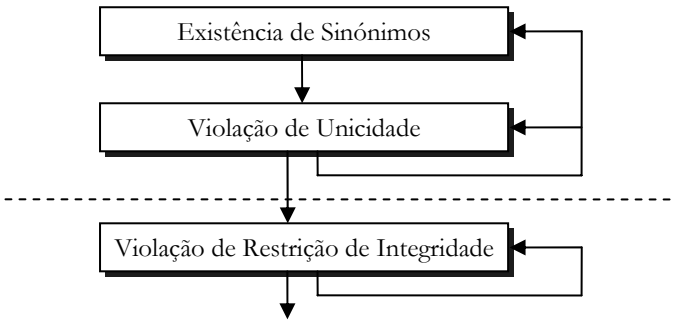


Figura 7.10 – Processo de execução das OD no contexto multi-valor do atributo

As OD de existência de sinónimos e violação de unicidade, ainda que em sequência (*i.e.*, primeiro a detecção de existência de sinónimos e depois a detecção de violação de unicidade) são executadas conjuntamente (*i.e.*, na mesma fase de execução). A execução da OD de violação de unicidade não obriga a que os sinónimos detectados sejam previamente corrigidos. Naturalmente, os tuplos nos quais sejam identificados sinónimos são excluídos da execução da operação seguinte de detecção de violação de unicidade. Nesta operação apenas se consideram os valores do atributo que não constituem sinónimos, uma vez que apenas esses são susceptíveis de serem iguais. A execução conjunta de ambas as operações permite reportar, de uma só vez, ambos os tipos de problemas ao utilizador. Na iteração seguinte de execução da sequência de operações, os tuplos excluídos da detecção de violação de unicidade por estarem envolvidos no problema de sinónimos são, então, considerados. Tendo como objectivo clarificar a abordagem que acabou de ser descrita, considere-se o exemplo que a seguir se apresenta. A Tabela 7.3 exibe o conteúdo da relação utilizada no exemplo. Suponha-se que o utilizador especificou para execução apenas as duas OD em causa (*i.e.*, detecção da existência de sinónimos e detecção de violação de unicidade). Da execução da primeira operação resulta a identificação da existência de sinónimos nos tuplos identificados com o *ID* 3 e 6 (*i.e.*, canalizador e picheleiro). Estes dois tuplos ficam, desde logo, excluídos da OD de violação de unicidade. Da execução desta operação, resulta a identificação de uma violação de unicidade nos tuplos identificados com o ID 2 e 7 (*i.e.*, *Pintor*). Só após terem sido executadas estas duas operações é que os PQD identificados são reportados ao utilizador.

Tabela 7.3 – Relação para exemplificação da execução das OD no contexto multi-valor do atributo

ID	Profissão
1	Electricista
2	Pintor
3	Canalizador
4	Médico
5	Professor
6	Picheleiro
7	Pintor

Naturalmente, a correcção que faria mais sentido para o problema dos sinónimos consistiria na eliminação de um dos tuplos. Suponha-se, por hipótese, que o utilizador altera o valor da profissão no tuplo identificado com o *ID* 6 para *Canalizador*. Desta forma, soluciona o problema da existência de sinónimos, mas introduz involuntariamente uma violação de unicidade no atributo. Quanto à violação de unicidade detectada, suponha-se que o valor da profissão no tuplo identificado com o *ID* 2 passa a ser *Pintor Automóvel* e no tuplo identificado com o *ID* 7 passa a ser *Pintor de Construção Civil*.

Após a realização das correcções, a sequência de execução de OD reinicia-se. No contexto do valor individual do atributo, neste reinício apenas são considerados os tuplos nos quais tinham sido detectados PQD. Ao invés, neste contexto, todos os valores do atributo (*i.e.*, de todos os tuplos) são novamente considerados. Se assim não fosse, não seriam detectados outros PQD, inadvertidamente introduzidos (*e.g.*: a resolução de um problema de sinónimos entre dois valores pode reintroduzir outro problema de sinónimos entre um destes valores e um outro valor de outro tuplo da relação).

No caso do exemplo apresentado, o reinício da execução das OD ocorre com a detecção da existência de sinónimos. Face às correcções efectuadas pelo utilizador, não é detectado qualquer problema do tipo existência de sinónimos. Quando isto acontece e à semelhança do que já foi explicado no contexto do valor individual do atributo, a OD em causa não voltará a ser executada em eventuais futuras iterações de execução. A execução da operação seguinte da sequência detecta a existência da já referida violação de unicidade, como consequência das correcções efectuadas (*i.e.*, *canalizador* encontra-se nos tuplos com *ID* 3 e 6). Após a correcção deste

problema pelo utilizador, uma nova iteração da sequência de operações realizar-se-á, mas envolvendo apenas a detecção de violação de unicidade.

Para ilustrar a forma como as duas operações são executadas, o exemplo apresentado foi manipulado em conformidade. Numa situação real, dificilmente o utilizador corrigiria um problema de sinónimos com a repetição do mesmo valor (no caso, *Canalizador*) em dois tuplos diferentes. Esta inusitada correcção introduziu intencionalmente uma violação de unicidade no atributo, meramente para ilustrar a forma de execução das operações. Não fosse esta correcção e ambos os problemas identificados, numa iteração de execução das OD, teriam sido solucionados em simultâneo pelo utilizador.

No caso da OD de violação de restrição de integridade, esta só é executada após os eventuais PQD de sinónimos e de violação de unicidade terem sido solucionados. Se assim não fosse, na detecção das violações de restrição de integridade podiam ser identificados problemas que não eram mais do que um mero reflexo dos dois anteriores. Para que tal não aconteça, a DC desses problemas tem de preceder a detecção das violações de restrição de integridade. Sempre que há uma nova iteração de execução de uma OD de violação de restrição de integridade, todos os valores do atributo são considerados novamente. A razão é a mesma do que a já mencionada no penúltimo parágrafo. No caso da sequência conter mais do que uma operação de violação de restrição de integridade, só se passa à execução da OD seguinte quando da execução da operação actual não resultar a detecção de qualquer problema. Por outras palavras, todos os problemas identificados por uma OD de violação de restrição de integridade têm de ser solucionados antes de se passar à detecção da seguinte. Como pode haver dependência entre as OD de violação de restrição de integridade, a abordagem adequada consiste em garantir que todos os PQD identificados pela execução de uma operação são solucionados, antes de se passar à execução da OD referente à restrição de integridade seguinte.

7.4.1.3 Tuplo

No NG do tuplo há apenas um único tipo de PQD: violação de restrição de integridade. Como tal, o processo de execução que a seguir se descreve, corresponde à situação em que existe mais do que uma OD de violação de restrição de integridade. Uma sequência de OD deste género obedece ao processo de execução já descrito no contexto do valor individual do atributo. Assim, a identificação de uma violação a uma restrição de integridade não impede a continuação da execução das restantes OD (de violação de restrição de integridade), de acordo com a ordem estabelecida pelo utilizador. A sequência de OD apenas é interrompida no tuplo em questão,

ficando pendente até que o problema em causa seja solucionado. Nos tuplos onde não tenha sido detectado o problema, não se justifica a sua interrupção. Nestes tuplos, a execução das operações pode continuar com toda a normalidade, uma vez que os outros tuplos que se encontram afectados com PQD não interferem na execução das OD que incidem sobre estes. Suponha-se que uma determinada sequência é composta por duas OD de violação de restrição de integridade. Se num dado tuplo da relação for detectada uma violação à primeira restrição de integridade, a segunda OD já não é executada nesse tuplo, ficando pendente até à correcção do problema. No entanto, atendendo a que em todos os outros tuplos da relação não foi detectada qualquer violação à primeira restrição de integridade, não há impedimentos à execução da segunda operação.

Após a realização das correcções às violações às restrições de integridade, a execução das OD reinicia-se, mas considera apenas os tuplos onde anteriormente tinham sido detectados PQD. Todos os outros tuplos nos quais não se detectou a existência de problemas, aquando da iteração anterior, são excluídos da execução das OD nesta nova iteração. As correcções entretanto efectuadas ao tuplos com PQD não os afectam, daí que não se justifique voltarem a ser submetidos às OD. Ao submeter novamente os tuplos nos quais tinham sido identificados PQD às OD, pretende-se garantir que estes foram efectivamente solucionados com as correcções efectuadas. Na execução de cada OD são considerados, além dos tuplos onde anteriormente tinham sido identificados os problemas em causa, também os tuplos em que a execução das operações precedentes da sequência já não detectou a existência de PQD. Esta abordagem permite a continuação da execução da sequência de OD nos tuplos onde esta tinha ficado pendente. A Figura 7.11 representa esquematicamente o processo de execução das OD que acabou de ser descrito. As diversas iterações que eventualmente possam ser necessárias até à resolução de todos os PQD, também se encontram representadas na figura.

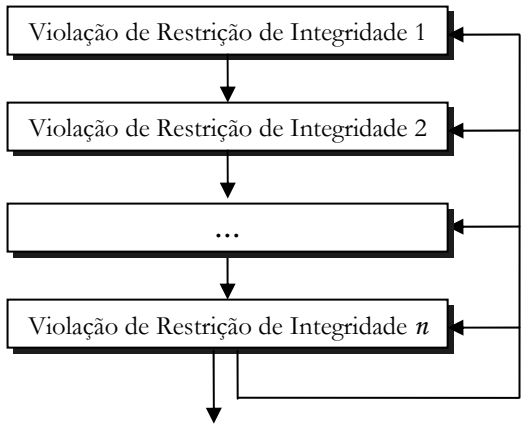


Figura 7.11 – Processo de execução das OD no contexto do NG do tuplo

Com as devidas adaptações, este processo é exactamente o mesmo do que o exemplificado, passo a passo, na Secção 7.4.1.1.

7.4.1.4 Relação

Face ao exposto na Secção 7.3.2.4, as dependências entre as OD permitem que se estabeleçam as seguintes sequências de execução: (*i*) violação de dependência funcional; tuplos duplicados; e, violação de restrição de integridade; ou, (*ii*) circularidade entre tuplos num auto-relacionamento; tuplos duplicados; e, violação de restrição de integridade. Perante os PQD que possam ser detectados por uma OD, neste NG não há qualquer outra operação subsequente da sequência que possa ser executada. Contrariamente ao sucedido nos NG anteriores, neste nível não podem ficar PQD pendentes e continuar-se com a execução das restantes OD da sequência. A execução de qualquer OD obriga a que todos os PQD respeitantes às operações anteriores da sequência tenham sido corrigidos. Como há dependências entre os valores, poderiam não ser identificados certos problemas ou erradamente serem reportados outros. Na Figura 7.12 encontra-se esquematizado graficamente o processo respectivo de execução das OD. As linhas a tracejado ilustram a separação existentes entre as três fases de execução das OD.

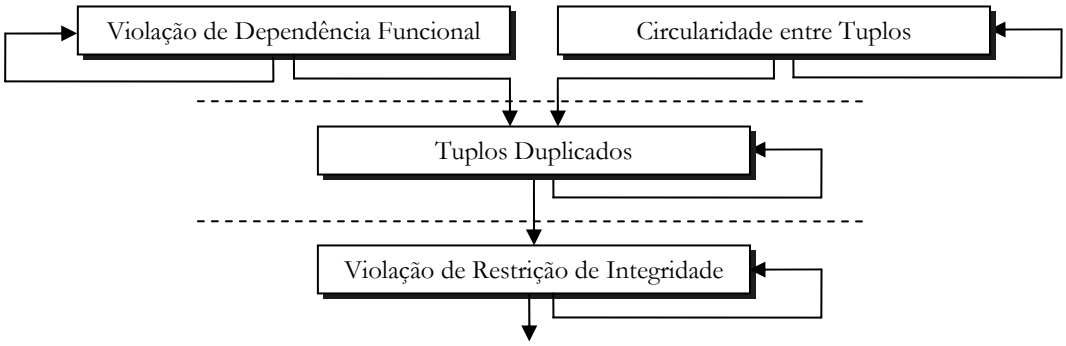


Figura 7.12 – Processo de execução das OD no NG da relação

Sempre que são efectuadas correcções no seguimento dos PQD detectados, a OD respectiva é executada novamente. Na figura anterior, isto corresponde às iterações representadas em cada tipo de PQD. Independentemente dos tuplos onde tenham sido detectados PQD, na nova iteração de execução da OD, todos os tuplos têm de ser novamente considerados. A realização de uma correcção pode introduzir inadvertidamente um outro PQD do mesmo tipo (*e.g.*: a correcção de uma violação de dependência funcional pode introduzir uma outra violação de dependência funcional). Só tendo em conta a totalidade dos tuplos existentes é possível garantir que, de facto, não existem problemas do tipo em questão.

No caso da sequência envolver mais do que uma OD de um dado tipo (*e.g.*: violação de dependência funcional; violação de restrição de integridade), estas são executadas de acordo com a ordem de especificação do utilizador. Uma determinada OD (de um dado tipo) apenas é executada quando da execução da operação anterior (do mesmo tipo) não resultar a identificação de problemas. Adopta-se esta abordagem em virtude de poderem existir dependências na execução das operações relativas ao mesmo tipo de problema. Desta forma, garante-se que todos os PQD resultantes da execução de uma OD se encontram solucionados, antes de se proceder à execução da operação seguinte.

7.4.1.5 Multi-Relação – Mono-Fonte / Multi-Fonte de Dados

Neste NG há apenas duas operações cuja execução é possível mesmo perante a existência de PQD, resultantes de OD prévias da sequência de dependências apresentada na Secção 7.3.2.5. As operações em causa são: existência de homónimos e heterogeneidade de unidades de medida.

No caso da OD de existência de homónimos, esta é efectuada em conjunto com a de existência de sinónimos. Em primeiro lugar é executada esta, imediatamente seguida da OD de existência de homónimos. A execução conjunta destas duas operações é possível, uma vez que os PQD detectados pela primeira não impedem a segunda de ser executada de forma fidedigna (*i.e.*, não serem detectados os homónimos existentes ou serem identificados homónimos que apenas são um mero reflexo dos sinónimos existentes). O facto destas OD serem efectuadas em conjunto permite a introdução da "optimização" já mencionada na Secção 7.3.2.5. Os pares de valores identificados como sinónimos ficam, desde logo, excluídos da execução da OD de homónimos.

No caso da OD de heterogeneidade de unidades de medida, esta é efectuada conjuntamente com a OD de tuplos duplicados. Em primeiro lugar é executada esta, imediatamente seguida da OD de heterogeneidade de unidades de medida. Esta operação tira partido dos resultados produzidos pela OD de tuplos duplicados para verificar se existe ou não heterogeneidade de unidades de medida entre os valores dos atributos. A execução da OD de heterogeneidade de unidades de medida encontra-se, inclusive, dependente da existência da OD de tuplos duplicados. Como tal, estas duas operações são executadas em conjunto.

Em todas as outras OD não é possível proceder à sua execução enquanto existirem PQD resultantes de uma operação prévia da sequência de dependências. A execução destas operações apenas é efectuada quando todos os PQD referentes às operações anteriores tenham sido solucionados. À semelhança do NG da relação, neste nível há dependências de execução entre as OD que obrigam a que cada problema seja manipulado isoladamente, antes de se avançar para o seguinte. Apenas adoptando esta abordagem é que se consegue detectar todos os PQD existentes e evitar a detecção de "falsos" problemas, *i.e.*, os que surgem como um mero reflexo de outros. A Figura 7.13 esquematiza graficamente o processo de execução das OD que fazem parte das sequências de dependências. As linhas a tracejado representam o requisito de todos os PQD do tipo anterior estarem solucionados, antes de se prosseguir com a detecção dos problemas do tipo seguinte da sequência.

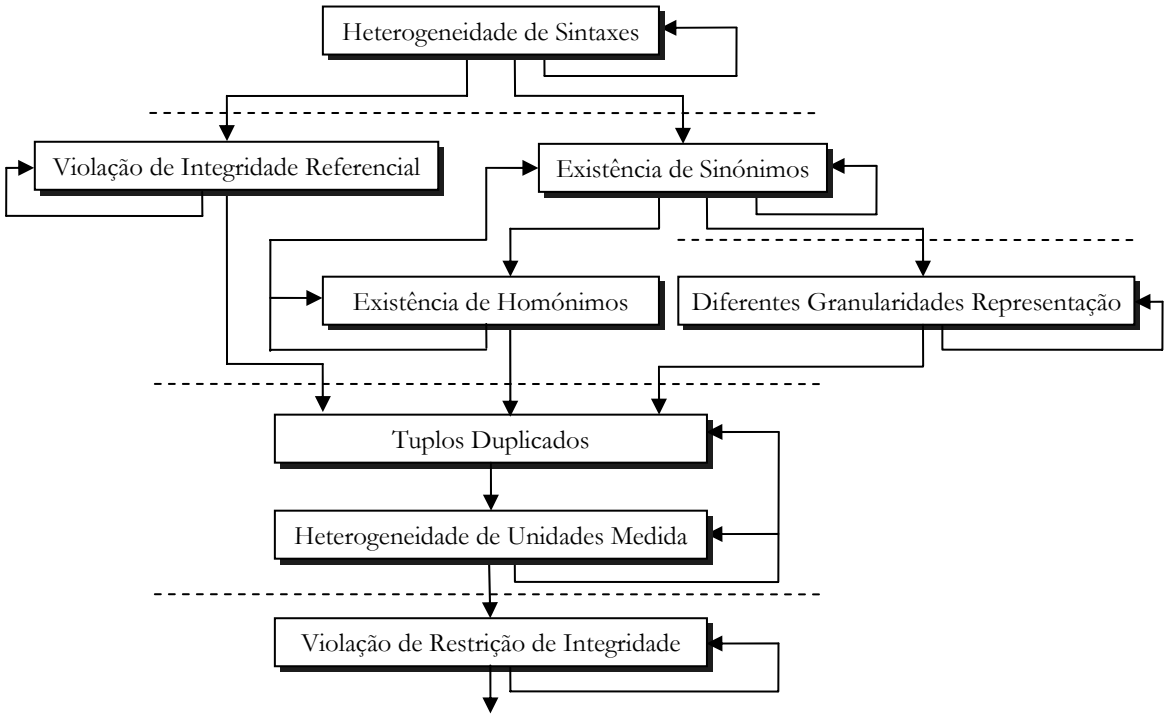


Figura 7.13 – Processo de execução das OD no NG multi-relação da fonte/multi-fonte de dados

Após terem sido realizadas correcções aos PQD, a operação responsável pela sua detecção volta a ser executada. As iterações representadas na figura anterior representam esta situação. No caso da violação de integridade referencial, apenas são considerados os tuplos das relações envolvidas nos quais o problema tinha sido detectado. As correcções efectuadas não são passíveis de introduzir este tipo de problema nos outros tuplos, nos quais já se confirmou que este não existe (na execução anterior da OD). Em todos os outros tipos de problemas é considerada novamente a totalidade dos tuplos das relações envolvidas, independentemente de onde tinham sido

detectados os PQD. As correcções efectuadas podem inadvertidamente ter introduzido novos problemas do tipo em questão entre os tuplos das relações (*e.g.*: a correcção de um problema de sinónimos pode introduzir um novo problema de sinónimos). Somente considerando a globalidade dos tuplos é possível assegurar que, de facto, não existem problemas do tipo em questão.

Caso a sequência de dependências envolva mais do que uma OD de um dado tipo (*e.g.*: violação de restrição de integridade; heterogeneidade de unidades de medida), a execução destas é feita obedecendo à ordem pela qual foram especificadas pelo utilizador. Uma OD de um dado tipo apenas é executada quando, da execução da operação anterior do mesmo tipo, não resultar a identificação de qualquer PQD. O motivo pelo qual se adoptou esta abordagem encontra-se explicado no último parágrafo da secção anterior.

7.4.2 Execução por Nível de Granularidade

A execução das OD obedece à sequenciação por NG apresentada na Secção 7.3.1. Assim, as primeiras operações a serem executadas referem-se ao NG do atributo, enquanto que as últimas dizem respeito ao NG multi-relação de diferentes fontes de dados. O início da execução das OD relativas ao NG seguinte só ocorre quando todos os PQD do nível actual estiverem solucionados (*i.e.*, quando as OD não detectarem a existência de problemas). Ainda que a execução das OD decorra de acordo com a sequência preconizada, podem ser necessárias múltiplas iterações que obrigam a que esta se reinicie novamente. Este reinício ocorre no nível mais elementar associado às operações especificadas pelo utilizador (normalmente, o NG do atributo). Para que este aconteça é suficiente que da execução de uma determinada OD, respeitante a um qualquer NG, resulte a identificação de um PQD. Após a realização das correcções aos PQD detectados, a abordagem descrita na secção anterior por NG envolve executar, novamente, as OD respectivas para confirmar se os problemas foram, efectivamente, solucionados. No entanto, ainda que não seja muito comum, a correcção de um PQD pode introduzir outro, cuja detecção já tenha sido dada como concluída. Na realidade, a correcção de um problema num NG pode introduzir involuntariamente outro PQD noutro nível que o precede.

Com o intuito de ilustrar a possibilidade que acabou de ser referida, considere-se o exemplo seguinte. Suponha-se que o utilizador especificou a execução de duas OD, referentes a NG diferentes: erro ortográfico (nível do atributo) e violação de dependência funcional (nível da relação). Ambas as operações possuem um atributo em comum (*e.g.*: *localidade*). Face à sequência preconizada de execução das OD por NG, começa-se pela que diz respeito ao atributo (*i.e.*,

detecção de erro ortográfico). Da execução desta operação não resulta a detecção de qualquer problema. Como não existem problemas no nível do atributo, prossegue-se para a detecção dos PQD do NG seguinte (*i.e.*, relação). A execução desta operação culmina na detecção de uma violação de dependência funcional, resultante da existência de localidades diferentes para o mesmo valor de código postal. Suponha-se que o utilizador procede à correcção deste problema, substituindo uma localidade por outra. Quando se reinicia a execução da OD, confirma-se que a violação de dependência funcional foi solucionada. No entanto, a correcção efectuada acabou por introduzir um erro ortográfico no atributo *localidade* que anteriormente não existia. Face ao exposto, ao invés de se executar unicamente as operações do NG em questão, justifica-se a execução das OD referentes aos níveis que o antecedem na sequência (*i.e.*, que se encontram a montante). Na realidade, para se garantir que uma determinada correcção de um tipo de problema (*e.g.*: violação de dependência funcional) não introduz um outro tipo de problema (*e.g.*: erro ortográfico) num NG a montante, é necessário executar novamente todas as OD relativas a esses níveis.

Face ao volume de dados que pode estar envolvido, a adopção de uma abordagem tão exaustiva poderia tornar impraticável a realização da LD. Isto surge como consequência das inúmeras iterações de execução integral das sequências de OD, respeitantes a cada NG que poderia ser necessário realizar. Para contornar este problema, adoptou-se a abordagem exposta nas secções anteriores. No reinício da execução de cada sequência de OD considera-se unicamente as que anteriormente culminaram na identificação de PQD e que dizem respeito apenas ao NG em questão. O grau de probabilidade de uma correcção causar um novo problema é reduzido. Ainda assim, como essa possibilidade existe, não pode ser descurada. Assim, como forma de assegurar que as correcções efectuadas não redundam na introdução de novos problemas nos NG que se encontram a montante, complementou-se a abordagem com o que a seguir se descreve. Quando da execução das sequências de OD, respeitantes a um determinado NG, já não resultar a detecção de qualquer PQD, então reinicia-se a execução das sequências de detecção de cada NG que se encontra a montante. Naturalmente, esta execução é novamente efectuada de acordo com a sequência por NG defendida neste trabalho (*i.e.*, primeiro as OD referentes ao nível do atributo e, em último, as relativas a múltiplas relações de diferentes fontes de dados). Assim, em vez de se executar as OD referentes aos NG precedentes sempre que é feita uma correcção, estas são executadas apenas uma única vez, no final da execução das OD do NG em questão (*i.e.*, quando já não for detectada a existência de qualquer problema). Eventuais problemas que possam ter sido inadvertidamente introduzidos nos NG a montante, fruto das correcções efectuadas, são

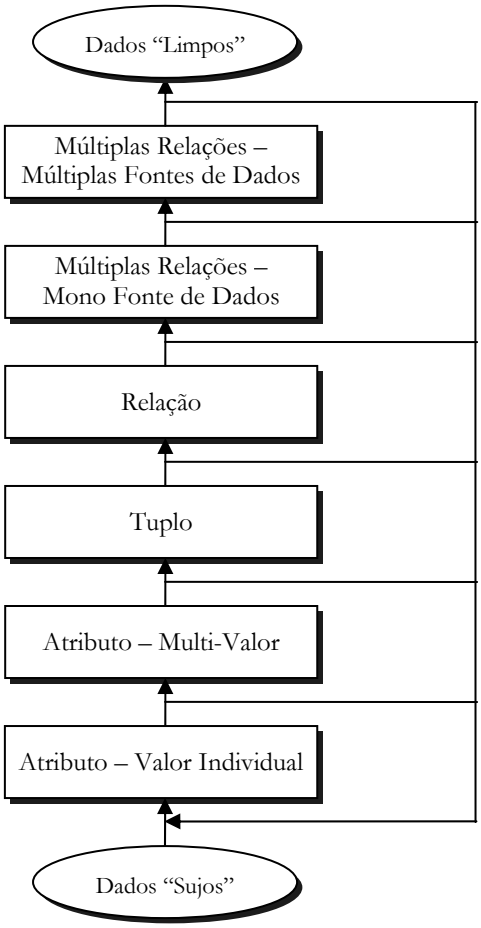


Figura 7.14 – Processo de execução das OD por NG

assim detectados. A Figura 7.14 ilustra graficamente o processo de execução descrito. Esta figura constitui uma reformulação à sequência de manipulação dos PQD apresentada na Figura 7.2.

Até ao momento, abordou-se a possibilidade de uma correcção de um problema num NG introduzir um outro PQD noutro nível. No limite, o mesmo pode acontecer no interior de cada NG, *i.e.*, a correcção de um problema pode causar um outro problema cuja detecção já tinha sido dada como concluída. Por exemplo, no contexto dos múltiplos valores do atributo (NG do atributo), a correcção de uma violação de unicidade pode redundar num problema de sinónimos que anteriormente não existia. Note-se que, de acordo com a sequência defendida, a execução da OD de sinónimos é efectuada antes da de violação de unicidade. Assim, quando se reinicia a execução das sequências de OD a partir do NG mais a montante, estas incluem as próprias sequências relativas ao NG que despoletou esse reinício. Por exemplo, caso o reinício de execução das sequências de OD nos NG precedentes seja despoletado pelas correcções realizadas ao nível da relação, as sequências referentes a este nível também são novamente executadas. Desta forma, qualquer problema que possa ter sido introduzido num NG por uma correcção efectuada a outro PQD do mesmo nível (a jusante na sequência de OD), acaba por ser detectado.

Este é o motivo que justifica que as sequências de OD referentes aos problemas no contexto do valor individual do atributo sejam novamente executadas, após a realização de correcções aos PQD identificados neste contexto. Na Figura 7.14 isto traduz-se graficamente na primeira iteração de execução das OD. Naturalmente que neste contexto apenas se justifica executar novamente as sequências de OD caso estas sejam compostas por mais do que uma operação.

Quando se reinicia a execução das sequências de OD no NG mais a montante, fruto das correcções efectuadas aos problemas noutro NG, os atributos e tuplo alterados condicionam essa execução. Em função destes, certas OD que integram cada sequência podem nem sequer ser alvo de execução. No caso de o serem, dependendo do NG a que pertencem, o seu âmbito de execução pode ficar restrito apenas aos tuplos que foram objecto de alterações. A Figura 7.15 representa, graficamente, uma alteração ao atributo a_2, do tuplo t_2, na relação R, efectuada para corrigir um PQD ao nível multi-relação da fonte de dados.

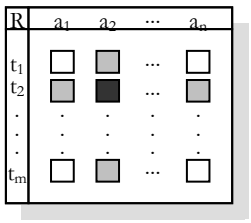


Figura 7.15 – Alteração ao valor do atributo a_2 no tuplo t_2

Assumindo que esta foi a única alteração efectuada à relação, o reinício da execução das sequências de OD nos NG do atributo (contexto do valor individual e contexto multi-valor), tuplo, relação e multi-relação da fonte de dados envolve unicamente as operações que incidem sobre o atributo a_2. Nos NG do atributo (apenas no contexto do valor individual) e do tuplo, a execução das OD ficam ainda mais limitadas a apenas aos valores dos atributos do tuplo t_2. Todas as outras OD que envolvem outros atributos da relação $\mathcal{R}$ não são executadas.

O comportamento a nível de execução das OD perante as alterações efectuadas, depende do NG a que estas pertencem. A apresentação deste comportamento, em cada NG, é efectuada nos parágrafos seguintes, acompanhada de um exemplo ilustrativo. Começa-se, precisamente, pela apresentação do exemplo. Suponha-se que foi detectado um problema de sinónimos nos valores de dois atributos, de dois tuplos, pertencentes a relações de fontes de dados diferentes. Este é um problema que pertence ao NG múltiplas relações de diferentes fontes de dados. O utilizador procede à correcção do problema numa das relações, igualando um dos valores ao outro valor. A Tabela 7.4 exibe o conteúdo da relação após a realização da correcção efectuada no atributo *Profissão*. O valor alterado pelo utilizador encontra-se assinalado a sombreado (*i.e.*, *Professor*).

Tabela 7.4 – Relação para exemplificação da repetição da execução das OD

ID	$atrib_2$	Profissão	…	$atrib_n$
1	xxx	Electricista	…	xxx
2	xxx	Mecânico	…	xxx
3	xxx	Canalizador	…	xxx
4	xxx	Médico	…	xxx
5	xxx	Professor	…	xxx
6	xxx	Advogado	…	xxx
7	xxx	Carpinteiro	…	xxx

Assuma-se que se encontram definidas OD para todos os outros NG. Assim sendo, as sequências de OD de PQD relativas a cada um desses níveis voltam a ser executadas, na procura dos eventuais problemas que possam resultar da correcção efectuada (*i.e.*, alteração de *Docente* para *Professor*). O reinício da execução destas sequências ocorre com as que dizem respeito ao NG do atributo, em particular, as que se referem ao contexto do seu valor individual.

No contexto do valor individual do atributo (NG do atributo), apenas se justifica repetir a execução das sequências de OD que envolvam atributos cujos valores tenham sido objecto de correcção. Além disto, o âmbito de execução fica ainda mais restringido a apenas aos tuplos cujos valores tenham sido alterados. Em todos os outros tuplos e em todos os outros atributos em que não se tenham verificados alterações, não se justifica repetir a execução das OD. No caso do exemplo apresentado, apenas é executada a sequência de OD relativa ao atributo *Profissão*, ficando a execução das operações que compõem esta sequência restrita ao valor que se encontra no tuplo identificado com o *ID* 5 (*i.e.*, *Professor*).

No contexto dos múltiplos valores do atributo (ainda no NG do atributo), apenas se justifica repetir a execução das sequências de OD que envolvam atributos cujos valores tenham sofrido alterações. As sequências de OD que envolvam atributos em que tais alterações não se tenham verificado, não voltam a ser executadas. No caso do exemplo, apenas é executada a sequência de OD que incide sobre o atributo *profissão*.

No NG do tuplo apenas se justifica repetir a execução das sequências de OD que envolvam, pelo menos, um atributo que tenha sido alvo de alterações. Tal como no contexto do valor individual

do atributo, o âmbito de execução fica ainda mais limitado a apenas aos tuplos cujos valores tenham sido alterados. Nos demais tuplos e em todos os atributos que tenham permanecido inalterados, não se procede à repetição da execução das OD relativas a este NG. No caso do exemplo apresentado, apenas é executada a sequência de OD que envolve o atributo *Profissão*, ficando limitada aos valores que se encontram no tuplo identificado com o *ID* 5.

No NG da relação, apenas há necessidade de executar as sequências de OD que envolvam um ou mais atributos cujos valores tenham sido objecto de correcções. As sequências de OD em que nenhum dos atributos envolvidos sofreu qualquer correcção, não voltam a ser executadas. No caso do exemplo, apenas é repetida a execução da sequência de OD que envolve o atributo *Profissão*.

Nos NG multi-relação de uma fonte de dados e multi-relação de diferentes fontes de dados verifica-se exactamente o mesmo do que no NG da relação. Assim, apenas há a necessidade de executar as sequências de OD que envolvam um ou mais atributos (independentemente da relação), cujos valores tenham sofrido alterações. Em todas as outras sequências em que nenhum dos atributos envolvidos tenha sido objecto de alterações, não se justifica uma nova execução. No caso específico do PQD violação de integridade referencial, o âmbito de execução da OD é ainda mais restrito, uma vez que considera apenas os tuplos cujos valores foram alterados. No caso do exemplo apresentado, apenas é executada novamente a sequência de OD que envolve o atributo *Profissão* da relação em causa.

O comportamento que acabou de ser descrito por NG, perante a necessidade de voltar a executar as OD desde o início (*i.e.*, desde o NG mais elementar), é similar ao que decorre do reinício das OD em cada NG e que tinha sido exposto na Secção 7.4.1. Importa, agora, apresentar o modo de operação deste processo de detecção que envolve reiniciar a execução das sequências de OD, quando se encontram envolvidos diferentes NG em simultâneo. Para melhor o ilustrar, considere-se o exemplo que a seguir se apresenta. Suponha-se que o utilizador especificou OD ao nível: do atributo (apenas no contexto do valor individual); do tuplo; da relação; e, multi-relação de uma fonte de dados. Ao nível do atributo e do tuplo não foi detectado qualquer PQD. No momento da execução da OD relativa ao NG da relação é detectado um PQD. Este problema é devidamente solucionado pelo utilizador. Há duas relações auxiliares que armazenam informação sobre os atributos e os tuplos que sofreram alterações com as correcções efectuadas. A estrutura destas relações será detalhada na secção seguinte. Por agora, é suficiente saber da sua existência e qual a sua finalidade. Ao executar-se novamente a OD respeitante ao NG da relação confirma-se que o problema anterior foi, de facto, solucionado. Uma vez que já não existem problemas ao

nível da relação, o processo de execução das sequências de OD reinicia-se no NG mais elementar (no caso, ao nível do atributo). O objectivo é garantir que as correcções efectuadas não introduziram inconscientemente outros problemas nos NG precedentes (*i.e.*, atributo e tuplo). O âmbito de execução das sequências de OD relativas ao NG do atributo fica restrito aos atributos e tuplos alterados pelas correcções efectuadas ao nível da relação. Suponha-se que da execução destas operações resulta a identificação de um novo PQD. O utilizador procede à correcção do problema identificado. Esta correcção também é registada nas relações auxiliares que armazenam as alterações efectuadas aos valores. Ao executar-se novamente a OD verifica-se que o problema foi efectivamente solucionado. Assumindo que a operação não é a primeira da sequência, o processo de detecção volta a reiniciar-se no NG do atributo. O objectivo é identificar potenciais problemas que possam ter sido introduzidos, apenas detectáveis pelas OD que se encontram a montante da actual operação. Nesta nova iteração de execução, apenas são considerados os valores alterados com as correcções efectuadas ao nível do atributo. Os valores alterados com as correcções efectuadas para suprir os problemas detectados ao nível da relação já foram verificados na iteração anterior de execução das OD deste NG (*i.e.*, atributo). Da nova iteração de execução das sequências de OD respeitantes ao nível do atributo, não resulta a identificação de qualquer problema. Assim, a detecção dos PQD prossegue com a execução das sequências de OD referentes ao NG do tuplo.

O âmbito de execução das OD referentes ao NG do tuplo fica restrito aos atributos e tuplos alterados pelas correcções efectuadas, inicialmente, ao nível da relação e, posteriormente, ao nível do atributo. Suponha-se que a execução destas operações redunda na identificação de novos PQD. O utilizador procede à correcção dos PQD detectados, sendo as alterações efectuadas a nível de atributos e tuplos armazenadas nas relações auxiliares respectivas. Da execução da sequência de OD constata-se a inexistência de problemas neste NG. Uma vez que já não existem problemas neste nível, o processo de detecção volta a reiniciar-se desde o nível do atributo. O objectivo é novamente garantir que as correcções efectuadas ao nível do tuplo não resultam em problemas ao nível do atributo. Na execução das OD que dizem respeito a este NG são apenas considerados os atributos e tuplos que sofreram alterações, fruto das correcções efectuadas no seguimento dos problemas detectados no NG do tuplo. Note-se que as correcções efectuadas ao nível da relação e, como consequência destas, as realizadas ao nível do atributo já foram objecto de execução das OD deste NG. Da execução das OD respeitantes ao NG do atributo não resulta a detecção de qualquer PQD. Assim, a detecção dos PQD prossegue com as operações relativas ao NG do tuplo. Uma vez que não foram detectados novos problemas ao nível do atributo, a execução das sequências de operações do NG do tuplo visa identificar os problemas que

eventualmente possam ter sido introduzidos pelas correcções efectuadas neste próprio nível, cuja detecção é efectuada pelas operações que se encontram a montante da que as desencadeou. Isto resulta da possibilidade de uma correcção no NG do tuplo poder introduzir um outro problema ao mesmo nível. Caso a sequência de OD no NG do tuplo fosse composta por uma única operação (assuma-se que não é o caso), não se justificava uma nova execução, uma vez que não foi efectuada qualquer correcção ao nível do atributo. A execução das OD respeitantes ao NG do tuplo não identifica qualquer problema. Assim sendo, a detecção dos PQD continua com a execução das operações referentes ao NG da relação.

Atendendo a que no NG da relação foi especificada uma única OD, a sua execução apenas é efectuada caso algum dos atributos que esta envolve, tenha sido alterado pelas correcções efectuadas nos NG do atributo e do tuplo. Estas alterações existem, tendo surgido como uma consequência das correcções efectuadas ao nível da relação que introduziram PQD ao nível do atributo e ao nível do tuplo. Da execução da OD, não resulta a identificação de qualquer problema. Caso a sequência neste NG fosse composta por mais do que uma operação, também os valores alterados no âmbito das correcções efectuadas aos problemas identificados neste nível (*i.e.*, relação), seriam incluídos na execução das OD. O objectivo consistiria em identificar os problemas que pudessem ter sido introduzidos a montante com as correcções efectuadas. No caso de todas as alterações efectuadas no contexto de um determinado NG surgirem como consequência da primeira OD da sequência, não se justifica a execução de qualquer operação nesse NG. A justificação encontra-se no facto das correcções efectuadas não serem susceptíveis de introduzirem outros problemas a montante, no próprio NG. No entanto, se os tuplos e atributos envolvidos nessa primeira OD da sequência tiverem sido objecto de alterações, fruto dos problemas encontrados por operações de outros NG, então já se justifica a sua execução, bem como das demais operações que façam parte da sequência do nível em questão.

Regressando ao exemplo, uma vez que não foi identificado qualquer problema ao nível da relação, a detecção avança para os PQD do NG seguinte (*i.e.*, múltiplas relações de uma fonte de dados). Assuma-se que da execução das sequências de OD relativas ao NG multi-relação de uma fonte de dados não resulta a detecção de qualquer problema. Uma vez que este constitui o último NG ao qual pertencem as operações especificadas pelo utilizador, o processo de detecção é dado por concluído. Neste momento, como já não se justifica a existência das relações auxiliares, criadas com a finalidade de armazenar as alterações efectuadas aos atributos e tuplos, estas são eliminadas. A Figura 7.16 ilustra as três iterações de execução das sequências de OD, ocorridas no âmbito do exemplo que acabou de ser descrito.

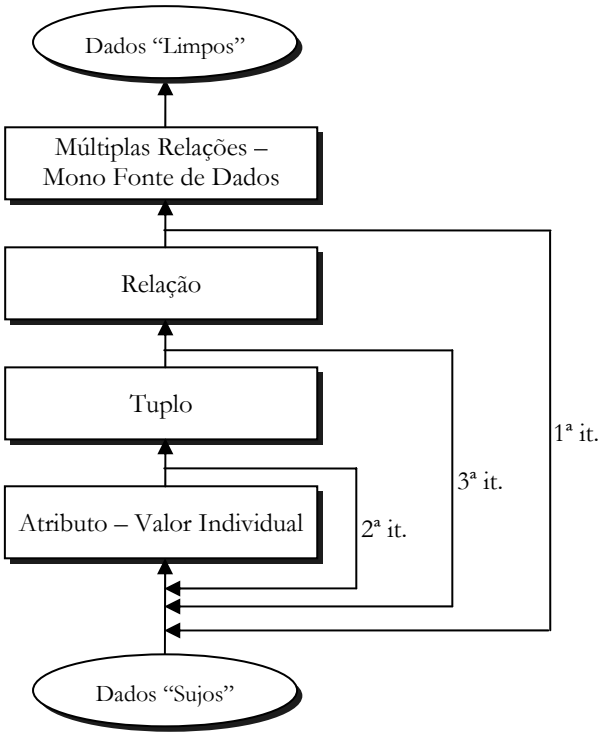


Figura 7.16 – Processo de execução das OD por NG

Só à custa deste método exaustivo e iterativo é possível assegurar que todos os PQD existentes nos dados foram, efectivamente, solucionados e que as correcções efectuadas não se traduziram involuntariamente na introdução de novos problemas. Face às OD de PQD especificadas pelo utilizador, é possível garantir que os dados se encontram, de facto, isentos desses problemas.

As formalizações das OD apresentadas no Capítulo 5 não reflectem estas iterações de execução que acabaram de ser descritas. As formalizações apenas reflectem a existência de iterações em cada NG e não entre NG diferentes. Entendeu-se que a apresentação das formalizações contemplando esta última situação ficaria excessivamente complexa. Em linhas muito gerais, pode dizer-se que o âmbito de execução das OD considera apenas os valores dos tuplos e atributos alterados pelas correcções entretanto efectuadas. Para o efeito, recorre-se a um mecanismo do tipo *join* entre as relações alvo de LD e as relações auxiliares que armazenam as alterações efectuadas.

7.5 Motor de Execução das Operações de Correcção

Como resultado da execução das sequências de OD podem ser reportados PQD ao utilizador. Cada problema pode ser solucionado por uma das seguintes vias: manualmente, *i.e.*, o utilizador efectua a correcção directamente na própria tabela recorrendo a instruções em *Structured Query*

Language (SQL); ou, automaticamente, *i.e.*, a correcção é efectuada com base numa OC especificada pelo utilizador. A semântica das OC que a arquitectura suporta, de base, encontra-se formalizada no capítulo anterior. Outras operações são susceptíveis de corrigir automaticamente os problemas detectados, tendo por base funções definidas pelo utilizador. No âmbito desta secção, apresenta-se a forma como este módulo da arquitectura procede à execução das OC especificadas pelo utilizador. Um outro aspecto que também é exposto nesta secção prende-se com o controlo das correcções (ou alterações) efectuadas aos valores. Esta informação é importante por dois motivos: (*i*) no reinicio de execução das sequências de OD é necessário saber-se quais as alterações realizadas na última iteração; e, (*ii*) suporta a anulação dos efeitos das OC, caso estas não produzam os resultados desejados.

7.5.1 Modo de Execução

O modelo de LD que se defende assenta na detecção e imediata correcção dos PQD. Como se viu na secção anterior, a execução das OD é efectuada de forma ascendente por NG. Como a correcção acompanha a detecção, a execução das OC também segue a mesma sequência gradual e ascendente por NG. No entanto, contrariamente ao que acontece nas OD em que a sua execução é feita de acordo com uma sequência predefinida em cada NG, tal não acontece nas OC. Confrontado com os PQD resultantes de uma iteração das sequências de OD respeitantes a um determinado NG, o utilizador especifica um conjunto de OC que visam a sua resolução. A ordem pela qual estas operações são executadas é perfeitamente irrelevante, uma vez que não há dependências na sua execução. Assim sendo, a sua execução é efectuada em paralelo. Uma excepção a esta regra ocorre quando existe mais do que uma OC para o mesmo tipo de PQD (*e.g.*: o utilizador especificou duas operações para a correcção do mesmo problema de violação de sintaxe). Nestes casos, pode fazer sentido que as operações sejam executadas segundo uma determinada ordem. Assim, quando o utilizador especifica mais do que uma OC para o mesmo PQD, independentemente de existir ou não dependência na execução destas, a sua execução é feita seguindo a ordem de especificação. Caso a OC Y dependa da execução prévia da OC X, compete ao utilizador colocar esta última antes daquela na especificação das OC a executar. Em todas as outras situações, não é relevante a ordem de execução das diversas OC. Com intuito de ilustrar esta independência de execução das OC, considere-se o exemplo a seguir apresentado. A Tabela 7.5 exibe o conteúdo da relação utilizada no exemplo. Nesta relação, o utilizador definiu as seguintes OD que incidem sobre o atributo *NomeFuncionário*: violação de sintaxe (*i.e.*, tem de ser composto por duas palavras separadas por um espaço); erro ortográfico; e, violação de domínio (*i.e.*, tem de pertencer ao conjunto de nomes de funcionários da empresa).

Tabela 7.5 – Relação para exemplificação da execução das OC

ID	$atrib_2$	NomeFuncionário	…	$atrib_n$
1	xxx	Justino Oliveira	…	xxx
2	xxx	Celeste Machado	…	xxx
3	xxx	Susana Loureiro	…	xxx
4	xxx	Ângelo Barrozo	…	xxx
5	xxx	Vítor Ramalho	…	xxx
6	xxx	Carvalho, António	…	xxx
7	xxx	Jorge Orlando	…	xxx

Da execução destas operações resultou a identificação de uma violação de sintaxe em "Carvalho, António", um erro ortográfico em "Ângelo Barrozo" e uma violação de domínio em "Celeste Machado". Confrontado com os problemas detectados, admita-se que o utilizador especificou três OC para a sua resolução. A OC de violação de sintaxe coloca o valor do atributo de acordo com a sintaxe pretendida. A OC de erro ortográfico substitui a palavra errada por aquela que lexicalmente mais próxima se encontra. A OC de violação de domínio recorre a uma função definida pelo utilizador que, socorrendo-se de outros atributos da relação e de uma relação auxiliar, actualiza o valor do atributo *NomeFuncionário*.

A ordem pela qual as três OC são executadas é absolutamente irrelevante. O facto da correcção de violação de sintaxe ser executada em primeiro, segundo ou terceiro lugar não tem qualquer implicação nas demais OC. O mesmo se passa com as outras OC. Isto acontece em virtude de não existir qualquer dependência de execução entre as OC. A existência desta resulta da própria independência de execução das operações que compõem a sequência de OD, o que permitiu que diferentes tipos de PQD fossem reportados de uma só vez. Como já referido, quando existe mais do que uma OC respeitante a um determinado tipo de problema, a ordem de execução das operações obedece à ordem de especificação por parte do utilizador. No caso específico da OC de heterogeneidade de unidades de medida, forçosamente esta tem de preceder a operação de remoção de tuplos duplicados. A existência de unidades de medida diferentes entre as duas relações constitui um obstáculo à consolidação dos *clusters* de tuplos duplicados existentes, num só tuplo. Só após ambos os atributos estarem sob uma unidade de medida comum faz sentido proceder à consolidação dos tuplos duplicados existentes. Assim, como há uma dependência

entre estas duas operações, são executadas sequencialmente. Em primeiro lugar, a OC de heterogeneidade de unidades de medida, seguida da operação de remoção de tuplos duplicados. À excepção destas, em todas as outras situações não há dependências de execução entre as OC, pelo que se optou por efectuar a sua execução em simultâneo. Desta forma, minimiza-se o tempo necessário à sua execução. Note-se que nos NG da relação e multi-relação (de uma ou de diferentes fontes de dados), na grande maioria das situações, a detecção de um PQD obriga à sua resolução antes de se avançar para a detecção do problema seguinte da sequência de OD. Naturalmente, nestas situações apenas há uma OC para execução.

7.5.2 Controlo das Correcções Efectuadas

Imediatamente antes da execução de qualquer OC é actualizada (ou criada caso não exista) uma relação temporária auxiliar que armazena o estado da relação. Estas relações são criadas em cada NG, sendo eliminadas quando se passa ao NG seguinte ou conclui a LD. Existe uma relação auxiliar para cada relação envolvida em OC. Como se verá mais adiante, uma relação auxiliar pode conter apenas os atributos da relação original envolvidos nas OC do NG em causa ou conter todos os atributos. Igualmente, a relação auxiliar pode replicar apenas os tuplos cujas OD do NG em questão identificaram PQD ou conter todos os tuplos da relação respectiva. A existência de relações auxiliares justifica-se por dois motivos: (*i*) identificar que correcções foram efectuadas às relações a que se encontram associadas; e, (*ii*) permitir anular as alterações efectuadas, resultantes da última OC efectuada.

De acordo com o exposto na secção anterior, pode ser necessário mais do que uma execução das sequências de OD para se poder assegurar que todos os PQD identificados estão, efectivamente, solucionados. A necessidade de repetir a execução das sequências de OD advém da possibilidade das correcções poderem introduzir outros PQD que, por se encontrarem a montante da operação actual (no mesmo ou noutro NG), já não são detectáveis. Assim, é necessário voltar a executar as sequências de OD, desde o NG mais elementar que corresponde às operações especificadas pelo utilizador. No entanto, em cada iteração, apenas são executadas as sequências de OD relativas aos atributos objecto de alterações na iteração anterior. Nos NG do atributo (apenas no contexto do valor individual) e do tuplo, a execução das operações fica ainda mais restrita a apenas aos tuplos que sofreram modificações na iteração anterior. Para tal, é necessário saber-se quais foram esses atributos e tuplos. O primeiro motivo que justifica a existência das referidas relações auxiliares é, precisamente, este. Por comparação entre a relação auxiliar (armazena o estado inicial) e a relação resultante das correcções efectuadas (contém o estado final), é possível identificar quais os tuplos

e atributos que sofreram alterações. No caso da relação resultante da OD, *i.e.*, a que armazena os PQD, conter a chave primária dos tuplos nos quais os problemas foram identificados, esta informação é usada na identificação das alterações. Caso contrário, é necessário efectuar uma comparação integral entre todos os tuplos de ambas as relações para identificar as alterações. Nem sempre uma correcção produz os resultados que se esperam. Contemplando esta possibilidade, é possível anular integralmente as alterações efectuadas. A anulação da correcção implica que se reponha o estado da relação antes da sua execução. Para que tal seja possível é necessário que o estado anterior esteja armazenado. Este constitui o segundo motivo que justifica a existência das relações auxiliares.

Actualmente, apenas é possível repor o estado anterior à realização das correcções efectuadas com o intuito de solucionar um determinado PQD. Não é possível repor qualquer outro estado anterior a este último. Por outro lado, a correcção de um dado PQD pode envolver mais do que uma OC, *i.e.*, uma sequência de operações. Perante a anulação das alterações efectuadas no âmbito da resolução de um PQD, são anulados os efeitos de todas as OC que compõem uma dada sequência. Por outras palavras, não é possível anular os efeitos a partir de um determinado ponto da sequência de OC. Actualmente, ao efectuar-se a anulação repõe-se o estado anterior à execução da primeira OC da sequência. As limitações enunciadas a nível de anulação dos efeitos das OC serão objecto de trabalho futuro.

Uma relação auxiliar é composta pelo conjunto de atributos envolvidos nas diversas OC que incidem sobre a relação respectiva. Comparativamente a esta, na relação auxiliar ficam excluídos os atributos que não são susceptíveis de sofrer alterações, *i.e.*, que não estão envolvidos em qualquer OC. Há, no entanto, uma situação que obriga a que a relação auxiliar contenha todos os atributos da relação a que se encontra associada. Para tal, basta que exista uma operação de remoção/eliminação no conjunto de operações que incidem sobre a relação. Quando isto acontece, é necessário armazenar os valores de todos os atributos na relação auxiliar para que, caso se mostre necessário, seja possível restaurar os tuplos eliminados. Além da relação auxiliar poder não ser uma réplica da relação a que se encontra associada na óptica do atributo, também pode não o ser na perspectiva do tuplo. Na relação auxiliar armazena-se apenas informação relativa aos tuplos que se encontram afectados por PQD, resultantes da execução das diversas OD que incidem sobre a relação em questão. Apenas os valores referentes a esses tuplos são susceptíveis de vir a ser objecto de alterações. No entanto, a execução de algumas OD (*e.g.*: violação de restrição de integridade) pode não produzir informação específica dos tuplos que se encontram afectados com o PQD. Quando isto acontece, a relação que armazena os PQD

detectados não contém informação sobre a chave primária dos tuplos respectivos. Isto significa que o problema existe, mas não se sabe exactamente qual o tuplo ou tuplos responsáveis (*e.g.*: o somatório de uma coluna que armazena valores percentuais não é igual a 100%). Nestas situações, na relação auxiliar armazena-se informação relativa a todos os tuplos da relação. Só assim se consegue identificar os valores alterados pelas OC e anular os seus efeitos, caso necessário. Ao procura-se que a relação auxiliar contenha apenas os atributos e tuplos indispensáveis ao controlo das alterações, há duas preocupações de minimização implícitas: (*i*) o espaço em disco necessário; (*ii*) o tempo necessário à sua criação. Note-se que não é só no contexto das correcções efectuadas por meios automáticos que se justifica a existência das relações auxiliares, mas também no contexto das correcções efectuadas manualmente pelo utilizador. Neste caso, assume uma importância ainda maior, visto ser o único meio de identificação das correcções efectuadas.

Além das relações auxiliares que armazenam o estado das relações antes da execução das OC, é necessário armazenar as correcções efectuadas. Como já se referiu, a identificação destas correcções resulta da comparação entre as relações auxiliares e as respectivas relações. O registo das correcções efectuadas é importante pelas seguintes razões: (*i*) quando todos os PQD se encontram solucionados num NG, o processo de detecção reinicia-se novamente desde o nível mais elementar, mas considerando unicamente os valores que entretanto sofreram correcções; (*ii*) é com base no conhecimento das alterações provocadas por uma dada OC que é possível anular os seus efeitos e assim restaurar a situação original. De referir que quando se anulam correções entretanto efectuadas, a informação armazenada sobre as correspondentes alterações é também eliminada.

Para suportar o armazenamento das alterações resultantes das diversas OC, desenvolveu-se o modelo de dados apresentado na Figura 7.17, sob a forma de um diagrama entidade – relação.

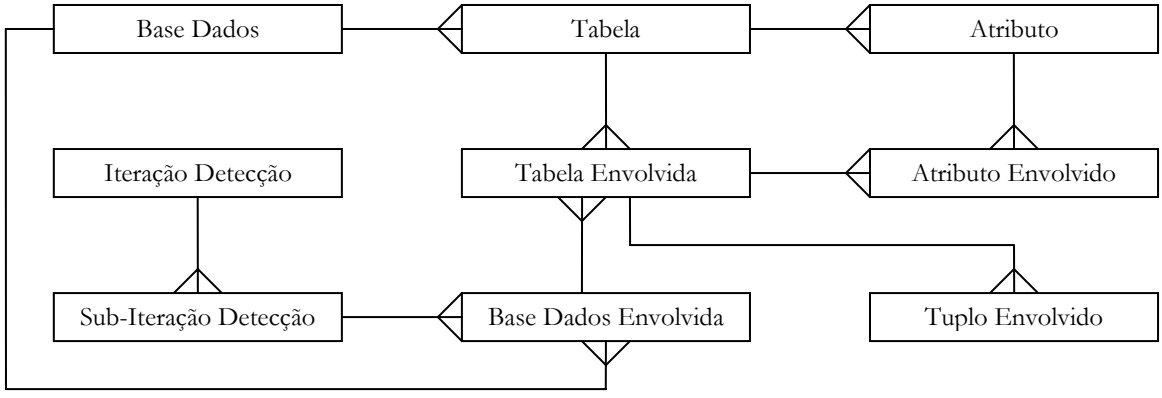


Figura 7.17 – Modelo de dados de suporte ao armazenamento das alterações efectuadas

A implementação do modelo de dados proposto traduz-se num conjunto de relações e no estabelecimento de relacionamentos chave primária – chave estrangeira entre estas. Nos parágrafos seguintes descreve-se a finalidade de cada relação, bem como o esquema de dados subjacente.

A relação *BaseDados* armazena o nome das BD envolvidas nas OC. Esta relação possui o seguinte esquema: *BaseDados*(*CódigoBD*, *Descritivo*). O atributo *CódigoBD* constitui a chave primária da relação, sendo um atributo de numeração automática que se inicia em 1. O atributo *Descritivo* contém o nome das BD. A relação *Tabela* armazena o nome das tabelas de cada BD envolvida em OC. Esta tabela possui o seguinte esquema *Tabela*(*CódigoBD*, *CódigoTabela*, *Descritivo*). O atributo *CódigoBD* constitui a chave estrangeira para a relação *BaseDados*. *CódigoTabela* constitui um número inteiro sequencial iniciado em 1 para cada *CódigoBD*. Em conjunto, estes dois atributos formam a chave primária desta relação. O atributo *Descritivo* contém o nome de cada relação que faz parte de uma BD. A relação *Atributo* armazena os nomes dos atributos das tabelas envolvidas em OC. Esta tabela possui o seguinte esquema *Atributo*(*CódigoBD*, *CódigoTabela*, *CódigoAtributo*, *Descritivo*). Os atributos *CódigoBD* e *CódigoTabela* formam, em conjunto, a chave estrangeira para a relação *Tabela*. *CódigoAtributo* constitui um número inteiro sequencial iniciado em 1 para cada combinação de *CódigoBD* e *CódigoTabela*. Em conjunto, estes três atributos formam a chave primária desta relação. O atributo *Descritivo* contém o nome de cada atributo que faz parte de uma relação.

A relação *IteraçãoDetecção* armazena as iterações de execução das sequências de OD. Quando todos os problemas detectados num NG se encontram solucionados, a execução das sequências de OD deve reiniciar-se novamente a partir do nível mais elementar. Sempre que isto acontece, está-se perante uma nova iteração de execução das OD. A relação *IteraçãoDetecção* possui o seguinte esquema: *IteraçãoDetecção*(*NrIteração*), em que *NrIteração* constitui um atributo de numeração automática representativo da iteração de detecção. Sendo o único atributo da relação, naturalmente constitui a sua chave primária.

A relação *SubIteraçãoDetecção* armazena as sub-iterações de execução das OD que ocorrem em cada iteração. Ao reiniciar a execução das sequências de OD podem ser detectados novos PQD que anteriormente não existiam. Quando estiverem solucionados obrigam novamente ao reinício de execução das OD. Como esta nova iteração deriva da iteração que ocorreu anteriormente designa-se de *sub-iteração*. O esquema da relação *SubIteraçãoDetecção* é o seguinte: *SubIteraçãoDetecção* (*NrIteração*, *NrSubIteração*). O atributo *NrIteração* constitui a chave estrangeira para a relação *IteraçãoDetecção*. *NrSubIteração* constitui um número inteiro sequencial iniciado em 1 para cada

NrIteração que representa a sub-iteração em cada iteração de detecção. Conjuntamente, estes dois atributos formam a chave primária desta relação.

A relação *BDEnvolvida* armazena informação sobre as BD envolvidas em cada OC. Esta relação possui o seguinte esquema: *BDEnvolvida*(*NrIteração*, *NrSubIteração*, *CódigoBD*). Os dois primeiros atributos, em conjunto, formam a chave estrangeira para a relação *SubIteraçãoDetecção*. O atributo *CódigoBD* constitui a chave estrangeira para a relação *BaseDados*. Em conjunto, os três atributos da relação formam a sua chave primária.

A relação *TabelaEnvolvida* armazena informação sobre as tabelas envolvidas em cada OC. O esquema desta relação é o que a seguir se apresenta: *TabelaEnvolvida*(*NrIteração*, *NrSubIteração*, *CódigoBD*, *CódigoTabela*). Em conjunto, os três primeiros atributos da relação constituem a chave estrangeira para a relação anterior. Os atributos *CódigoBD* e *CódigoTabela* formam, em conjunto, a chave estrangeira para a relação *Tabela*. A chave primária da relação é composta pelo conjunto formado pelos quatro atributos.

A relação *AtributoEnvolvido* armazena informação sobre os atributos envolvidos em cada OC. Esta relação possui o esquema que a seguir se apresenta: *AtributoEnvolvido*(*NrIteração*, *NrSubIteração*, *CódigoBD*, *CódigoTabela*, *CódigoAtributo*, *NrOperaçãoDetecção*). Os quatro primeiros atributos desta relação formam, em conjunto, a chave estrangeira para a relação *TabelaEnvolvida*. Os atributos *CódigoBD*, *CódigoTabela* e *CódigoAtributo* formam a chave estrangeira para a relação *Atributo*. Em conjunto, os cinco primeiros atributos formam a chave primária desta relação. Além destes, a relação contém ainda um outro atributo: *NrOperaçãoDetecção*. Este atributo armazena informação sobre o número da OD que despoletou a presente correcção ao atributo (a cada OD especificada pelo utilizador é atribuído um número inteiro sequencial entre 1 e ***n*** que serve de identificador). Esta informação é utilizada na reposição do estado anterior à realização das correcções efectuadas, no seguimento dos PQD identificados pelas OD.

Por fim, a relação *TuploEnvolvido* armazena informação sobre os tuplos que sofreram alterações como consequência da execução das OC. O esquema desta relação é o que a seguir se apresenta: *TuploEnvolvido*(*NrIteração*, *NrSubIteração*, *CódigoBD*, *CódigoTabela*, *NrSequencial*, *ChavePrimáriaTuplo*, *NrOperaçãoDetecção*, *TipoOperação*). Tal como na relação anterior, o conjunto dos quatro primeiros atributos desta relação formam a chave estrangeira para a relação *TabelaEnvolvida*. O atributo *NrSequencial* constitui um número inteiro sequencial iniciado em 1 para cada combinação dos atributos *NrIteração*, *NrSubIteração*, *CódigoBD* e *CódigoTabela*. O conjunto formado pelos cinco primeiros atributos constitui a chave primária desta relação. Além destes, há ainda outros três

atributos. O atributo *ChavePrimáriaTuplo* contém a chave primária do tuplo cujos valores foram objecto de alterações. Este atributo é do tipo textual, pelo que no caso da chave primária ser formada por mais do que um atributo, os valores respectivos são armazenados neste atributo, devidamente, separados pelos caracteres "||". No caso de todos os tuplos da relação terem sido alterados, neste atributo usa-se à mnemónica "*All*" para o representar. Assim, dispensa-se que o armazenamento dessa informação seja efectuado para cada tuplo. O atributo *NrOperaçãoDetecção* armazena informação sobre o número da OD subjacente à presente correcção aos valores do tuplo. Tal como na relação anterior, esta informação é utilizada na reposição do estado anterior à realização das correcções efectuadas. Por último, o atributo *TipoOperação* armazena o tipo de operação efectuada no tuplo, *i.e.*, correcção dos valores ou remoção/eliminação do próprio tuplo (*e.g.*: resultante de uma operação de remoção de sinónimos).

7.6 Diagrama de Sequência

Tendo sido descrita a arquitectura e expostos detalhadamente os principais módulos que a compõem, a descrição do modelo proposto de LD finda com a apresentação do diagrama de sequência respectivo, representado na Figura 7.18. Esta diagrama de sequência recapitula e sintetiza o modelo que se defende para a LD.

Como se pode observar no diagrama, o início da LD ocorre com a especificação de um conjunto de OD que visam identificar os PQD existentes nos diferentes NG. De seguida, as OD especificadas são sequenciadas por e em cada NG. Daqui resulta um conjunto de sequências de OD para cada NG, cuja execução em cada nível pode ser efectuada em simultâneo. As primeiras sequências de OD executadas dizem respeito ao NG mais elementar que corresponde às operações especificadas (normalmente, o NG do atributo – contexto do valor individual). No caso de serem identificados PQD pelas OD, estes podem ser solucionados pela via manual, pela via automática ou um misto das duas. A resolução manual implica que o utilizador solucione os PQD directamente na própria fonte de dados. Na resolução automática, o utilizador procede à especificação de um conjunto de OC que visam solucionar os PQD identificados. O passo seguinte envolve a execução destas OC. Independentemente da abordagem adoptada na resolução dos PQD, as OD responsáveis pela sua identificação voltam a ser executadas para certificar que os problemas foram, de facto, solucionados. A execução das sequências de OD prossegue a partir dos pontos em que estas tinham sido interrompidas para que se efectuassem as necessárias correcções. No caso de serem identificados novos PQD, repete-se todo o procedimento que acabou de ser descrito. Quando da execução de todas as sequências de OD

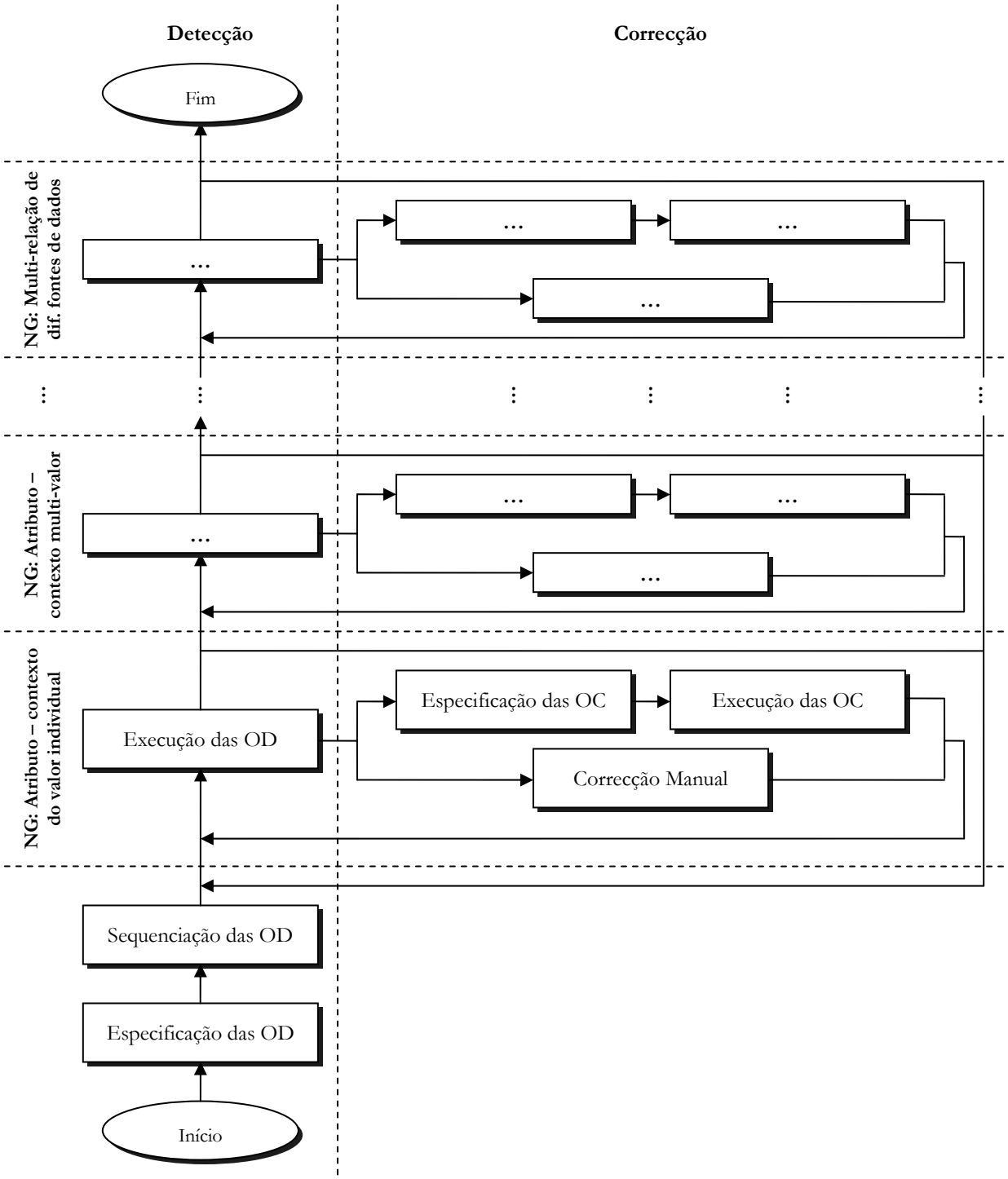


Figura 7.18 – Diagrama de sequência do modelo proposto de LD

referentes a um NG já não resultar a identificação de qualquer PQD, a execução das OD reinicia-se no NG mais elementar, na procura de novos problemas que possam ter sido involuntariamente introduzidos com as correcções efectuadas. Neste reinício, apenas são executadas as OD que incidem sobre atributos cujos valores tenham sido objecto de alteração com as correcções entretanto realizadas. No caso de ser detectado algum PQD, o procedimento atrás descrito volta a desenrolar-se. Caso contrário, a detecção dos PQD prossegue para o NG

seguinte. Neste novo NG repete-se tudo aquilo que até agora foi descrito. A LD termina quando da execução das OD referentes ao NG de maior complexidade que corresponde às operações especificadas (no limite, o NG multi-relação de diferentes fontes de dados), não resultar a identificação de qualquer PQD.

7.7 Protótipo Desenvolvido

Nesta secção apresentam-se, de forma sucinta, os aspectos considerados de maior relevância relacionados com o protótipo desenvolvido no âmbito deste trabalho de doutoramento. O protótipo, baptizado com a denominação *SmartClean*, materializa a arquitectura proposta e, consequentemente, o modelo que se defende para a LD, bem como as OD e OC formalizadas nos dois capítulos anteriores.

O *SmartClean* foi desenvolvido integralmente usando a linguagem de programação *Java*. Uma vez que da taxionomia elaborada no Capítulo 4 resultou uma organização hierárquica dos PQD por NG, a adopção de uma linguagem de programação que permitisse implementar mecanismos de herança e polimorfismo tornou-se um requisito. Estes mecanismos são típicos de linguagens que suportam programação orientada a objectos. Entre as várias hipóteses disponíveis, a linguagem *Java* foi aquela que evidenciou ter um melhor balanceamento entre facilidade de desenvolvimento e performance de execução. Face ao enorme volume de dados que o protótipo pode ter de manipular, as questões relacionadas com a performance são importantes. Por falar em dados, a linguagem *Java* também obedeceu a um outro requisito essencial, *i.e.*, fornecer um suporte adequado ao acesso e manipulação de dados armazenados em BD relacionais de tipos diferentes (*e.g.*: *SQL Server*; *Oracle*; *MySQL*). Este suporte é fornecido na linguagem *Java* por uma *API* designada de *JDBC*. Como se pôde verificar nas secções anteriores, defende-se que a execução de algumas OD e OC pode ser efectuada em paralelo. A linguagem *Java* também fornece um suporte adequado à execução simultânea de várias tarefas, por intermédio de um mecanismo simples de criação de *threads*. Um outro aspecto que abonou a favor da escolha do *Java* como linguagem de programação prende-se com o seu cariz multi-plataforma, o que permite a portabilidade do protótipo para diferentes sistemas operativos. O desenvolvimento do *SmartClean* traduziu-se em cerca de vinte e cinco mil linhas de código, distribuídas por mais de uma centena de classes. Sempre que possível, tirou-se partido das potencialidades intrínsecas oferecidas pela própria linguagem na implementação das OD e OC. A título de exemplo, no caso da correcção das violações de sintaxe, tirou-se partido das potencialidades de transformação baseadas em expressões regulares, oferecidas por uma *API* que acompanha a linguagem *Java* (no caso, *regex*).

As OD e OC especificadas para execução pelo utilizador começam por ser submetidas aos respectivos compiladores. Estes compiladores traduzem essas operações, representadas sob a forma declarativa, para uma representação interna usada nos demais módulos da arquitectura (caso não estejam afectadas por erros sintácticos). Esta representação interna não é mais do que código *Java* gerado dinamicamente, cuja execução resulta na DC dos PQD existentes.

Da arquitectura apresentada na Figura 7.1 faz parte uma biblioteca de funções de LD definidas pelo utilizador. Estas funções, cuja necessidade de desenvolvimento advém dos requisitos específicos de LD existentes em cada domínio concreto, também são implementadas em *Java*, no caso sob a forma de métodos estáticos e organizados em *packages*. Um problema que obriga frequentemente o utilizador a proceder ao desenvolvimento de funções é o das violações de restrição de integridade. Por natureza, este tipo de problema é específico de cada domínio de aplicação. Mesmo assim, nas violações de restrição de integridade que ocorrem nos diferentes NG foram identificados aspectos comuns, tendo estes sido formalizados nos dois capítulos anteriores. O que é comum nas violações de restrição de integridade em cada NG foi implementado num *template* de código *Java*. Assim, o utilizador apenas tem de implementar numa função aquilo que é específico e necessário à detecção ou correcção das violações à restrição de integridade em causa. Desta forma, procura-se minimizar o esforço de implementação a que o utilizador é sujeito.

Actualmente, a principal limitação do *SmartClean* é a inexistência de uma interface gráfica. Daí que não seja possível apresentar imagens que ilustrem o protótipo desenvolvido. No estado actual, a especificação das OD é efectuada num ficheiro de texto. Na Figura 7.19, do lado esquerdo, exemplifica-se a especificação de um conjunto de OD. Aquando da sua execução, os PQD identificados são colocados em tabelas de uma BD, especialmente criada para o efeito. Compete ao utilizador consultar estas tabelas e analisar os PQD identificados. Na Figura 7.20 ilustra-se as tabelas resultantes da execução de um conjunto de OD numa BD em *MySQL* e a respectiva consulta dos PQD. As correcções que o utilizador entenda efectuar podem ser efectuadas por uma de duas vias: (*i*) manualmente, recorrendo a comandos de SQL (*e.g.*: *update*; *delete*); ou (*ii*) automaticamente, especificando as OC que devem ser efectuadas. Tal como as OD, as de correcção também são especificadas pelo utilizador num ficheiro de texto. Na Figura 7.19, do lado direito, ilustra-se a especificação de um conjunto de OC que visam solucionar os PQD detectados no contexto do valor individual do atributo. Há a perfeita noção de que uma interface gráfica deve suportar todas estas operações, auxiliando o utilizador nas suas tarefas (*e.g.*: a especificação das OD ou OC deve ser assistida por um editor gráfico). No entanto, não só o

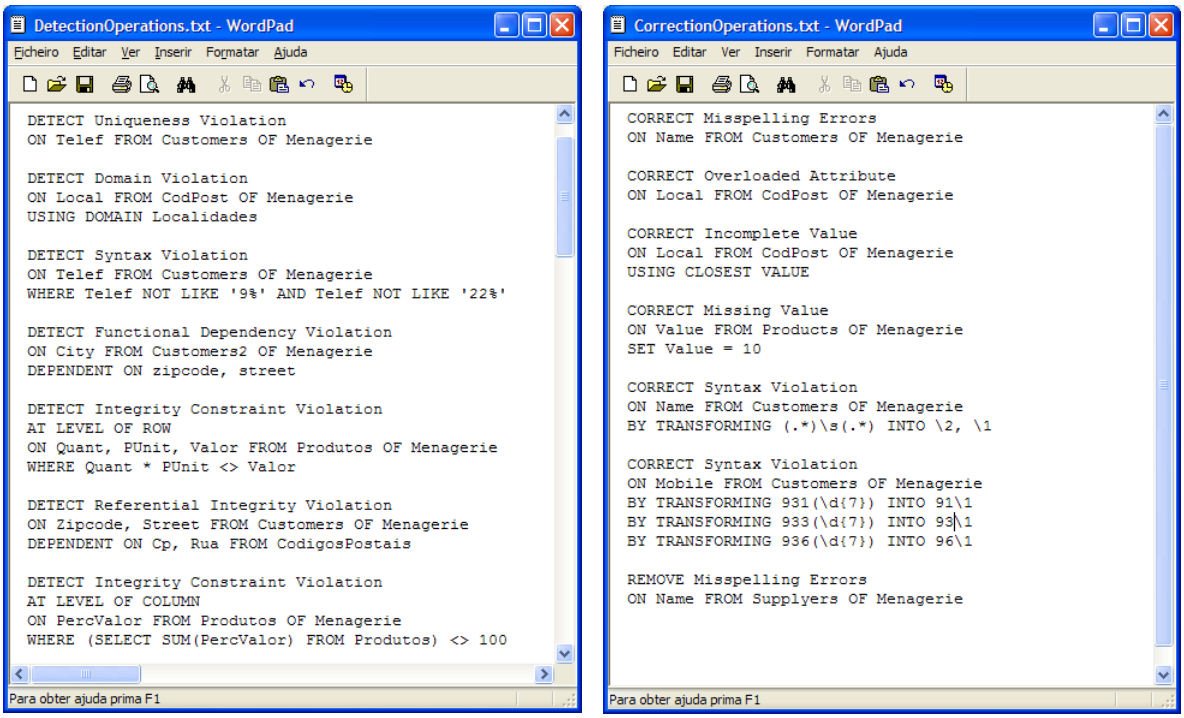


Figura 7.19 – Exemplificação da especificação de OD (lado esquerdo) e OC (lado direito)

trabalho de implementação em particular, como todo este trabalho de doutoramento no geral, esteve sujeito a restrições temporais. Perante estas restrições e no caso concreto da implementação, optou-se por dar primazia aquilo que foi considerado essencial na arquitectura proposta. Tudo o resto que não impede que se possa demonstrar a validade da arquitectura, não foi intencionalmente implementado. A interface gráfica foi o aspecto preterido, uma vez que o seu desenvolvimento é sempre moroso. O inconveniente é que a sua inexistência dificulta a utilização do protótipo. O incremento da sua usabilidade será objecto de trabalho no futuro próximo. Perspectiva-se que o desenvolvimento futuro da interface seja pacífico, uma vez que a camada gráfica a desenvolver (sob a forma de um conjunto de classes *Java*) assentará sobre a camada de processamento, entretanto já implementada. Desta forma, consegue-se uma clara separação entre as duas camadas, o que simplifica a manutenção de ambas.

Nos módulos de sequênciação das OD, execução incremental das OD e execução das OC centraram-se os principais esforços de implementação, uma vez que representam o núcleo da arquitectura proposta e, consequentemente, do modelo de LD subjacente. Estes módulos foram implementados seguindo fielmente o que foi exposto nas três secções anteriores, daí que não haja comentários a efectuar.

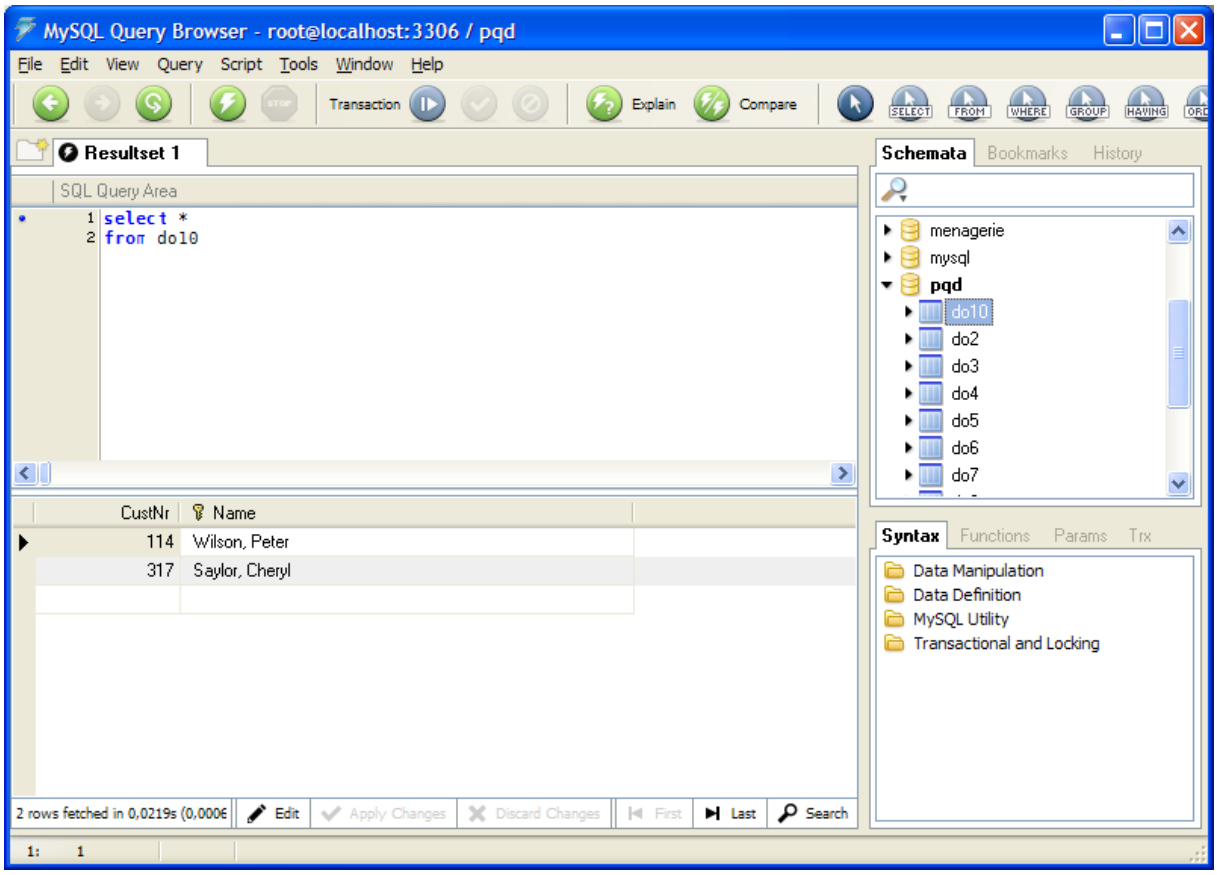


Figura 7.20 – Consulta dos PQD resultantes da execução das OD

7.8 Conclusão

Neste capítulo expôs-se, com o necessário pormenor, o modelo que se defende para a DC dos PQD. Esta exposição efectuou-se por intermédio da arquitectura proposta, desenvolvida de acordo com esse modelo de LD. A principal originalidade do modelo reside na sequência proposta de manipulação dos PQD. Esta sequência resultou das dependências de manipulação (*i.e.*, DC) identificadas entre os diversos tipos de PQD. O modelo baseia-se nos NG dos PQD que resultaram da taxionomia desenvolvida no âmbito deste trabalho de doutoramento. Assim, os problemas são manipulados gradualmente por NG, desde o mais elementar (*i.e.*, atributo) até ao de maior complexidade (*i.e.*, múltiplas relações de diferentes fontes de dados). A detecção de um PQD num NG encontra-se dependente da resolução prévia dos problemas que existem nos níveis anteriores. Se tal não acontecer, podem não ser detectados certos PQD ou erradamente serem reportados outros que não são mais do que um mero reflexo de problemas existentes noutro NG. Em cada NG, os problemas também são detectados e, de imediato, solucionados de acordo com uma sequência predefinida. A não observância desta sequência em cada NG possui, igualmente, as consequências já referidas (*i.e.*, não serem detectados certos problemas ou

redundantemente serem reportados outros). Pelo que se conhece, esta é a primeira vez que é proposta uma sequência de manipulação que cobre os diversos tipos de problemas de qualidade que podem afectar os dados.

ESTUDO DE CASO DE LIMPEZA DE DADOS 8

Neste capítulo procede-se à apresentação do estudo de caso de Limpeza de Dados (LD), efectuado com o intuito de testar e validar o modelo proposto e as operações de LD (*i.e.*, de Detecção e Correcção (DC)). Estes aspectos foram detalhados nos três capítulos anteriores e encontram-se materializados no protótipo desenvolvido, designado de *SmartClean*. É este sistema de LD que origina a realização do presente estudo de caso. O capítulo inicia-se com a apresentação do caso, baseado numa Base de Dados (BD) relativa a dádivas/colheitas de sangue. De seguida, são apresentadas sucessivamente as várias operações de LD efectuadas em cada Nível de Granularidade (NG), de acordo com a sequência de manipulação dos Problemas de Qualidade dos Dados (PQD) defendida no modelo proposto. Assim, começa-se por expor as operações efectuadas ao nível do atributo e termina-se com as que envolvem relações de BD diferentes.

8.1 Introdução

Nos Capítulos 5 e 6 apresentaram-se as formalizações, respectivamente, das Operações de Detecção (OD) e Operações de Correcção (OC) dos diversos tipos de PQD. No capítulo anterior apresentou-se o modelo proposto para a LD que articula a execução de ambas as operações. Estes três capítulos materializaram-se num protótipo, baptizado com a designação *SmartClean*. O presente capítulo surge na sequência natural dos três anteriores. Após as formalizações efectuadas, após a sua concretização numa ferramenta informática, apenas faltava demonstrar o interesse e validade de todo o trabalho desenvolvido numa situação real de LD. O objectivo deste capítulo é, justamente, esse.

O caso eleito para testar o interesse e validade do *SmartClean* e do modelo de LD que se encontra subjacente é do domínio das dádivas de sangue concedidas a uma determinada instituição (anónima) que procede a colheitas. Uma vez que não houve a possibilidade de aceder a um indivíduo com conhecimentos sobre o domínio e os dados em questão, a realização das operações de LD ficou a cargo do autor desta dissertação. Antes da realização de qualquer operação de LD, começou-se por efectuar uma ambientação, no geral, ao domínio em causa e, em particular, à BD em questão (*i.e.*, a que será objecto das operações de LD). A ambientação ao

domínio efectuou-se fundamentalmente com base em informação disponível na *Internet*. A ambientação à BD ocorreu por intermédio da realização de variados inquéritos em SQL.

Da ambientação ao domínio e à BD resultou a identificação de um conjunto de OD que faziam sentido serem executadas. Estas operações cobriam os diferentes NG, desde o atributo até ao nível que envolve múltiplas relações de BD diferentes. A sua especificação efectuou-se de uma só vez, ficando a cargo do *SmartClean* a sua organização e execução por NG. À medida que os PQD foram identificados e reportados, como resultado da execução das OD, efectuaram-se algumas correcções manuais e especificaram-se OC para solucionar automaticamente a generalidade dos problemas.

A execução das operações obedeceu à abordagem ascendente (*i.e.*, *bottom-up*) preconizada no modelo de LD defendido e implementada no *SmartClean*. Assim, começou-se pela execução das operações relativas ao NG mais elementar (*i.e.*, atributo) e terminou-se no NG de maior complexidade (*i.e.*, múltiplas relações de diferentes BD). Sempre que da execução de uma OD resultou a identificação de PQD que obrigaram à realização de correcções (por via automática ou manual[19]), isso redundou numa nova execução da OD em causa. Esta é uma das características do modelo proposto de LD. Na generalidade dos problemas alvo de acções correctivas, a nova execução das OD já não resultou na identificação do PQD. Nas secções seguintes relativas aos PQD detectados, apenas se apresentam os resultados da nova execução quando desta ainda resultaram PQD.

8.2 Apresentação do Caso

O estudo de caso de LD efectuado no âmbito deste trabalho de doutoramento incide sobre uma BD que contém informação sobre dádivas/colheitas de sangue, respectivos dadores e as correspondentes análises efectuadas a essas dádivas. Em torno destas três tabelas principais existem outras que armazenam informação adicional (*e.g.*: profissões dos dadores; estado civil dos dadores; tipo de dádiva) e com as quais se estabelecem relacionamentos. Na totalidade, a BD é composta por 12 tabelas. Esta BD é propriedade de uma instituição que se dedica a colheitas de sangue. A informação existente na BD diz respeito ao período compreendido entre 01/01/2000 e 31/06/2002. Doravante, esta BD será designada de BD de Dádivas (BDD). Na Figura 8.1

[19] A resolução manual significa que não possível executar uma OC que solucionasse automaticamente os PQD detectados. Assim, estes foram corrigidos directamente pelo utilizador (no caso, o autor da dissertação), caso a caso.

apresenta-se o diagrama entidade – relação referente à BDD. No diagrama constam também os atributos que caracterizam cada entidade.

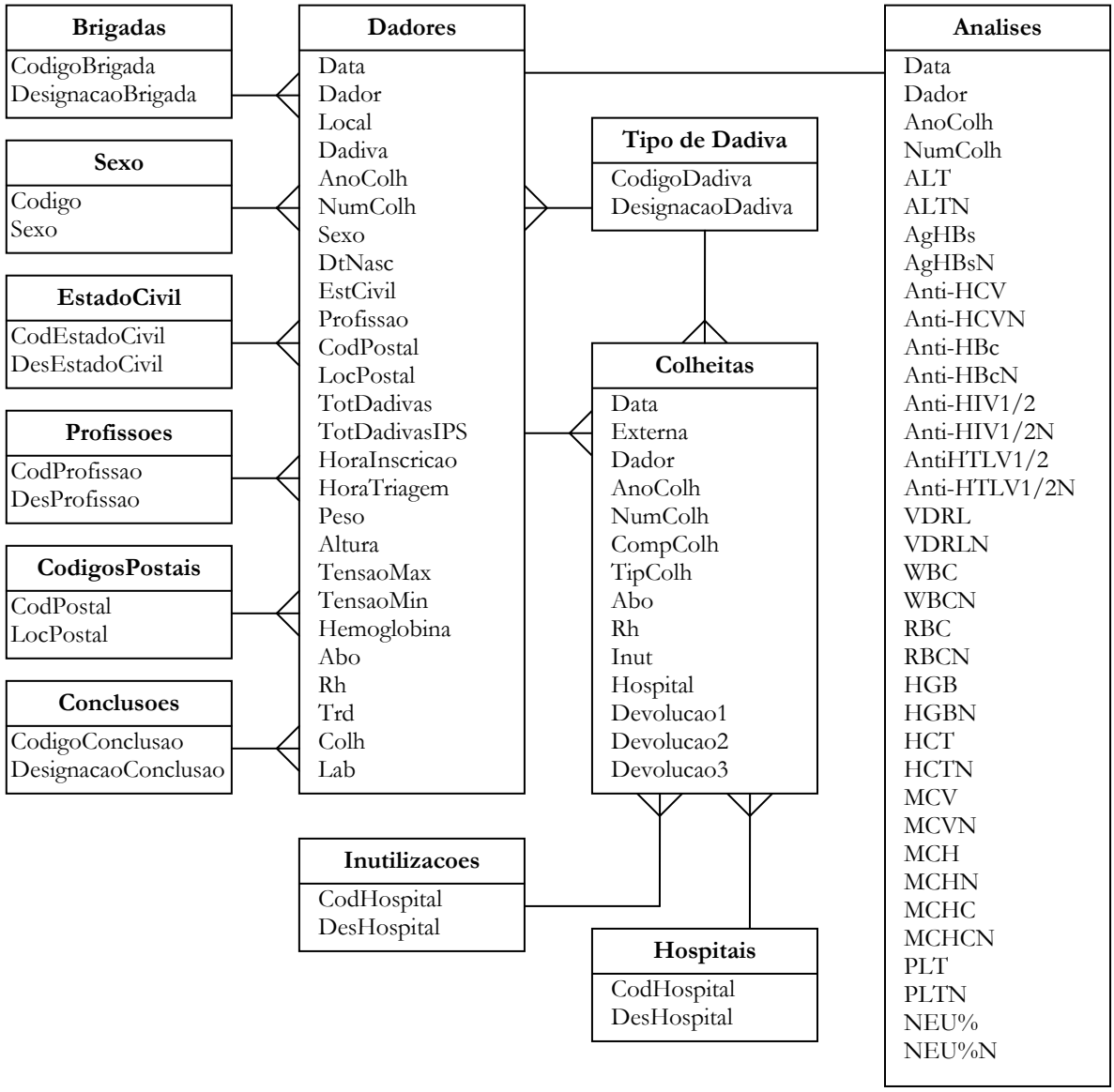


Figura 8.1 – Diagrama entidade – relação da BDD

Na sequência da apresentação do diagrama, justifica-se que sejam efectuadas as seguintes observações:

- Nem todas as relações existentes no modelo de dados respeitam a terceira forma normal. Sempre que há uma dádiva é registada informação redundante sobre o dador na relação *Dadores*. Independentemente do número de vezes que um dador tenha dado sangue é sempre registado o sexo, data de nascimento, a altura, o grupo sanguíneo (atributo *Abo*) e o factor *Rh* (*i.e.*, se o sangue é do tipo positivo ou negativo). Estes elementos não são susceptíveis de sofrer alterações ao longo do tempo, pelo que só deveriam ser

armazenados uma única vez para cada dador. Ao serem registados múltiplas vezes, há a possibilidade de serem introduzidas inconsistências (*i.e.*, violações de dependência funcional). Por outro lado, a relação *Dadores* só deveria conter informação sobre os dadores, quando também contém informação sobre cada dádiva (*e.g.*: local da dádiva; número da colheita; hora de inscrição; hora de triagem).

- Na relação *Colheitas* existem atributos como o grupo sanguíneo (atributo *Abo*) e o factor *Rh* que são intrínsecos ao dador, daí que não se justifique que sejam armazenados sempre que se efectua uma colheita para um dado dador. Mais uma vez, isto constitui uma abordagem redundante que, no limite, pode propiciar a introdução de inconsistências nestes valores.
- Existe uma relação *CodigosPostais* cuja finalidade é armazenar códigos postais e respectivas localidades. No entanto, esta mesma informação encontra-se armazenada na relação *Dadores*. A justificação é a seguinte: o mesmo valor de código postal pode estar associado a diferentes localidades, logo esta também tem de ser armazenada na tabela *Dadores*. Se apenas fosse armazenado o código postal nesta tabela, não seria possível saber qual a localidade do dador (a não ser que existisse apenas uma localidade para o código postal em questão). No entanto, esta forma de organização introduz redundância e eventualmente até inconsistência nos dados. Uma correcção efectuada na tabela *CodigosPostais* não se reflecte na tabela *Dadores*. A inclusão de um atributo numérico que funcionasse como identificador único de cada tuplo (*i.e.*, chave primária) em *CódigosPostais* teria permitido que o relacionamento entre ambas as tabelas pudesse ser estabelecido com base neste. Desta forma, garantidamente não existiriam os referidos problemas de redundância e inconsistência nos dados.

Face a estas observações, não se consegue vislumbrar uma justificação para o modelo de dados subjacente à BDD ter sido construído desta forma que não seja a de uma concepção deficiente. Recorde-se que no âmbito deste trabalho de doutoramento as preocupações com a qualidade centram-se unicamente ao nível da instância/valor dos dados. Ainda que se reconheça a sua importância, a qualidade do modelo de dados não constitui uma preocupação. Naturalmente, a qualidade deste modelo contribui para a existência ou não de certos problemas ao nível dos valores dos dados (*e.g.*: numa relação que se encontra na terceira forma normal não é possível existirem violações de dependência funcional).

Fisicamente, a BDD encontra-se implementada em *MySQL*. Nenhuma tabela desta BD possui chaves primárias definidas. No modelo relacional esta é uma situação pouco habitual. Nalgumas situações, a existência da chave primária é o garante da inexistência de redundância nas tabelas (*e.g.*: definir os atributos *CodPostal* e *LocPostal* como chaves da tabela *CodPostais*, impede que a

mesma combinação de valores destes atributos possa ser introduzida em duplicado). Como consequência, apesar dos relacionamentos entre as entidades que constam do modelo de dados apresentado na Figura 8.1, estes não se encontram implementados na BDD sob a forma de restrições que assegurem a sua integridade referencial.

Para se ficar com uma ideia da dimensão da BDD, na Tabela 8.1 apresenta-se o número de tuplos que fazem parte de cada tabela. Simultaneamente, fornece-se também uma descrição sucinta do seu conteúdo.

Tabela 8.1 – Informação relativa a cada tabela da BDD

Tabela	N.º tuplos	Significado/finalidade
Analises	98341	Análises efectuadas aos parâmetros sanguíneos
Brigadas	535	Locais onde se efectuam colheitas/dádivas
CodigosPostais	1987	Código postal dos dadores
Colheitas	246208	Colheitas/dádivas efectuadas
Conclusoes	122	Conclusões da aceitabilidade ou não dos dadores
Dadores	98341	Dadores de sangue
EstadoCivil	7	Estado civil dos dadores
Hospitais	272	Hospital de recolha/devolução da dádiva
Inutilizacoes	29	Motivo para a inutilização da dádiva
Profissoes	4036	Profissão dos dadores
Sexo	2	Sexo dos dadores
TipoDadiva	3	Tipo de dádiva efectuada pelos dadores

Como não existiam chaves primárias definidas nas tabelas, acrescentou-se um atributo numérico adicional que cumpre essa finalidade de identificador único. Este é um requisito de funcionamento do *SmartClean*. Assim sendo, todas as tabelas da BDD passaram a conter um atributo adicional aos que se encontram representados na Figura 8.1.

Naturalmente, os atributos que constituem a tabela *Dadores* da BDD não revelam as identidades dos dadores, mantendo-os em anonimato absoluto. Neste caso de estudo, essa informação não

tem qualquer interesse, daí que nem sequer tenha sido fornecida pela instituição que gentilmente cedeu a BDD.

8.3 Limpeza de Dados ao Nível do Atributo

As operações de LD efectuadas ao nível do atributo das diversas tabelas da BDD, em consonância com a taxionomia elaborada no Capítulo 4, ocorreram em contextos distintos: valor individual do atributo e múltiplos valores do atributo. As operações efectuadas em cada um destes contextos são expostas nas duas subsecções seguintes.

8.3.1 Contexto do Valor Individual

Começa-se, então, por apresentar as operações de LD efectuadas no contexto do valor individual dos atributos pertencentes às diversas tabelas da BDD.

8.3.1.1 Tabela *Conclusoes*

Operações de Detecção

A Tabela 8.2 apresenta as OD efectuadas no contexto do valor individual dos atributos da tabela *Conclusoes*.

Tabela 8.2 – OD executadas no contexto do valor individual dos atributos da tabela *Conclusoes*

Atributo	Valor em falta	Violação de sintaxe	Erro ortográfico
CodigoConclusao	×	×	
DesignacaoConclusao	×		×

Para efeitos ilustrativos, a seguir fornece-se a especificação sintáctica da generalidade destas operações.

```
DETECT MISSING-VALUE
ON CodigoConclusao FROM Conclusoes OF BDD

DETECT SYNTAX-VIOLATION
ON CodigoConclusao FROM Conclusoes OF BDD
WHERE CodigoConclusao NOT LIKE '[A-Z][0-9][0-9][0-9]'
```

```
DETECT MISSPELLING-ERRORS
ON DesignacaoConclusao FROM Conclusoes OF BDD
USING DICTIONARY DicPortugues
USING METRIC Jaro-Winkler
```

Da execução das OD de valor em falta em cada um dos atributos e da execução da OD de violação de sintaxe não resultou a identificação de qualquer PQD. A Tabela 8.3 apresenta os erros ortográficos identificados pela OD respectiva.

Tabela 8.3 – Resultados da OD de erro ortográfico no atributo *DesignacaoConclusao*

ID	Erro Ortográfico	Valores Similares	Grau de Semelhança
21	Homssexualidade	Homossexualidade	0.9854167
23	psicosocial	psicossocial	0.9888889
23	psicosocial	psico-social	0.98611116
31	Patologai	Patologia	0.98518515
31	Patologai	Catalogai	0.80423284
34	psqiquiátrica	psiquiátrica	0.95726496
35	infeciosa	infecciosa	0.98333335
38	autoimune	auto imune	0.98
38	autoimune	auto - imune	0.95
116	Inotoxicação	Intoxicação	0.95353544

Na primeira coluna da tabela encontram-se os valores das chaves primárias (atributo *ID*) dos tuplos nos quais se detectaram erros ortográficos no atributo em questão. A coluna seguinte contém os erros ortográficos propriamente ditos. Na terceira coluna encontram-se os valores existentes no dicionário que apresentam semelhanças aos erros ortográficos. Na última coluna há um número que relaciona os valores similares aos erros ortográficos, em função do grau de semelhança entre estes, resultante da métrica utilizada na OD (no caso, *Jaro-Winkler* [Winkler, 1999]).

Operações de Correcção

A correcção dos erros ortográficos no atributo *DesignacaoConclusao* envolveu a sua substituição pelos valores que, comparativamente a estes, apresentam o maior grau de semelhança. Para tal, executou-se a OC que a seguir se apresenta.

```
CORRECT MISSPELLING-ERRORS
ON DesignacaoConclusao FROM Conclusoes OF BDD
```

De acordo com a semântica apresentada no Capítulo 6, esta OC substitui o erro ortográfico pelo valor que possui o maior grau de semelhança. No caso de existir mais do que um valor com o mesmo grau de semelhança, não é efectuada correcção automática.

8.3.1.2 Tabela *Inutilizacoes*

Operações de Detecção

Na Tabela 8.4 apresentam-se as OD efectuadas no contexto do valor individual dos atributos da tabela *Inutilizacoes*.

Tabela 8.4 – OD executadas no contexto do valor individual dos atributos da tabela *Inutilizacoes*

Atributo	Valor em falta	Violação de sintaxe	Erro ortográfico
CodigoInutilizacao	×	×	
DescricaoInutilizacao	×		×

A execução das OD de valores em falta em cada um dos atributos, violação de sintaxe no atributo *CodigoInutilizacao* (deve ser composto por dois dígitos numéricos) e erro ortográfico no atributo *DescricaoInutilizacao* não redundou na detecção de qualquer PQD.

8.3.1.3 Tabela *Hospitais*

Operações de Detecção

A Tabela 8.5 exibe as OD efectuadas no contexto do valor individual dos atributos da tabela *Hospitais*.

Tabela 8.5 – OD executadas no contexto do valor individual dos atributos da tabela *Hospitais*

Atributo	Valor em falta	Violação de sintaxe	Erro ortográfico
CodHospital	×	×	
DesHospital	×		×

Da execução da OD de valor em falta em ambos os atributos da tabela *Hospitais* e da execução da OD de violação de sintaxe não resultou a identificação de qualquer PQD. A Tabela 8.6 apresenta os erros ortográficos identificados pela OD respectiva, executada no atributo *DesHospital*.

Tabela 8.6 – Resultados da OD de erro ortográfico no atributo *DesHospital*

ID	Erro Ortográfico	Valores Similares	Grau de Semelhança
10	Psiquiatrico	Psiquiátrico	0.9777778
80	Reynaldo	Reinaldo	0.93333334
133	Saude	Saúde	0.8933333

O significado de cada coluna desta tabela é o mesmo do que o fornecido no momento da apresentação da Tabela 8.3. O grau de semelhança resulta novamente da métrica *Jaro-Winkler*.

Operações de Correcção

À semelhança da correcção efectuada no atributo *DesignacaoConclusao* da tabela *Conclusoes*, neste atributo *DesHospital* a correcção dos erros ortográficos também envolveu a sua substituição pelos valores que, relativamente a estes, possuem o maior grau de semelhança. A especificação sintáctica desta OC é idêntica à apresentada anteriormente para o atributo *DesignacaoConclusao*.

8.3.1.4 Tabela *Brigadas*

Operações de Detecção

Na Tabela 8.7 encontram-se as OD efectuadas no contexto do valor individual dos atributos da tabela *Brigadas*.

Tabela 8.7 – OD executadas no contexto do valor individual dos atributos da tabela *Brigadas*

Atributo	Valor em falta	Violação de sintaxe
CodigoBrigada	×	×
DesignacaoBrigada	×	

A execução das OD de valor em falta em cada um dos atributos não identificou a existência de qualquer problema deste tipo. A execução da OD de violação de sintaxe visou identificar os valores que se encontram no atributo *CodigoBrigada* que não são constituídos, unicamente, por duas letras alfabéticas. Da execução desta OD, também não resultou a identificação de qualquer PQD.

8.3.1.5 Tabela *CodigosPostais*

Operações de Detecção

As OD efectuadas no contexto do valor individual dos atributos da tabela *CodigosPostais* encontram-se na Tabela 8.8

Tabela 8.8 – OD executadas no contexto do valor individual dos atributos da tabela *CodigosPostais*

Atributo	Valor em falta	Violação de domínio
CodPostal	×	×
LocPostal	×	

A especificação sintáctica da OD de violação de domínio no atributo *CodPostal* é a seguir apresentada.

```
DETECT DOMAIN-VIOLATION
ON CodPostal FROM CodigosPostais OF BDD
USING DOMAIN Codigos_Postais
```

A operação socorre-se do domínio de valores válidos do código postal. Este domínio encontra-se disponível no *SmartClean*, tendo sido construído a partir de um ficheiro de texto descarregado a partir do *site* dos CTT, que contém todos os códigos postais e respectivas localidades existentes em Portugal. Como os códigos postais armazenados na tabela *CodigosPostais* se encontram no

formato usado no passado, composto por apenas quatro dígitos, na construção do domínio de valores válidos suprimiram-se os três dígitos adicionais referentes à rua (que totalizam os sete dígitos actualmente utilizados em Portugal).

Da execução das OD de valor em falta em cada um dos atributos da tabela *CodigosPostais* não resultou a detecção de qualquer PQD. A execução da OD de violação de domínio resultou na identificação de 56 valores do atributo *CodPostal* que violam o domínio. Na Tabela 8.9 ilustram-se alguns desses valores.

Tabela 8.9 – Resultados da OD de violação de domínio no atributo *CodPostal*

ID	CodPostal
32	1850
299	2901
595	3965
1387	4915
1771	5495
1846	6720
1914	7855
1943	8985
1976	9870

Operações de Correcção

Com o objectivo de solucionar automaticamente o maior número possível de violações de domínio, executou-se a OC que a seguir se apresenta.

```
CORRECT DOMAIN-VIOLATION
ON CodPostal FROM CodigosPostais T1 OF BDD
SET CodPostal = CP
FROM Codigos_Postais, DO18 T2
WHERE LocPostal = Localidade AND T1.CodPostal = T2.CodPostal
```

A lógica que se encontra implícita a esta OC é a seguinte: Para os valores do atributo *CodPostal* identificados como constituindo violações ao domínio dos códigos postais (encontram-se armazenados na relação que resultou da execução da OD (no caso, *DO18*), da qual a Tabela 8.9

constitui uma amostra), recorre-se aos valores respectivos do atributo *LocPostal* para, no domínio dos códigos postais, determinar e actualizar o valor do código postal que corresponde a essa localidade. A realização das correcções encontra-se dependente da existência de correspondência entre os valores dos atributos *LocPostal* (tabela *CodigosPostais*) e *Localidade* (tabela *Codigos_Postais* – serve de suporte ao armazenamento do domínio dos códigos postais), o que nem sempre sucedeu.

Operações de Detecção (nova iteração)

Nesta nova iteração, apenas a OD de violação de domínio foi alvo de execução. Em particular, apenas os 56 valores que constituíam violações de domínio foram alvo de execução da OD. Após a realização da correcção referida, a execução da OD apenas detectou 9 violações de domínio no atributo *CodPostal*. A execução da OC anterior não solucionou estas violações de domínio em virtude de estar armazenado um lugar no atributo *LocPostal* e não a respectiva localidade. A resolução destes PQD efectuou-se manualmente, uma vez que implicou a realização de pesquisas na *Internet* para determinar a que localidade pertence cada lugar. O valor do atributo *LocPostal* foi, então, actualizado com a localidade em substituição do lugar.

8.3.1.6 Tabela *Profissoes*

Operações de Detecção

A Tabela 8.10 apresenta as OD efectuadas no contexto do valor individual dos atributos da tabela *Profissoes*.

Tabela 8.10 – OD executadas no contexto do valor individual dos atributos da tabela *Profissoes*

Atributo	Valor em falta	Violação de sintaxe	Erro ortográfico
CodProfissao	×	×	
DesProfissao	×		×

A OD de violação de sintaxe efectuada sobre o atributo *CodProfissao* é apresentada de seguida.

```
DETECT SYNTAX-VIOLATION
ON CodProfissao FROM Profissoes OF BDD
WHERE CodProfissao NOT LIKE '[0-9][0-9]-[0-9][0-9].[0-9][0-9]'
```

A execução das OD de valor em falta em cada um dos atributos não redundou na detecção de qualquer PQD. Na Tabela 8.11 apresentam-se os PQD identificados na sequência da execução da OD de violação de sintaxe.

Tabela 8.11 – Resultados da OD de violação de sintaxe no atributo *CodProfissao*

ID	CodProfissao
1	00-00-05
2	00-00-06
392	01-01-2001
667	02-02-1999
798	03-60-00
974	05-05-2005
1305	07-07-2007

Na Tabela 8.12 apresentam-se os erros ortográficos identificados no atributo *DesProfissao* como resultado da execução da OD respectiva.

Tabela 8.12 – Resultados da OD de erro ortográfico no atributo *DesProfissao*

ID Tuplo	Erro Ortográfico	Valores Similares	Grau de Semelhança
32	METEREOLOGISTA	METEOROLOGISTA	0.94065934
57	ELECTROTECNICOS	ELECTROTÉCNICOS	0.98222226
95	SEGURANCA	SEGURANÇA	0.97037035
137	MACANICOS	MECÂNICOS	0.8666667
234	SAUDE	SAÚDE	0.8933333
246	VETERINARIOS	VETERINÁRIOS	0.9777778
290	SERVICO	SERVIÇO	0.952381
364	PEDIATRICA	PEDIÁTRICA	0.9155556
415	SECUNDARIO	SECUNDÁRIO	0.97333336

O significado de cada coluna da tabela é o mesmo do que o fornecido aquando da apresentação da Tabela 8.3. Mais uma vez, o grau de semelhança resulta da métrica *Jaro-Winkler* utilizada na OD.

Operações de Correcção

A correcção das violações de sintaxe detectadas envolveu: (*i*) a substituição do segundo hífen por um ponto; (*ii*) a supressão dos dois primeiros dígitos do último conjunto (após o segundo hífen) quando este é composto por quatro dígitos. Para o efeito, executou-se a OC a seguir apresentada.

```
CORRECT SYNTAX-VIOLATION
ON CodProfissao FROM Profissoes OF BDD
BY TRANSFORMING (\d\d)-(\d\d)-(\d\d) INTO \1-\2.\3
BY TRANSFORMING (\d\d)-(\d\d)-(\d\d)(\d\d) INTO \1-\2.\4
```

A correcção dos erros ortográficos detectados no atributo *DesProfissao* envolveu a sua substituição pelos valores semelhantes identificados no dicionário. Esta OC é semelhante à apresentada anteriormente para o atributo *DesignacaoConclusao* da tabela *Conclusoes*.

8.3.1.7 Tabela *Colheitas*

Operações de Detecção

Na Tabela 8.13 apresentam-se as OD efectuadas no contexto do valor individual dos atributos da tabela *Colheitas*.

Tabela 8.13 – OD executadas no contexto do valor individual dos atributos da tabela *Colheitas*

Atributo	Valor em falta	Violação de domínio
Data	×	×
Dador	×	×
AnoColh	×	
NumColh	×	×
Abo		×
Rh		×

Das diversas OD efectuadas na tabela *Colheitas*, ilustra-se a de violação de domínio que incide sobre o atributo *Abo*.

```
DETECT DOMAIN-VIOLATION
ON Abo FROM Colheitas OF BDD
WHERE Abo NOT IN ('A','O','B', 'AB')
```

Da execução das OD de valor em falta em cada um dos atributos assinalados não resultou a detecção de qualquer PQD. A execução das OD de violação de domínio nos atributos *Data* (os valores têm de estar compreendidos entre 01-01-2000 e 30-06-2002), *Dador* (os valores têm de ser superiores a zero), *NumColh* (os valores têm de ser superiores a zero) e *Rh* (os valores têm de ser iguais a "+" ou "-"), também não redundou na detecção de qualquer PQD. A Tabela 8.14 apresenta as violações de domínio identificadas no atributo *Abo* pela OD respectiva (que precede este parágrafo).

Tabela 8.14 – Resultados da OD de violação de domínio no atributo *Abo*

ID	Abo
4155	A1
4177	A1
4178	A1
4179	A1
4186	A1
4187	A1
4188	A1
33940	A1
33941	A1
104252	A1
104253	A1

Operações de Correcção

Na correcção destas violações ao domínio do atributo *Abo*, optou-se por substituir os valores respectivos pelo grupo sanguíneo *A*. Para tal, executou-se a OC que a seguir se apresenta.

```
CORRECT DOMAIN-VIOLATION
ON Abo FROM Colheitas OF BDD
SET Abo = 'A'
```

8.3.1.8 Tabela *Dadores*

Operações de Detecção

As OD efectuadas no contexto do valor individual dos atributos da tabela *Dadores* encontram-se presentes na Tabela 8.15.

Tabela 8.15 – OD executadas no contexto do valor individual dos atributos da tabela *Dadores*

Atributo	Valor em falta	Violação de domínio	Violação de sintaxe
Data	×	×	
Dador	×	×	
AnoColh	×		
NumColh	×	×	
Sexo	×		
DtNasc	×		
Profissão			×
CodPostal	×	×	
TotDadivas	×	×	
TotDadivasInst	×	×	
HoraInscricao	×		
Peso	×	×	
Altura		×	
TensaoMax		×	
TensaoMin		×	
Abo		×	
Rh		×	

Da execução das OD de valor em falta nos atributos assinalados na tabela anterior (de preenchimento supostamente obrigatório) não resultou a detecção de qualquer PQD. A execução das OD de violação de domínio nos atributos *Data*, *Dador*, *NumColh* e *Rh* não detectou a existência de qualquer problema nestes atributos. A execução destas operações visou identificar violações ao domínio de valores dos atributos, de acordo com o já enunciado para os mesmos atributos da tabela *Colheitas*. A execução das OD de violação de domínio nos atributos *TotDadivas* e *TotDadivasInst* (em ambos os atributos o total de dádivas tem de ser igual ou superior a zero) não redundou na detecção de qualquer PQD. A execução da OD de violação de domínio no atributo *Abo* culminou na identificação da existência do valor *A1* em dois tuplos, cujos valores das chaves primárias (atributo *ID*), respectivamente, são 31369 e 39177.

Na Tabela 8.16 apresentam-se algumas das 101 violações de domínio identificadas no atributo *CodPostal*, como consequência dos respectivos valores não pertencerem ao domínio.

Tabela 8.16 – Resultados da OD de violação de domínio no atributo *CodPostal*

ID	CodPostal
15800	1235
52411	1590
77913	4754
14596	4915
8502	5051
29779	5315
29298	6380
18754	9090

Na Tabela 8.17 apresentam-se algumas das 312 violações de domínio detectadas no atributo *Peso* (o peso não pode ser inferior a 50 kg (um peso igual ou superior a 50 kg é uma das condições para se ser dador), nem superior a 250 kg).

Na Tabela 8.18 apresentam-se algumas das 59 violações de domínio detectadas no atributo *Altura* (a altura não pode ser inferior a 1.3 nem superior a 2.5).

Tabela 8.17 – Resultados da OD de violação de domínio no atributo *Peso*

ID	Peso
73359	-71
31796	0
13449	0,5
59785	1
2441	6
44134	28
39755	472
2031	599
13577	698
7018	797
29088	987,5

Tabela 8.18 – Resultados da OD de violação de domínio no atributo *Altura*

ID	Altura
9133	0
17416	0,27
28579	0,55
77835	0,63
48651	0,7
74751	0,85
92989	3
76348	3,8
9291	5
7504	6,55

Na Tabela 8.19 apresentam-se algumas das 175 violações de domínio identificadas no atributo *TensaoMax* (a tensão máxima não pode ser inferior a 70 nem superior a 240).

Tabela 8.19 – Resultados da OD de violação de domínio no atributo *TensaoMax*

ID	TensaoMax
68896	0
27270	10
89680	26
87304	38
85066	61
15495	411
80940	534
67056	625
5840	850
60525	970

Na Tabela 8.20 apresentam-se algumas das 250 violações de domínio detectadas no atributo *TensaoMin* (a tensão mínima não pode ser inferior a 40 nem superior a 120).

Tabela 8.20 – Resultados da OD de violação de domínio no atributo *TensaoMin*

ID	TensaoMin
60001	0
40774	10
47284	39
37095	160
91697	368
7798	556
57766	790
70960	990

No atributo *Profissao* da tabela *Dadores* executou-se uma OD de violação de sintaxe semelhante à efectuada no atributo *CodProfissao* da tabela *Profissoes*. O objectivo consistia em identificar violações ao seguinte formato de código de profissão: *nn-nn.nn*, em que *n* representa um dígito numérico. Da execução da operação resultou a identificação de 11955 valores que violam esta sintaxe. Na Tabela 8.21 apresentam-se os tipos distintos de violação de sintaxe detectados.

Tabela 8.21 – Resultados da OD de violação de sintaxe no atributo *Profissao*

ID	Profissao
51620	00-00-06
32968	07-07-2007
45821	02-02-1999
90075	01-01-2001
65702	05-05-2005

Operações de Correcção

À semelhança do efectuado no atributo *Abo* da tabela *Colheitas*, a correcção das duas violações de domínio detectadas no atributo *Abo* da tabela *Dadores*, envolveu a substituição dos respectivos valores pelo valor *A*. Para o efeito, executou-se uma OC equivalente à já apresentada na tabela *Colheitas*.

Na correcção automática das violações de domínio no atributo *CodPostal*, executou-se a OC que a seguir se apresenta. Esta operação é idêntica à efectuada na tabela *CodigosPostais* (já apresentada) para solucionar o mesmo tipo de problema.

```
CORRECT DOMAIN-VIOLATION
ON CodPostal FROM Dadores T1 OF BDD
SET CodPostal = CP
FROM Codigos_Postais, DO45 T2
WHERE LocPostal = Localidade AND T1.CodPostal = T2.CodPostal
```

Na correcção das violações de domínio identificadas no atributo *Peso* executou-se a operação que a seguir se apresenta.

```
CORRECT DOMAIN-VIOLATION
ON Peso FROM Dadores OF BDD
SET Peso = PesoMedioDador(ID, "Dadores", "ID", "Peso")
```

Como se pode observar, esta OC envolveu a invocação de uma função definida pelo utilizador (no caso, *PesoMedioDador*). Esta função foi desenvolvida especificamente para manipular o PQD em questão. Assim, para cada valor que constituía violação de domínio, utilizou-se o valor do atributo *ID* (chave primária da tabela *Dadores*) para identificar na tabela *Dadores* o respectivo dador. Se a tabela continha outros tuplos referentes ao mesmo dador, estes foram utilizados no cálculo da média dos valores do atributo *Peso* nesses tuplos. Neste cálculo, apenas se consideraram os valores que não violavam o domínio do atributo. Na situação contrária, calculou-se a média dos pesos dos indivíduos do mesmo sexo e idade. Utilizou-se o valor resultante do cálculo da média para substituir o valor que violava o domínio.

Na correcção das violações de domínio detectadas no atributo *Altura*, executou-se a operação apresentada de seguida.

```
CORRECT DOMAIN-VIOLATION
ON Altura FROM Dadores OF BDD
SET Altura = AlturaDador(ID, "Dadores", "ID", "Altura")
```

Esta OC também envolveu a invocação de uma função definida pelo utilizador (no caso *AlturaDador*). A semântica que se encontra subjacente a esta função é a que se descreve a seguir. Para cada valor que constituía violação de domínio, utilizou-se o respectivo valor do atributo *ID* para se identificar na tabela *Dadores* o dador em causa. Na situação da tabela conter outros tuplos referentes ao mesmo dador, utilizou-se o valor do atributo *Altura* desses tuplos. Sempre que existia diferenças nestes valores, utilizou-se o princípio da maioria, *i.e.*, seleccionou-se a altura que se encontrava na maioria dos tuplos. Quando não existiu uma altura com um número de ocorrências superior às restantes, não se efectuou qualquer tipo de correcção, tendo sido retornado o próprio valor que constituía a violação ao domínio. Em ambos os casos, os valores do atributo *Altura* existentes nos demais tuplos, apenas foram considerados quando não representavam uma violação de domínio. O resultado retornado pela função actualizou o valor do atributo.

Na correcção das violações de domínio identificadas no atributo *TensaoMax*, executou-se uma operação similar às duas anteriores, envolvendo a invocação de uma função definida pelo utilizador (no caso, *TensaoMaxMediaDador*). A semântica implícita a esta função, com as devidas adaptações, é idêntica à já descrita para a correcção da violação de domínio ao peso do dador. Basicamente, os valores do atributo *TensaoMax* que violavam o domínio foram substituídos pelo valor médio da tensão máxima existente noutros tuplos para o mesmo dador. Na ausência desta

informação, actualizou-se o atributo com a média dos valores da tensão máxima dos indivíduos do mesmo sexo e idade.

No atributo *TensaoMin* executou-se uma OC semelhante à anterior para resolução das violações de domínio detectadas, mais uma vez envolvendo o recurso a uma função definida pelo utilizador (no caso, *TensaoMinMediaDador*). Com as necessárias adaptações, a semântica que se encontra subjacente a esta função é similar à descrita para a função *TensaoMaxMediaDador*.

Na resolução dos PQD de violação de sintaxe no atributo *Profissao* recorreu-se à mesma OC (com as devidas adaptações) que tinha sido definida para a correcção das violações de sintaxe no atributo *CodProfissao* da tabela *Profissoes*.

Operações de Detecção (nova iteração)

Nesta nova iteração, apenas as OD respeitantes aos atributos onde tinham sido identificados PQD voltaram a ser objecto de execução. Mais especificamente, o seu âmbito restringiu-se apenas aos valores nos quais tinham sido detectados PQD. Da execução destas operações, apenas a referente à detecção de violações de domínio no atributo *CodPostal* voltou a reportar a existência de 37 problemas. Isto significa que das 101 violações inicialmente detectadas, a OC anteriormente realizada solucionou 64 problemas. O motivo pelo qual as 37 violações de domínio subsistiram no atributo *CodPostal* e o modo como foram manualmente solucionadas, são os mesmos do que os já descritos no âmbito do atributo *CodPostal* da tabela *CodigosPostais*.

8.3.1.9 Tabela *Analises*

Operações de Detecção

Na Tabela 8.22 encontram-se as OD efectuadas no contexto do valor individual dos atributos da tabela *Analises*.

A execução das OD de valor em falta nos quatro atributos não resultou na detecção de qualquer PQD. Igualmente, as OD de violação de domínio também não culminaram na identificação de problemas. A execução destas operações visou identificar as violações ao domínio dos atributos, de acordo com as regras já enunciadas para os mesmos atributos da tabela *Colheitas*.

Tabela 8.22 – OD executadas no contexto do valor individual dos atributos da tabela *Analises*

Atributo	Valor em falta	Violação de domínio
Data	×	×
Dador	×	×
AnoColh	×	
NumColh	×	×

8.3.2 Contexto Multi-Valor

Seguidamente, são apresentadas as operações de LD efectuadas nas diversas tabelas da BDD no contexto dos múltiplos valores dos seus atributos.

8.3.2.1 Tabela *Conclusoes*

Operações de Detecção

A Tabela 8.23 apresenta as OD efectuadas no contexto dos múltiplos valores de cada atributo da tabela *Conclusoes*.

Tabela 8.23 – OD executadas no contexto dos múltiplos valores dos atributos da tabela *Conclusoes*

Atributo	Violação de unicidade
CodigoConclusao	×
DesignacaoConclusao	×

A especificação sintáctica de uma destas OD é apresentada de seguida.

```
DETECT UNIQUENESS-VIOLATION
ON CodigoConclusao FROM Conclusoes OF BDD
```

A execução da OD de violação de unicidade no atributo *CodigoConclusão* não originou a identificação de qualquer PQD. Por seu turno, a execução do mesmo tipo de OD no atributo *DesignacaoConclusao* resultou na identificação dos PQD apresentados na Tabela 8.24.

Tabela 8.24 – Resultados da OD de violação de unicidade no atributo *DesignacaConclusao*

ID	DesignacaoConclusao
56	ALT elevada
46	ALT elevada
119	Comportamento de risco
22	Comportamento de risco
122	Genérico
53	Genérico
36	Patologia hematológica
83	Patologia hematológica
35	Patologia infecciosa / parasitária
91	Patologia infecciosa / parasitária
33	Patologia neurológica
89	Patologia neurológica
28	Patologia respiratória
84	Patologia respiratória
37	Patologia reumatológica
93	Patologia reumatológica

Operações de Correcção

Na resolução das violações de unicidade detectadas no atributo *DesignacaoConclusao* executou-se a operação a seguir apresentada.

```
REMOVE UNIQUENESS-VIOLATION
ON DesignacaoConclusao FROM Conclusoes OF BDD
```

De acordo com a semântica apresentada no Capítulo 6, esta operação elimina os tuplos redundantes de que resultam as violações de unicidade e, caso a tabela em questão se encontre envolvida nalgum relacionamento, efectua as necessárias alterações que asseguram a manutenção da integridade referencial. Existe um relacionamento entre a tabela *Conclusões* (atributo

CodigoConclusao) e a tabela *Dadores* (atributos *Trcl*, *Colh* e *Lab*), ainda que este não se encontre definido na BDD. A inexistência da definição deste relacionamento impede que as necessárias alterações que asseguram a integridade referencial sejam efectuadas automaticamente pela operação em questão. Assim, antes da sua execução, definiu-se a existência desse relacionamento entre as tabelas na BDD.

8.3.2.2 Tabela *Inutilizacoes*

Operações de Detecção

Na Tabela 8.25 apresentam-se as OD efectuadas no contexto dos múltiplos valores de cada atributo da tabela *Inutilizacoes*.

Tabela 8.25 – OD executadas no contexto dos múltiplos valores dos atributos da tabela *Inutilizacoes*

Atributo	Violação de unicidade
CodigoInutilizacao	×
DescricaoInutilizacao	×

A execução das OD de violação de unicidade nos atributos *CodigoInutilizacao* e *DescricaoInutilizacao* não redundou na detecção deste tipo de PQD.

8.3.2.3 Tabela *Hospitais*

Operações de Detecção

As OD efectuadas no contexto dos múltiplos valores de cada atributo da tabela *Hospitais* encontram-se na Tabela 8.26.

Tabela 8.26 – OD executadas no contexto dos múltiplos valores dos atributos da tabela *Hospitais*

Atributo	Violação de unicidade
CodHospital	×
DesHospital	×

Da execução da OD em questão no atributo *CodHospital* não resultou a detecção de PQD. No caso do atributo *DesHospital* detectou-se um problema de violação de unicidade, resultante da existência do valor *Nefroclínica*, respectivamente, nos tuplos com valor de chave primária igual a 208 e 214.

Operações de Correcção

Uma vez que apenas se detectou um PQD (os dois tuplos identificados representam a violação de unicidade), a sua resolução efectuou-se manualmente. Assim, após se ter constatado que a eliminação do tuplo com a chave primária igual a 208 não acarretava violações à integridade referencial da tabela *Colheitas*, procedeu-se à sua eliminação. Note-se que apesar de não estar definida na BDD, existe um relacionamento entre a tabela *Hospitais* (atributo *CodHospital*) e a tabela *Colheitas* (atributo *Hospital*).

8.3.2.4 Tabela *Brigadas*

Operações de Detecção

A Tabela 8.27 apresenta as OD efectuadas no contexto dos múltiplos valores de cada atributo da tabela *Brigadas*.

Tabela 8.27 – OD executadas no contexto dos múltiplos valores dos atributos da tabela *Brigadas*

Atributo	Violação de unicidade
CodigoBrigada	×
DesignacaoBrigada	×

A execução da OD de violação de unicidade no atributo *CodigoBrigada* não culminou na detecção de PQD deste tipo. A execução do mesmo tipo de OD no atributo *DesignacaoBrigada* resultou na identificação das violações de unicidade apresentadas na Tabela 8.28.

Tabela 8.28 – Resultados da OD de violação de domínio no atributo *DesignacaBrigada*

ID	DesignacaoBrigada
85	Associação Dadores EDP – Braga
91	Associação Dadores EDP – Braga
100	EN – Electricidade do Norte SA.
103	EN – Electricidade do Norte SA.
275	Amorim & Irmãos
511	Amorim & Irmãos

Operações de Correcção

Na resolução das violações de unicidade detectadas no atributo *DesginacaoBrigada*, executou-se uma operação idêntica à executada para a resolução do mesmo tipo de problema no atributo *DesignacaoConclusao* da tabela *Conclusoes*. Para que todas as alterações necessárias que garantem a manutenção da integridade referencial entre as tabelas *Brigadas* (atributo *CodigoBrigada*) e *Dadores* (atributo *Local*) fossem efectuadas automaticamente pela operação em questão, estabeleceu-se o relacionamento entre estas tabelas na BDD, uma vez que este não existia.

8.3.2.5 Tabela *Profissoes*

Operações de Detecção

Na Tabela 8.29 apresenta-se as OD efectuadas no contexto dos múltiplos valores de cada atributo da tabela *Profissoes*.

Tabela 8.29 – OD executadas no contexto dos múltiplos valores dos atributos da tabela *Profissoes*

Atributo	Violação de unicidade
CodProfissao	×
DesProfissao	×

Da execução das OD de violação de unicidade em ambos os atributos não resultou a identificação da existência deste tipo de PQD.

8.4 Limpeza de Dados ao Nível do Tuplo

As violações de restrição de integridade constituem o único PQD susceptível de ocorrer ao nível do tuplo. Seguidamente, apresentam-se as operações de LD efectuadas nas diversas tabelas da BDD com o intuito de detectar e solucionar este tipo de PQD.

8.4.1 Tabela *Colheitas*

Operações de Detecção

Na tabela *Colheitas*, a seguinte restrição de integridade tem de ser respeitada: o valor do atributo referente ao ano da colheita (*i.e.*, *AnoColh*) tem de ser igual ao valor do atributo relativo ao ano da data da colheita (*i.e.*, *Data*). Seguidamente, apresenta-se a OD executada com o objectivo de identificar as violações a esta restrição de integridade.

```
DETECT INTEGRITY-CONSTRAINT-VIOLATION
AT LEVEL OF ROW
ON Data, AnoColh FROM Colheitas OF BDD
WHERE YEAR(data) <> AnoColh
```

Da execução desta OD resultou a identificação da violação de restrição de integridade que se apresenta na Tabela 8.30.

Tabela 8.30 – Resultados da OD de violação de restrição de integridade na tabela *Colheitas*

ID	Data	AnoColh
92242	2000-10-19	99

Operações de Correcção

Uma vez que apenas se detectou um único PQD, a sua correcção efectuou-se manualmente. O valor do atributo *AnoColh* do tuplo em questão foi actualizado para *2000*. Neste caso, não fazia sentido especificar-se uma OC para solucionar uma só violação de restrição de integridade.

8.4.2 Tabela *Analises*

Operações de Detecção

A restrição enunciada para a tabela *Colheitas* também se verifica na tabela *Analises*. Nesta tabela, a execução da OD correspondente não redundou na identificação de violações à restrição de integridade enunciada.

8.4.3 Tabela *Dadores*

Operações de Detecção

A seguir apresentam-se as cinco OD de violação de restrição de integridade efectuadas no âmbito desta tabela.

Restrição de Integridade n.º 1

À semelhança da tabela *Analises*, a restrição enunciada para a tabela *Colheitas* aplica-se igualmente à tabela *Dadores*. Da execução da OD correspondente nesta tabela, também não resultou a detecção de qualquer violação à restrição de integridade.

Restrição de Integridade n.º 2

Na tabela *Dadores*, a seguinte restrição tem de ser respeitada: o valor do atributo relativo ao total de dádivas (*i.e.*, *TotDadivas*) tem de ser igual ou superior ao valor do atributo referente ao total de dádivas à instituição em causa (*i.e.*, *TotDadivasInst*). A execução da OD correspondente, redundou na identificação das violações apresentadas na Tabela 8.31.

Tabela 8.31 – Resultados da OD de violação da 2ª restrição de integridade na tabela *Dadores*

ID	TotDadivas	TotDadivasInst
3002	10	18
22003	10	18
24460	1	2
31957	10	18
63506	10	18
83430	1	2

Restrição de Integridade n.º 3

Nesta tabela a seguinte restrição de integridade ao nível do tuplo tem de ser respeitada: o valor do atributo relativo à pressão arterial máxima (*i.e.*, *TensaoMax*) tem de ser superior ao valor do atributo referente à pressão arterial mínima (*i.e.*, *TensaoMin*). Da execução desta OD resultou a identificação da violação de restrição de integridade que se apresenta na Tabela 8.32.

Tabela 8.32 – Resultados da OD de violação da 3ª restrição de integridade na tabela *Dadores*

ID	TensaoMin	TensaoMax
35479	100	100

Restrição de Integridade n.º 4

Além das já apresentadas, nesta tabela há ainda a seguinte restrição de integridade ao nível do tuplo: O valor do atributo relativo à hora de triagem (*i.e.*, *HoraTriagem*) tem de ser superior ao valor do atributo referente à hora de inscrição (*i.e.*, *HoraInscrição*). Seguidamente, apresenta-se a OD que se encontra subjacente a esta restrição de integridade.

```
DETECT INTEGRITY-CONSTRAINT-VIOLATION
AT LEVEL OF ROW
ON HoraInscricao, HoraTriagem FROM Dadores OF BDD
WHERE HoraInscricao > HoraTriagem
```

Na Tabela 8.33 ilustram-se algumas das 132 violações identificadas a esta restrição de integridade, resultantes da execução desta OD.

Tabela 8.33 – Resultados da OD de violação da 4ª restrição de integridade na tabela *Dadores*

ID	HoraInscricao	HoraTriagem
9111	10:27:00	10:10:00
19443	16:35:00	16:24:00
29891	11:22:00	11:02:00
37699	10:24:00	10:17:00
58682	12:37:00	12:31:00
79470	17:10:00	17:03:00
86952	13:07:00	13:01:00

Restrição de Integridade n.º 5

Por último, há uma restrição que obriga a que um dador tenha idade compreendida entre os 18 e os 65 anos de idade. Na identificação das violações que ocorrem a esta restrição de integridade executou-se a seguinte OD.

```
DETECT INTEGRITY-CONSTRAINT-VIOLATION
AT LEVEL OF ROW
ON Data, DtNasc FROM Dadores OF BDD
WHERE DateDiff(Data, DtNasc)/365.25 < 18 OR DateDiff(Data, DtNasc)/365.25 > 65
```

A execução desta OD não redundou na detecção de qualquer violação à restrição de integridade em questão.

Operações de Correcção

As OC executadas com o intuito de solucionar as violações às restrições de integridade detectadas são, a seguir, apresentadas.

Restrição de Integridade n.º 2

Na resolução das violações detectadas a esta restrição de integridade, executou-se a OC que a seguir se apresenta.

```
CORRECT INTEGRITY-CONSTRAINT-VIOLATION
AT LEVEL OF ROW
ON TotDadivas, TotDadivasInst FROM Dadores OF BDD
SET TotDadivas = TotDadivasInst
```

Assim, considerou-se que todas as dádivas foram efectuadas directamente à instituição em causa.

Restrição de Integridade n.º 3

Apesar de ter sido identificada uma única violação à restrição de integridade n.º 3, na sua correcção executou-se a operação que a seguir se apresenta. Esta OC invoca a mesma função definida pelo utilizador, desenvolvida para a correcção de violações de domínio no atributo *TensaoMax*. A possibilidade da sua reutilização na resolução deste PQD justifica a especificação desta OC.

```
CORRECT INTEGRITY-CONSTRAINT-VIOLATION
AT LEVEL OF ROW
ON TensaoMin, TensaoMax FROM Dadores OF BDD
SET TensaoMax = TensaoMaxMediaDador(ID, "Dadores", "ID", "TensaoMax")
```

A semântica desta operação é equivalente à semântica da OC de violação de domínio no atributo *TensaoMax*. Basicamente, substituiu-se o valor do atributo *TensaoMax* que viola a restrição de integridade pelo valor médio da tensão máxima existente noutro tuplos para o mesmo dador. Na ausência desta informação, actualizou-se o atributo com a média dos valores da tensão máxima dos indivíduos do mesmo sexo e idade.

Restrição de Integridade n.º 4

Na correcção das violações detectadas à restrição de integridade n.º 4, recorreu-se à operação que a seguir se apresenta.

```
CORRECT INTEGRITY-CONSTRAINT-VIOLATION
AT LEVEL OF ROW
ON HoraInscricao, HoraTriagem FROM Dadores OF BDD
SET Temp = HoraInscricao, HoraInscricao = HoraTriagem, HoraTriagem = Temp
```

Esta OC procedeu à troca dos valores que se encontram nos atributos *HoraInscricao* e *HoraTriagem*, o que soluciona as violações identificadas à restrição de integridade. Na sua base, encontra-se a constatação de que houve uma troca dos valores, no momento da sua introdução nos respectivos atributos.

8.5 Limpeza de Dados ao Nível da Relação

Nesta secção apresentam-se as operações de LD efectuadas ao nível das tabelas da BDD, quando consideradas individualmente. As operações executadas encontram-se representadas na Tabela 8.34.

Tabela 8.34 – OD executadas no NG da relação

Tabela	Violação de dep. funcional	Tuplos duplicados	Violação restr. integridade
CodigosPostais		×	
Colheitas	×	×	
Dadores	×	×	×
Análises		×	

8.5.1 Tabela *CodigosPostais*

Operações de Detecção

A OD de tuplos duplicados executada nesta tabela é a seguir apresentada.

```
DETECT DUPLICATE-TUPLES
FROM CodigosPostais T1, CodigosPostais T2 OF BDD
USING CodPostal, LocPostal
WHERE T1.CodPostal = T2.CodPostal AND T1.LocPostal = T2.LocPostal
```

Nota: Este tipo de operação envolve a realização de um produto cartesiano sobre a tabela. Para evitar a realização de comparações desnecessárias e diminuir ao tempo de execução, no momento de compilação da OD, o *SmartClean* acrescentou a seguinte condição à cláusula *Where* da operação: T1.ID > T2.ID. Naturalmente, isto só aconteceu porque esta não tinha sido especificada.

Da execução desta operação não resultou a detecção da existência de tuplos duplicados na tabela *CodigosPostais*.

8.5.2 Tabela *Colheitas*

Operações de Detecção

Ao nível do PQD violação de dependência funcional efectuaram-se duas OD nesta tabela: (*i*) a um determinado dador tem de corresponder sempre o mesmo grupo sanguíneo (atributo *Abo*); (*ii*) ao mesmo dador tem de corresponder sempre o mesmo factor *Rh* (atributo *Rh*). De seguida, ilustra-se a primeira OD de violação de dependência funcional.

```
DETECT FUNCTIONAL-DEPENDENCY-VIOLATION
ON Abo FROM Colheitas OF BDD
DEPENDENT ON Dador
```

Da execução desta OD resultou a identificação das violações de dependência funcional que se apresentam na Tabela 8.35.

Tabela 8.35 – Resultados da execução da 1ª OD de violação da dependência funcional na tabela *Colheitas*

ID	Dador	Abo
68474	60370	A
68475	60370	A
68476	60370	A
54078	60370	O
54079	60370	O
54080	60370	O
205622	60370	O
205623	60370	O
205624	60370	O
206718	60370	O
206719	60370	O
206720	60370	O
231860	77660	A
191203	77660	O
191204	77660	O
191205	77660	O

A execução da segunda OD de violação de dependência funcional redundou na identificação de 10 PQD deste tipo. Na Tabela 8.36 apresenta-se os valores envolvidos numa dessas violações de dependência funcional.

Tabela 8.36 – Resultados da execução da 2ª OD de violação da dependência funcional na tabela *Colheitas*

ID	Dador	Rh
38106	21803	-
38107	21803	-
88364	21803	-
88365	21803	-
88366	21803	-
136650	21803	-
136651	21803	-
136652	21803	-
231471	21803	-
231472	21803	-
231473	21803	-
182214	21803	+
182215	21803	+
182216	21803	+

Nesta tabela efectuou-se uma OD que visou identificar a existência de tuplos duplicados. Considera-se que um tuplo é duplicado de outro quando simultaneamente os valores dos atributos relativos à data, dador e componente colhido (atributo *CompColh*) são iguais. A execução da OD não detectou a existência de qualquer PQD deste tipo.

Operações de Correcção

Na correcção destas violações de dependência funcional, executou-se a operação que a seguir se apresenta.

```
CORRECT FUNCTIONAL-DEPENDENCY-VIOLATION
ON Abo FROM Colheitas OF BDD
DEPENDENT ON Dador
```

De acordo com a semântica apresentada no Capítulo 6 para este tipo de OC, a resolução das violações de dependência funcional baseia-se na aplicação do princípio da maioria. Na correcção das violações de dependência funcional no atributo *Abo*, o valor do grupo sanguíneo dos dadores *60370* e *77660* passa a ser *O* em todos os tuplos que a estes se referem. Na correcção das violações de dependência funcional no atributo *Rh* executou-se uma operação idêntica à efectuada no atributo *Abo*.

8.5.3 Tabela *Dadores*

Operações de Detecção

Ao nível do PQD violação de dependência funcional efectuaram-se cinco OD nesta tabela: (*i*) a um determinado dador tem de corresponder sempre o mesmo sexo; (*ii*) a um determinado dador tem de corresponder sempre a mesma data de nascimento; (*iii*) a um determinado dador tem de corresponder sempre a mesma altura; (*iv*) a um determinado dador tem de corresponder sempre o mesmo grupo sanguíneo; e, (*v*) a um determinado dador tem de corresponder sempre o mesmo factor *Rh*. À excepção da terceira OD (*i.e.*, relacionada com a altura do dador), a execução das demais operações não culminou na detecção de qualquer PQD do tipo em questão. Da execução da terceira operação resultou a detecção de 40 violações de dependência funcional. Na tabela 8.37 apresentam-se os valores envolvidos em duas dessas violações.

Tabela 8.37 – Resultados da execução da 3ª OD de violação da dependência funcional na tabela *Dadores*

ID	Dador	Altura
6434	12581	1,7
42345	12581	1,78
60110	12581	1,78
86462	12581	1,78
7892	54925	1,69
22561	54925	1,79
40834	54925	1,79
75010	54925	1,79
89483	54925	1,79

Nesta tabela efectuou-se uma OD para identificar os tuplos duplicados que eventualmente pudessem existir. Um tuplo diz-se duplicado de outro quando simultaneamente os valores dos atributos referentes à data e ao dador são iguais. A execução da OD permitiu confirmar a inexistência de tuplos duplicados.

Ainda na Tabela *Dadores*, efectuaram-se OD com o objectivo de detectar violações às seguintes restrições de integridade: (*i*) um indivíduo do sexo masculino apenas pode dar sangue de três em três meses (*i.e.*, quatro vezes por ano); (*ii*) um indivíduo do sexo feminino apenas pode dar sangue de quatro em quatro meses (*i.e.*, três vezes por ano). De seguida, apresenta-se a OD referente à primeira restrição de integridade.

```
DETECT INTEGRITY-CONSTRAINT-VIOLATION
AT LEVEL OF RELATION
ON T1.ID, T2.ID, T1.Dador, T1.Data, T2.Data, FROM Dadores T1, Dadores T2 OF BDD
WHERE T1.ID > T2.ID AND T1.Dador = T2.Dador AND T1.Sexo = 1 AND
        Abs(DateDiff(T1.Data, T2.Data)) < 90
```

Da execução desta OD, bem como da OD relativa à segunda restrição de integridade não resultou a detecção de qualquer PQD.

Operações de Correcção

Na resolução da violação de dependência funcional detectada no atributo *Altura* da tabela *Dadores*, executou-se uma OC semelhante às efectuadas nos atributos *Abo* e *Rh* da tabela *Colheitas*.

8.5.4 Tabela *Analises*

Operações de Detecção

À semelhança das tabelas anteriores, nesta tabela também se executou uma OD com o objectivo de identificar a existência de eventuais tuplos duplicados. Um tuplo é considerado duplicado de outro quando simultaneamente os valores dos atributos referentes à data e ao dador são iguais. Da execução da OD não resultou a identificação da existência de qualquer tuplo duplicado.

8.6 Limpeza de Dados ao Nível Multi-Relação da Base de Dados

Nesta secção apresentam-se as operações de LD efectuadas ao nível das múltiplas tabelas da BDD. No contexto dos diversos PQD que podem ocorrer ao nível de múltiplas relações, nesta

BD apenas faz sentido procurar detectar violações de integridade referencial e violações de restrição de integridade. Na Tabela 8.38 apresentam-se as OD efectuadas entre cada par de tabelas da BDD.

Tabela 8.38 – OD executadas no NG multi-relação da BDD

Tabelas	Violação integr. referencial	Violação restr. Integridade
Dadores – Brigadas	×	
Dadores – TipoDadiva	×	
Dadores – Conclusoes	×	
Dadores – Sexo	×	
Dadores – EstadoCivil	×	
Dadores – Profissoes	×	
Dadores – CodigosPostais	×	
Dadores – Analises	×	×
Dadores – Colheitas	×	×
Colheitas – Analises	×	×
Colheitas – TipoDadiva	×	
Colheitas – Inutilizacoes	×	
Colheitas – Hospitais	×	

De seguida, apresentam-se os resultados obtidos com a execução das diversas OD efectuadas em cada um dos tipos de problemas (*i.e.*, violação de integridade referencial e violação de restrição de integridade).

8.6.1 Violação de Integridade Referencial

Operações de Detecção

Há excepção do par de tabelas *Dadores – Analises* em que se estabelece um relacionamento de *um para um*, em todos os outros pares, o relacionamento existente é de *um para muitos*. Com o objectivo de identificar eventuais violações de integridade referencial entre cada par de tabelas,

efectuou-se as OD respectivas. A título ilustrativo, a seguir apresenta-se uma dessas OD (no caso, para o relacionamento existentes entre as tabelas de *Dadores* e *Brigadas*).

```
DETECT REFERENTIAL-INTEGRITY-VIOLATION
ON Local FROM Dadores OF BDD
DEPENDENT ON CodigoBrigada FROM Brigadas
```

Da execução das OD respectivas não resultou a detecção de qualquer PQD do tipo em questão entre os pares de tabelas *Dadores – TipoDadiva*; *Dadores – Conclusões*; *Dadores – Sexo*; *Dadores – EstadoCivil*; *Dadores – Profissoes*; *Dadores – CodigosPostais*; *Dadores – Analises*; *Dadores – Colheitas*; *Colheitas – Análises*; *Colheitas – TipoDadiva*; e, *Colheitas – Hospitais*.

Na Tabela 8.39 apresentam-se algumas das 37 violações de integridade referencial resultantes da execução da OD respectiva no par de tabelas *Dadores – Brigadas*. De referir que todas as violações detectadas surgem como consequência da existência do valor *QM* no atributo *Local*.

Tabela 8.39 – Resultados da OD de violação de integridade referencial entre *Dadores* e *Brigadas*

ID	Local
17449	QM
17458	QM
17469	QM
17474	QM
17479	QM

Na Tabela 8.40 apresentam-se as violações de integridade referencial detectadas resultantes da execução da OD respectiva no par de tabelas *Colheitas – Inutilizacoes*.

Tabela 8.40 – Resultados da OD de violação de integridade referencial entre *Colheitas* e *Inutilizacoes*

ID	Inut
72662	34
11758	74

Operações de Correcção

A resolução da violação de integridade referencial entre as tabelas *Dadores* e *Brigadas* consistiu na inserção na tabela *Brigadas* de um tuplo com o valor do atributo *CodigoBrigada* igual a *QM* e o valor do atributo *DesignacaoBrigada* igual a *Desconhecido*.

A correcção da violação de integridade referencial entre as tabelas de *Colheitas* e *Inutilizacoes* seguiu a mesma abordagem do que a utilizada na resolução das violações de integridade referencial entre as tabelas anteriores. Assim, inseriram-se dois tuplos na tabela *Inutilizacoes* com o atributo *CodigoInutilizacao*, respectivamente, igual a *34* e *74* e o atributo *DescricaoInutilizacao* igual a "*Causa desconhecida*". Ainda que longe de ser a solução ideal, esta abordagem soluciona as violações à integridade referencial. No futuro, poderia vir a ser dada um outro tratamento a estas correcções que, de momento, podem ser consideradas como provisórias.

8.6.2 Violação de Restrição de Integridade

Operações de Detecção

Entre as tabelas *Dadores* e *Análises* executou-se uma OD de violação de restrição de integridade que visou identificar se para os mesmos valores dos atributos *Data* e *Dador* de cada uma das tabelas, os valores do atributo *NumColh* em ambas as tabelas são diferentes. A OD inerente a esta verificação é a seguir apresentada:

```
DETECT INTEGRITY-CONSTRAINT-VIOLATION
AT LEVEL OF MULTIPLE RELATIONS
ON T1.Data, T1.Dador, T1.NumColh, T2.Data, T2.Dador, T2.NumColh FROM Dadores
    T1, Analises T2 OF BDD
WHERE T1.Data=T2.Data AND T1.Dador=T2.Dador AND T1.NumColh<>T2.NumColh
```

A execução desta OD não redundou na detecção de qualquer violação à restrição de integridade enunciada.

Entre as tabelas de *Colheitas* e *Analises* executou-se uma OD de violação de restrição de integridade com os mesmos objectivos da anterior. Desta forma, pretendia-se verificar a existência de valores diferentes no atributo *NumColh* para os mesmos valores dos atributos *Data* e *Dador* de cada uma das tabelas. Da execução desta OD de violação de restrição de integridade, resultaram os PQD que se apresentam na Tabela 8.41.

Tabela 8.41 – Resultados da OD de violação de restrição de integridade entre *Colheitas* e *Analises*

T1.ID	T2.ID	T1.Data	T1.Dador	T1.NumColh	T2.NumColh
139156	53489	2001-06-05	35296	65264	65262
139157	53483	2001-06-05	29389	65265	65263
139160	53540	2001-06-05	51495	65268	65266
139163	53474	2001-06-05	20376	65271	65269
139164	53481	2001-06-05	29078	65272	65270
139200	54902	2001-06-19	66499	65296	65295
189407	61979	2001-08-28	27239	65408	65407
189700	72920	2001-12-04	38506	66561	65561

Na tabela anterior, as duas primeiras colunas contêm os valores das chaves primárias dos tuplos, respectivamente, das tabelas *Colheitas* e *Analises*, onde residem as violações à restrição de integridade. A terceira e quarta coluna contêm os valores em que há igualdade entre os atributos *Data* e *Dador* de ambas as tabelas. As duas últimas colunas apresentam as diferenças identificadas nos valores referentes ao número da colheita (*i.e.*, atributo *NumColh*) em ambas as tabelas, para o mesmo conjunto de valores formado pelos atributos *Data* e *Dador* de cada tabela.

Entre as tabelas *Dadores* e *Colheitas*, executou-se uma OD de violação de restrição de integridade com o objectivo de identificar se para os mesmos valores dos atributos *Data* e *Dador* de cada uma das tabelas, os valores do atributo *Abo* de ambas as tabelas diferem. A execução desta OD não culminou na detecção de qualquer PQD.

Ainda entre as tabelas *Dadores* e *Colheitas*, executou-se uma OD de violação de restrição de integridade que visou identificar se os valores do atributo *Rh*, de ambas as tabelas, são diferentes para os mesmos valores dos atributos *Data* e *Dador* de cada uma das tabelas. Novamente, da execução da OD não resultou a detecção de qualquer violação a esta restrição de integridade.

Operações de Correcção

Na correcção das violações de restrição de integridade detectadas entre as tabelas de *Colheitas* e *Analises*, executou-se a OC que a seguir se apresenta.

```
CORRECT INTEGRITY-CONSTRAINT-VIOLATION
AT LEVEL OF MULTIPLE RELATIONS
ON T1.Data, T1.Dador, T1.NumColh, T2.Data, T2.Dador, T2.NumColh FROM Colheitas
    T1, Analises T2 OF BDD
SET T2.NumColh = T1.NumColh
```

Nota: Apenas são alvo desta OC os tuplos da tabela *Analises* em que o PQD foi detectado.

O valor do atributo *NumColh* é definido no momento da colheita e registado na respectiva tabela (*i.e.*, *Colheitas*). Como tal, considera-se que este valor merece uma maior credibilidade do que o que se encontra no atributo respectivo da tabela *Analises*. Assim, o atributo *NumColh* desta tabela é actualizado com o valor do atributo *NumColh* da tabela *Dadores*. Note-se que para a mesma combinação de valores dos atributos *Data* e *Dador*, o valor do atributo *NumColh* tem de ser forçosamente o mesmo em ambas as tabelas.

8.7 Limpeza de Dados ao Nível Multi-Relação de Diferentes Bases de Dados

Por último, são apresentadas as operações de LD envolvendo simultaneamente tabelas da BDD e tabelas de uma outra BD.

Operações de Detecção

Na execução destas operações de LD recorreu-se à BD que no *SmartClean* suporta o armazenamento dos diversos domínios de valores válidos dos atributos (denominada de *Domínios*). Estes domínios encontram-se armazenados sob a forma de tabelas. Em particular, tirou-se partido da tabela que armazena o domínio dos códigos postais (denominada de *Codigos_Postais*) para efectuar as verificações de integridade dos valores do atributo *LocPostal* das tabelas *CodigosPostais* e *Dadores*. Assim, em qualquer uma destas tabelas, o valor do atributo *LocPostal* tem de pertencer ao conjunto de valores do atributo *Localidade* da tabela *Codigos_Postais* para o código postal em questão (*i.e.*, valor do atributo *CodPostal* da tabela *CodigosPostais* ou *Dadores* igual ao valor do atributo *CP* da tabela *Codigos_Postais*). Caso isto não aconteça, está-se perante uma violação de restrição de integridade. Por exemplo, na tabela *Codigos_Postais* (BD *Domínios*) há, unicamente, duas localidades (atributo *localidade*) válidas para o código postal (atributo *CP*) *4445*: *Alfena* e *Ermesinde*. A existência do valor *Gandra* no atributo *LocPostal* da tabela *CodigosPostais* (BDD) para um valor do atributo *CodPostal* igual a *4445* representa uma violação à restrição de integridade enunciada.

A OD que envolve a tabela *CodigosPostais* (da BDD) e a tabela *Codigos_Postais* (da BD *Domínios*) é a seguir apresentada.

```
DETECT INTEGRITY-CONSTRAINT-VIOLATION
AT LEVEL OF MULTIPLE RELATIONS
ON CodPostal, LocPostal, CP FROM CodigosPostais, Codigos_Postais OF BDD, Domínios
WHERE CodPostal = CP AND LocPostal NOT IN (SELECT Localidade
                                           FROM Codigos_Postais
                                           WHERE CP = CodPostal)
```

Da execução desta OD resultaram 129 violações de restrição de integridade. Na Tabela 8.42 apresentam-se algumas das violações detectadas.

Tabela 8.42 – Resultados da OD de violação de restrição de integridade entre *CodigosPostais* e *Codigos_Postais*

ID	CodPostal	LocPostal
40	2000	ARNEIRO DAS MILHEIRICAS
340	3050	CASAL COMBA
350	3060	POCARICA
628	4415	SEIXO ALVO
1513	5100	CAMBRES
1841	6420	VILA FRANCA DAS NAVES
1872	7340	ARRONCHES
1943	8985	MARTINLONGO
1961	9555	GINETES

A execução de uma OD com igual objectivo entre a tabela *Dadores* (BDD) e a tabela *Codigos_Postais* (BD *Domínios*) redundou na identificação de 516 violações de restrições de integridade. Algumas dessas violações de restrição de integridade encontram-se na Tabela 8.43.

Tabela 8.43 – Resultados da OD de violação de restrição de integridade entre *Dadores* e *Codigos_Postais*

ID	CodPostal	LocPostal
92938	2400	MARRAZES
91084	3060	CORTICEIRA DE CIMA
97698	4415	CARVALHOS
33089	4460	RATES
95783	4750	ALVELOS
82975	4800	VERMIL
29747	4815	INFIAS
4688	7480	AVIZ
56058	9370	CALHETA

Operações de Correcção

Na correcção das violações de restrição de integridade detectadas entre as tabelas *CodigosPostais* (BDD) e *Codigos_Postais* (BD *Domínios*) executou-se a OC que a seguir se apresenta.

```
CORRECT INTEGRITY-CONSTRAINT-VIOLATION
AT LEVEL OF MULTIPLE RELATIONS
ON CodPostal, LocPostal, CP, Localidade FROM CodigosPostais, Codigos_Postais OF
    BDD, Domínios
SET LocPostal = Localidade
FROM DO103 T1, (SELECT T1.CP, Localidade
                FROM Domínios.Codigos_Postais T1,(SELECT CP, COUNT(*)
                                                  FROM Domínios.Codigos_Postais
                                                  GROUP BY CP
                                                  HAVING COUNT(*) = 1) T2
                WHERE T1.CP = T2.CP) T3
WHERE T1.CodPostal = T3.CP
```

A semântica que se encontra subjacente a esta OC é a seguir descrita. Para cada violação de restrição de integridade identificada (encontra-se na tabela resultante da OD que no caso é *DO103*) é utilizado o valor respectivo do atributo *CodPostal* para efectuar um *join* com o atributo *CP* (tabela *Codigos_Postais*) resultante de um inquérito que identificou os códigos postais que correspondem a uma só localidade. Nestes casos, o valor respectivo do atributo *Localidade* é

utilizado na actualização do atributo *LocPostal*. Nos restantes casos (*i.e.*, mais de uma localidade para o mesmo valor de código postal), o valor deste atributo permanece inalterado, uma vez que há mais do que uma possibilidade de correcção.

Na correcção das violações de restrição de integridade detectadas entre as tabelas *Dadores* (BDD) e *Codigos_Postais* (BD *Domínios*) executou-se uma OC idêntica à anterior.

Operações de Detecção (nova iteração)

Da nova execução da OD de violação de restrição de integridade entre as tabelas de *CodigosPostais* (BDD) e *Codigos_Postais* (BD *Domínios*) resultou a identificação de 37 problemas deste tipo. Assim sendo, a OC solucionou 92 violações de restrição de integridade. Após uma análise às violações detectadas, constatou-se que nos valores do atributo *LocPostal* se encontravam lugares em vez de localidades (*e.g.*: *Carvalhos* é um lugar da localidade de *Pedroso*). A OC não solucionou estes problemas em virtude de existir mais do que uma localidade possível para o mesmo valor de código postal. Uma vez que não se conseguiu encontrar uma BD que contivesse informação sobre os lugares e as respectivas localidades a que estes pertencem[20], não foi possível continuar a solucionar os PQD por via automática. Como o número de violações ainda existente não era excessivamente elevado, recorreu-se à via manual para proceder à sua correcção. A abordagem seguida foi do género da descrita no contexto do atributo *CodPostal* da tabela *CodigosPostais*.

A nova execução da OD de violação de restrição de integridade entre as tabelas de *Dadores* (BDD) e *CodigosPostais* (BD *Domínios*) redundou na identificação de 180 problemas deste tipo. Isto significa que a OC respectiva solucionou 336 violações de restrição de integridade. O motivo da existência destes PQD é o mesmo do que o referido no parágrafo anterior. Como se trata de um número demasiado elevado de violações à restrição de integridade para ser manipulado por via manual, executou-se a OC que a seguir se apresenta.

```
CORRECT INTEGRITY-CONSTRAINT-VIOLATION
AT LEVEL OF MULTIPLE RELATIONS
ON CodPostal, LocPostal, CP, Localidade FROM CodigosPostais, Codigos_Postais OF
    BDD, Domínios
SET LocPostal = LocalidadeMaisFrequente()
```

Nota: Apenas são alvo da OC os tuplos da tabela *CodigosPostais* em que se detectou o PQD.

[20] Mesmo que tivesse sido possível, haveria situações não susceptíveis de manipulação automática, uma vez que o mesmo lugar pode corresponder a mais do que uma localidade. Todavia, diminuiria ao número de situações que requerem intervenção manual.

Como se pode observar, esta OC envolve a execução de uma função definida pelo utilizador (no caso, *LocalidadeMaisFrequente*). A semântica subjacente a esta operação é a seguir descrita. Para cada violação à restrição de integridade que envolve os atributos *CodPostal* e *LocPostal*, o valor deste último atributo é substituído pela localidade que apresenta maior frequência (*i.e.*, número de ocorrências) relativamente ao valor do atributo *CodPostal* em questão. No caso de existir mais do que uma localidade nestas condições, a função aleatoriamente retorna uma destas. Naturalmente, as correcções efectuadas desta forma não apresentam o mesmo grau de credibilidade do que as realizadas manualmente.

8.8 Conclusão

Neste capítulo apresentou-se um estudo de caso com o objectivo de evidenciar o interesse e validade da ferramenta *SmartClean* e do modelo proposto de LD, de acordo com o qual esta foi desenvolvida. Para o efeito, executou-se um conjunto alargado de OD (*i.e.*, 104) e respectivas OC (*i.e.*, 22) sempre que se identificaram PQD na BD que suportou a realização do estudo (*i.e.*, a BDD). De referir que ambos os tipos de operações foram especificados pelo autor desta dissertação que não possui conhecimentos específicos sobre o domínio (*i.e.*, dádivas de sangue) em geral e, em particular, sobre a BD em questão. Um perito da área teria certamente especificado OD adicionais e eventualmente efectuado correcções diferentes. Ainda assim, cumprem perfeitamente a finalidade com que foram criadas, *i.e.*, comprovar a utilidade e validade do *SmartClean* na DC dos PQD.

A situação ideal era que o presente caso de LD tivesse permitido demonstrar todas as potencialidades de DC disponíveis no *SmartClean*, resultantes da implementação das respectivas operações formalizadas nos Capítulos 5 e 6. No entanto, num caso real esta aspiração não é realista. Dependendo da BD, há operações que justificam serem executadas, enquanto outras não. Na BDD foi precisamente isso que aconteceu, *i.e.*, nem todas as OD e OC disponíveis puderam ser executadas (*e.g.*: na BD em questão não fazia o mínimo sentido a execução da OD de circularidade entre tuplos). Como consequência, algumas potencialidades do *SmartClean* na DC de certos PQD não puderam ser demonstradas. Ainda assim, o número de operações executadas foi significativo, o que permitiu ilustrar as capacidades da ferramenta de identificar e solucionar diferentes tipos de PQD. A demonstração de todas as potencialidades do *SmartClean* só seria possível de alcançar considerando outros casos de LD. No futuro, este é um dos trabalhos que será efectuado, de modo a certificar que a ferramenta suporta diferentes requisitos de LD existentes em variadas BD.

Um aspecto que não se referiu ao longo do capítulo prende-se com o tempo de execução das OD e OC. No geral, não há informação individualizada sobre o tempo de execução de cada operação. De acordo com o modelo proposto de LD, as OD e OC são executadas por NG. Em cada NG, as operações que não possuem dependências entre si são executadas em simultâneo ou paralelo. Quando há dependências, as operações são executadas em sequência. Actualmente, o *SmartClean* não fornece informação sobre o instante de tempo em que uma terminou e se iniciou a seguinte. Assim, o único tipo de análise que pode ser efectuado aos tempos de execução é considerando o conjunto de operações em cada NG. Apenas foi efectuada análise às OD, uma vez que as OC têm um campo de aplicação limitado (*i.e.*, apenas aos tuplos onde se detectaram PQD), daí que o seu tempo de execução em cada NG nunca tenha ido além de uns breves segundos (*i.e.*, menos de cinco segundos). Os resultados apresentados na Tabela 8.44 dizem respeito à primeira execução das OD, *i.e.*, à iteração que implica o maior tempo de execução, em virtude de todos os tuplos das relações serem considerados.

Tabela 8.44 – Tempo necessário à execução das OD em cada NG

NG	N.º OD	Tempo execução (mm:ss)
Atributo – valor individual	62	6:08
Atributo – multi-valor	10	0:23
Tuplo	7	0:42
Relação	7	58:09
Multi-relação da BD	16	2:17
Multi-relação de diferentes BD	2	0:11

Claramente, o NG da relação foi aquele em que as OD demoraram mais tempo a serem executadas. Note-se que neste NG foram executadas quatro OD de tuplos duplicados e duas de violação de dependência funcional. As tabelas *Colheitas*, *Dadores* e *Analises* possuem um número considerável de tuplos, pelo que a realização de um produto Cartesiano de comparações entre estes, mesmo que parcial, é sempre morosa. Por outro lado, as OD de violação de dependência funcional também evidenciaram ser de execução demorada. Em todos os outros NG, face ao número de OD executadas, os tempos acabaram por ser reduzidos. O tempo total despendido na execução das OD foi de uma hora, sete minutos e cinquenta segundos. Tendo em conta o número de operações executadas, este valor afigura ser perfeitamente aceitável. Vários factores influenciam o tempo de execução das operações de LD. Entre estes encontram-se o desempenho

do motor da BD e as características do computador onde estas foram executadas. Neste estudo de caso de LD utilizou-se um computador com potencialidades medianas: *Pentium* IV; 3.0 GHz; 2 GBytes de RAM. Num computador com outras potencialidades, o tempo de execução seria ainda inferior.

O tempo global despendido na DC dos PQD engloba, não só este tempo de execução das operações, mas também a ambientação ao domínio e à BDD, a identificação das OD que faziam sentido serem executadas e respectiva especificação, a definição das estratégias de correcção para os PQD detectados e respectiva especificação das operações, bem como a implementação de certas funções na linguagem de programação *Java*. O tempo global despendido na LD da BDD rondou os dois dias de trabalho (sensivelmente, 20h).

Ainda que a exposição deste caso não o evidencie, a realização do processo de LD envolveu múltiplas iterações. Esta é uma característica diferenciadora do modelo de LD que se propõe e que se encontra implícito no *SmartClean*. Assim, sempre que é efectuada uma correcção como consequência de um PQD identificado, a detecção reinicia-se com a execução da OD que o originou. No caso de estudo, as correcções efectuadas no seguimento dos PQD detectados pela primeira iteração das OD, no geral, solucionaram todos esses problemas de imediato. Apenas em três situações, a segunda iteração das OD voltou novamente a identificar a existência de PQD. A resolução dos PQD relativos a uma destas situações envolveu a especificação de uma nova OC. Nas outras duas situações, a resolução dos PQD respectivos obrigou a uma intervenção manual. Este caso de LD permitiu confirmar aquilo que à partida já se sabia. Por mais OC que possam ser disponibilizadas, em certos PQD não há outra alternativa à sua resolução que não seja uma manipulação individualizada (*i.e.*, caso a caso) pelo ser humano.

Quando todos os PQD estão solucionados num NG, a execução das OD reinicia-se novamente desde o nível mais elementar, para os valores que entretanto sofreram alterações. O objectivo é identificar potenciais problemas que possam ter sido inadvertidamente introduzidos. Esta é outra característica do modelo de LD proposto sobre o qual o *SmartClean* se encontra alicerçado. Esta característica não é evidenciada no estudo de caso efectuado, uma vez que desse reinício de execução das OD nunca resultou a identificação de qualquer outro problema que tivesse sido introduzido involuntariamente a "montante", fruto das correcções entretanto efectuadas.

Apenas com esta abordagem iterativa e exaustiva de DC que faz parte do modelo de LD proposto é possível assegurar que os dados se encontram "limpos", *i.e.*, isentos de PQD. O presente caso de estudo permitiu demonstrar a validade desse modelo de LD.

INTEROPERABILIDADE DAS OPERAÇÕES DE LIMPEZA DE DADOS

9

Neste capítulo começa-se por descrever detalhadamente a abordagem que permite a interoperabilidade das operações de Limpeza de Dados (LD) entre Bases de Dados (BD) diferentes. De seguida, é apresentada uma das suas principais potencialidades que consiste na especificação das operações conceptuais de LD (*i.e.*, ao nível da ontologia) a diferentes níveis de granularidade de instanciação. Outra das principais potencialidades, envolvendo a herança e sobreposição das operações conceptuais de LD, é exposta imediatamente a seguir. Por último, é apresentado o modelo de dados que suporta o armazenamento e manipulação do conhecimento e dos meta-dados em que a abordagem se encontra alicerçada.

9.1 Introdução

A LD obriga a que o utilizador especifique manualmente as operações de Detecção e Correcção (DC) a efectuar. O utilizador começa por definir as Operações de Detecção (OD) que pretende executar. Após a execução destas, o utilizador especifica as operações de correcção automáticas a realizar nos dados para solucionar os problemas identificados. Em alternativa, o utilizador pode decidir ou ser forçado a corrigir manualmente alguns dos problemas (caso a caso), em virtude de não serem susceptíveis de correcção automática.

Normalmente, a realização de um processo de LD envolve a especificação manual de um número apreciável de operações de DC. Infelizmente, não há uma solução mágica para a LD. Um sistema informático completamente automático que receba dados, detecte e corrija os problemas de qualidade existentes nestes, sem obrigar a qualquer intervenção por parte do utilizador é uma miragem. Há aspectos semânticos envolvidos, daí que não seja possível transformar a LD num processo totalmente automático. É necessário saber qual o conteúdo semântico de um atributo para que seja possível efectuar as operações de DC adequadas (*e.g.*: se um determinado atributo armazena códigos postais, então justifica-se a realização de operações que detectem violações à sua sintaxe e ao seu domínio). Normalmente, o nome do atributo disponível nos metadados de uma BD não é suficiente para concluir acerca do seu conteúdo. Por vezes, o nome do atributo é muito telegráfico senão mesmo opaco, *i.e.*, não possui qualquer significado (*e.g.*: *atrib1*), pelo que nada se pode concluir quanto ao que armazena. Por outro lado, a especificação do que se

considera um Problema de Qualidade de Dados (PQD) e a forma de como o solucionar são também tarefas que dependem do utilizador. No exemplo dado anteriormente, o utilizador tem de especificar o que é considerado uma violação de sintaxe e uma violação de domínio ao código postal. Para os problemas de sintaxe e de domínio detectados é necessário que o utilizador especifique a forma de os corrigir automaticamente (caso esta exista). Por estes motivos, a especificação das operações de DC por parte do utilizador é algo de inultrapassável, apesar de constituir uma tarefa altamente trabalhosa. Em suma, o utilizador necessita de especificar os problemas que pretende detectar, a forma de os detectar, bem como o modo de os corrigir.

A abordagem seguida nos protótipos de investigação e nas ferramentas comerciais actualmente existentes passa pela especificação das operações de DC ao nível do esquema dos dados. As operações são especificadas com base nos nomes dos atributos, das tabelas e das BD. Esta mesma abordagem foi adoptada no âmbito do modelo de LD concebido para a especificação das operações de LD. A abordagem "prende" as operações ao esquema de uma determinada BD, sendo perfeitamente adequada caso estas sejam apenas para executar nessa BD. No entanto, frequentemente isto não corresponde à realidade. A generalidade das operações de LD são genéricas o suficiente para poderem ser executadas em diferentes BD. Estas operações podem estar limitadas a BD que pertençam ao mesmo domínio ou podem ser de tal forma genéricas que são independentes deste (*e.g.*: uma dada organização pode ter várias BD acerca das suas actividades, organizadas em função da sua distribuição geográfica pela várias delegações comerciais; uma empresa de consultoria na área da Qualidade de Dados (QD) enfrenta os mesmos PQD inúmeras vezes em BD de diferentes clientes cujo ramo de actividade é o mesmo). Nestes casos, a especificação das operações ao nível do esquema dos dados não constitui a abordagem apropriada. As mesmas operações ou, pelo menos, subconjuntos destas fazem sentido ser aplicadas em diferentes BD pertencentes ao mesmo domínio. Atendendo a que as operações foram especificadas para uma BD em concreto, efectuar a sua reutilização noutras BD não é fácil. Diversas alterações são necessárias para que seja possível reutilizar uma operação de limpeza noutra BD, uma vez que os nomes das tabelas e atributos normalmente não são os mesmos. No caso de tabelas ou atributos estarem em falta, as operações de limpeza correspondentes têm de ser eliminadas. Isto faz com que as alterações a efectuar se revistam ainda de maior dificuldade, não constituindo meras operações de "localizar e substituir".

Neste capítulo apresenta-se uma extensão ao modelo de LD exposto no Capítulo 7 que suporta a interoperabilidade das operações de DC entre BD diferentes, de acordo com o proposto em [Oliveira *et al.*, 2006c]. A extensão apresentada consiste numa abordagem que tira partido das

operações de LD especificadas de uma forma abstracta/conceptual ao nível de uma *ontologia*[21] do domínio. Estas operações encontram-se independentes do esquema de qualquer BD. Tendo por base um conjunto de mapeamentos definidos pelo utilizador, entre o esquema de uma BD e a ontologia do domínio, um conjunto de operações conceptuais é automaticamente instanciado à BD em causa, sendo a sua execução proposta ao utilizador. A abordagem permite que as operações de limpeza conceptuais possam ser facilmente reutilizadas em diferentes BD.

9.2 Descrição da Abordagem Proposta

A abordagem proposta para a interoperabilidade das operações de LD assenta na existência de ontologias dos domínios a que pertencem as BD nas quais as operações são executadas. Tal como se ilustra na Figura 9.1, a representação da realidade, *i.e.*, do mundo real é efectuada sob a forma de ontologias dos domínios (*e.g.*: seguros), compostas por: conceitos de cada domínio (*e.g.*: segurado); propriedades de cada conceito do domínio (*e.g.*: código postal); e, relacionamentos entre os conceitos do domínio (*e.g.*: segurado possui apólice).

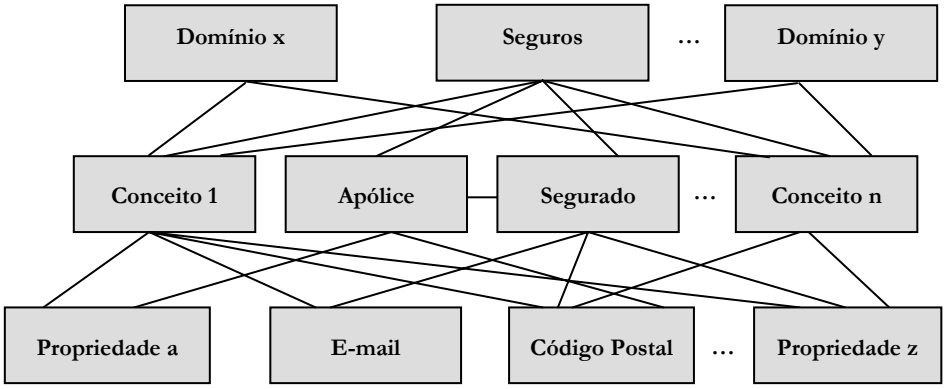


Figura 9.1 – Representação da realidade através de ontologias

Apesar da criação da ontologia do domínio constituir o primeiro passo da abordagem proposta, as dificuldades inerentes a essa criação não constituem preocupação no âmbito da abordagem, embora se reconheça a sua importância. O objectivo consiste, apenas, em detalhar a ideia de usar ontologias como forma de alcançar a interoperabilidade das operações de LD em BD diferentes.

A interoperabilidade das operações de DC é alcançada mediante a sua especificação abstracta ou conceptual, *i.e.*, ao nível da ontologia do domínio em causa. Para o efeito são usadas as mesmas

[21] Uma *ontologia* é uma teoria de conteúdo sobre os conceitos, propriedades dos conceitos e relacionamentos entre conceitos existentes num determinado domínio de conhecimento [Chandrasekaran, 1999].

linguagens declarativas definidas no Capítulo 5 – Secção 5.2.1 e Capítulo 6 – Secção 6.2.1, apenas com as seguintes diferenças, de modo a permitir a especificação conceptual das operações: a BD é substituída pelo domínio; a tabela é substituída pelo conceito; e, o atributo é substituído pela propriedade. Cada elemento da ontologia (*e.g.*: propriedade) pode participar em diversas operações de LD. No exemplo apresentado na Figura 9.2, duas operações conceptuais de LD (*i.e.*, detecção de violação de sintaxe e de violação de domínio) encontram-se definidas para a propriedade *Código Postal*. A especificação das operações conceptuais de LD constitui o segundo passo da abordagem.

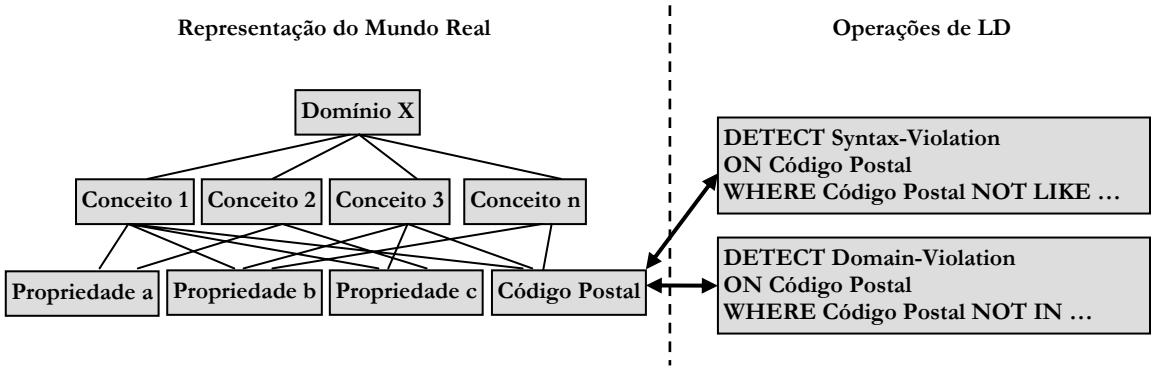


Figura 9.2 – Especificação conceptual das operações de LD

Na BD na qual se pretende detectar e corrigir os PQD, o utilizador estabelece manualmente mapeamentos entre o seu esquema de dados e a ontologia do domínio respectiva. Os mapeamentos entre os dois níveis (*i.e.*, da ontologia e do esquema de dados) são estabelecidos entre: (*i*) uma BD e um domínio; (*ii*) uma tabela e um conceito; e, (*iii*) um atributo e uma propriedade. Um mapeamento entre uma BD e um domínio significa que aquela contém dados deste domínio. Um mapeamento entre uma tabela e um conceito significa que aquela contém dados que correspondem a este conceito. Por último, um mapeamento entre um atributo e uma propriedade significa que aquele contém valores referentes a esta propriedade. A Figura 9.3 ilustra um mapeamento entre o atributo *cp* e a propriedade *código postal*. O estabelecimento dos mapeamentos constitui a terceira etapa na abordagem proposta.

Em função dos mapeamentos estabelecidos, automaticamente são identificadas as operações conceptuais de LD que podem ser executadas, sendo instanciadas ao nível do esquema da BD em causa. No exemplo considerado, as operações de LD associadas à propriedade *código postal* são automaticamente instanciadas em resultado do mapeamento estabelecido, *i.e.*, são transformadas do nível da ontologia (ou nível ontológico) para o nível do esquema da BD em questão. Supondo que no exemplo considerado o nome da tabela é *Segurado*s e o nome da BD é *Seguros*, são automaticamente instanciadas as duas OD apresentadas na Figura 9.4. Estas operações são

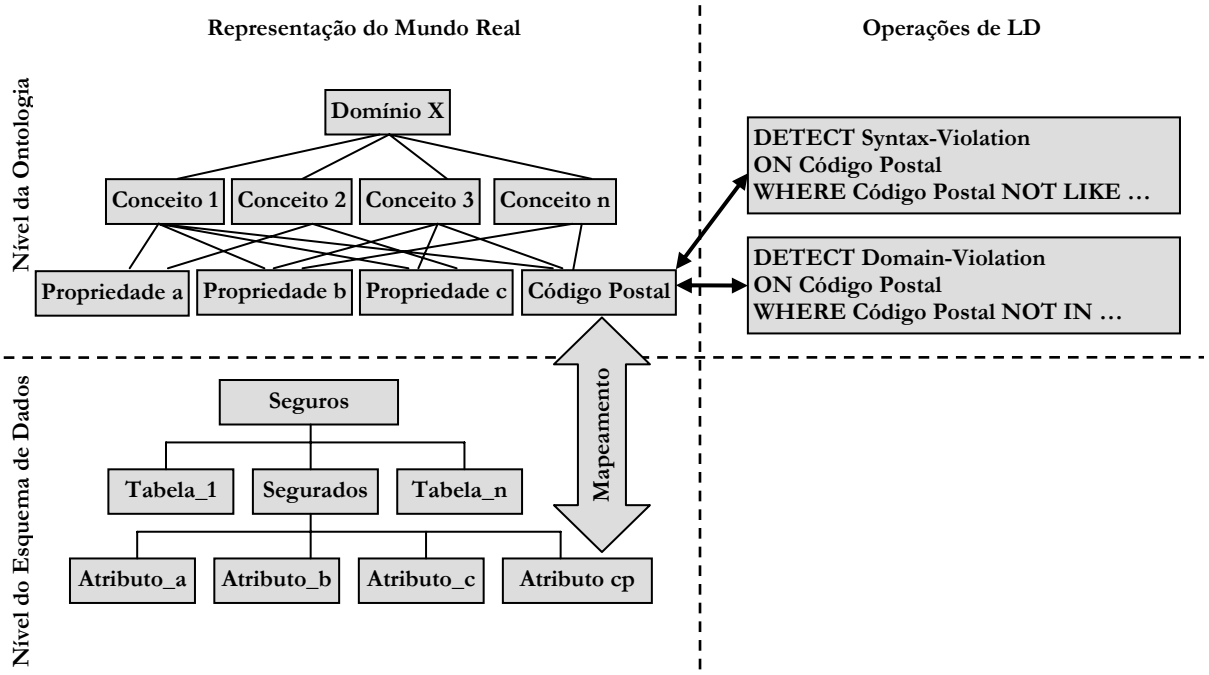


Figura 9.3 – Estabelecimento de um mapeamento entre um atributo e uma propriedade

propostas ao utilizador para execução na BD. A instanciação das operações constitui o último passo da abordagem proposta. Após terem sido instanciadas, as operações de LD são executadas de acordo com a sequência de manipulação preconizada no modelo proposto de LD apresentado no Capítulo 7.

Nas operações de LD ao nível do esquema dos dados, as cláusulas FROM e OF definem, respectivamente, a tabela e a BD onde a operação será executada. De acordo com o definido no

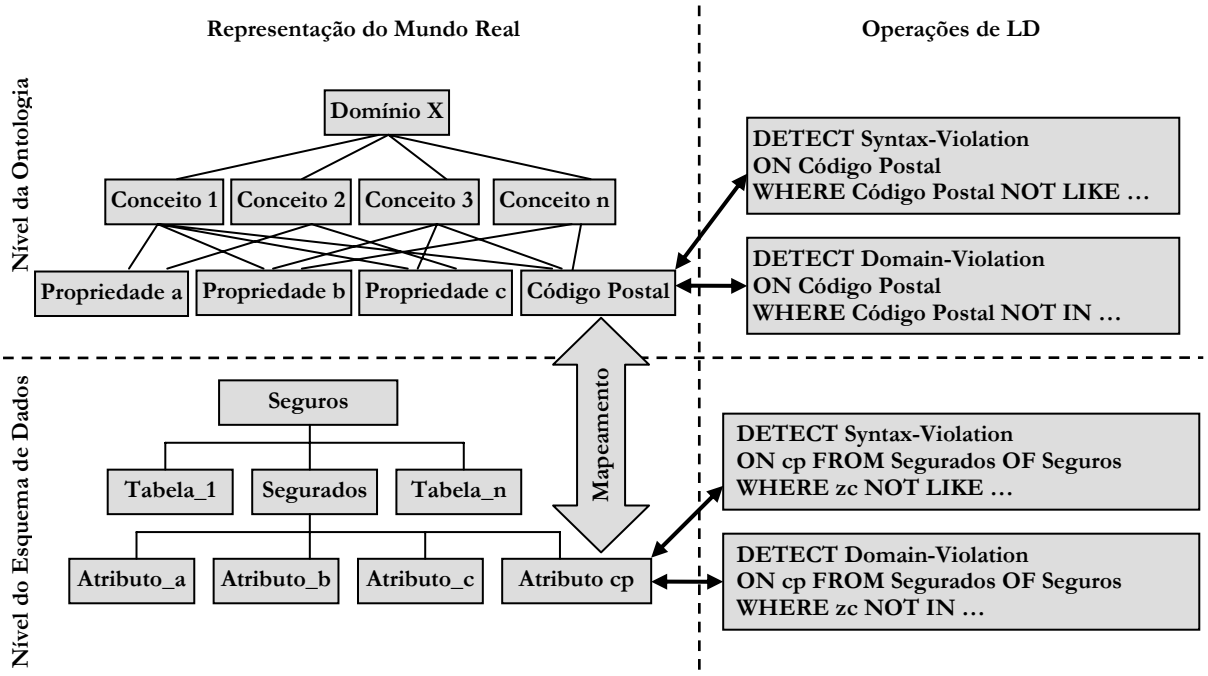


Figura 9.4 – Instanciação das operações conceptuais de LD

Capítulo 5 – Secção 5.2.1 e Capítulo 6 – Secção 6.2.1, ao nível do esquema de dados estas cláusulas são sempre obrigatórias. Ao nível da ontologia, as mesmas cláusulas não surgem nas OD apresentadas no exemplo, o que significa que o seu âmbito de aplicação é genérico o suficiente para ser aplicado em diferentes conceitos de diferentes domínios. Este assunto será abordado na secção seguinte.

O exemplo usado é simplista uma vez que utiliza uma só propriedade. No entanto, a filosofia de funcionamento preconizada na abordagem é a mesma, independentemente de se utilizar uma ou várias propriedades. Caso vários conceitos sejam utilizados na operação de LD conceptual ou ontológica, esta apenas é instanciada caso existam mapeamentos entre todos os conceitos envolvidos e tabelas da BD e, simultaneamente, existam mapeamentos entre as propriedades e os atributos destas tabelas.

Os metadados da BD desempenham um papel importante, uma vez que são utilizados para evitar a instanciação de operações de LD desnecessárias. A título de exemplo considere-se a existência de um atributo definido como de preenchimento obrigatório. Naturalmente, não é possível a existência de valores em falta neste atributo. A instanciação de uma OD de valores em falta no atributo não faz qualquer sentido. Os metadados são úteis uma vez que impedem a instanciação de algumas operações de LD desnecessárias. Apenas se consideram metadados que podem ser obtidos directamente a partir da BD. Outras fontes de metadados (*e.g.*: dicionários de dados) não são consideradas uma vez que podem não estar correctas. Naturalmente, está-se a partir do pressuposto que o "motor" da BD é fiável, *i.e.*, que assegura devidamente este género de restrições. Nas BD comercializadas actualmente este pressuposto parece ser realista.

As operações de LD especificadas ao nível da ontologia são aquelas susceptíveis de reutilização noutras BD. Naturalmente, se uma operação é específica de uma BD (*e.g.*: uma OD que identifica os códigos de produto que violam uma sintaxe muito específica utilizada numa dada empresa), não deve ser representada ao nível da ontologia. A possibilidade de reutilização da operação é muito reduzida, senão mesmo inexistente. Nestes casos, as operações devem ser especificadas directamente ao nível do esquema dos dados, complementando as instanciadas automaticamente. A decisão de especificar uma operação ao nível da ontologia ou apenas ao nível do esquema da BD depende desta poder ser ou não reutilizável noutras BD. Esta avaliação compete única e exclusivamente ao utilizador.

9.3 Granularidade das Operações Ontológicas de Limpeza de Dados

As operações ontológicas ou conceptuais de DC podem ser especificadas a diferentes níveis de granularidade. Em particular, as operações podem ser especificadas ao nível da: (*i*) propriedade (independentemente do conceito e domínio envolvidos); (*ii*) da propriedade num conceito (independentemente do domínio envolvido); e, (*iii*) da propriedade num conceito num domínio.

O primeiro nível é o que apresenta o cariz mais genérico, uma vez que as operações ontológicas de LD são especificadas unicamente ao nível da propriedade. Para que as operações possam ser instanciadas automaticamente é suficiente a existência de mapeamentos para as suas propriedades, independentemente do domínio e conceitos em questão. A título de exemplo, suponha-se a existência de uma operação de violação de sintaxe associada à propriedade *e-mail*. Naturalmente, não importa se é um *e-mail* de um cliente, fornecedor ou funcionário, um *e-mail* tem de respeitar uma sintaxe. Esta operação não depende de conceitos e, muito menos, de um domínio específico. Como tal, o estabelecimento de um mapeamento entre um atributo e esta propriedade é suficiente para a sua instanciação automática, independentemente do domínio e conceitos representados na BD.

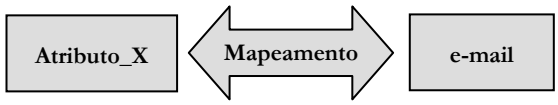
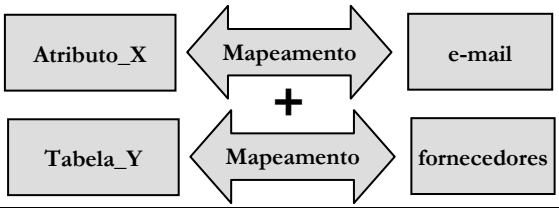
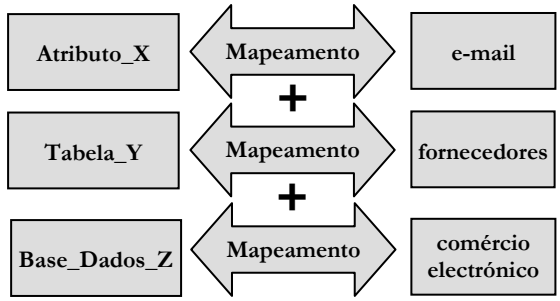
No nível intermédio de granularidade, as operações ontológicas de LD são especificadas para uma propriedade de um determinado conceito. Estas operações só são instanciadas automaticamente caso existam simultaneamente mapeamentos para as suas propriedades e conceitos. O domínio subjacente não é relevante na instanciação das operações. Por exemplo, suponha-se a existência de uma OD de violação de unicidade na propriedade *e-mail* do conceito *fornecedor*. Dois fornecedores não podem ter o mesmo endereço de *e-mail* (a não ser que sejam o mesmo). De qualquer forma, em ambos os casos há um PQD (violação de unicidade ou registos duplicados). A OD apenas é instanciada quando existe um mapeamento entre um atributo e a propriedade *e-mail* e simultaneamente um mapeamento entre a tabela a que pertence o atributo e o conceito *fornecedor*.

Por último, no nível de granularidade mais específico, as operações ontológicas de LD são especificadas para uma propriedade, de um conceito, de um determinado domínio. A sua instanciação automática obriga à existência simultânea de mapeamentos para as suas propriedades, conceitos e domínios. A título exemplificativo, considere-se a existência de uma OD de valor em falta para a propriedade *e-mail*, do conceito *fornecedor*, no domínio do *comércio electrónico*. O *e-mail* é opcional na generalidade dos domínios, mas no caso do comércio electrónico

é obrigatório, uma vez que é o meio de comunicação utilizado para as transacções comerciais. A operação de LD em causa apenas é instanciada quando existe um mapeamento entre um atributo e a propriedade *e-mail* e, ao mesmo tempo, existem mapeamentos entre a tabela e o conceito *fornecedor* e entre a BD e o domínio do *comércio electrónico*.

A Tabela 9.1 sumaria os três níveis de granularidade de especificação das operações ontológicas de LD e os mapeamentos necessários para que possam ser instanciadas ao nível do esquema dos dados.

Tabela 9.1 – Mapeamentos necessários à instanciação das operações conceptuais de LD

Granularidade	Operação Conceptual de LD	Mapeamentos Necessários à Instanciação da Operação
Propriedade	DETECT Syntax-Violation ON e-mail WHERE …	
Propriedade do Conceito	DETECT Uniqueness-Violation ON e-mail FROM fornecedores	
Propriedade do Conceito do Domínio	DETECT Missing-Value ON e-mail FROM fornecedores OF comércio electrónico	

9.4 Herança e Sobreposição das Operações Ontológicas de Limpeza de Dados

O nível ontológico ou conceptual da abordagem proposta permite uma representação hierárquica do mundo real. Assim, um domínio pode ser sub-domínio de outro domínio (*e.g.*: a facturação constitui um sub-domínio da gestão comercial), um conceito pode ser sub-conceito de outro conceito (*e.g.*: cliente constitui um sub-conceito de entidade comercial) e uma propriedade pode ser sub-propriedade de outra propriedade (*e.g.*: apelido constitui uma sub-propriedade de nome). Esta é uma forma natural de modelar o mundo real. Por outro lado, esta organização hierárquica suporta duas importantes características da abordagem: herança e sobreposição das operações de

DC. Estas características são exemplificadas recorrendo a uma propriedade (*código postal*) e a duas sub-propriedades desta (*código postal Português* e *código postal dos Estados Unidos da América* (*E.U.A*)), de acordo com o ilustrado na Figura 9.5.

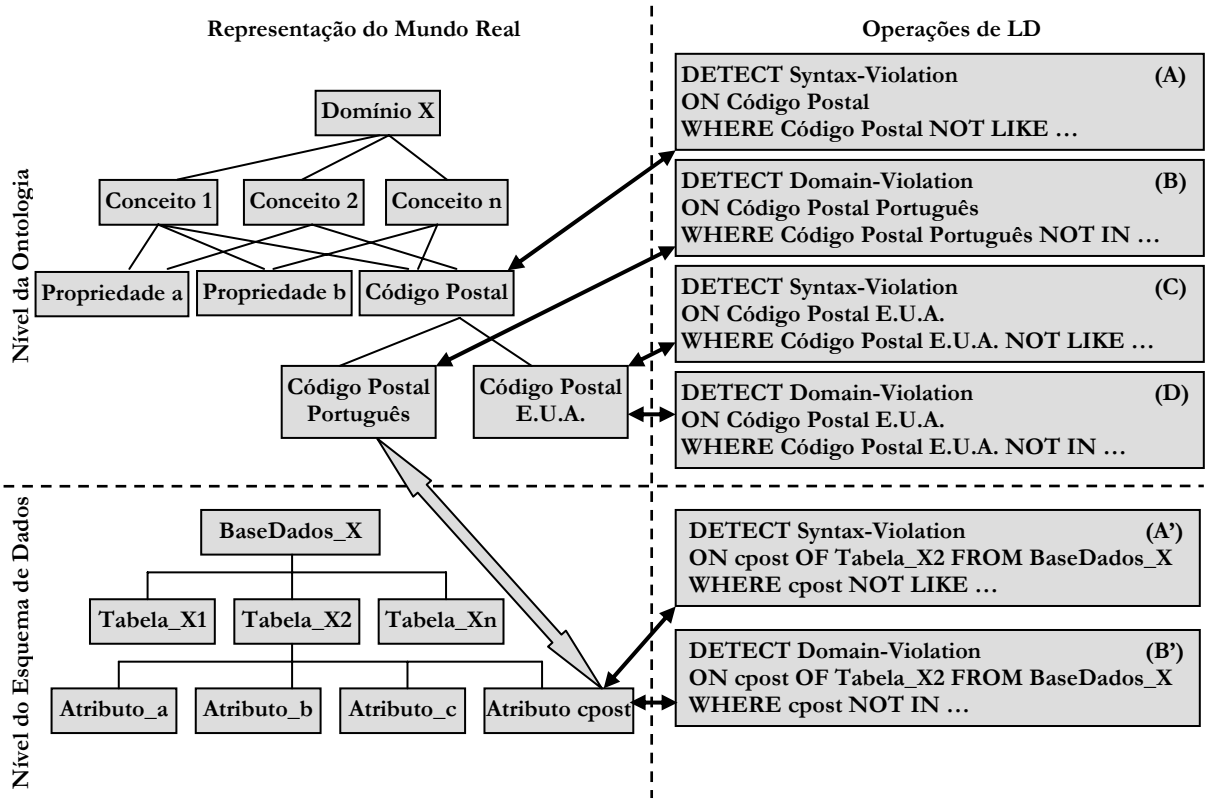


Figura 9.5 – Herança na instanciação das operações conceptuais de LD

Uma OD de violação de sintaxe (operação A) encontra-se especificada para a propriedade *código postal*. Suponha-se que esta operação detecta violações à sintaxe mais utilizada internacionalmente na representação de um código postal. Caso exista um mapeamento entre um dado atributo e esta propriedade, a operação ontológica de LD é instanciada automaticamente para a correspondente operação ao nível do esquema de dados. A propriedade *código postal* encontra-se especializada em duas sub-propriedades: *código postal Português* e *código postal dos E.U.A.* Estas especializações da propriedade *código postal* pretendem cobrir casos específicos. Uma OD de violação de domínio (operação B) encontra-se definida para a primeira sub-propriedade. O objectivo desta operação consiste em identificar os valores que não correspondem um código postal Português existente. Caso exista um mapeamento entre um atributo e esta sub-propriedade (ver Figura 9.5), duas OD são automaticamente instanciadas: a violação de domínio (operação B') e também a violação de sintaxe (operação A'). Esta última operação é herdada da propriedade pai da sub-propriedade (*código postal*).

Como se pode constatar na Figura 9.6, na sub-propriedade *código postal dos E.U.A.* encontram-se especificadas duas OD: uma violação de sintaxe (operação C) e uma violação de domínio (operação D). Suponha-se que os códigos postais dos E.U.A. não correspondem à sintaxe definida ao nível da propriedade *código postal*, o que obriga à especificação de uma nova OD de violação de sintaxe. A detecção de violação de domínio nos códigos postais dos E.U.A tem uma finalidade similar à detecção de violação de domínio nos códigos postais Portugueses. Caso exista um mapeamento entre um atributo e esta última sub-propriedade (ilustrado na Figura 9.6), a OD de violação de sintaxe definida ao nível da sub-propriedade (operação C) sobrepõe-se à OD de violação de sintaxe definida ao nível do *código postal* (operação A), sendo automaticamente instanciada (operação C'). Além desta, a OD de violação de domínio (Operação D) é também instanciada (operação D').

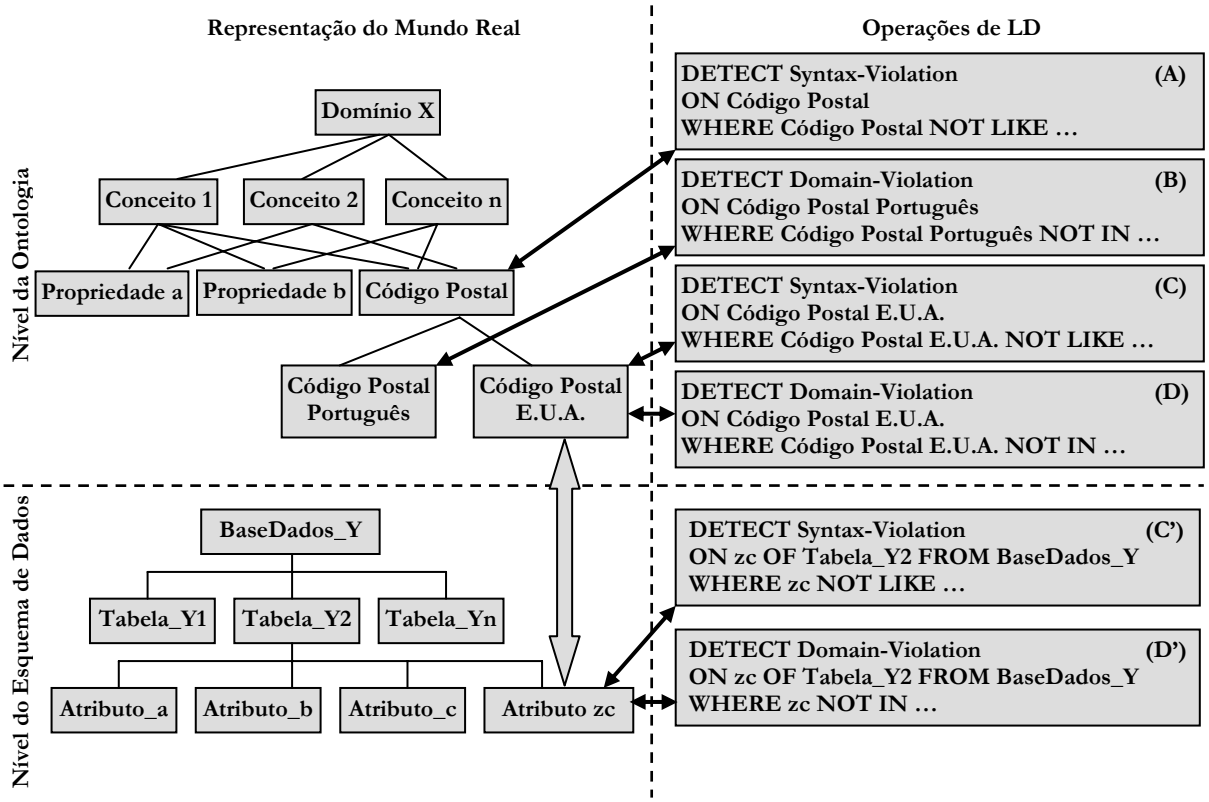


Figura 9.6 – Sobreposição na instanciação das operações conceptuais de LD

A instanciação automática das operações conceptuais de LD obriga a que em cada propriedade ou sub-propriedade esteja especificada uma só operação do mesmo tipo (*e.g.*: violação de sintaxe; violação de domínio). Se tal não acontecer, *i.e.*, se existir mais do que uma operação candidata, compete ao utilizador seleccionar a operação que deve ser instanciada. No exemplo atrás considerado, caso duas OD de violação de sintaxe estivessem associadas à propriedade *código postal* em vez de uma só, teria sido necessária a intervenção do utilizador.

Ainda que não seja uma situação comum, podem ser estabelecidos diversos mapeamentos entre um atributo e várias propriedades (ou sub-propriedades) diferentes. Conceptualmente, isto significa que o atributo armazena valores que correspondem ambas as propriedades. No exemplo anterior, considerou-se a existência de um mapeamento entre o atributo *zipc* e a sub-propriedade *código postal dos E.U.A.* No caso de também existir um mapeamento entre o mesmo atributo e a sub-propriedade *código postal Português*, isto significa que o atributo armazena códigos postais referentes a ambos os países. Quando há mais do que uma operação do mesmo tipo (*e.g.*: detecção de violação de sintaxe) entre as diversas propriedades envolvidas nos mapeamentos, as operações são combinadas, o que resulta na instanciação de uma só operação. A combinação das operações resulta da reunião dos elementos das várias cláusulas. No caso da cláusula *Where* as diversas condições referentes às diferentes operações do mesmo tipo são ligadas pelo operador lógico conjuntivo (*i.e.*, *and*). No exemplo anterior, isto corresponde a reunir as condições que se encontram na cláusula *Where* das operações do mesmo tipo (*i.e.*, operações A e C para a detecção de violação de sintaxe e operações B e D para a detecção de violação de domínio).

A exemplificação da abordagem foi efectuada exclusivamente com base em OD. No entanto, com as devidas adaptações, as OC são objecto do mesmo tipo de manipulação do que as OD.

9.5 Modelo de Dados de Suporte à Interoperabilidade das Operações de Limpeza

Conceptualmente, o modelo de dados que suporta a interoperabilidade das operações de LD encontra-se organizado em quatro quadrantes, tal como se apresenta na Figura 9.7. Estes quadrantes suportam o armazenamento e manipulação de todo o conhecimento (*e.g.*: as ontologias) e dos meta-dados (*e.g.*: informação sobre o esquema dos dados) em que a abordagem assenta.

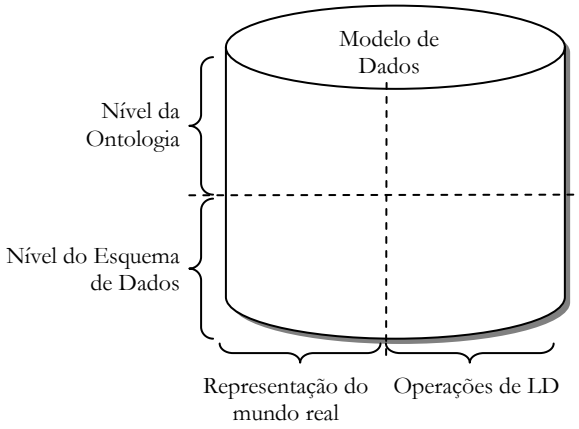


Figura 9.7 – Representação conceptual do modelo de dados

O quadrante superior esquerdo suporta o armazenamento das representações dos diferentes domínios ao nível da ontologia, *i.e.*, suporta o armazenamento das ontologias dos domínios. O quadrante inferior esquerdo também permite armazenar as representações dos diferentes domínios, mas ao nível do esquema da BD. Uma dada ontologia do domínio pode corresponder a diferentes representações ao nível do esquema dos dados. Em particular, um elemento da ontologia (*i.e.*, domínio; conceito; propriedade) pode estar representado sob múltiplas formas ao nível do esquema dos dados (*i.e.*, BD; tabela; atributo) (*e.g.*: a propriedade *Código Postal* pode ser representada como sendo o atributo: *CodigoPostal*; *CodPostal*; *CodigoPost*; *CodPost*; *CP*).

Como explicado na secção anterior, o utilizador estabelece manualmente os mapeamentos entre os dois níveis, *i.e.*, dos elementos do esquema de dados para os elementos da ontologia do domínio respectiva. Estes mapeamentos são armazenados no repositório resultante do modelo de dados aqui proposto. No futuro, pretende-se tirar partido da informação existente sobre estes, para estabelecer automaticamente mapeamentos entre os dois níveis, perante uma nova BD. O objectivo passa por reduzir o número de mapeamentos que o utilizador necessita de especificar manualmente. Os mapeamentos são muito importantes, uma vez que é baseado nestes que as operações conceptuais de LD são instanciadas ao nível do esquema da BD onde vão ser executadas. Por agora, a abordagem baseia-se exclusivamente em mapeamentos manuais. No futuro, perspectiva-se a inclusão de análises de semelhança léxica [Rahm e Bernstein, 2001] entre o esquema da BD que será objecto de LD e a informação existente sobre esquemas de dados no repositório de conhecimento e meta-dados. A título exemplificativo, suponha-se que existe informação neste repositório sobre um mapeamento anteriormente estabelecido entre o atributo *codpost* e a propriedade *código postal*. Como o atributo *codpostal* (de uma nova BD que irá ser sujeita a LD) é lexicalmente similar a *codpost*, automaticamente pode estabelecer-se um mapeamento entre o primeiro e a propriedade *código postal*.

O quadrante superior direito suporta o armazenamento das operações de LD representadas ao nível conceptual, *i.e.*, ao nível da ontologia. Por fim, o quadrante inferior esquerdo permite armazenar as operações de LD ao nível do esquema das próprias BD. As operações de LD efectivamente efectuadas em cada BD são armazenadas a este nível. Caso seja necessário, as operações podem ser facilmente re-executadas, repetindo todo o processo de LD. O repositório de operações de LD considerado na arquitectura apresentada no Capítulo 7 – Secção 7.2 corresponde a este último quadrante.

A Figura 9.8 detalha o modelo de dados através de um diagrama *Unified Modelling Language* (UML) em notação simplificada. O quadrante superior esquerdo contém as classes necessárias para

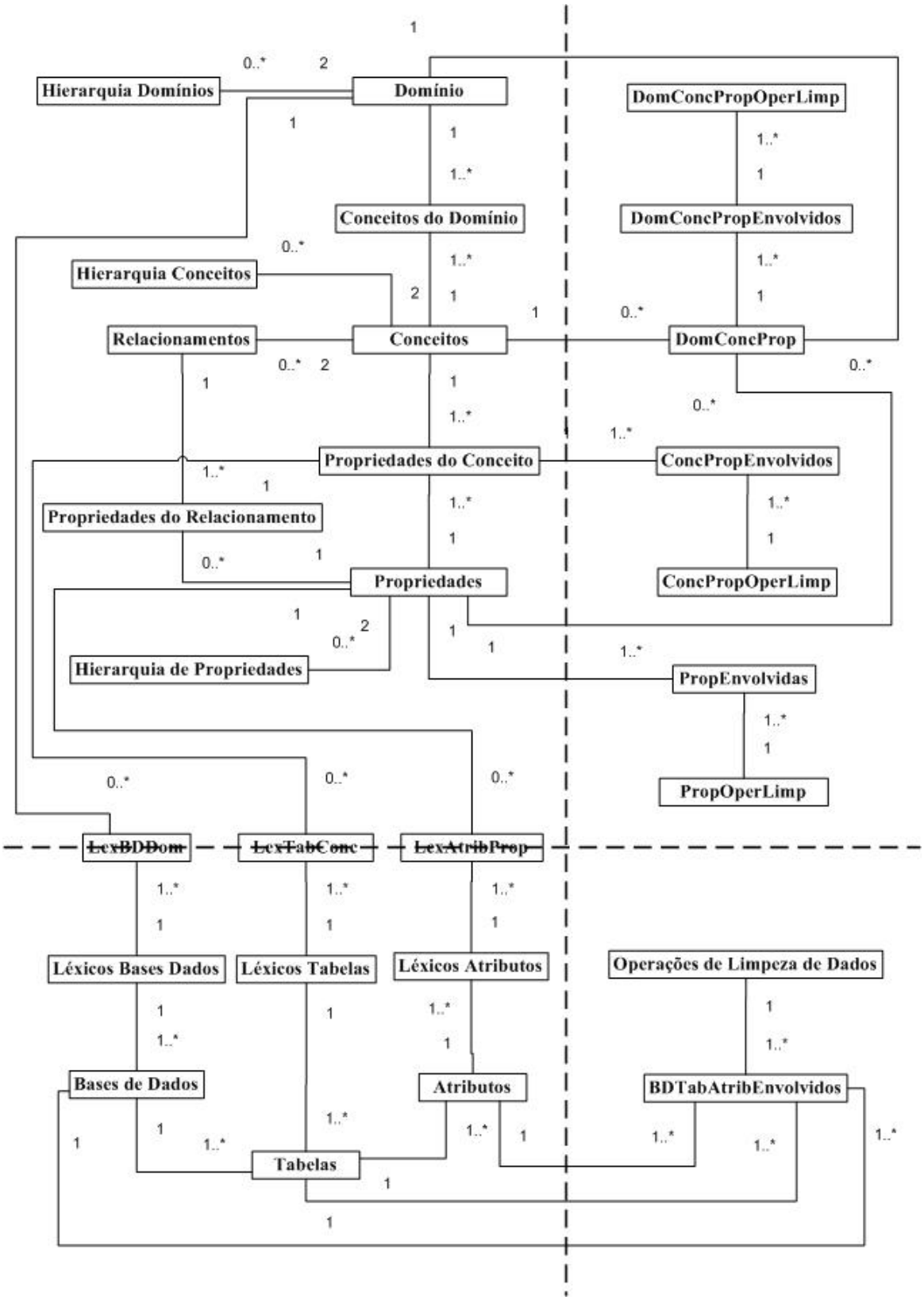


Figura 9.8 – Diagrama UML em notação simplificada do modelo de dados

suportar a manipulação dos domínios, dos conceitos envolvidos, das propriedades dos conceitos e dos relacionamentos entre conceitos. Outras três classes encontram-se incluídas para suportar a

manipulação das possíveis hierarquias existentes ao nível de domínios, conceitos e propriedades. O quadrante superior direito inclui as classes necessárias para suportar a manipulação das operações de LD definidas ao nível da ontologia do domínio. Há classes para suportar a manipulação das operações ontológicas definidas ao nível da: (*i*) propriedade; (*ii*) propriedade do conceito; e, (*iii*) propriedade do conceito do domínio. O quadrante inferior esquerdo contém as classes necessárias para suportar a manipulação dos elementos referentes ao esquema (*i.e.*, nome da BD; nomes das tabelas; nomes dos atributos) das diferentes BD sobre as quais se executaram operações de DC. Como já se referiu, futuramente pretende-se explorar esta informação de modo a estabelecerem-se automaticamente mapeamentos entre os níveis ontológico e do esquema de dados. Há três classes que conectam estes dois níveis do modelo de dados, surgindo no diagrama na fronteira dos respectivos quadrantes. O quadrante inferior direito inclui as classes necessárias à manipulação das operações de LD efectivamente executadas nas BD objecto de melhoria da QD. Estas operações encontram-se representadas ao nível do esquema de cada BD.

9.6 Conclusão

Neste capítulo apresentou-se uma extensão ao modelo de LD proposto no Capítulo 7, baseada numa abordagem que suporta a interoperabilidade das operações de LD entre BD diferentes. Esta interoperabilidade é alcançada à custa da especificação conceptual das operações de LD ao nível da ontologia do domínio. Estas operações tornam-se independentes de um qualquer esquema de BD. As operações são especificadas com base nas linguagens declarativas definidas para a DC dos PQD, mas sendo substituídos os elementos relativos ao esquema dos dados (*i.e.*, BD; tabelas; e, atributos) pelos elementos referentes à ontologia (*i.e.*, domínio; conceitos; e, propriedades). Cada operação de limpeza possui uma granularidade de especificação, indo esta do genérico ao específico. A granularidade de especificação de uma operação de LD pode ser a: (*i*) propriedade; (*ii*) propriedade do conceito; ou, (*iii*) propriedade do conceito do domínio. A organização hierárquica suportada pelo nível ontológico da abordagem permite a herança e sobreposição das operações conceptuais de LD. Os elementos do esquema da BD encontram-se conectados aos elementos da ontologia através de um conjunto de mapeamentos manualmente estabelecidos pelo utilizador. Com base nestes, as operações definidas ao nível conceptual são automaticamente seleccionadas e instanciadas ao nível do esquema da BD em questão. A representação conceptual das operações de LD propicia a sua interoperabilidade em diferentes BD, *i.e.*, permite que a mesma operação abstracta possa ser adaptada a cada situação concreta e reutilizada inúmeras vezes.

O desenvolvimento de uma ontologia do domínio é algo de difícil e complexo. Naturalmente, apenas faz sentido proceder ao seu desenvolvimento se os ganhos excederem os custos. Na abordagem proposta, as ontologias são utilizadas para suportar a especificação das operações a um nível conceptual, e assim permitir a sua reutilização em diferentes BD. Caso as operações sejam de tal forma específicas que apenas possam ser executadas numa única BD, a abordagem proposta não é adequada. Nestes casos, o desenvolvimento de uma ontologia acarreta custos elevados e não há qualquer proveito. A abordagem apropriada consiste em especificar as operações de LD somente ao nível do esquema de dados. Caso a generalidade das operações sejam genéricas o suficiente para serem aplicadas em diferentes BD, então justifica-se o desenvolvimento da ontologia e a especificação das operações a este nível. Nestes casos, os ganhos que advêm da reutilização das operações conceptuais de LD são superiores aos custos.

Algumas operações de LD envolvem a invocação de funções definidas pelo utilizador (*e.g.*: para verificar se dois registos são duplicados). Alguns ajustamentos aos parâmetros destas funções (*e.g.*: limiar de detecção de duplicados) podem ser necessários para os ajustar à BD específica na qual a operação vai ser executada. Estes ajustamentos são efectuados pelo utilizador. O objectivo da abordagem consiste em identificar e instanciar automaticamente as operações que devem ser executadas numa BD, em função dos mapeamentos estabelecidos, apresentando-as ao utilizador. Este pode aceitar algumas destas operações sem lhes efectuar alterações, enquanto outras podem necessitar de alguns ajustamentos nos parâmetros das funções. No entanto, é mais fácil analisar as operações apresentadas e efectuar os ajustamentos que se mostrem necessários, do que: especificar na íntegra todas as operações a efectuar; ou, pesquisar e efectuar as necessárias alterações ao nível do esquema em operações de LD especificadas no passado. Isto é particularmente verdade naquelas operações especificadas inúmeras vezes em diferentes BD. Além de poder efectuar ajustamentos, o utilizador também pode eliminar as operações instanciadas cuja execução não pretenda efectuar (*e.g.*: não se efectuar uma determinada OD porque esse problema em concreto, garantidamente, não existe).

Há a consciência de que a aproximação apresentada neste capítulo ainda necessita de muito trabalho até alcançar um estado em que possa ser implementada e integrada no protótipo *SmartClean*. Entre outras tarefas, há que formalizar como são identificadas e instanciadas automaticamente as operações conceptuais de LD, em função dos mapeamentos estabelecidos pelo utilizador. Apesar do cariz inacabado, o interesse e originalidade da abordagem já mereceu reconhecimento em conferências científicas internacionais da área (*e.g.*: [Oliveira *et al.*, 2006c]). Este é o motivo pelo qual se procedeu à sua inclusão nesta dissertação de doutoramento.

10 CONCLUSÃO

Neste capítulo começa-se por procurar demonstrar que todos os objectivos que estiveram na origem deste trabalho de doutoramento foram cumpridos. Seguidamente, é efectuada uma comparação entre o resultado mais visível do trabalho realizado (*i.e.*, o protótipo *SmartClean*) e as ferramentas de Limpeza de Dados (LD) usadas como referencial, com o intuito de clarificar as semelhanças e diferenças. As limitações inerentes ao trabalho efectuado são enunciadas de seguida. Por fim, são perspectivados alguns desenvolvimentos futuros que pretendem dar continuidade ao trabalho realizado.

10.1 Objectivos Alcançados

No capítulo inicial desta dissertação de doutoramento foram enunciados os objectivos que estiveram na génese do trabalho efectuado (Secção 1.3). Estes objectivos foram identificados no seguimento dos problemas e limitações encontrados nas soluções de LD à data existentes. Nos parágrafos seguintes procura-se demonstrar, pela mesma ordem com que foram apresentados, que cada um dos objectivos foi concretizado com sucesso.

Relativamente ao primeiro objectivo, na realidade, formalizou-se a semântica operacional inerente a cada Operação de Detecção (OD) ou Operação de Correcção (OC) em que foi possível perspectivar, à partida, uma possível implementação. Estas formalizações foram usadas no protótipo desenvolvido na implementação da Detecção e Correcção (DC) dos Problemas de Qualidade dos Dados (PQD). Uma vez que o protótipo disponibiliza estas operações de base, não é necessário recorrer à sua implementação em funções definidas pelo utilizador, como acontece nos protótipos de investigação anteriormente existentes. No caso das ferramentas comerciais, normalmente nem sequer possuem esta potencialidade de extensibilidade. As formalizações são fornecidas para a generalidade dos PQD incluídos na taxionomia desenvolvida, uma vez que na grande maioria das situações é possível disponibilizar-se, desde logo, uma manipulação automática. Isto é especialmente verdade a nível de detecção, uma vez que em certas OC, fruto da sua especificidade, não há mesmo outra alternativa que não seja a de recorrer à intervenção manual ou a funções definidas pelo utilizador. Nos casos em que se justifica o

desenvolvimento destas funções de carácter diverso (*e.g.*: para a detecção de violações de restrições de integridade), foi possível identificar aspectos comuns na semântica subjacente. Esta semântica partilhada por diferentes operações de DC foi também formalizada. Para a função definida pelo utilizador fica apenas aquilo que é específico para a detecção ou correcção do PQD em questão.

Quanto ao segundo objectivo, de facto, foi concebido e desenvolvido um modelo vocacionado exclusivamente para a DC dos PQD, *i.e.*, para a LD. O modelo obedece a todos os requisitos formulados inicialmente, daí que se possa considerar que este objectivo tenha sido totalmente concretizado. Em particular, o modelo:

- **Estabelece uma sequência de execução para as operações de DC** – Os PQD são identificados e de imediato solucionados, seguindo uma sequência predefinida em cada Nível de Granularidade (NG) da taxionomia. Começa-se por manipular os problemas que ocorrem no NG mais elementar (*i.e.*, atributo) e só se passa ao nível seguinte quando todos os problemas estiverem solucionados. Adoptando uma abordagem ascendente (*i.e.*, *bottom-up*), todos os NG são sucessivamente manipulados, até se chegar ao que envolve os problemas de maior complexidade (*i.e.*, múltiplas relações de diferentes fontes de dados). A necessidade de solucionar de imediato os PQD advém da possibilidade de ocultarem ou induzirem outros problemas. Esta sequência permite estabelecer automaticamente a ordem de execução da generalidade das operações de DC, sendo apenas necessário que o utilizador proceda à sua especificação. A sequência considera todos os PQD que fazem parte da taxionomia, contrariamente ao que acontece nas ferramentas comerciais actualmente existentes em que o número de problemas coberto pela sua sequência de execução é diminuto. As operações apenas necessitam de ser sequenciadas pelo utilizador quando num NG há mais do que uma operação do mesmo tipo (*e.g.*: violação de restrição de integridade). Nos protótipos de investigação anteriormente existentes, as operações de LD são efectuadas num contexto de transformação de dados, competindo ao utilizador especificar manualmente as dependências entre estas, sob a forma de um grafo dirigido de transformações.

  Esta sequência de execução das operações de DC resulta na tese defendida nesta dissertação de doutoramento. De acordo com esta, a adopção de uma sequência de manipulação predefinida, em que os PQD são detectados e corrigidos seguindo uma aproximação ascendente, desde o NG mais elementar do modelo relacional até ao de maior complexidade, constitui uma forma adequada e eficaz de identificar e solucionar os PQD existentes. Por um lado, a sequência de manipulação é *adequada* porque reflecte as dependências de detecção – correcção existentes entre os problemas que, forçosamente, têm de ser respeitadas. Em muitos casos, a existência desta dispensa a usual intervenção

do utilizador na definição da sequência de execução das operações de LD. Por outro lado, a sequência é *eficaz* porque garante a DC dos PQD existentes. Os testes realizados durante o período de desenvolvimento do protótipo e o estudo de caso de LD demonstram a validade da tese defendida.

- **Suporta a realização das operações de DC directamente na Base de Dados (BD), não obrigando à realização de transformação de dados** – Contrariamente ao que acontece nas actuais soluções de LD (*i.e.*, englobando protótipos de investigação e ferramentas comerciais) em que a DC dos PQD é efectuada num contexto que implica transformação de dados (*i.e.*, alterações ao nível do esquema dos dados), no modelo proposto as operações de LD são efectuadas na própria BD. No final da LD, os PQD existentes encontram-se solucionados, sem que para isso tenha sido necessária qualquer operação de transformação de dados.

Além de obedecer aos requisitos iniciais, o modelo de LD desenvolvido possui ainda as seguintes características:

- **Define um mecanismo de execução incremental das OD** – Nos NG do atributo, designadamente no contexto do valor individual, e do tuplo, sempre que é detectado um PQD num tuplo, todas as subsequentes OD da sequência de execução que envolvem o atributo ou atributos em causa ficam pendentes. A detecção dos problemas prossegue, com toda a normalidade, em todos os outros atributos e tuplos nos quais não foram identificados PQD. O objectivo é reportar ao utilizador o máximo número de problemas possível em cada execução das sequências de OD. Após terem sido efectuadas as correcções, de forma manual (*i.e.*, caso a caso) ou automática (*i.e.*, com base nas OC especificadas pelo utilizador), a execução das sequências de OD reinicia-se nos mesmos pontos onde tinham sido detectados os problemas. Todas as operações pendentes da sequência de execução vão agora ser executadas, na procura de outros PQD. Caso sejam detectados, uma nova iteração de correcção – detecção volta a ocorrer, repetindo-se tudo aquilo que até agora se descreveu. De notar que nesta nova iteração apenas são executadas as OD relativas aos atributos e aos tuplos nos quais tinham sido detectado PQD, resultantes da iteração anterior. Este mecanismo de execução incremental tira partido da existência de uma sequência de execução das OD. No caso de não serem identificados PQD, o processo é dado como concluído no NG actual, prosseguindo no nível seguinte.

- **Baseia-se em linguagens declarativas para a especificação das operações de DC** – As operações de DC dos PQD que fazem parte da taxionomia são especificadas tendo

por base duas linguagens declarativas especialmente desenvolvidas para o efeito. A sintaxe inerente a cada uma das linguagens foi formalmente apresentada.

Relativamente ao terceiro objectivo do trabalho, efectivamente foi implementado um sistema de LD de carácter experimental (*i.e.*, um protótipo), denominado de *SmartClean* que materializa fielmente a arquitectura resultante do modelo proposto. Este protótipo constitui o resultado mais visível do trabalho realizado, demonstrando que a arquitectura é susceptível de ser materializada numa ferramenta informática. Por outro lado, o *SmartClean* foi utilizado num caso real de LD. Os PQD foram identificados e solucionados, seguindo a sequência preconizada e de forma progressiva, em função das operações especificadas pelo utilizador. Após a utilização do protótipo, os dados ficaram isentos dos problemas que os afectavam inicialmente, registando-se a pretendida melhoria da qualidade. A aplicação do protótipo a um caso real permitiu demonstrar, na prática, as suas potencialidades de DC dos PQD. Consequentemente, foi também demonstrada a validade da arquitectura e do modelo de LD que se encontra na sua génese. Isto significa que este objectivo também foi alcançado.

Da concretização dos dois objectivos anteriores resultaram as principais contribuições alcançadas com a realização deste trabalho de doutoramento (*i.e.*, o modelo de LD e o protótipo). O mérito científico destas contribuições foi reconhecido em publicações internacionais (*e.g.*: [Oliveira *et al.*, 2006b]). Além destas, foi possível alcançar outras duas contribuições que surgiram como resultados laterais do trabalho realizado e que designadamente são:

- **Taxionomia de PQD relacionais** – Um dos objectivos do trabalho consistia em conceber e desenvolver um modelo para a DC dos PQD representados no formato relacional. Naturalmente, o primeiro passo nesse sentido consistiu em identificar todos esses problemas. Apesar de na literatura já existirem taxionomias de PQD, por comparação entre estas, constatou-se que nenhuma era completa. Como se pretendia definir um modelo que contemplasse os diversos problemas de qualidade susceptíveis de afectarem os dados, a criação de uma taxionomia o mais completa possível tornou-se um requisito. Assim, desenvolveu-se uma nova taxionomia assente numa abordagem de identificação dos PQD. Nesta taxionomia, os PQD surgem agrupados em função do NG do modelo relacional em que ocorrem. A taxionomia desenvolvida é mais genérica e abrangente do que as existentes anteriormente, obedecendo ao requisito que despoletou o seu desenvolvimento. Ao contrário do que acontece nas taxionomias anteriores, um conjunto de definições formais acompanha a taxionomia proposta, eliminando possíveis subjectividades quanto ao significado de cada problema que a compõe. Apesar da sua elaboração não constituir um objectivo inicial do trabalho, no decurso deste, a taxionomia revelou-se uma condição para a sua continuidade. Nesta perspectiva, ainda que a

taxionomia de PQD se apresente como um resultado lateral obtido na persecução do objectivo principal (*i.e.*, conceber e desenvolver um modelo de LD), não deixa de constituir uma contribuição do presente trabalho, como se comprova pela receptividade que mereceu em conferências internacionais da área (*e.g.*: [Oliveira *et al.*, 2005c]).

Note-se que a importância da taxionomia no contexto do trabalho realizado vai além da mera identificação dos PQD. A organização por NG estabelecida pela taxionomia acabou por estar na base da tese defendida, *i.e.*, da sequência de manipulação dos PQD e também do mecanismo que permite a execução incremental das operações de DC. Como tal, a taxionomia acaba por estar presente em diversas vertentes do modelo proposto de LD.

- **Abordagem que suporta a interoperabilidade das operações de DC** – Independentemente de se tratarem de protótipos de investigação ou ferramentas comerciais, as soluções informáticas de LD actualmente existentes operam ao nível do esquema dos dados. O mesmo acontece com o protótipo *SmartClean*, desenvolvido no âmbito deste trabalho de doutoramento. No entanto, este modo de funcionamento "prende" as operações de DC a cada BD específica. Não é fácil aplicar as operações de LD noutra BD, pois os seus esquemas de dados são normalmente diferentes. Em suma, não é trivial efectuar a sua reutilização noutras BD. Esta limitação é importante, uma vez que em BD que pertencem ao mesmo domínio, muitas vezes faz sentido aplicar-se as mesmas operações de DC ou, pelo menos, um determinado subconjunto destas. Como forma de a superar, nesta dissertação concebeu-se uma abordagem que suporta a interoperabilidade das operações de DC entre BD diferentes. Em vez de definidas ao nível do esquema de dados, as operações são especificadas conceptualmente ao nível dos elementos das ontologias, usando os domínios, os conceitos e as propriedades envolvidas. Um conjunto de mapeamentos definidos pelo utilizador entre os dois níveis (*i.e.*, esquema de dados e ontologia) permite identificar e instanciar automaticamente as operações conceptuais de LD susceptíveis de execução em cada BD específica. Desta forma, é suportada a interoperabilidade das operações, o que permite a sua reutilização em diferentes BD. Apesar desta abordagem não constituir um objectivo inicial do trabalho, durante a execução do mesmo, esta surgiu como resultado da exploração de uma oportunidade de investigação entretanto identificada. Por este razão, considera-se que se trata de um resultado lateral do trabalho realizado. Actualmente, esta abordagem pode ser encarada como uma extensão ao modelo desenvolvido, sob a forma de uma nova camada que futuramente virá a ser integrada neste. Apesar de não constituir um trabalho acabado, a originalidade da abordagem já mereceu aprovação e reconhecimento em conferências científicas internacionais da área (*e.g.*: [Oliveira *et al.* 2006c]). É por este motivo que se considera que constitui uma contribuição deste trabalho de doutoramento.

Como comentário de conclusão pode dizer-se que o saldo final deste trabalho afigura ser francamente positivo. Assim, além de terem sido concretizados os objectivos enunciados no início desta dissertação, foi possível obter outros dois resultados inicialmente não programados, mas com relevância científica. Ainda assim, existe a consciência de que muito há a fazer para consolidar e dar seguimento ao trabalho efectuado no âmbito deste doutoramento. Algum desse trabalho futuro é perspectivado na última secção deste capítulo.

10.2 Comparação com Trabalho Relacionado

Não se poderia deixar de efectuar uma comparação entre o trabalho efectuado e o trabalho relacionado. Esta comparação é fulcral para que se possa evidenciar o que há em comum e, principalmente, o que há de diferente. Assim, nesta secção apresenta-se uma comparação entre o resultado mais visível do trabalho realizado, *i.e.*, o *SmartClean* (e consequentemente do modelo de LD que lhe está subjacente) e as ferramentas de LD descritas no Capítulo 3 – Secção 3.5. Estas ferramentas têm a sua origem em duas comunidades distintas: científica/académica e comercial/ empresarial. A comparação com as ferramentas oriundas de cada comunidade é efectuada nas duas subsecções seguintes.

10.2.1 Protótipos de Investigação

Nesta secção efectua-se uma comparação entre o *SmartClean* e os protótipos de investigação apresentados no Capítulo 3 – Secção 3.5.1 (*i.e.*, *Ajax*; *Arktos II*; *IntelliClean*; *FraQL*; e, *Potter's Wheel*). A análise comparativa é efectuada em duas vertentes: características intrínsecas e cobertura a nível de DC dos PQD que constituem a taxionomia apresentada no Capítulo 4.

10.2.1.1 Análise das Características

A Tabela 10.1 permite comparar as características do *SmartClean* relativamente aos demais protótipos de investigação de LD analisados no âmbito deste trabalho de doutoramento. Esta tabela constitui uma evolução da Tabela 3.2. Em relação a esta, a Tabela 10.1 é composta por uma coluna adicional relativa às características do *SmartClean*. As características adoptadas como termo de comparação são as mesmas que se encontram na Tabela 3.2. O significado de cada característica encontra-se definido na referida secção.

Tabela 10.1 – Características do SmartClean comparativamente aos protótipos de investigação de LD

	Ajax	***Arktos II***	***Intelli Clean***	***Potter's Wheel***	***FraQL***	***Smart Clean***
Tipos de fontes suportadas	BDR, FT	BDR	BDR	BDR, FT	BDR	BDR
Preocupações de optimização	Sim	Sim	Não	Sim	Não	Sim
Capacidades extensibilidade	Sim	Sim	Sim	Sim	Sim	Sim
Suporte excepções	Sim	Sim	Não	Não	Não	Sim
Interface com o utilizador	Não Gráfico	Gráfico	Não gráfico	Gráfico	Não gráfico	Não gráfico
Potencialidades de anulação	Sim	Não	Não	Sim	Não	Sim
Execução incremental	Sim	Não	Não	Não	Não	Sim
Defin. sequência execução operaç.	Manual	Manual	Auto.	Manual	Manual	Auto.

Os resultados exibidos na Tabela 10.1 suscitam os comentários que a seguir se apresentam.

- À semelhança do que acontece na maioria dos protótipos de investigação analisados, o *SmartClean* também tem em conta os aspectos relacionados com a optimização da execução das operações de DC. Estas preocupações traduzem-se na: (*i*) execução paralela das OD que pertencem a sequências de dependências diferentes; (*ii*) execução em simultâneo das OC especificadas para solucionar os PQD detectados; (*iii*) possibilidade de na detecção de duplicados se recorrer ao método da vizinhança ordenada com multi-passagem; e, (*iv*) implementação eficiente das OD e OC, a nível algorítmico, seguindo-se as formalizações expostas, respectivamente, nos Capítulos 5 e 6. Apesar destas, naturalmente que há margem para a introdução de outras optimizações na execução das operações de DC. Este aspecto será objecto de trabalho futuro.

- O suporte concedido pelo *SmartClean* à ocorrência de excepções é muito simples. O mecanismo concebido baseia-se nas potencialidades de manipulação de excepções existentes na linguagem de programação *Java*. Sempre que ocorre uma situação não prevista que implicaria a interrupção da execução da operação, esta é ignorada, sendo o tuplo ou tuplos causadores e o respectivo motivo da excepção registados numa tabela auxiliar criada na BD que armazena os PQD identificados. Assim, a execução da operação

em causa continua com toda a normalidade. Compete ao utilizador analisar o conteúdo destas tabelas e solucionar estas situações anómalas ou adaptar a operação de LD respectiva para que esta manipule devidamente as situações excepcionais.

- Como já se referiu no Capítulo 7 – Secção 7.7, a inexistência de uma interface gráfica é uma limitação que será suprida no futuro. Na maioria dos protótipos analisados esta limitação também existe.

- O *SmartClean* possui potencialidades de anulação das alterações efectuadas como consequência da execução das OC. Esta potencialidade não se encontra disponível na maioria dos protótipos analisados.

- A generalidade dos protótipos considerados não suporta a execução incremental das operações de LD, contrariamente ao que acontece no *SmartClean*. Neste aspecto, apenas o *Ajax* constitui excepção.

- O *SmartClean* estabelece automaticamente a sequência de execução das operações de LD especificadas pelo utilizador. Apenas quando há mais do que uma operação de um dado tipo (*e.g.*: violação de restrição de integridade) com um âmbito de incidência comum (*e.g.*: o mesmo atributo) é que é necessário que o utilizador defina a sua ordem de execução. Neste aspecto, apenas o *IntelliClean* segue um caminho similar. No entanto, como se poderá constatar na secção seguinte, a cobertura dada por este protótipo aos diversos tipos de PQD não é comparável à que é suportada pelo *SmartClean*. Face à diversidade de operações que este último protótipo suporta, a definição da sequência de execução das operações revestiu-se de maior complexidade.

10.2.1.2 Análise da Cobertura Dada aos Problemas de Qualidade dos Dados

A Tabela 10.2, a seguir apresentada, constitui uma evolução da Tabela 4.3. Esta última foi criada com a finalidade de ilustrar a cobertura dada aos PQD, a nível de DC, pelos cincos protótipos de investigação de LD. Em relação a esta tabela, a Tabela 10.2 é composta por uma coluna adicional que exibe a cobertura dada aos PQD pelo *SmartClean*. Esta nova tabela permite comparar as potencialidades de DC do *SmartClean* relativamente a esses protótipos de investigação. Os significados dos acrónimos usados na tabela são: D – Detecção; C – Correcção; OBS – Operação Baseada em *Structured Query Language* (SQL); FDU – Função Definida pelo Utilizador; OP – Operação Predefinida; e, NS – Não Suportado.

Tabela 10.2 – Cobertura dada aos PQD pelo *SmartClean* por comparação aos protótipos de investigação de LD

PQD	Tipo Op.	*Ajax*	*Arktos II*	*Intelli Clean*	*FraQL*	*Potter's Wheel*	*Smart Clean*
Valor em falta	**D**	OBS	OP	NS	OBS	OP	OP
	C	OBS/ FDU	FDU	FDU	OBS	NS	OP/ FDU
Violação de sintaxe	**D**	OBS/ FDU	OP	NS	NS	OP	OP/ FDU
	C	FDU	OP	FDU	OP/ FDU	OP	OP/ FDU
Erro ortográfico	**D**	FDU	FDU	NS	NS	OP	OP
	C	FDU	FDU	FDU	NS	NS	OP
Violação de domínio	**D**	OBS	OP	NS	NS	FDU	OP
	C	OBS/ FDU	FDU	FDU	OP/ FDU	NS	OP/ FDU
Violação de unicidade	**D**	OBS	OP	NS	NS	FDU	OP
	C	OBS/ FDU	FDU	NS	NS	NS	OP
Existência de sinónimos	**D**	FDU	FDU	NS	NS	FDU	OP
	C	FDU	FDU	FDU	NS	NS	OP
Violação de restrição de integridade	**D**	OBS/ FDU	FDU	OP/ FDU	NS	FDU	OP/ FDU
	C	OBS/ FDU	FDU	NS	NS	NS	OP/ FDU
Violação de dependência funcional	**D**	OBS	FDU	OP	NS	FDU	OP
	C	FDU	FDU	NS	NS	NS	OP
Circularidade entre tuplos num auto-relac.	**D**	FDU	FDU	NS	NS	NS	OP
	C	FDU	FDU	NS	NS	NS	FDU
Tuplos duplicados	**D**	OBS/ FDU	FDU	OP/ FDU	OBS/ FDU	NS	OP/ FDU
	C	OBS/ FDU	FDU	OP/ FDU	OBS/ FDU	NS	OP/ FDU
Existência de homónimos	**D**	OBS	FDU	NS	NS	NS	OP
	C	FDU	FDU	NS	NS	NS	FDU

Tabela 10.2 – Cobertura dada aos PQD pelo *SmartClean* por comparação aos protótipos de investigação de LD (cont.)

PQD	Tipo Op.	*Ajax*	*Arktos II*	*Intelli Clean*	*FraQL*	*Potter's Wheel*	*Smart Clean*
Dif. granularidades de representação	**D**	OBS/ FDU	FDU	NS	NS	FDU	OP
	C	OBS/ FDU	FDU	FDU	NS	NS	FDU
Violação de integridade referencial	**D**	OBS	OP	NS	NS	NS	OP
	C	FDU	FDU	NS	NS	NS	FDU
Heterogeneidade de sintaxes	**D**	FDU	FDU	NS	NS	NS	OP
	C	FDU	FDU	NS	NS	NS	OP/ FDU
Heterogeneidade de unidades de medida	**D**	FDU	FDU	NS	NS	NS	OP
	C	FDU	FDU	NS	NS	NS	OP

Esta tabela demonstra que o *SmartClean* proporciona uma cobertura mais abrangente à LD do que qualquer outro dos protótipos de investigação considerados. De facto, o *SmartClean* suporta de base a detecção de todos os PQD incluídos na taxionomia proposta no Capítulo 4. Nos protótipos de investigação que servem de referencial, há diversos tipos de PQD cuja detecção nem sequer é suportada. Noutras situações, esse suporte apenas existe virtualmente, uma vez que depende da especificação de FDU. O suporte concedido pelo *SmartClean* a nível de correcção é inferior ao de detecção. Nem todos os PQD podem ser solucionados de raiz, a partir das OC disponibilizadas. Nalguns PQD (*e.g.*: existência de circularidade num auto-relacionamento) é necessário recorrer à implementação de FDU para os solucionar[22]. Nestes problemas não foi possível perspectivar uma OC que pudesse ser, desde logo, disponibilizada. Ainda assim, o *SmartClean* suporta de base a correcção da grande maioria dos PQD. Quando se compara estas potencialidades de correcção com as dos protótipos de referência, constata-se que estas são superadas. Nesses protótipos, há vários tipos de PQD cuja correcção não é suportada. Outros protótipos evidenciam potencialidades de correcção virtuais, uma vez que a sua materialização se encontra dependente da implementação de FDU. Quer a nível de detecção, quer a nível de correcção, o *SmartClean* disponibiliza um leque muito apreciável de operações "prontas a usar" na identificação e resolução dos PQD. Nenhum outro protótipo evidencia potencialidades similares.

[22] Naturalmente, a correcção de qualquer PQD pode sempre ser efectuada por via manual, *i.e.*, directamente pelo próprio utilizador.

Nalguns casos o *SmartClean* concede um suporte à manipulação dos PQD que é um misto de operação predefinida e FDU. Na tabela anterior, estas situações surgem representadas através dos acrónimos OP/FDU. Quando a semântica da OD ou OC é demasiado complexa para ser suportada pela operação predefinida, não há outra alternativa que não seja o recurso à sua implementação numa FDU. Em todas os outros casos, o suporte à detecção ou correcção dos PQD é concedido pela operação que se fornece de raiz. As potencialidades de extensibilidade do *SmartClean*, também existentes nos outros protótipos, foram introduzidas para colmatar necessidades específicas de LD que não é possível prever de antemão. No entanto, esta potencialidade permite que qualquer PQD possa ser detectado ou corrigido por intermédio de FDU, mesmo aqueles para os quais já se fornece suporte. Na tabela anterior, esta possibilidade nem sequer se encontra representada (*e.g.*: a detecção de valores em falta ser efectuada através de uma FDU), uma vez que não foi para esta finalidade que se incluiu as potencialidades de extensibilidade do *SmartClean*.

10.2.2 Ferramentas Comerciais

Nesta secção apresenta-se a comparação entre o *SmartClean* e as ferramentas comerciais descritas no Capítulo 3 – Secção 3.5.2 (*i.e.*, *WinPure Clean and Match 2007*; *ETI Data Cleanser*; *Trillium Quality*; *dfPower Quality*; e, *HIQuality*). Mantendo a coerência com a secção anterior, a análise comparativa é efectuada de acordo com as mesmas perspectivas (*i.e.*, características intrínsecas e cobertura de manipulação dos PQD).

10.2.2.1 Análise das Características

A Tabela 10.3 possibilita a comparação das características do *SmartClean* relativamente às ferramentas comerciais de LD estudadas. Esta tabela constitui uma evolução da Tabela 3.3. Comparativamente a esta, a Tabela 10.3 inclui uma coluna extra onde se encontram as características do *SmartClean*. O conjunto de características utilizado como referencial de comparação é o mesmo da Tabela 3.3. O significado de cada característica encontra-se definido no Capítulo 3 – Secção 3.5.1.6.

Tabela 10.3 – Características do *SmartClean* comparativamente às ferramentas comerciais de LD

	WinPure	***ETI Data Cleanser***	***TS Quality***	***dfPower Quality***	***HI Quality***	***Smart Clean***
Tipos de fontes suportadas	BDR, FT	DT	DT	DT	BDR, FT	BDR
Preocupações de optimização	Sim	Sim	Sim	Sim	Sim	Sim
Capacidades extensibilidade	Não	Não	Não	Não	Não	Sim
Suporte excepções	Não	Sim	Sim	Sim	Sim	Sim
Interface com o utilizador	Gráfico	Gráfico	Gráfico	Gráfico	Gráfico	Não gráfico
Potencialidades de anulação	Não	Não	Não	Não	Não	Sim
Execução incremental	Não	Não	Não	Não	Não	Sim
Defin. sequência execução operaç.	Autom.	Autom.	Autom.	Autom.	Autom.	Autom.

Os resultados expostos nesta tabela justificam os comentários que a seguir se apresentam.

- Tal como sucede em todas as ferramentas comerciais consideradas, o *SmartClean* também evidencia preocupações a nível da optimização da execução das operações de DC. Estas preocupações já foram enunciadas na Secção 10.2.1.1 deste capítulo. No caso das ferramentas comerciais, ainda que estas optimizações sejam publicitadas pelas empresas proprietárias, em concreto não foi possível apurar exactamente em que consistem. Uma das optimizações que por vezes surge vagamente mencionada, prende-se com a adopção de métodos eficientes de detecção de duplicados que não obrigam à realização de um produto Cartesiano de comparações entre os tuplos. Nesta perspectiva, o *SmartClean* também disponibiliza o método da vizinhança ordenada com multi-passagem.

- O *SmartClean* possui potencialidades de extensibilidade, ao contrário do que acontece nas ferramentas comerciais analisadas. A impossibilidade de incorporar FDU impede-as de detectar outros problemas para além daqueles que suportam de base. Apesar do seu cariz orientado para o utilizador não perito (*i.e.*, sem conhecimentos de programação), a inexistência desta potencialidade constitui uma limitação importante.

- Contrariamente ao que acontece no *SmartClean*, todas as ferramentas comerciais possuem uma interface gráfica de interacção com o utilizador. Esta limitação actual do *SmartClean* perante as ferramentas comerciais será suprida num futuro próximo, como já se referiu.

- O *SmartClean* permite a anulação automática das alterações efectuadas no seguimento da execução de OC, caso os seus efeitos não sejam os esperados. Nas ferramentas comerciais esta potencialidade não existe, o que obriga a que seja especificada uma operação de LD que restaure a situação inicial. Dependendo das alterações efectuadas, nem sempre é possível efectuar o restauro por esta via.

- Nas ferramentas comerciais em causa, não há a possibilidade de executar as operações de LD incrementalmente, ao invés do que acontece no *SmartClean*. Nestas ferramentas, a execução da sequência de operações de LD é sempre efectuada na íntegra, abrangendo todos os tuplos das relações envolvidas. O modelo de LD sobre o qual o *SmartClean* se encontra alicerçado é, por natureza, incremental.

- À semelhança do que acontece no *SmartClean*, nas ferramentas comerciais, a definição da sequência de execução das operações de LD é efectuada automaticamente. No *SmartClean* apenas é necessário que o utilizador defina a ordem de execução das operações quando há mais do que uma operação de um dado tipo (*e.g.*: violação de restrição de integridade), cujo âmbito de incidência é comum (*e.g.*: o mesmo atributo). Como se poderá verificar na secção seguinte, as ferramentas comerciais acabam por conceder um suporte muito reduzido aos diferentes tipos de PQD. Este suporte não é comparável ao fornecido pelo *SmartClean*. Face ao maior o número de operações suportadas, o estabelecimento de uma sequência de execução adequada revestiu-se de uma maior dificuldade.

10.2.2.2 Análise da Cobertura Dada aos Problemas de Qualidade dos Dados

A Tabela 10.4, a seguir apresentada, representa uma evolução da Tabela 4.4. Esta última tem como finalidade exibir a cobertura, a nível de DC, que as cinco ferramentas comerciais em causa concedem aos PQD. Relativamente a esta, a Tabela 10.4 possui uma coluna adicional que ilustra a cobertura dada pelo protótipo *SmartClean* aos PQD. Esta tabela permite comparar as potencialidades de DC do *SmartClean* em relação às ferramentas comerciais. Os significados dos acrónimos utilizados nesta tabela são exactamente os mesmos do que os apresentados para a Tabela 10.2.

Tabela 10.4 – Cobertura dada aos PQD pelo *SmartClean* por comparação a cinco ferramentas comerciais de LD

PQD	Tipo Op.	*WinPure*	*ETI Data Cleanser*	*Trillium Quality*	*dfPower Quality*	*HI Quality*	*Smart Clean*
Valor em falta	**D**	OP	NS	OP[1]	NS	NS	OP
	C	NS	NS	OP[1]	NS	NS	OP/ FDU
Violação de sintaxe	**D**	OP[2]	NS	NS	NS	OP	OP/ FDU
	C	OP[2]	OP	OP	OP	OP	OP/ FDU
Erro ortográfico	**D**	NS	NS	NS	NS	NS	OP
	C	NS	NS	NS	NS	NS	OP
Violação de domínio	**D**	OP[3]	OP[4,5]	OP[4,6,7]	OP[4]	OP[4,5,8,9]	OP
	C	OP[3]	OP[10,11]	OP[7,10]	OP[10]	OP[4,5,10,11]	OP/ FDU
Violação de unicidade	**D**	OP	NS	NS	NS	NS	OP
	C	NS	NS	NS	NS	NS	OP
Existência de sinónimos	**D**	NS	NS	NS	NS	NS	OP
	C	OP	OP	OP	OP	OP	OP
Violação restr. integridade	**D**	NS	NS	NS	NS	NS	OP/ FDU
	C	NS	NS	NS	NS	NS	OP/ FDU
Violação dep. funcional	**D**	NS	NS	NS	NS	NS	OP
	C	NS	NS	NS	NS	NS	OP
Circular. entre tuplos auto-rel.	**D**	NS	NS	NS	NS	NS	OP
	C	NS	NS	NS	NS	NS	FDU
Tuplos duplicados	**D**	OP[12]	OP	OP	OP	OP[12]	OP/ FDU
	C	OP[12]	OP	OP	OP	OP[12]	OP/ FDU
Existência de homónimos	**D**	NS	NS	NS	NS	NS	OP
	C	NS	NS	NS	NS	NS	FDU
Dif. granular. representação	**D**	NS	NS	NS	NS	NS	OP
	C	NS	NS	NS	NS	NS	FDU

Tabela 10.4 – Cobertura dada aos PQD pelo *SmartClean* por comparação a cinco ferramentas comerciais de LD (cont.)

PQD	Tipo Op.	*WinPure*	*ETI Data Cleanser*	*Trillium Quality*	*dfPower Quality*	*HI Quality*	*Smart Clean*
Violação integr. referencial	**D**	NS	NS	NS	NS	NS	OP
	C	NS	NS	NS	NS	NS	FDU
Heterogen. de sintaxes	**D**	NS	NS	NS	NS	NS	OP
	C	NS	NS	NS	NS	NS	OP/ FDU
Heterogen. de un. medida	**D**	NS	NS	NS	NS	NS	OP
	C	NS	NS	NS	NS	NS	OP

Nota: O significado das anotações numéricas apresentadas nesta tabela encontra-se no Capítulo 4 – Secção 4.5.2. Não se considerou relevante voltar a efectuar a sua repetição neste local.

A tabela evidencia que o *SmartClean* oferece uma cobertura bem mais abrangente do que qualquer outra das ferramentas comerciais de LD consideradas. Contrariamente ao que acontece no *SmartClean*, estas ferramentas não suportam a detecção da generalidade dos PQD que compõem a taxionomia exposta no Capítulo 4. A nível de correcção, apesar de ligeiramente melhor, o panorama não é substancialmente diferente. A correcção da grande maioria dos PQD também não é suportada pelas ferramentas analisadas. A isto acresce ainda o facto de que nalguns PQD o suporte concedido, a nível de DC, é apenas parcial, em virtude deste se encontrar limitado a dados de um determinado tipo (*e.g.*: nomes de pessoas; endereços postais; endereços de *e-mail*). Como as ferramentas em causa não possuem potencialidades de extensibilidade, não é possível incorporar FDU para a detecção ou correcção de outros PQD que não sejam suportados de raiz. Nenhuma das ferramentas comerciais de LD consideradas oferece uma panóplia de OD e OC "prontas a usar" de dimensão tão considerável quanto as disponíveis no *SmartClean*.

10.3 Problemas e Limitações

Naturalmente, alguns problemas e limitações podem ser apontados ao trabalho realizado. Nos pontos seguintes efectua-se a sua apresentação.

- O modelo de LD foi desenvolvido especificamente para os PQD que fazem parte da taxionomia. Como tal, apenas estes problemas possuem suporte a nível de DC. Como não foi possível provar formalmente que a taxionomia é completa, *i.e.*, que cobre todos os PQD que podem afectar os dados, também não é possível garantir que o modelo de LD

desenvolvido suporta a DC de todo e qualquer problema. Ainda que se considere improvável, no caso de vir a ser identificado um novo PQD que justifique a sua inclusão na taxionomia, o modelo de LD teria de ser adaptado em conformidade. Ainda assim, isto não implicaria alterações de maior. Em função do NG a que o problema pertencesse, haveria que efectuar a sua inserção no local apropriado na respectiva sequência de manipulação preconizada. Naturalmente, as respectivas OD e OC do PQD em causa também teriam que ser formalizadas (caso tal fosse desde logo possível) e posteriormente implementadas no *SmartClean*.

- O modelo de LD apresentado é, por natureza, iterativo. Por um lado, estas iterações ocorrem em cada NG, até que não sejam detectados PQD. Quando se verifica uma nova iteração, apenas são executadas as OD que incidem sobre atributos nos quais tinham sido identificados PQD. No caso dos NG do atributo (contexto do valor individual) e do tuplo, a execução das OD é ainda mais restrita a apenas aos tuplos nos quais os PQD tinham sido identificados. Por outro lado, quando se constata que todos os PQD estão solucionados num NG, a sequência de execução das OD volta a reiniciar-se desde o nível mais elementar que corresponde às operações especificadas pelo utilizador. O objectivo desta iteração é identificar possíveis PQD que possam ter sido involuntariamente introduzidos noutros NG a montante, fruto das correcções efectuadas. Nesta nova iteração, apenas são executadas as OD que incidem sobre atributos que entretanto sofreram alterações/correcções. Mesmo adoptando estas abordagens que procuram minimizar o custo computacional, a nível do tempo despendido nas sucessivas iterações de execução das OD, este constitui o maior problema deste modelo de LD. Naturalmente, quanto maior o número de iterações, maior o custo computacional associado. Caso as correcções efectuadas no seguimento da primeira execução das sequências de OD não introduzam outros PQD (como se verificou no estudo de caso, esta constitui a situação mais usual), apenas há uma iteração adicional de execução das OD em cada NG e uma outra desde o nível mais elementar. Em termos de custo computacional, estas iterações representam o "preço a pagar" para se ter a certeza que todos os PQD foram solucionados e que os dados estão, efectivamente, "limpos". No estudo de caso apresentado no Capítulo 8, o custo computacional não se mostrou minimamente relevante. Todavia, a dimensão da BD utilizada não é muito significativa, especialmente quando comparada com certas BD que armazenam milhões de tuplos. A avaliação do custo computacional em BD de maiores dimensões será objecto de trabalho futuro.

- No modelo proposto de LD, a manipulação dos PQD obedece a uma sequência preestabelecida por e em cada NG. As OD especificadas pelo utilizador são executadas de acordo com essa sequência de manipulação. Independentemente da situação de LD

em causa (*i.e.*, domínio; BD), os PQD são sempre manipulados em função dessa sequência. A existência desta evita que tenha de ser o utilizador a definir as dependências de execução entre as operações, como actualmente acontece nos protótipos de investigação. Naturalmente, quando a definição das dependências fica a cargo do utilizador, qualquer sequência de execução das operações é permitida. Em certas situações, pode tirar-se partido desta flexibilidade para optimizar a execução das operações. Por exemplo, ao começar-se por detectar e corrigir as violações à restrição de integridade ao nível do tuplo que obriga a que o valor do atributo *valor* seja igual ao produto entre os valores dos atributos *quant* e *preço_unit*, pode aproveitar-se também para solucionar eventuais valores em falta no atributo *valor*. Ao conceber-se um modelo de LD assente numa sequência de manipulação dos PQD que é sempre a mesma, independentemente do domínio e da BD, não se tira partido de certas especificidades que poderiam optimizar a sua manipulação. Ainda que este problema de eficiência de manipulação dos PQD possa ocorrer, em contrapartida, o utilizador apenas necessita de centrar os seus esforços na especificação das operações a realizar e não nas dependências de execução que possam existir entre estas. Apenas perante operações do mesmo tipo (*e.g.*: violação de dependência funcional) que possuam atributos em comum é necessário que o utilizador as coloque segundo a ordem de execução pretendida.

- De acordo com o modelo proposto, as OD relativas aos diferentes NG são especificadas na íntegra no início do processo de LD. Uma vez iniciada a execução destas no NG mais elementar que corresponde às operações especificadas, não é possível proceder-se à especificação de outras OD referentes a um qualquer NG que, no entretanto, tenham sido identificadas como necessárias. A necessidade destas pode surgir como consequência dos PQD que vão sendo reportados. Actualmente, a única alternativa que permite a especificação de novas OD implica a interrupção do processo de LD em curso. As operações pretendidas são especificadas e todo o processo reinicia-se novamente, com as consequências que daí advêm de voltarem a ser executadas OD que já o tinham sido anteriormente. No futuro, esta limitação será objecto de trabalho para que seja possível, pelo menos, especificar OD para o NG em questão e para os que se encontram a jusante deste.

10.4 Trabalho Futuro

Não só neste capítulo, como também ao longo dos cinco capítulos anteriores, foram referidos aspectos susceptíveis de virem a ser alvo de desenvolvimentos futuros, identificados no decurso do trabalho realizado. Nos pontos seguintes recapitulam-se os considerados mais relevantes e apresentam-se outros desenvolvimentos futuros que ainda não tinham sido referidos.

- **Implementação do modelo proposto de LD como um *Web Service*** – O modelo proposto de LD deu origem ao protótipo *SmartClean*. Este protótipo foi concebido para funcionar isoladamente no computador que serve de suporte à realização da LD (cariz de *stand-alone application*). Esta orientação coaduna-se perfeitamente à finalidade com que o *SmartClean* foi desenvolvido no âmbito deste trabalho de doutoramento, *i.e.*, materializar e permitir validar o modelo proposto de LD. No futuro planeia-se a implementação deste modelo sob a forma de um *Web Service*. Quem necessitar de detectar e corrigir PQD poderá contratar este serviço de LD através da *Internet*.

- **Extensão do modelo de LD desenvolvido para dados semi-estruturados** – O modelo de LD desenvolvido encontra-se vocacionado especificamente para dados representados no formato relacional. A DC de problemas de qualidade em dados semi-estruturados (*e.g.*: em XML[23]) envolve uma maior complexidade do que em dados estruturados (*e.g.*: no formato relacional). A justificação para isto encontra-se na rigidez imposta pelos esquemas de dados associados às representações estruturadas. Nas representações semi-estruturadas, há uma certa flexibilidade quanto ao que é possível armazenar-se, o que dificulta a LD. A LD representados em XML é precisamente uma das áreas que recentemente tem centrado a atenção de alguns investigadores desta comunidade. Nesta perspectiva, um desenvolvimento futuro envolverá estender/adaptar o modelo de LD proposto para dados representados num formato semi-estruturado.

- **Alargamento do modelo de LD desenvolvido para tipos de dados complexos** – O modelo de LD desenvolvido encontra-se limitado a valores cujo tipo de dados é atómico ou elementar (*i.e.*, em cada atributo de cada tuplo encontra-se um único valor). Actualmente, alguns sistemas de BD relacionais (*e.g.*: *SQL Server*; *Oracle*) suportam tipos de dados complexos (*e.g.*: tipo de dados tabela/matriz) que permitem que num atributo possam estar armazenados simultaneamente múltiplos valores. Um desenvolvimento futuro deste trabalho consistirá em fornecer o adequado suporte à DC dos PQD existentes nos atributos cujos tipos de dados são complexos.

- **Desenvolvimento de uma interface gráfica para o *SmartClean*** – Actualmente, o *SmartClean* é um protótipo de LD que não possui uma interface gráfica. As operações de DC são especificadas directamente num ficheiro de texto, utilizando as linguagens declarativas desenvolvidas. A existência de um editor gráfico que auxilie à introdução das operações aumentará muito a usabilidade do protótipo. Por outro lado, os PQD identificados são "despejados" para tabelas de uma BD especialmente criada para o efeito, enquanto que as correcções manuais têm que ser efectuadas directamente pelo

[23] XML é a abreviatura de *Extensible Markup Language*, constituindo uma linguagem de anotação descritiva extensível.

utilizador na própria BD alvo de LD. O utilizador necessita de efectuar inquéritos SQL para visualizar os problemas detectados e as correcções automaticamente efectuadas. Naturalmente, esta não é a forma adequada de interagir com o utilizador, daí que o desenvolvimento de uma interface gráfica, na qual seja possível efectuar todas estas operações de forma integrada, constitua mesmo uma prioridade.

- **Implementação da abordagem que suporta a interoperabilidade das operações de DC** – A abordagem foi descrita no capítulo anterior, mas devido às limitações temporais que caracterizam um trabalho deste género, não foi possível proceder à sua implementação e consequente integração no protótipo *SmartClean*. A implementação da abordagem permitirá demonstrar na prática as potencialidades de: sugestão automática das operações de LD a efectuar, em função dos mapeamentos manualmente estabelecidos pelo utilizador entre o nível do esquema de dados e o nível da ontologia; e, reutilização automática das operações de LD.

- **Especificação e implementação do mecanismo de mapeamento automático entre o nível do esquema de dados e o nível ontológico** – No estado actual da abordagem, todos os mapeamentos entre os elementos do esquema de dados (*i.e.*, BD; tabelas; atributos) e os elementos da ontologia (*i.e.*, domínios; conceitos; propriedades) são estabelecidos manualmente pelo utilizador. O modelo subjacente a esta abordagem de suporte à interoperabilidade das operações de LD permite que se tire partido de técnicas existentes de análise de semelhança léxica. O objectivo é identificar automaticamente semelhanças léxicas entre os elementos do esquema de dados em causa (*i.e.*, no qual vão ser efectuadas as operações de LD) e os elementos dos esquemas de dados anteriores sobre os quais se armazenou informação. Com base nas semelhanças é possível inferir automaticamente iguais mapeamentos aos que foram estabelecidos anteriormente por via manual. A forma de funcionamento deste mecanismo necessita de ser convenientemente especificada. Naturalmente, o passo seguinte passará pela implementação deste mecanismo, de forma integrada, no protótipo *SmartClean*.

- **Aplicação do *SmartClean* a novos casos de LD** – Face às perspectivas de trabalho futuro apresentadas nos pontos anteriores, o protótipo *SmartClean* ficará substancialmente diferente do actual. Assim sendo, faz todo o sentido que seja aplicado em novos casos de melhoria da qualidade dos dados, para validar as novas extensões (*e.g.*: a interoperabilidade das operações de LD e o mecanismo de mapeamento automático) ao modelo de LD proposto nesta dissertação. Por outro lado, como já se referiu neste capítulo, é importante avaliar o desempenho do modelo de LD em BD de maiores dimensões do que a usada no estudo de caso realizado. O objectivo consiste em continuar a testar e validar o modelo proposto de LD em novas e diferentes situações que requerem a DC dos PQD.

FERRAMENTAS COMERCIAIS DE MELHORIA DA QUALIDADE DOS DADOS

Apêndice A

Neste apêndice apresenta-se uma breve descrição de algumas ferramentas comerciais utilizadas no processo de melhoria da qualidade dos dados (exposto no Capítulo 2). No âmbito deste trabalho de doutoramento procedeu-se ao seu estudo e respectiva experimentação. Estas ferramentas suportam a realização das diversas actividades efectuadas no processo de melhoria da qualidade dos dados, nomeadamente: *data profiling*; análise de dados; transformação de dados; e, limpeza de dados.

A.1 *Datiris Profiler*

A ferramenta *Datiris Profiler* [Dataris, 2007] efectua *data profiling* em bases de dados dos seguintes tipos: *Oracle*; *SQL Server*; e, *Access*. Através de uma conexão *ODBC* é ainda possível aceder a outros tipos de bases de dados. O primeiro passo a efectuar quando se utiliza esta ferramenta consiste em especificar a fonte de dados e a(s) tabela(s) a analisar. A ferramenta permite manipular simultaneamente múltiplas tabelas (*batch profiling*), podendo ser especificadas diferentes opções para cada uma destas. No passo seguinte, procede-se à escolha dos atributos a analisar. Normalmente, o interesse do *data profiling* encontra-se centrado num subconjunto dos atributos existentes. De seguida, há que seleccionar os registos a considerar. Para o efeito, estão disponíveis três opções: todos os registos; apenas os *n* primeiros registos; e, um subconjunto dos registos da tabela, em função de um determinado critério. Após terem sido especificadas estas opções, o utilizador dá início ao processo de *data profiling* propriamente dito.

Os resultados são apresentados organizados por atributo. Cada atributo surge como linha de uma tabela, contendo as colunas os resultados das diversas análises efectuadas. Entre outras informações, nesta tabela é possível encontrar informação sobre: os atributos que estão definidos como constituintes da chave primária; a unicidade dos valores que se encontram nos atributos; a percentagem de valores que se encontram preenchidos em cada atributo; o intervalo de valores existente nos atributos numéricos, o que permite identificar valores que violam o domínio; e, os diferentes padrões sintácticos que existem nos valores. Os resultados apresentados na tabela são extremamente úteis para a identificação dos problemas de qualidade dos dados. Em qualquer

atributo é possível efectuar uma análise de frequência dos seus valores (em valor absoluto e percentual). A ferramenta permite visualizar os registos em que estes valores se encontram (operação de *drill down*). Uma potencialidade interessante da ferramenta consiste na identificação dos padrões (formatos) que existem nos valores dos atributos. A existência de padrões estranhos nos valores representa problemas de qualidade dos dados. Os registos onde um determinado padrão ocorre também podem ser visualizados. Os registos afectados por problemas de qualidade podem ser exportados para um ficheiro de texto ou para um ficheiro com o formato *Microsoft Excel*, para posterior análise. Os resultados obtidos do processo de *data profiling* podem ser devidamente anotados com comentários, para os tornar mais explícitos. Estes resultados podem ser armazenados num repositório existente para o efeito, na sua totalidade ou apenas os *n* primeiros valores obtidos.

A ferramenta possui ainda outras potencialidades, como: (*i*) definir *data profiling templates* que podem ser especificados uma só vez e aplicados múltiplas vezes em várias bases de dados; (*ii*) permitir adicionar descrições às entidades e atributos, de modo a tornar o seu significado mais expressivo (*e.g.*: o atributo *dp* representa a *descrição do produto*); (*iii*) e, suportar a exportação dos resultados obtidos no processo de *data profiling* para o *Microsoft Excel*, onde estes podem sofrer outras manipulações (*e.g.*: criação de gráficos).

A.2 *WizRule*

A *WizRule* [WizSoft, 2007b] é uma ferramenta de análise dos dados, baseada em técnicas de mineração de dados (*i.e.*, *data mining*). Na sua base encontra-se o seguinte pressuposto: na maioria dos casos, os problemas de qualidade dos dados manifestam-se como excepções à regra. Por exemplo, se todas as vendas efectuadas a um determinando cliente foram efectuadas pelo mesmo vendedor, à excepção de uma que foi efectuada por um vendedor associado a outros clientes, esta pode ser considerada uma excepção à regra. Usando um algoritmo matemático, a ferramenta *WizRule* infere todas as regras que se encontram subjacentes a um conjunto de dados. Estas regras permitem detectar os valores que constituem excepções. Sobre estes valores recaem suspeitas de constituírem problemas de qualidade dos dados, o que justifica uma análise cuidada por parte do utilizador.

O utilizador necessita apenas de seleccionar a tabela que pretende analisar, sendo todo o trabalho de análise efectuado automaticamente. O utilizador pode ajustar alguns parâmetros de análise como a *probabilidade mínima das regras se-então* e o *número mínimo de casos de uma regra*. O utilizador

também pode alterar o tipo de regras que pretende que sejam geradas. A ferramenta permite a obtenção dos seguintes tipos de regras:

- *regras fórmula* – Um exemplo deste tipo de regras é: $\mathcal{A} = \mathcal{B} * \mathcal{C}$, em que $\mathcal{A}$ = total; $\mathcal{B}$ = quantidade; e, $\mathcal{C}$ = preço unitário. O número de variáveis diferentes que podem integrar a fórmula encontra-se limitado a um máximo de cinco. Este tipo de regras é acompanhado pelo seu nível de precisão, resultante do rácio entre o número de registos em que a regra se verifica e o número total de registos.
- *regras se-então* – Um exemplo deste tipo de regras é: Se *Cliente* é *Loja dos PC's* e *FamiliaProduto* é *Portátil* Então *Desconto = 0.07*. A ferramenta permite obter todas as regras existentes, independentemente do número de condições que estas possam ter. Este tipo de regras é acompanhado pela sua probabilidade, resultante do rácio entre o número de registos em que o antecedente e consequente da regra é verdadeiro, e o número total de registos em que o antecedente é verdadeiro (independentemente do consequente o ser ou não). Este tipo de regras é também acompanhado pelo nível de significância das regras, representativo do seu grau de validade. A significância resulta da subtracção à unidade do erro de probabilidade, sendo este uma quantificação da probabilidade da regra existir acidentalmente no conjunto de dados em questão.
- *regras ortográficas* – Um exemplo deste tipo de regras é: O valor *Loja do PC's* aparece 52 vezes no atributo *Cliente* – há 2 caso(s) que contêm valores similares. O objectivo principal destas regras é contribuir para a identificação de erros ortográficos. Um determinado valor é considerado como erro ortográfico se é semelhante a um outro valor que conste no mesmo atributo, sendo a sua frequência muito baixa e a frequência deste último muito elevada.

Após a fase de descoberta, a ferramenta *WizRule* identifica as excepções às regras. No entanto, nem todas as excepções são assinaladas como sendo possíveis problemas de qualidade. Para evitar o reporte de falsos problemas de qualidade, é calculado o grau de improbabilidade de cada excepção. Este valor indica em que medida a excepção é improvável relativamente ao conjunto de regras descobertas. O cálculo é efectuado com base no número e no nível de significância das regras que permitem identificar a excepção. Quanto menor for o grau de improbabilidade, maior a probabilidade da excepção ser mesmo um problema de qualidade dos dados. Estes casos são considerados como sendo possíveis problemas de qualidade, sendo apresentados no ecrã através de um relatório de fácil compreensão. Compete ao utilizador proceder à sua análise e aferir se, de facto, representam problemas de qualidade dos dados.

A.3 *Oracle Data Integrator*

A ferramenta *Oracle Data Integrator* [Oracle Data Integrator, 2007] permite a definição e execução de processos de integração e transformação de dados. A ferramenta pode ser usada em inúmeros contextos de manipulação de dados, nomeadamente: na criação de armazém de dados; na migração de dados; e, na replicação de dados. A definição das operações é efectuada através de uma abordagem declarativa, o que simplifica o seu desenvolvimento e manutenção. A abordagem separa a definição do processo, efectuada sob a forma de regras declarativas, dos detalhes de implementação. A ferramenta permite aceder e integrar dados provenientes de qualquer sistema de base de dados relacional existente, bem como de outras tecnologias que também manipulam dados (*e.g.*: *ERP*; *CRM*; fontes de dados *XML*), seguindo sempre a mesma metodologia. A qualidade dos dados é assegurada, uma vez que os problemas são detectados e solucionados antes da sua inserção na base de dados de destino.

O processo de integração e transformação de dados que a ferramenta permite implementar inicia-se com a especificação das operações a efectuar. Para cada operação, é necessário definir a tabela na qual serão armazenados os resultados (tabela de destino) e a tabela ou tabelas que contêm os dados a transformar (tabelas de origem). A ferramenta permite estabelecer automaticamente o mapeamento entre os atributos das tabelas de origem e os atributos da tabela de destino, com base na igualdade dos nomes dos atributos. Os restantes mapeamentos têm que ser estabelecidos manualmente. O utilizador define graficamente os *joins* necessários entre as tabelas de origem, caso estes não se encontrem definidos nos meta-dados das tabelas. O utilizador pode definir filtros que limitam os valores das tabelas de origem a considerar no processo de transformação. O utilizador também define as regras de transformação dos valores a armazenar na tabela de destino, expressas sob a forma de declarações SQL. Por último, o utilizador também pode definir as verificações a efectuar aos dados de modo a assegurar a sua integridade e consistência. A ferramenta permite detectar e armazenar as violações de integridade que possam existir (*e.g.*: a regras de negócio; à integridade referencial), para posterior análise e correcção. Os dados apenas são carregados na tabela de destino se estas verificações forem bem sucedidas.

Quando a operação de transformação estiver devidamente especificada, o utilizador pode despoletar a sua execução. Os resultados obtidos da transformação podem ser analisados, *i.e.*, os registos que transitaram para a tabela de destino e os registos que não passaram nas verificações às restrições de integridade. Os valores destes registos podem ser objecto de alteração manual, podendo a operação de transformação ser novamente executada, agora apenas para estes registos.

A ferramenta permite ao utilizador sequenciar e automatizar a execução simultânea de várias destas operações de transformação de dados.

A.4 *WizSame*

WizSame [WizSoft, 2007c] é uma ferramenta que identifica registos duplicados exactos ou aproximados. A detecção de duplicados pode ser efectuada nos registos de uma tabela ou entre os registos de duas tabelas. Para que seja possível detectar duplicados é necessário que o utilizador especifique o significado de duplicado. Por defeito, todos os atributos são considerados na detecção dos duplicados. Desta forma, para que dois registos sejam considerados duplicados é necessário que os valores de todos os seus atributos sejam iguais ou semelhantes. O utilizador pode optar por excluir um ou mais atributos da detecção. Por outro lado, o utilizador também pode estabelecer, por atributo, que os seus valores têm de ser exactamente iguais para que os registos sejam classificados como duplicados. A ferramenta também permite ao utilizador definir várias condições ligadas pelos operadores *And* e *Or*, estipulando ele próprio as situações em que há duplicidade.

Entre dois registos diferentes, conclui-se que há semelhança nos valores de um atributo quando:

- Há diferença de apenas um caracter entre os valores do atributo, seja este numérico ou textual. Esta diferença corresponde a três situações distintas: (*i*) um caracter do valor do atributo num registo é substituído por outro caracter no valor do atributo noutro registo (*e.g.*: *Susana* e *Suzana*); (*ii*) um caracter do valor do atributo num registo encontra-se ausente no valor do atributo noutro registo (*e.g.*: *David* e *Davide*); dois caracteres adjacentes do valor do atributo num registo encontram-se pela ordem inversa no valor do atributo noutro registo (*e.g.*: *Paulo* e *Pualo*).
- São usados sinónimos nos valores do atributo (*e.g.*: num registo o valor do atributo profissão é *docente*, enquanto que noutro registo o valor do mesmo atributo é *docente*). A ferramenta *WizSame* contém um dicionário de sinónimos ao qual o utilizador pode adicionar os termos sinónimos que quizer, para além daqueles que já existem de raíz.
- Há transposições nos valores do atributo entre os registos, *i.e.*, os *tokens* são os mesmos mas a ordem pela qual aparecem é diferente (*e.g.*: *Hipermercados Modelo-Continente* e *Modelo-Continente Hipermercados*). A ferramenta *WizSame* efectua todas as comparações possíveis entre os *tokens* que constituem o valor de cada atributo.

Após a detecção de duplicados é elaborado um relatório que contém todos os pares identificados. Este relatório permite ao utilizador: (*i*) visualizar os pares de registos no ecrã; (*ii*) imprimir os

pares de registos; (*iii*) criar um ficheiro de texto que contém todos os pares de registos; (*iv*) pesquisar se um determinado registo está incluído nalgum par de duplicados (permite verificar se foram identificados duplicados para um determinado registo); e, (*v*) seleccionar um conjunto de pares de registos para ser impresso ou armazenado num ficheiro de texto. Esta última opção é útil para delimitar a análise a um conjunto de pares de registos duplicados.

A ferramenta permite ao utilizador eliminar os registos duplicados e criar um ficheiro de texto que não contém redundâncias.

A.5 *matchIT*

A ferramenta *matchIT* [helpIT, 2007] focaliza-se na manipulação de duplicados em dados sobre clientes, permitindo: (*i*) detectar e remover registos duplicados existentes numa tabela; (*ii*) integrar dados provenientes de múltiplas tabelas, identificando e eliminando os duplicados existentes; e, (*iii*) detectar e remover os registos de uma tabela que existem noutra tabela (*i.e.*, identificar e eliminar sobreposições). Nestes dois últimos casos, não é obrigatório que as tabelas possuam um esquema exactamente igual, sendo permitidas algumas diferenças. A ferramenta disponibiliza um conjunto de assistentes que auxiliam o utilizador ao longo de todo o processo de manipulação de duplicados. A ferramenta opera sobre uma cópia dos dados, pelo que a primeira operação executada envolve a sua importação. A ferramenta suporta o acesso a dados que se encontram no formato *Microsoft Access*, *Microsoft Excel*, bem como aos formatos textuais habituais (*e.g.*: separado por vírgulas, separado por tabulações). Outros tipos de fontes de dados podem ser acedidas recorrendo a uma conexão *ODBC*. A ferramenta permite visualizar os dados e corrigir o conteúdo semântico (*e.g.*: linha de endereço; telefone; departamento), inferido automaticamente para cada atributo. O conteúdo de cada atributo é importante na identificação dos registos duplicados.

A detecção e remoção de duplicados pode ser efectuada com objectivo diferentes, designadamente de obter apenas um registo por: (*i*) contacto ou indivíduo; (*ii*) empresa ou família; ou, (*iii*) endereço. A identificação dos duplicados é efectuada exclusivamente com base nas seguintes chaves: (*i*) chave fonética do último nome, juntamente com o código postal; (*ii*) chave fonética do último nome, juntamente com a primeira inicial, juntamente com a chave fonética da rua; e, (*iii*) número da rua, juntamente com a chave fonética do endereço. Para que dois registos sejam considerados possíveis duplicados, é necessário haver correspondência pelo menos numa destas chaves. Sempre que isto acontece é gerado um valor representativo do seu

grau de semelhança. No cálculo deste valor são usados os valores de alguns atributos relevantes. Apesar da forma de cálculo do grau de semelhança se encontrar predefinida, o utilizador pode proceder à sua alteração. Caso o valor seja inferior ao limiar predefinido, a ferramenta não considera os registos como sendo potenciais duplicados.

Após a detecção dos possíveis duplicados são apresentados dois relatórios: resumo dos dados, ilustrando a qualidade e natureza dos valores que se encontram nos atributos (*e.g.*: valores distintos existentes nos atributos e sua distribuição percentual); e, resumo de correspondências, apresentando os duplicados encontrados na tabela. O relatório resumo de correspondências possibilita a visualização dos pares de duplicados lado a lado, evidenciado as semelhanças e diferenças existentes entre os valores dos atributos. Usando esta potencialidade, o utilizador pode: (*i*) classificar um determinado par como não sendo duplicados; (*ii*) efectuar alterações manualmente aos registos; (*iii*) copiar e colar valores dos atributos entre os registos; (*iv*) combinar os valores dos dois registos num só; e, (*v*) eliminar um dos registos. O relatório resumo de correspondências também possibilita a visualização ou impressão de uma listagem geral dos registos identificados como possíveis duplicados. Compete ao utilizador estabelecer o valor do limiar do grau de semelhança a partir do qual os tuplos são assinalados como verdadeiros duplicados, entre os que foram identificados como potenciais duplicados.

Após terem sido assinalados os registos duplicados, é possível proceder-se à criação de um ficheiro isento de redundâncias. Ao utilizador é permitido seleccionar os atributos a incluir neste ficheiro, podendo este ser criado de acordo com diversos formatos (*e.g.*: *xls*, *dbf*, *ASCII*). Os registos assinalados como duplicados também podem ser armazenados em ficheiro. Igualmente, este ficheiro pode ser criado segundo diversos formatos suportados. Além da questão dos duplicados, a ferramenta também permite uniformizar nomes e endereços em maiúsculas e minúsculas, de forma apropriada.

A.6 *Centrus Merge/Purge*

A ferramenta *Centrus Merge/Purge* [Group 1 Software, 2007] visa a identificação e remoção de registos duplicados numa tabela ou entre tabelas. Os seguintes tipos de fontes de dados são suportados por esta ferramenta: *Oracle*; *dBase*; *FoxPro*; *Clipper* e *ASCII* de tamanho fixo ou delimitado por caracter.

Esta ferramenta implementa fielmente o método da vizinhança ordenada, na variante multi-passagem (ver Capítulo 3 – Secção 3.4.1.3). O utilizador começa por especificar uma ou mais chaves de ordenação, baseadas num ou em vários atributos. Estas chaves são baseadas nos valores, partes de valores (*e.g.*: primeiros três dígitos do valor do atributo) ou transformações dos valores que constam dos atributos. As chaves de ordenação definem as diferentes formas de sequenciamento dos registos que serão usadas na identificação dos duplicados. O seu objectivo é colocar registos semelhantes, próximos uns dos outros.

O utilizador também define o critério que permite identificar os duplicados, o que envolve especificar o atributo ou par de atributos entre os quais deve haver correspondência ou equivalência nos seus valores, bem como o tipo de correspondência que deve existir. A correspondência entre os pares de atributos pode ser analisada a vários níveis, em virtude dos vários algoritmos disponíveis. Entre estes, encontram-se: (*i*) *frequência de caracteres* – a frequência de ocorrência dos diferentes caracteres presentes no valor de cada atributo é comparada; (*ii*) *distância de edição* – número de operações de edição necessárias (*i.e.*, inserções, alterações ou eliminações de caracteres) para transformar um valor no outro; (*iii*) *correspondência de nomes* – identifica correspondências entre diferentes representações do mesmo nome (*e.g.*: devido ao uso de abreviaturas); (*iv*) *comparação numérica* – determina se dois valores numéricos são iguais ou não; e, (*v*) *soundex* – os valores dos atributos são comparados em função do som resultante da sua pronúncia. O critério de identificação de duplicados é aplicado por cada chave de ordenação existente. Sempre que dois registos são identificados como duplicados, esta informação é armazenada. É com base nesta informação que os diversos grupos de registos duplicados são formados. Em cada um destes grupos será eleito um registo "mestre", em função de um determinado critério de prioridade (*e.g.*: prioridade atribuída a cada atributo; presença de valor no atributo; escolha aleatória). A ferramenta também permite efectuar consolidação de dados, *i.e.*, os valores dos atributos dos registos que fazem parte de cada grupo de duplicados são consolidados, dando origem a um novo registo. Em cada atributo dos registos consolidados, o utilizador define o critério de consolidação que estabelece como surgem estes valores.

A ferramenta permite a criação de múltiplas tabelas onde são colocados os diferentes tipos de resultados obtidos do processo de manipulação dos duplicados. Entre estas, encontram-se as tabelas que suportam o armazenamentos dos: (*i*) *registos únicos* – registos para os quais não foram identificados duplicados; (*ii*) *duplicados mestre* – registos seleccionados em cada grupo de duplicados; (*iii*) *duplicados subordinados* – registos preteridos em cada grupo de duplicados, *i.e.*, aqueles que não são duplicados mestre; e, (*iv*) *registos consolidados* – registos que resultam da

combinação dos valores que se encontram nos vários tuplos duplicados que compõem cada grupo. Os tipos de fontes nos quais estas tabelas podem ser criadas são os mesmos já mencionados anteriormente. A ferramenta também permite a elaboração de relatórios sobre os diversos tipos de resultados obtidos (*e.g.*: registos únicos; registos consolidados). Além de poderem ser visualizados e impressos, estes relatórios também podem ser armazenados sob a forma de um ficheiro *ASCII*.

Apêndice B

ÁLGEBRA RELACIONAL

Neste apêndice apresenta-se uma descrição muito sucinta sobre álgebra relacional e algumas das operações que esta disponibiliza. O critério usado na selecção destas baseou-se na sua utilização na formalização semântica das operações de detecção e correcção apresentadas, respectivamente, nos Capítulos 5 e 6 desta dissertação.

B.1 Conceito

A álgebra relacional é uma linguagem composta por operações que incidem sobre uma ou mais relações, para dar origem a uma outra relação. Da execução das operações não resulta qualquer alteração à relação ou relações originais. O resultado de uma operação pode ser utilizado noutra operação, *i.e.*, as operações em álgebra relacional podem estar encaixadas/imbricadas entre si, tal como acontece nas operações aritméticas. Há diversas variações de sintaxe nas operações de álgebra relacional. Nesta dissertação utiliza-se uma notação simbólica usual na representação das operações. Também há muitas variantes das operações incluídas na álgebra relacional. Em [Codd, 1972] foram inicialmente apresentadas oito operações, mas desde então, muitas outras foram propostas. No âmbito deste trabalho, adoptaram-se as operações consideradas necessárias à formalização das operações de detecção e correcção. Essencialmente, estas operações de álgebra relacional podem ser divididas em dois grupos: básicas e complementares. Cada um destes grupos de operações é apresentado nas duas secções seguintes.

B.2 Operações Básicas

As operações básicas são aquelas que se encontram na generalidade das propostas de operações de álgebra relacional. Estas operações satisfazem a generalidade das necessidades de manipulação dos dados. Em álgebra relacional há cinco operações básicas: selecção; projecção; produto Cartesiano; diferença; e, união. Nas secções seguintes são apresentadas sucintamente as quatro primeiras. A operação de união fica excluída em virtude de não ser utilizada no contexto das formalizações das operações de detecção e correcção.

B.2.1 Selecção

A operação de selecção, simbolicamente representada por $\sigma_{cond}(\mathcal{R})$, constitui uma operação unária que actua sobre uma relação $\mathcal{R}$ e produz uma nova relação que contém apenas os tuplos de $\mathcal{R}$ que satisfazem a condição *cond* especificada. Esta operação funciona como um filtro sobre os tuplos de $\mathcal{R}$.

B.2.2 Projecção

A operação de projecção, simbolicamente representada por $\pi_{atr_1,\ldots,atr_n}(\mathcal{R})$, constitui uma operação unária que actua sobre uma relação $\mathcal{R}$ e produz uma nova relação que contém todos os valores dos atributos atr_1, …, atr_n, em que estes constituem um subconjunto dos atributos de $\mathcal{R}$.

B.2.3 Produto Cartesiano

A operação produto cartesiano, simbolicamente representada por $\mathcal{R} \times \mathcal{S}$, constitui uma operação binária que produz uma nova relação resultante da combinação de todos os tuplos da relação $\mathcal{R}$, com todos os tuplos da relação $\mathcal{S}$.

B.2.4 Diferença

A operação diferença, simbolicamente representada por $\mathcal{R} - \mathcal{S}$, constitui uma operação binária que produz uma nova relação composta pelos tuplos que se encontram na relação $\mathcal{R}$, mas que não se encontram na relação $\mathcal{S}$.

B.3 Operações Complementares

As operações complementares de álgebra relacional são aquelas que podem ser expressas em função das operações básicas (como é o caso do *join*) ou que, entre as diversas propostas de operações de álgebra relacional existentes na literatura, foram consideradas úteis para a formalização da semântica inerente às operações de detecção e correcção.

B.3.1 *Join*

Nas expressões em álgebra relacional apresentadas nos Capítulos 5 e 6 desta dissertação, recorre-se à utilização de dois tipos de operações *join*: *theta-join* e *join* natural. Cada um destes tipos é apresentado nas subsecções seguintes.

B.3.1.1 *Theta-Join*

A operação *theta-join*, simbolicamente representada por $\mathcal{R} \triangleright\triangleleft_{cond} \mathcal{S}$, constitui uma operação binária que produz uma nova relação que contém os tuplos que satisfazem a condição *cond*, resultantes do produto Cartesiano entre os tuplos da relação $\mathcal{R}$ e os tuplos da relação $\mathcal{S}$. A condição *cond* possui o seguinte formato: $\mathcal{R}.a_1 \theta \mathcal{S}.b_1$, em que θ constitui um operador de comparação (*i.e.*, $<$; $\leq$; $>$; $\geq$; $=$; $\neq$). Os símbolos a_1 e b_1 representam atributos, respectivamente, de $\mathcal{R}$ e $\mathcal{S}$.

B.3.1.2 *Join* Natural

A operação *join* natural, simbolicamente representada por $\mathcal{R} \triangleright\triangleleft \mathcal{S}$, constitui uma operação binária que produz uma nova relação que contém os tuplos resultantes do produto Cartesiano entre os tuplos da relação $\mathcal{R}$ e os tuplos da relação $\mathcal{S}$, em que os valores dos atributos comuns entre $\mathcal{R}$ e $\mathcal{S}$ são iguais. Na relação que resulta desta operação apenas surge uma ocorrência de cada atributo comum a ambas as relações.

B.3.2 Renomeação

A operação de renomeação, simbolicamente representada por $\rho_{\mathcal{S}(b_1 \leftarrow a_1,\ldots,b_n \leftarrow a_n)}(\mathcal{R})$, constitui uma operação unária que produz uma nova relação $\mathcal{S}$ que constitui uma cópia da relação $\mathcal{R}$, em que o atributo a_1 de $\mathcal{R}$ é renomeado para b_1 em $\mathcal{S}$ e assim sucessivamente até ao atributo a_n de $\mathcal{R}$ que é renomeado para b_n em $\mathcal{S}$.

B.3.3 Agregação

A operação de agregação, simbolicamente representada por ${}_{atr_1,\ldots,atr_n}\gamma_{func\_agrega}(\mathcal{R})$, constitui uma operação unária que produz uma nova relação em que os tuplos da relação $\mathcal{R}$ se encontram agrupados por $atr_1,\ldots, atr_n$ e que contém os valores resultantes da aplicação da função de agregação *func_agrega* (*e.g.*: média; máximo; mínimo) a cada um destes grupos. Na nova relação, cada grupo de tuplos de $\mathcal{R}$ corresponde a um só tuplo.

B.3.4 Ordenação

A operação de ordenação, simbolicamente representada por $\tau_{atr_1,\ldots,atr_n}(\mathcal{R})$, constitui uma operação unária que produz uma nova relação em que os tuplos da relação $\mathcal{R}$ se encontram ordenados em função dos valores dos atributos $atr_1, \ldots, atr_n$. O primeiro critério de ordenação baseia-se nos valores de atr_1 e o último critério de ordenação baseia-se nos valores de atr_n.

A operação de ordenação, simbolicamente representada por $\tau_{atr_1,\ldots,atr_n}(\mathcal{R})$, constitui uma operação unária que produz uma nova relação em que os tuplos da relação $\mathcal{R}$ se encontram ordenados em função dos valores dos atributos atr_1, …, atr_n. O primeiro critério de ordenação baseia-se nos valores de atr_1 e assim sucessivamente até ao último critério de ordenação baseado nos valores de atr_n.

REFERÊNCIAS BIBLIOGRÁFICAS

Akao, Y. (1990) – *Quality Function Deployment: Integrating Customer Requirements into Product Design*. Productivity Press Inc., 1990.

Atzeni, P. e Antonellis, V. (1983) – *Relational Database Theory*. The Benjamin/Cummings Publishing Company, 1983.

Baeza-Yates, R. e Ribeiro-Neto, B. (1999) – *Modern Information Retrieval*. ACM Press, 1999.

Ballou, D. e Tayi, G. (1999) – Enhancing Data Quality in Data Warehouse Environments. *Communications of the ACM*, 42:1 (1999), pp. 73-78.

Ballou, D.; Madnick, S. e Wang, R. Y. (2003) – Special Section: Assuring Information Quality. *Journal of Management Information and Systems*, 20:3 (2003), pp. 9-11.

Ballou, D.; Wang, R.; Prazer, H. e Tayi, G. (1998) – Modeling Information Manufacturing Systems to Determine Information Product Quality. *Management Science*, 44:4 (1998), pp. 462-484.

Barateiro, J. e Galhardas, H. (2005) – A Survey of Data Quality Tools. *Datenbank-Spektrum*, 14 (2005), pp. 15-21.

Baskarada, S.; Koronios, A. e Gao, J. (2006) – Towards a Capability Maturity Model for Information Quality Management: A TDQM Approach. In *Proceedings of the 11th International Conference on Information Quality*, Boston (EUA), Novembro de 2006. pp. 499-510.

Batini, C. e Scannapieco, M. (2006) – *Data Quality: Concepts, Methodologies and Techniques*. Springer, 2006. ISBN 978-3-540-33172-8.

Batini, C.; Lenzerini, M. e Navathe, S. (1986) – A Comparative Analysis of Methodologies for Database Schema Integration. *Computing Surveys*, 18:4 (1986), pp. 323-364.

Bilenko, M. e Mooney, R. J. (2003) – Adaptive Duplicate Detection Using Learnable String Similarity Measures. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Washington (EUA), Agosto de 2003. pp. 39-48.

Bovee, M.; Srivastava, R. P. e Mak, B. (2003) – A Conceptual Framework and Belief-Function Approach to Assessing Overall Information Quality. *International Journal of Intelligent Systems*, 18 (2003), pp. 51-74.

Brackstone, G. (1999) – Managing Data Quality in a Statistical Agency. *Survey Methodology*, 25:2 (1999), pp. 139-149.

Bressan, S.; Goh, C.; Fynn, K.; Jakobisiak, M. Hussein, K.; Kon, H.; Lee, T.; Madnick, S.; Pena, T.; Qu, J.; Shum, A. e Siegel, M. (1997) – The COntext INterchange Mediator Prototype. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Arizona (EUA), Maio de 1997. pp. 525-527.

Calvanese, D.; Giacomo, G.; Lenzerini, M.; Nardi, D. e Rosati, R. (1999) – A Principled Approach to Data Integration and Reconciliation in Data Warehousing. In *Proceedings of the International Workshop on Design and Management of Data Warehouses*, Heidelberg (Alemanha), Junho de 1999. pp. 16.1-16.11.

Chandrasekaran, B. (1999) – What are Ontologies and Why Do We Need Them?. *IEEE Intelligent Systems and Their Applications*, 9:1 (1999), pp. 20-26.

Chaudhuri, S e Dayal, U. (1997) – An Overview of Data Warehousing and OLAP Technology. *SIGMOD Record*, 26:1 (1997), pp. 65-74.

Chengalur-Smith, I.; Ballou, D. e Prazer, H. (1999) – The Impact of Poor Data Quality Information on Decision Making: An Exploratory Analysis. *IEEE Transactions on Knowledge and Data Engineering*, 11:6 (1999), pp. 853-864.

Cholvy, L. e Moral, S. (2001) – Merging Databases: Problems and Examples. *International Journal of Intelligent Systems*, 16 (2001), pp. 1193-1221.

Cochinwala, M.; Kurien, V.; Lalk, G. e Shasha, D. (2001) – Efficient Data Reconciliation. *Information Sciences*, 137 (2001), pp. 1-15.

Cohen, W. e Richman, J. (2001) – Learning to Match and Cluster Entity Names. In *Proceedings of the ACM SIGIR'01 Workshop on Mathematical/Formal Methods in Information Retrieval*, Louisiana (EUA), Setembro de 2001.

Cohen, W. e Richman, J. (2002) – Learning to Match and Cluster Large High-Dimensional Data Sets for Data Integration. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Alberta (Canada), Julho de 2002. pp. 475-480.

Costa, A. (2006) – A Gestão da Qualidade dos Dados em Ambientes de Data Warehousing na Prossecução da Excelência da Informação. *Dissertação de Mestrado em Sistemas de Dados e Processamento Analítico*, orientada pelo Prof. Doutor Orlando Belo. Universidade do Minho, 2006.

Damerau, F. (1964) – A Technique for Computer Detection and Correction of Spelling Errors. *Communications of the ACM*, 7:3 (1964), pp. 659-664.

Dasu, T. e Johnson, T. (2003) – *Exploratory Data Mining and Data Cleaning*. Willey and Sons, 2003.

Dasu, T.; Johnson, T.; Muthukrishnan, S. e Shkapenyuk, V. (2002) – Mining Database Structure; Or, How to Build a Data Quality Browser. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, Wisconsin (EUA), Junho de 2002. pp. 240-251.

Dasu, T.; Vesonder, G. T. e Wright, J. R. (2003) – Data Quality Through Knowledge Engineering. In *Proceedings of the SIGKDD'03 Conference*, Washington, August 2003. pp. 705-710.

DataFlux (2007a) – Accelerate to Compliance, Data Governance and MDM. Disponível em http://www.dataflux.com/Resources/file-stream.asp?rid=146, no dia 03/04/2007, às 22:40.

DataFlux (2007b) – Data Profiling: The Diagnosis for Better Enterprise Information. Disponível em http://www.dataflux.com/Resources/file-stream.asp?rid=139, no dia 03/04/2007, às 23:00.

Datiris (2007) – Datiris Profiler: Features and Benefits. Disponível em http://www.datiris.com/features.shtml, no dia 06/04/2007, às 22:20.

Deming, W. (1986) – *Out of Crisis*. MIT Center for Advanced Enginnering Study, 1986.

Eckerson, W. (2002) – Data Quality and the Bottom Line - Achieving Business Success through a Commitment to High Quality Data. In *The Data Warehousing Institute's Data Quality Report*, 2002.

English, L. (1999) – Improving Data Warehouse and Business Information Quality: *Methods for Reducing Costs and Increasing Profits*. Willey and Sons, 1999.

ETI (2007a) – ETI Data Cleanser: Process Driven Data Cleansing and Matching for the Transparent Enterprise. Disponível em http://www.eti.com/data_sheets/ ds_ETI_DataCleanser.pdf, no dia 02/04/2007, às 23:20.

ETI (2007b) – ETI Data Quality Solutions: Analysis, Insight and Accuracy with ETI Data Profiler. Disponível em http://www.eti.com/data_sheets/ ds_ETI_DataProfiler.pdf, no dia 02/04/2007, às 23:40.

Fayyad, U.; Piatetsky-Shapiro, G. e Smyth, P. (1996) – The KDD Process for Extracting Useful Knowledge from Volumes of Data. *Communications of the ACM*, 39:11 (1996), pp. 27-34.

Feekin, A. e Chen, Z. (2000) – Duplicate Detection Using K-way Sorting Method. In *Proceedings of the 2000 ACM Symposium on Applied Computing*, Como (Italy), March 2000. pp. 323-327.

Fellegi, I. e Sunter, A. (1969) – A Theory for Record Linkage. *Journal of American Statistical Association*, 64 (1969), pp. 1183-1210.

Fisher, C. W. e Kingma, B. R. (2001) – Criticality of Data Quality as Exemplified in Two Disasters. *Information and Management* , 39 (2001), pp. 109-116.

Fox, C.; Levitin, A. e Redman, T. (1994) – The Notion of Data and Its Quality Dimensions. *Information Process Management*, 30:1 (1994), pp. 9-19.

Galhardas, H.; Florescu, D.; Shasha, D. e Simon, E (2000) – AJAX: An Extensible Data Cleaning Tool. In *Proceedings of the ACM SIGMOD on Management of Data*, Dallas (EUA), Maio de 2000. pp. 590.

Galhardas, H.; Florescu, D.; Shasha, D. e Simon, E. (2000) – Declaratively Cleaning your Data Using AJAX. In *16èmes Journées Bases de Données Avancées*, Blois(France), October 2000.

Galhardas, H.; Florescu, D.; Shasha, D.; Simon, E. e Saita, C.A. (2001) – Declarative Data Cleaning: Language, Model and Algorithms. In *Proceedings of the 27th Very Large Databases Conference*, Roma (Itália), Setembro de 2001. pp. 371-380.

Genero, M.; Poels, G. e Piattini, M. (2002) – Defining and Validating Measures for Conceptual Data Model Quality. In *Proceedings of the 14th International Conference on Computer Advances in Software Engineering*, Toronto (Canada), Maio de 2002. pp. 724-727.

Gertz, M.; Ozsu, M. T.; Saake, G. e Sattler, K.-U. (2004) – Report on the Dagstuhl Seminar "Data Quality on the Web". *SIGMOD Record*, 33:1 (2004), pp. 127-132.

Giovani Rubert Librelotto (2005) – Topic Maps: da Sintaxe à Semântica. *Dissertação de Doutoramento em Informática*. Universidade do Minho (Portugal), 2005.

Gravano, L.; Ipeirotis, P. G.; Koudas, N. e Srivastava, D. (2001) – Approximate String Joins in a Database (Almost) for Free. In *Proceedings of the 27th Very Large Databases Conference*, Roma (Itália), 2001. pp. 491-500.

Gravano, L.; Ipeirotis, P. G.; Koudas, N. e Srivastava, D. (2003) – Text Joins in an RDBMS for Web Data Integration. In *Proceedings of the 12th International Conference on World Wide Web*, Budapest (Hungria), Maio de 2003. pp. 90-101.

Group 1 Software (2007) – Centrus Merge/Purge. Disponível em http://www.centrus.com/documents/MergePurge.pdf, no dia 07/04/2007, às 23:25.

Guyon, I.; Matic, N. e Vapnik, V. (1996) – Discovering Informative Patterns and Data Cleaning. In *Advances in Knowledge Discovery and Data Mining*. American Association for Artificial Intelligence, 1996, pp. 181-203. ISBN 0-262-56097-6.

Hamming, R. (1950) – Error Detecting and Error Correcting Codes. *Bell System Tech Journal*, 9 (1950), pp. 147-160.

Hellerstein, J.; Stonebraker, M. e Caccia. R. (1999) – Independent, Open Enterprise Data Integration. *IEEE Technical Bulletin on Data Engineering – Special Issue on Data Transformation*, 22:1 (1999), pp. 43-49.

helpIT (2007) – The matchIT Suite version 5.0. Disponível em http://helpit.com/assets/brochure_pdfs/matchitsuite.pdf, no dia 07/04/2007, às 23:10.

Hernández, M. A. e Stolfo, S. J. (1995) – The Merge/Purge Problem for Large Databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Jose – California (EUA), May 1995. pp. 127-138.

Hernández, M. A. e Stolfo, S. J. (1998) – Real-World Data is Dirty: Data Cleansing and The Merge/Purge Problem. *Data Mining and Knowledge Discovery*, 2 (1998), pp. 9-37.

Hipp, J.; Güntzer, U. e Grimmer, U. (2001) – Data Quality Mining: Making a Virtue of Necessity. In *Proceedings of the 6th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, California (EUA), 2001. pp. 52-57.

Huang, K.; Lee, Y. e Wang, R. Y. (1999) – *Quality Information and Knowledge*. Prentice-Hall, Inc., 1999.

Human Inference (2007a) – The Power of Knowledge. Disponível em http://www.humaninference.com/filebank/originals/original_1394.pdf, no dia 02/04/2007, às 22:30.

Human Inference (2007b) – HIQuality Inspect: Data Quality Profiling. Disponível em http://www.humaninference.com/filebank/originals/original_1275.pdf, no dia 02/04/2007, às 22:50.

Human Inference (2007c) – HIquality Transform: Data Filtering and Transformation. Disponível em http://www.humaninference.com/filebank/originals/original_864.pdf, no dia 02/04/2007, às 23:50.

Human Inference (2007d) – HIquality Name: Name Cleansing and Validation. Disponível em http://www.humaninference.com/filebank/originals/original_627.pdf, no dia 03/04/2007, às 0:05.

Human Inference (2007e) – HIquality Address: Rapid Addressing and Standardization. Disponível em http://www.humaninference.com/filebank/originals/original_524. pdf, no dia 03/04/2007, às 0:15.

Human Inference (2007f) – HIquality Identify: Fault-tolerant searching, matching and duplicate detection. Disponível em http://www.humaninference.com/filebank/originals/original_472.pdf, no dia 03/04/2007, às 0:25.

Human Inference (2007g) – HIquality Merge: Consolidation of Information. Disponível em http://www.humaninference.com/filebank/originals/original_1254.pdf, no dia 03/04/2007, às 0:35.

Human Inference (2007h) – HIquality Enrich: Makes your Information Reflect Reality. Disponível em http://www.humaninference.com/filebank/originals/original_1213.pdf, no dia 04/04/2007, às 0:00.

Hylton, J. (1996) – Identifying and Merging Related Bibliographic Records. *MSc Dissertation*. MIT, 1996.

Inmon, W.; Rudin, K.; Buss, C. e Sousa, R. (1998) – *Data Warehouse Performance*. Willey, 1998. ISBN 471298085.

Ives, B.; Olson, M. e Baroudi, J. (1983) – The Measurement of User Information Satisfaction. *Communications of the ACM*, 26:10 (1983), pp. 785-793.

Jarke, M.; Lenzerini, M.; Vassiliou, Y. e Vassiliadis, P. (1995) – *Fundamentals of Data Warehouses*. Springer Verlag, 1995.

Jaro, M. (1989) – Advances in Record-Linkage Methodology as Applied to Matching the 1985 Census of Tampa, Florida. *Journal of American Statistical Association*, 84:406 (1989), pp. 414-420.

Jin, L.; Koudas, N. e Li, C. (2004) – NNH: Improving Performance of Nearest-Neighbor Searches Using Histograms. In *Proceedings of 9th International Conference on Extending Database Technology*, Creta (Grécia), Março de 2004. pp. 385-402.

Jin, L.; Koudas, N.; Li, C. e Tung, A. (2005) – Indexing Mixed Types for Approximate Retrieval. In *Proceedings of the 31st International Conference on Very Large Databases*, Trondheim (Norway), Setembro de 2005. pp. 793-804.

Kahn, B.; Strong, D. e Wang, R. Y. (2002) – Information Quality Benchmarks: Product and Services Performance. *Communications of the ACM*, 45:4 (2002), pp. 184-192.

Kilss, B. e Alvey, W. (1985) – Record Linkage Techniques. In *Proceedings of the Workshop on Exact Matching Methodologies*, Virginina (EUA), Maio de 1985. pp. 1-4.

Kim, W.; Choi, B.-J.; Hong, E.-K.; Kim, S.-K. e Lee, D. (2003) – A Taxonomy of Dirty Data. *Data Mining and Knowledge Discovery*, 7 (2003), pp. 81-99.

Kimbal, R.; Reeves, L.; Ross, M. e Thornthwaite, W. (1998) – *The Data Warehouse Lifecycle Toolkit*. John Wiley & Sons, Inc., 1998.

Koronios, A.; Lin, S. e Gao, J. (2005) – A Data Quality Model for Asset Management in Engineering Organizations. In *Proceedings of the 10th International Conference on Information Quality*, Boston (EUA), Novembro de 2005. pp. 27-51.

Kubica, J. e Moore, A. (2003) – Probabilistic Noise Identification and Data Cleaning. In *Proceedings of the 3rd IEEE International Conference on Data Mining*, Florida (EUA), Novembro de 2003. pp. 131-138.

Lee, M. e Ling, T. W. (1997) – Resolving Constraint Conflicts in the Integration of Entity-Relationship Schemas. In *Proceedings of the 16th International Conference on Conceptual Modeling*, California (EUA), Novembro de 1997. pp. 394-407.

Lee, M. L.; Ling, T. W. e Low, W. L. (2000a) – IntelliClean: A Knowledge-Based Intelligent Data Cleaner. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* , Boston (EUA), August 2000. pp. 290-294.

Lee, M. L.; Lu, H.; Ling, T. W. e Ko, Y. T. (1999) – Cleansing Data for Mining and Warehousing. In *Proceedings of the 10th International Conference on Database and Expert Systems Applications*, Florence (Italy), September 1999. pp. 751-760.

Lee, Y. e Strong, D. (2003) – Process Knowledge and Data Quality Outcomes. In *Proceedings of the 8th International Conference on Information Quality*, Boston (EUA), Outubro de 2003. pp. 96-107.

Lee, Y.; Bowen, P.; Funk, J.; Jarke, M.; Madnick, S. e Wand, Y. (2000b) – Data Quality in Internet Time, Space, and Communities. In *Proceedings of the 21st International Conference on Information Systems*, Queensland (Austrália), Dezembro de 2000 . pp. 713-716.

Levenshtein, V. (1966) – Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Doklady Akademii Nauk SSSR*, 163:4 (1966), pp. 845-848.

Liu, X. (1999) – Progress in Intelligent Data Analysis. *Applied Intelligence*, 11 (1999), pp. 235–240.

Lopes, A. e Babbitt, T. (1999) – Knowledge Management: Differing Ideals, Differing IT Implications. In *Proceedings of the Americas Conference on Information Systems*, Milwaukee (EUA), Agosto de 1999. pp. 477-479.

Low, W. L.; Lee, M. L. e Ling, T. W. (2001) – A Knowledge-Based Approach for Duplicate Elimination in Data Cleaning. *Information Systems*, 26 (2001), pp. 585-606.

Luebbers, D.; Grimmer, U. e Jarke, M. (2003) – Systematic Development of Data Mining-Based Quality Tools. In *Proceedings of the 29th Very Large Databases Conference*, Berlin (Alemanha), Setembro de 2003. pp. 548-549.

Maletic, J. I. e Marcus, A. (2000a) – Automated Identification of Errors in Data Sets. In *Technical Report n.º TR-CS-00-02*, Universidade de Memphis, 2000.

Maletic, J. I. e Marcus, A. (2000b) – Data Cleansing: Beyond Integrity Analysis. In *Proceedings of the 5th International Conference on Information Quality*, Boston (EUA), Outubro de 2000. pp. 200-209.

McCallum, A.K.; Nigam, K. e Ungar, L. (2000) – Efficient Clustering of High-Dimensional Data Sets with Application to Reference Matching. In *Proceedings of the 6th International Conference on Knowledge Discovery and Data Mining*, Boston (EUA), Agosto de 2000. pp. 169-178.

Mellisa Data (2007) – MatchUp. Disponível em http://www.melissadata.com/products/matchup.htm, no dia 07/04/2007, às 23:25.

Metais, E.; Kedad, Z.; Comyn-Wattiau, I. e Bouzeghoub, M. (1997) – Using Linguistic Knowledge in View Integration: Toward a Third Generation of Tools. *Data and Knowledge Enginnering*, 23 (1997), pp. 59-78.

Milano, D.; Scannapieco, M. e Catarci, T. (2005) – Using Ontologies for XML Data Cleaning. *Lecture Notes on Computer Science*, 3762 (2005), pp. 562-571.

Missier, P.; Lalk, G.; Verykios, V.; Grillo, F.; Lorusso, T. e Angeletti, P. (2003) – Improving Data Quality in Practice: A Case Study in the Italian Public Administration. *Distributed and Parallel Databases*, 13 (2003), pp. 135-160.

Monge, A. E. (2000) – Matching Algorithms Within a Duplicate Detection System. *Bulletin of the Technical Committe on Data Engineering – Special Issue on Data Cleaning*, 23:4 (2000), pp. 14-20.

Monge, A. E. e Elkan, C. P. (1997) – An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records. In *Proceedings of the SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*, Arizona (EUA), Maio de 1997. pp. 23-29.

Müller, H. e Freytag, J.-C. (2003) – Problems, Methods, and Challenges in Comprehensive Data Cleansing. *Technical Report HUB-IB-164*, Humboldt-Universität zu Berlin, Institut für Informatik, 2003.

Naumann, F. (2002) – *Quality-Driven Query Answering for Integrated Information Systems*. Springer Verlag, 2002. ISBN 978-3-540-43349-1.

Naumann, F. e Häussler, M. (2002) – Declarative Data Merging With Conflict Resolution. In *Proceedings of the 7th International Conference on Information Quality*, Boston (EUA), Novembro de 2002. pp. 212-224.

Newcombe, H. e Kennedy, J. (1962) – Record Linkage: Making Maximum Use of the Discriminating Power of Information. *Communications of the ACM*, 5 (1962), pp. 563-567.

Newcombe, H.; Kennedy, J.; Axford, S. e James, A. (1959) – Automatic Linkage of Vital Records. *Science*, 130 (1959), pp. 954-959.

Oliveira, P.; Rodrigues, F. e Henriques, P. (2004) – Limpeza de Dados: Uma Visão Geral. In *Proceedings of the Data Gadgets 2004 Workshop – Bringing Up Emerging Solutions for Data Warehousing Systems* (em conjunto com a *JISBD'04*), Málaga (Espanha), Novembro de 2004. pp. 39-51.

Oliveira, P.; Rodrigues, F. e Henriques, P. (2005b) – A Framework for Detection and Correction of Data Quality Problems. In *Proceedings of the Data Gadgets 2005 International Workshop - Bringing Up Emerging Solutions for Data Warehousing Systems* (em conjunto com a *JISBD'05*), Granada (Espanha), Setembro de 2005. pp. 59-74.

Oliveira, P.; Rodrigues, F. e Henriques, P. (2005c) – A Formal Definition of Data Quality Problems. In *Proceedings of the 10th International Conference on Information Quality*, MIT - Boston (EUA), Novembro de 2005. pp. 13-26.

Oliveira, P.; Rodrigues, F. e Henriques, P. (2006a) – Data Profiling versus Data Quality Problems. In *Proceedings of the 2nd International Workshop on Data and Knowledge Quality* (em conjunto com a *EGC'06*), Lille (França), Janeiro de 2006. pp. 9-15.

Oliveira, P.; Rodrigues, F. e Henriques, P. (2006b) – Data Cleaning by Reusing Domain Knowledge. In *Knowledge and Decision Technologies*. pp. 67-74. ISBN 972-8688-39-3.

Oliveira, P.; Rodrigues, F. e Henriques, P. (2006c) – An Ontology-Based Approach for Data Cleaning. In *Proceedings of the 11th International Conference on Information Quality*, MIT - Boston (EUA), Novembro de 2006. pp. 307-320.

Oliveira, P.; Rodrigues, F. e Henriques, P. (2006d) – A Taxonomy of Instance-Level Quality Problems on Relational Data. In *Proceedings of the 4th German Information Quality Management Conference*, Frankfurt (Alemanha), Novembro de 2006.

Oliveira, P.; Rodrigues, F.; Henriques, P. e Galhardas, H. (2005a) – A Taxonomy of Data Quality Problems. In *Proceedings of the 2nd International Workshop on Data and Information Quality* (em conjunto com a *CAISE'05*), Porto (Portugal), Junho de 2005. pp. 219-233.

Olson, J. (2003) – Data Quality: The Accuracy Dimension. Morgan Kaufmann Publishers, 2003.

Oracle Data Integrator (2007) – Oracle Data Integrator User's Guide. Disponível em http://www.oracle.com/technology/products/oracle-data-integrator/10.1.3/htdocs/documentation/oracledi_users.pdf, no dia 02/04/2007, às 23:40.

Orr, K. (1998) – Data Quality and Systems Theory. *Communications of the ACM*, 41:2 (1998), pp. 66-71.

Pande, P.; Newman e Cavanagh (2000) – *The SIX SIGMA Way*. McGraw-Hill, 2000.

Papakonstantinou, H.; Quass, D.; Rajaraman, A.; Sagiv, Y.; Ullman, J.; Vassalos, V. e Widom, J. (1997) – The TSIMMIS Approach to Mediation: Data Models and Languages. *Journal of Intelligent Information Systems*, 8:2 (1997), pp. 117-132.

Parssian, A.; Sarkar, S. e Jacob, V. S. (2004) – Assessing Data Quality for Information Products – Impact of Selection, Projection, and Cartesian Product. *Management Science*, 50:7 (2004), pp. 967-982.

Pipino, L. L.; Lee, Y. W. e Wang, R. Y. (2002) – Data Quality Assessment. *Communications of the ACM*, 45:4 (2002), pp. 211-218.

Rahm, E. e Bernstein, P. (2001) – A Survey of Approaches to Automatic Schema Matching. *The Very Large Databases Journal*, 10 (2001), pp. 334-350.

Rahm, E. e Do, H. H. (2000) – Data Cleaning: Problems and Current Approaches. *Bulletin of the Technical Committe on Data Engineering – Special Issue on Data Cleaning*, 23:4 (2000), pp. 3-13.

Raman, V. e Hellerstein, J. M. (2001) – Potter's Wheel: An Interactive Data Cleaning System. In *Proceedings of the 27th Very Large Databases Conference*, Roma (Itália), Setembro de 2001. pp. 381-390.

Redman, T. C. (1996) – *Data Quality for the Information Age*. Artech House, 1996.

Redman, T. C. (1998) – The Impact of Poor Data Quality on the Typical Enterprise. *Communications of the ACM*, 41:2 (1998), pp. 79-82.

Redman, T. C. (2004) – Data: An Unfolding Quality Disaster. In DM Review Magazine.

Salaun, Y. e Flores, K. (2001) – Information Quality: Meeting the Needs of the Consumer. *International Journal of Information Management*, 21:1 (2001), pp. 21-37.

Sarawagi, S. e Bhamidipaty, A. (2002) – Interactive Deduplication using Active Learning. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Alberta (Canada), 2002. pp. 269-278.

Sattler, K. U.; Conrad, S. e Saake, G. (2000) – Adding Conflict Resolution Features to a Query Language for Database Federations. In *Proceedings of the 3rd International Workshop on Engineering Federated Information Systems*, Dublin (Irlanda), Junho de 2000. pp. 41-52.

Sattler, K.-U. e Schallehn, E. (2001) – A Data Preparation Framework Based on a Multidatabase Language. In *Proceedings of the International Database Engineering and Applications Symposium*, Grenoble (França), Julho de 2001. pp. 219-228.

Scannapieco, M. (2004) – DaQuinCIS: Exchanging and Improving Data Quality in Cooperative Information Systems. *PhD Dissertation*. Università degli Studi di Roma "La Sapienza", 2004.

Scannapieco, M.; Virgillito, A.; Marchetti, C.; Mecella, M. e Baldoni, R. (2004) – The DaQuinCIS Architecture: A Platform for Exchanging and Improving Data Quality in Cooperative Information Systems. *Information Systems*, 29 (2004), pp. 551-582.

Shankaranarayanan, G. (2005) – Towards Implementing Total Data Quality Management in a Data Warehouse. *Journal of Information Technology Management*, 16:1 (2005), pp. 21-30.

Shankaranarayanan, G. e Watts-Sussman, S. (2003) – A Relevant Believable Approach for Data Quality Assessment. In *Proceedings of the 8th International Conference on Information Quality*, Boston (EUA), Outubro de 2003. pp. 178-189.

Shankaranarayanan, G.; Wang, R. e Ziad, M. (2000) – IP-MAP: Representing the Manufacture of an Information Product. In *Proceedings of the 5th International Conference on Information Quality*, Boston (EUA), Outubro de 2000. pp. 1-16.

Shankaranarayanan, G.; Ziad, M e Wang, R. Y. (2003) – Managing Data Quality in Dynamic Decision Making Environments: An Information Product Approach. *Journal of Database Management*, 14:4 (2003), pp. 14-32.

Shanks, G. e Darke, P. (1998) – Understanding Data Quality in a Data Warehouse. *Australian Computer Journal*, 30:4 (1998), pp. 122-128.

Simitsis, A.;Vassiliadis, P. e Sellis, T. (2005) – Optimizing ETL Processes in Data Warehouses. In *Proceedings of the 21st International Conference on Data Engineering*, Tóquio (Japão), Abril de 2005. pp. 564-575.

Strong, D. M (1997) – IT Process Designs for Improving Information Quality and Reducing Exception Handling: A Simulation Experiment. *Information and Management*, 31 (1997), pp. 251-263.

Strong, D. M.; Lee, Y. W. e Wang, R. Y. (1997) – Data Quality In Context. *Communications of the ACM*, 40:5 (1997), pp. 103-110.

Tejada, S.; Knoblock, C. A. e Minton, S. (2001) – Learning Object Identification Rules for Information Integration. *Information Systems*, 26 (2001), pp. 607-633.

Tejada, S.; Knoblock, C. A. e Minton, S. (2002) – Learning Domain-Independent String Transformation Weights for High Accuracy Object Identification. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Alberta (Canada), Julho de 2002. pp. 350-359.

Trillium Software (2007a) – TS Quality Version 10.0, Enterprise Services Edition. Disponível em http://www.trilliumsoftware.com/site/content/resources/library/pdf_detail.asp?id=147&pdfRecorded=1, no dia 03/04/2007, às 22:15.

Trillium Software (2007b) – TS Discovery Version 5.0: Product Overview. Disponível em http://www.trilliumsoftware.com/site/content/resources/library/pdf_detail.asp?id=148&pdfRecorded=1, no dia 03/04/2007, às 22:35.

Tsur, S. e Zaniolo, C. (1986) – LDL: A Logic-Based Data-Language. In *Proccedings of the 17th International Conference on Very Large Data Bases*, Kyoto (Japão), Agosto de 1986. pp. 33-41.

Umar, A.; Karabatis, G.; Ness, L.; Horowitz, B. e Elmagardmid, A. (1999) – Enterprise Data Quality: A Pragmatic Approach. *Information Systems Frontiers*, 1:3 (1999), pp. 279-301.

Vassiliadis, P.; Simitsis, A.; Georgantas, P. e Terrovitis, M. (2003) – A Framework for the Design of ETL Scenarios. In *Proceedings of the 15th Conference on Advanced Information Systems Engineering (CAiSE'03)*, Klagenfurt (Áustria), Junho de 2003. pp. 520-535.

Verykios, V. S.; Moustakides, G. V. e Elfeky, M. G. (2003) – A Bayesian Decision Model for Cost Optimal Record Matching. *The Very Large Databases Journal*, 12 (2003), pp. 28-40.

Wand, Y. e Wang, R. Y. (1996) – Anchoring Data Quality Dimensions in Ontological Foundations. *Communications of the ACM*, 39:11 (1996), pp. 86-95.

Wang, R. e Kon, H. (1993) – Towards Total Data Quality Management (TDQM). In *Information Technology in Action: Trends and Perspectives*. Prentice-Hall, Inc., 1993, pp. 179-197.

Wang, R. Y. (1998) – A Product Perspective on Total Data Quality Management. *Communications of the ACM*, 41:2 (1998), pp. 58-65.

Wang, R. Y.; Storey, V. e Firth, C. (1995) – A Framework For Analysis of Data Quality Research. *IEEE Transactions on Knowledge and Data Engineering*, 7:4 (1995), pp. 623-640.

Wang, R.; Lee, Y.; Pipino, L. e Strong, D. (1998) – Manage your Information as a Product. *Sloan Management Review*, 39:4 (1998), pp. 95-105.

Wang, R.; Madnick, S.; Harris, W. e Allen, T. (2003) – An Information Product Approach for Total Information Awareness. In *Proceedings of the IEEE Aerospace Conference*, Montana (EUA), 2003. pp. 3005-3020.

Wang. R. Y. e Strong, D. (1996) – Beyond Accuracy: What Data Quality Means to Data Consumers. *Journal of Management Information and Systems*, 124 (1996), pp. 5-34.

Winkler, W. E. (1999) – The State of Record Linkage and Current Research Problems. In *Statistics of Income Division – Internal Revenue Service Publication R99/04*. Disponível em http://www.census.gov/srd/papers/pdf/rr99-04.pdf, no dia 30/05/2007, às 22:50.

Winkler, W. E. (2003) – Data Cleaning Methods. In *Proceedings of the ACM Workshop on Data Cleaning*, Record Linkage, and Object Identification, Washington (EUA), Agosto de 2003. pp. 1-6.

Winkler, W. E. (2004) – Methods for Evaluating and Creating Data Quality. *Information Systems*, 29:7 (2004), pp. 531-550.

WinPure (2007) – WinPure: Data Cleaning Made Easy. Disponível em http://www.winpure.com/CAM2007.pdf, no dia 06/04/2007, às 22:50.

WizSoft (2007a) – WizWhy White Paper. Disponível em http://www.wizsoft.com/default.asp?win=7&winsub=31, no dia 06/04/2007, às 23:45.

WizSoft (2007b) – WizRule White paper. Disponível em http://www.wizsoft.com/default.asp?win=8&winsub=35, no dia 06/04/2007, às 23:55.

WizSoft (2007c) – WizSame White Paper. Disponível em http://www.wizsoft.com/default.asp?win=9&winsub=37, no dia 07/04/2007, às 22:55.

Wohed, P. (2000) – Schema Quality, Schema Enrichment, and Reuse in Information Systems Analysis. *Dissertação de Doutoramento em Informática*. Universidade de Stockholm, 2000.