Universidade Técnica de Lisboa
Instituto Superior Técnico

Processo Iterativo de Migração de Dados

Aplicado à Informação de Controlo de Segurança de Barragens de Betão

José Eduardo de Mendonça Tomás Barateiro

Licenciado em Engenharia Informática e de Computadores

Dissertação para a obtenção do Grau de Mestre em Engenharia Informática e de Computadores

Orientador: Doutora Helena Isabel de Jesus Galhardas

Júri

Presidente: Doutor Alberto Rodrigues da Silva, Instituto Superior Técnico

Vogais: Doutora Helena Isabel de Jesus Galhardas, Instituto Superior Técnico
Doutor Orlando Belo, Universidade do Minho
Doutor Pável Calado, Instituto Superior Técnico

Julho de 2007

Resumo

Com a evolução das tecnologias de informação, é essencial efectuar migrações de dados para garantir o correcto funcionamento das novas aplicações. A eficiência e correcção das aplicações depende, fortemente, da qualidade dos dados armazenados.

A criação de processos de migração de dados através da implementação de *Software* específico é bastante complexa. Por isso, é fundamental disponibilizar mecanismos de apoio à realização de tarefas de tratamento e migração de dados.

A *framework Ajax* disponibiliza vários operadores para limpeza e transformação de dados. Não sendo uma ferramenta exclusivamente orientada para a migração de dados, pretende-se que a sua utilização na migração de um projecto real permita validar a respectiva aplicação. Todas as lacunas dos operadores do *Ajax* podem, assim, ser preenchidas através do refinamento da lógica dos operadores.

No âmbito do projecto *gestBarragens*[SGBP05], desenvolveu-se um processo de migração que inclui o carregamento de um sistema de informação legado e vários sistemas simples relacionados com informação acerca da segurança de barragens de betão em Portugal.

Palavras-chave: migração, transformação de dados, grafo de transformações, qualidade de dados, limpeza de dados, *ETL*.

Agradecimentos

Agradeço aos meus pais e família, pelo suporte incondicional em todo o meu percurso académico e pelo apoio nos momentos mais difíceis no desenvolvimento deste trabalho. Um especial agradecimento à Cristina pelo apoio prestado na revisão da tese.

Uma menção especial à minha orientadora, Helena Galhardas, que me ajudou a definir o "'caminho"' a seguir e foi determinante no resultado final desta tese.

Também quero agradecer a todas as pessoas envolvidas no projecto *gestBarragens*, em especial aos colegas Hugo Matos e Jorge Gonçalves, aos Professores Alberto Silva e Helena Galhardas e aos colegas do LNEC Ana Fonseca, António Amante, Eliane Portela e Tavares de Castro que contribuiram para o sucesso deste projecto.

Um obrigado especial também a todos os amigos que não estando relacionados directamente com este trabalho, estiveram sempre presentes.

Finalmente, um especial obrigado à Andreia pelo apoio incondicional e a pela força transmitida.

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O objectivo da migração de dados consiste em converter um conjunto de dados fonte, armazenados com uma estrutura específica, num determinado suporte de dados (e.g. ficheiros ASCII, base de dados relacional), para uma nova estrutura e/ou novo suporte de dados, sem perda de informação. Nesta tese, pretende-se descrever o processo de migração de um conjunto de dados de controlo de segurança de barragens de betão realizado no âmbito do projecto *gestBarragens* [SGBP05], nomeadamente os aspectos relativos à sua natureza iterativa. O projecto *gestBarragens* foi desenvolvido numa parceria entre o Laboratório Nacional de Engenharia Civil (LNEC), a EDP-Produção EM e o Instituto de Engenharia de Sistemas e Computadores Investigação e Desenvolvimento em Lisboa (INESC-ID) e teve como principal objectivo a criação de um sistema de informação para gestão e manutenção de informação relativa ao controlo de segurança de barragens de betão em Portugal.

Neste Capítulo, apresenta-se, em primeiro lugar, a motivação para recorrer à utilização de processos de migração de dados. Em segundo lugar, descrevem-se as principais metodologias usadas na migração de aplicações e dados legados. De seguida, apresenta-se o enquadramento deste trabalho no âmbito do projecto *gestBarragens*. Finalmente, descreve-se o problema a resolver, as contribuições e a organização desta tese.

1.1 Motivação

As organizações actuais manipulam quantidades cada vez maiores de dados que representam conhecimento fundamental para o seu desenvolvimento. Os sistemas de informação suportam a introdução e manutenção dos dados de cada organização e constituem a base de toda a informação do negócio da organização.

O facto dos sistemas de informação serem cada vez mais críticos para as organizações actuais torna mais visíveis os problemas que lhes estão inerentes. Os problemas mais comuns que os sistemas de informação podem apresentar são os seguintes:

- sistemas de *Hardware* obsoletos, lentos e demasiado dispendiosos no que respeita aos respectivos custos de manutenção.
- custos de manutenção de *Software* elevados, o que fica a dever-se, principalmente, às dificuldades de interpretação do código fonte e da estrutura dos sistemas implementados em tecnologias obsoletas, e à falta de documentação.
- falta de mecanismos de integração com outros sistemas, pelo que a integração tem que ser feita, normalmente, à custa de esforço manual.
- grande dificuldade, ou até mesmo impossibilidade, de expandir os sistemas, devido à introdução de novos requisitos.
- reduzida qualidade dos dados devido à forma pouco estruturada, ou mesmo não estruturada, como são armazenados.

Os sistemas de informação que resistem significativamente à modificação e evolução, por apresentarem os problemas enumerados anteriormente, designam-se por sistemas de informação legados [WDB+97]. Muitas organizações optam por migrar os sistemas de informação legados para novos ambientes que providenciam facilidades de manutenção e de adaptação a novos requisitos de negócio. Estes novos sistemas devem manter a mesma funcionalidade, bem como toda a informação armazenada pelos sistemas de informação legados. De modo a manter a informação armazenada pelos sistemas de informação legados, é necessário migrar os dados legados. Em [oPAG96], define-se Migração de

Dados como um conjunto de tarefas desenhadas e organizadas para transferir dados digitais de uma configuração de *Hardware/Software* para outra ou de uma geração de tecnologia para a geração seguinte.

O carregamento de um *Data Warehouse* [CD97], a partir dos dados armazenados num sistema operacional, é um processo em tudo semelhante ao processo de migração de dados. Os dados armazenados pelo sistema operacional equivalem aos dados mantidos pelos sistemas de informação legados, que têm que ser armazenados com uma estrutura diferente. Antes de efectuar o carregamento dos dados é necessário aplicar-lhes um conjunto de transformações de forma a corresponderem ao esquema do sistema alvo. Na construção de um *Data Warehouse*, designa-se por processo de Extracção, Transformação e Carregamento (ETL, do inglês *Extraction*, *Transformation and Loading*) a sequência de passos responsável por extrair os dados dos sistemas operacionais e transformá-los, a fim de permitir o carregamento do *Data Warehouse*.

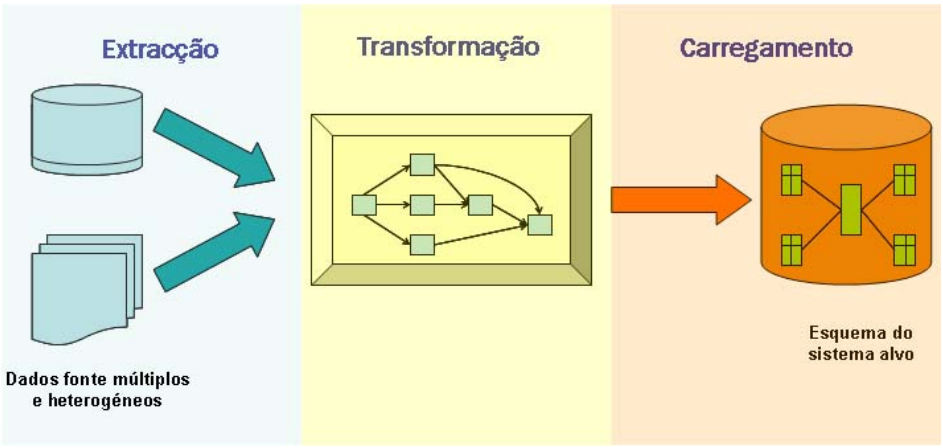

Figura 1.1: ETL - Esquema do mecanismo de Extracção, Transformação e Carregamento

Na Figura 1.1 apresentam-se as etapas de um processo ETL. A fase de *Extracção* é responsável por extrair os dados do sistema fonte. Em muitos casos, é necessário extrair dados de múltiplas fontes (por exemplo, ficheiros ASCII, Excel e bases de dados relacionais) que podem usar diferentes estruturas e formatos de dados.

A etapa de *Transformação* envolve a aplicação de um conjunto de regras e transformações aos dados fonte, de forma a derivar os dados que devem ser carregados no sistema alvo. É considerada a fase mais complexa de um processo ETL, já que deve

garantir a validação, correcção e consistência dos dados.

Finalmente, a fase de *Carregamento* é responsável por carregar os dados, anteriormente transformados, no sistema alvo. Este processo pode variar muito em função dos requisitos da organização. De facto, pode ser levado a cabo o carregamento em múltiplas bases de dados ou ser executado o carregamento incremental dos dados.

Um processo de migração de dados pode ser visto como um processo ETL com algumas restrições. Existem duas diferenças fundamentais que os distinguem. A primeira diferença está relacionada com o período de aplicabilidade dos processos de migração de dados e dos processos ETL. O processo ETL associado ao carregamento de um *Data Warehouse* é um processo cíclico que é, tipicamente, executado várias vezes, de forma periódica, com o objectivo de actualizar os dados armazenados no *Data Warehouse*. Já a migração de dados é um processo de transformação que ocorre, de uma só vez, para migrar um conjunto de dados fonte. Assim que esse conjunto de dados seja convertido para o sistema alvo, o processo de migração termina.

A segunda diferença importante entre a migração de dados e um processo ETL relaciona-se com a necessidade de limpeza de dados. Normalmente, num processo ETL as transformações são de esquema, isto é, apenas se altera o suporte e/ou a estrutura de armazenamento dos dados. Assim, num processo ETL são necessárias menos correcções para tratar inconsistências e problemas nos dados fonte. Para além disso, tendo em conta que as rotinas de limpeza de dados são bastante complexas, deve evitar-se a sua aplicação no âmbito de um processo de carregamento cíclico. Preferencialmente, a limpeza dos dados deve ser tratada ao nível do sistema operacional conduzindo, deste modo, a um aumento da qualidade dos dados tanto no sistema operacional como no *Data Warehouse*.

A qualidade dos dados armazenados pelos sistemas de informação legados é, geralmente, reduzida devido, principalmente, à falta de restrições de integridade dos sistemas de armazenamentos (normalmente ficheiros) e à falta de mecanismos de validação na introdução de dados no sistema. Assim, a migração de dados de um sistema de informação legado deve ser acompanhada de uma fase de limpeza de dados. Em [Sol], refere-se mesmo que um projecto de migração constitui uma oportunidade perfeita para limpar os dados antes de os mover para o novo sistema. A aplicação de mecanismos

de limpeza de dados pode ter como objectivo a melhoria da qualidade dos dados mas, muitas vezes, é necessária para que os dados transformados satisfaçam as restrições impostas pelo esquema de dados do sistema alvo.

1.2 Metodologias

Um projecto de migração de um sistema de informação legado cobre diferentes áreas de investigação. Bing Wu et. al [WLB+97] enunciam várias áreas de investigação que podem ser envolvidas:

> *Legacy system migration encompasses many research areas. A single migration project could, quite legitimately, address areas of reverse engineering, business reengineering, schema mapping and translation, data transformation, application development, human computer-interaction and testing.*

Atendendo à complexidade e ao risco de falha de um projecto de migração de um sistema de informação legado, é essencial definir uma metodologia detalhada que permita, de forma estruturada, implementar uma migração. Uma primeira abordagem para migrar um sistema de informação legado consiste em implementar, de raiz, todo o sistema aplicacional e de bases de dados. Esta metodologia é conhecida como a aproximação *Big Bang* ou *Cold Turkey* [BLW+97]. Na realidade, o risco de falha nesta abordagem é elevado, sendo utilizada apenas em casos excepcionais.

A metodologia de *Forward Migration*, também conhecida como *Database First* consiste em migrar inicialmente os dados e, posteriormente, e de forma incremental, migrar os componentes aplicacionais e interfaces. Os sistemas legados e o novo sistema operam em simultâneo durante a fase de migração. Pode ser necessário utilizar uma *gateway*[1] que permita às aplicações legadas aceder à base de dados do sistema alvo.

Utilizando o método de *Reverse Migration*, também conhecido por *Database Last*, as aplicações e interfaces legadas são migradas gradualmente para a nova plataforma, mantendo os dados legados no seu formato original. A migração dos dados é o último

[1] Uma *gateway* pode ser definida como um módulo de *Software* que opera entre componentes de *Software* para mediar a comunicação entre os mesmos

passo do processo de migração. Uma *gateway* inversa permite às novas aplicações aceder aos dados legados.

O método *Composite Database*, é um método híbrido que se baseia nas duas abordagens anteriores. As aplicações legadas são re-implementadas, de forma gradual, usando a tecnologia do novo sistema. O sistema legado e o novo sistema coexistem durante a migração. É de referir que, nesta abordagem, é necessário existir uma *gateway*, que permita ao sistema legado aceder aos dados do sistema alvo, e uma *gateway* inversa, que permita ao novo sistema aceder aos dados legados. Pode haver duplicação de dados entre as bases de dados legada e alvo. Para manter a integridade dos dados, é utilizado um *Coordenador* que intercepta todas as modificações efectuadas nos dados do sistema legado e do sistema alvo (e.g. remoção de um registo).

É importante notar que as metodologias que usam *gateways* herdam toda a complexidade de implementação destas, já que têm que estabelecer a comunicação entre sistemas que operam em ambientes heterogéneos.

1.3 O projecto *gestBarragens*

A segurança das barragens de betão é de grande importância, a fim de evitar anomalias ou mesmo eventuais catástrofes. Em Portugal, o controlo de segurança das barragens de betão está legislado pelo Regulamento de Segurança de Barragens (RSB) [RdSdB90] e resulta da interacção entre várias entidades que exercem diferentes papéis, conforme representado na Figura 1.2 [Gom99].

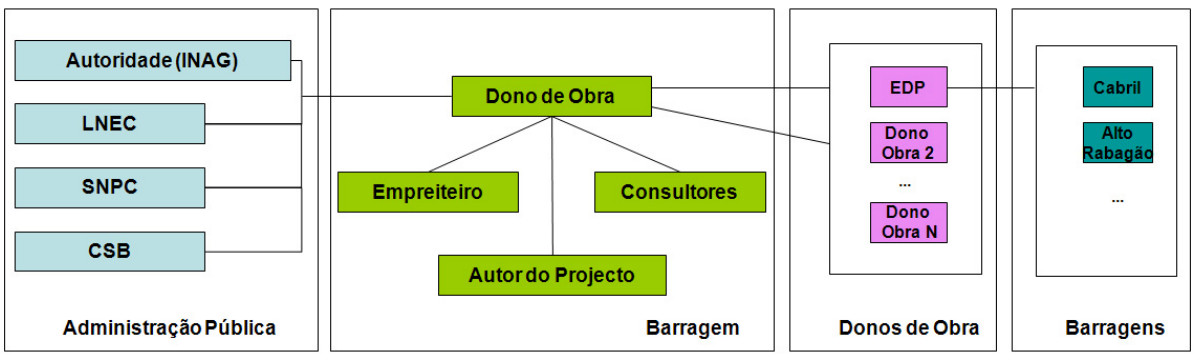

Figura 1.2: Entidades envolvidas no Controlo de Segurança de Barragens

Em Portugal, as entidades envolvidas no Controlo de Segurança das Barragens são: o Instituto da Água (INAG), como organismo da administração central com competência genérica ao nível do controlo de segurança das barragens; o LNEC; o Serviço Nacional de Protecção Civil (SNPC); a Comissão de Segurança de Barragens (CSB) e os vários Donos de Obra, que possuem uma ou mais barragens.

Existem várias barragens distribuídas pelo país, que podem ter diferentes donos de obra e diferentes entidades responsáveis pela observação e controlo da segurança. Por exemplo, na barragem de Castelo do Bode a EDP-Produção EM assume o papel de dono de obra e a EPAL tem responsabilidades relativamente às infraestruturas de captação de água para abastecimento.

O LNEC é responsável por centralizar e constituir um arquivo com a informação de controlo de segurança de todas as barragens em Portugal[2]. Para gerir e manter esta informação, desenvolveu-se, no LNEC, no final da década de 70, um sistema de informação denominado Sistema de Informação para Observação de Barragens de Betão (SIOBE) [HS93]. A linguagem de desenvolvimento utilizada foi o FORTRAN e a plataforma de funcionamento era o MS-DOS. Os dados eram armazenados em ficheiros binários e ASCII, com modelos de representação de baixo nível. A informação era introduzida no sistema de forma manual.

Com a necessidade de introdução de novos requisitos na gestão e exploração da informação, tornaram-se mais visíveis alguns problemas do SIOBE, que se devem, principalmente, à fraca estruturação no armazenamento dos dados e à tecnologia de desenvolvimento obsoleta. Por exemplo, a qualidade dos dados era reduzida devido à falta de restrições de integridade no arquivo de dados e à interface de registo de dados que não incluía validação adequada. Por outro lado, existiam várias dificuldades de manutenção do sistema.

Com vista à substituição do SIOBE, desenvolveu-se um novo sistema de informação, designado por *gestBarragens* [SGP02]. O novo sistema de informação mantém todas as

[2]RSB - Artigo 6º. Laboratório Nacional de Engenharia Civil. Sempre que no âmbito da alínea a) do n.º 2 do artigo 5.º lhe seja atribuída uma intervenção de carácter sistemático, compete ao LNEC: (...) d) Promover a constituição, na sua sede, de um arquivo informático dos dados dos sistemas de observação das barragens e explorá-lo de modo a manter um conhecimento actualizado do seu comportamento [RdSdB90]

funcionalidades do SIOBE e inclui novas funcionalidades como a criação automática de relatórios com informação na forma de gráficos e tabelas, ou a apresentação de gráficos nos modelos das barragens, através de um sistema de informação geográfico. A informação mantida pelo SIOBE teve que ser migrada para o novo sistema de informação, assim como informação que existia em ficheiros separados que não eram geridos pelo SIOBE (e.g. ficheiros Excel).

1.3.1 Sistema de Observação em Barragens

O Sistema de Observação de uma barragem é constituído pelo conjunto de instrumentos e dispositivos (por exemplo, fio de prumo, dreno, piezómetro) instalados na barragem. Estes instrumentos permitem determinar *grandezas*[3] (e.g., extensões, temperaturas, caudais) relativas às acções, propriedades dos materiais e respostas das estruturas da barragem às fundações [Por01]. Por exemplo, os instrumentos do tipo fio de prumo permitem determinar deslocamentos da estrutura da barragem no plano horizontal.

O Sistema de Observação instalado numa barragem deve permitir a detecção atempada de anomalias de comportamento, como por exemplo um deslocamento elevado da estrutura da barragem ou a abertura excessiva de uma determinada fissura. Esta detecção permite minimizar, ou mesmo prevenir, os efeitos nocivos das anomalias. A observação de barragens é de crucial importância para a sua segurança, podendo evitar a ocorrência de um eventual acidente ou, pelo menos, minimizar as suas consequências.

A observação das grandezas está associada a procedimentos específicos de medição, vulgarmente denominados por *observações*. A medição ou observação é o processo através do qual, empírica ou objectivamente, se atribuem valores numéricos a propriedades de objectos ou a acontecimentos reais. Na prática, o processo de medição em Engenharia de Barragens tem dois objectivos: observar o comportamento do sistema físico, neste caso da barragem, e controlar o comportamento desse sistema [Por01].

As observações em barragens podem ser feitas através de medição directa e de medição indirecta. Na medição directa, o valor da grandeza a medir é obtido direc-

[3]Grandeza é um atributo de um fenómeno, corpo ou substância susceptível de ser caracterizado qualitativamente e determinado quantitativamente [dQ96]

tamente a partir de um determinado instrumento. A medição directa não envolve a medição de outras grandezas funcionalmente relacionadas com a grandeza a medir, como acontece na medição indirecta.

A medição directa em barragens é realizada nas *Observações* efectuadas no Sistema de Observação. As *Observações Geodésicas* constituem o exemplo de medição indirecta em barragens.

1.3.2 Observações

A medição nas barragens portuguesas, é feita através da observação dos diferentes *instrumentos* instalados nas barragens. A Tabela A.1, no Apêndice A, contém uma listagem dos diferentes tipos de instrumentos que se encontram instalados nas barragens portuguesas (por exemplo, fios de prumo e drenos). Alguns tipos de instrumentos, como por exemplo os fios de prumo, encontram-se agrupados num *grupo* de instrumentos. Neste caso, cada instrumento pertence a um e um só grupo.

As medições dos instrumentos são efectuadas durante uma *campanha*. Uma campanha é um período de tempo durante o qual se efectuam medições, através de um conjunto de instrumentos, numa determinada barragem. Normalmente, uma campanha tem uma duração de dois a cinco dias, dependendo do número de instrumentos existentes na barragem e do número de instrumentos que são observados. Durante uma campanha, os observadores efectuam a medição de cada um dos instrumentos instalados na barragem. Para efectuar essa medição, é necessário recorrer a um equipamento de medição, vulgarmente chamado *instrumento de leitura*, que depende do tipo de instrumento a observar. Existem campanhas onde se observa apenas um subconjunto representativo dos instrumentos instalados na barragem. Estas campanhas designam-se por *campanhas expeditas*.

Uma *leitura* corresponde a um valor medido, por um determinado instrumento de leitura, num determinado instrumento instalado na barragem. A partir das leituras de cada instrumento, é possível determinar grandezas físicas, que se designam por *resultados*. O Apêndice B apresenta um exemplo no qual se descreve de forma detalhada a informação de leituras e resultados para o instrumento fio de prumo.

1.3.3 Observações Geodésicas

O objectivo das observações geodésicas consiste em determinar, ao longo do tempo, a posição de um conjunto discreto e representativo de pontos da estrutura das barragens. A posição dos diferentes pontos é determinada com base nos deslocamentos que cada ponto apresenta ao longo do tempo. Para proceder à observação geodésica em barragens, é necessário instalar *redes geodésicas* nas barragens. As redes geodésicas podem ser *altimétricas*, *planimétricas* ou *tridimensionais* e são compostas por um conjunto de pontos materializados na estrutura da barragem, que se designam por *vértices*, e por um conjunto de *ligações* que estabelecem a relação entre os vértices.

Os vértices podem desempenhar diferentes funções em cada rede geodésica. Por exemplo, nas redes planimétricas e tridimensionais, os vértices podem ser *pontos estação*, que são aqueles em que são estacionados instrumentos de medição; *pontos origem*, que definem a origem do referencial: e *pontos visados*, onde são estacionados, unicamente, alvos de pontaria óptica. Nas redes altimétricas os vértices podem exercer a função de *ponto atrás* e de *ponto à frente*.

As redes geodésicas são observadas, periodicamente, no âmbito de uma *campanha geodésica*. A observação de cada ligação numa campanha geodésica dá origem a uma leitura. É possível realizar quatro tipos de leituras diferentes: *(i) desníveis ortométricos*, diferenças de altitude entre dois pontos das redes altimétricas; *(ii) ângulos horizontais*, ângulos no plano horizontal entre dois pontos alvo das redes planimétricas ou tridimensionais; *(iii) distâncias* entre dois pontos nas redes planimétricas ou tridimensionais e *(iv) ângulos verticais*, ângulos realizados no plano vertical entre dois pontos alvo e com origem no *zénite*[4].

A Figura 1.3 apresenta um exemplo de uma rede planimétrica composta por ligações entre vários vértices (existe, por exemplo, uma ligação entre os vértices PFD1 e PFE1). Ilustra também a medição de um ângulo horizontal, no qual se coloca o instrumento de medição, neste caso, um teodolíto, no ponto estação (PFD1) e se determinam os ângulos para os vértices PE1, BC, CD, DE, EF, FG e PC que são os pontos visados. O observador determina os ângulos horizontais entre cada ponto visado e o ponto de

[4]Ponto em que a vertical de um lugar encontra a esfera celeste.

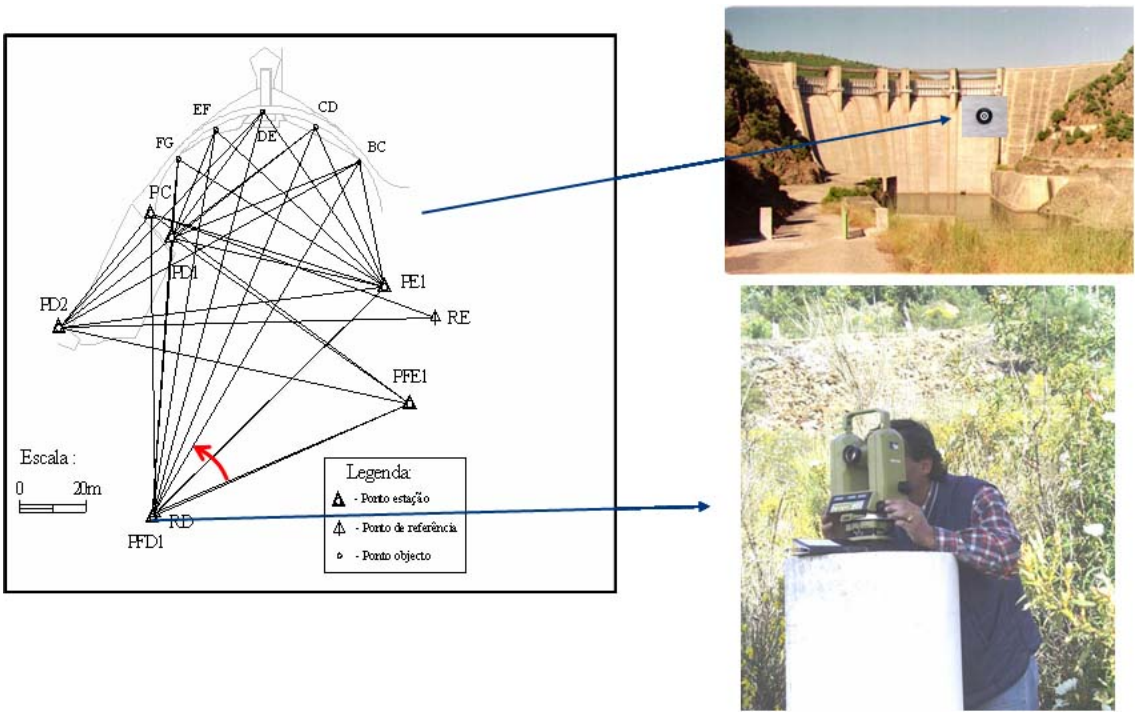

Figura 1.3: Medição de ângulos

referência (PFE1), de forma consecutiva.

A partir da observação das redes, isto é, dos valores dos ângulos, desníveis e distâncias determinados, é possível calcular os deslocamentos dos vértices. Nas redes altimétricas só é possível determinar deslocamentos na direcção z, que se designam por dz, nas redes planimétricas em x e y (dx e dy) e, nas redes tridimensionais em x, y e z (dx, dy e dz). Note-se que existem três programas, desenvolvidos em FORTRAN no LNEC, que efectuam estes cálculos, nomeadamente, 1DNETA para as redes altimétricas, 2DNETA para as redes planimétricas e 3DNETA para as redes tridimensionais.

Finalmente, a partir dos deslocamentos calculados, é possível determinar coordenadas (através de métodos de variação de coordenadas) que correspondem aos resultados das observações geodésicas. Estas coordenadas têm apenas uma direcção (x) para as redes altimétricas, duas direcções (x e y) para as redes planimétricas e três direcções (x, y e z) para as redes tridimensionais.

As leituras observadas no âmbito das campanhas geodésicas são registadas, manualmente, em cadernetas de campo em papel. Os instrumentos mais modernos registam

num ficheiro de *log* todas as leituras realizadas. Posteriormente, as leituras e os resultados calculados são registados em ficheiros Excel, que constituem o registo informático de toda a informação geodésica. Estes ficheiros contêm também a definição das redes geodésicas e das campanhas realizadas em cada barragem. Note-se que existe um ficheiro Excel para cada barragem, por exemplo, um ficheiro para a barragem do Alqueva e outro ficheiro para a barragem de Castelo do Bode.

1.3.4 Exemplo de migração do gestBarragens

Para controlar a segurança das barragens de betão existem vários tipos de instrumentos, tais como drenos e fios de prumo, que podem ser instalados em cada barragem. O sistema SIOBE permite armazenar e gerir informação de 21 tipos de instrumentos distintos.

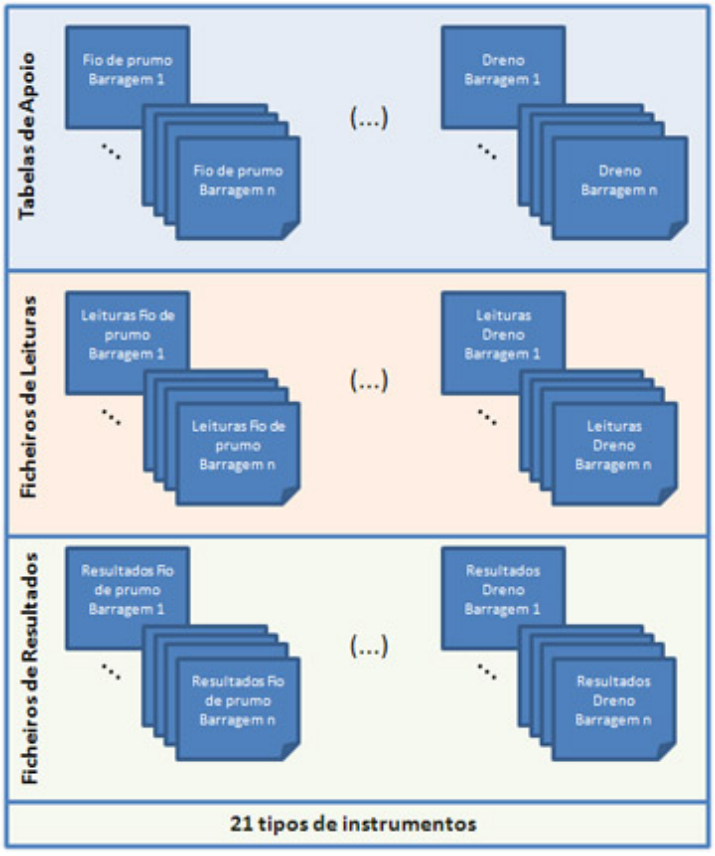

Figura 1.4: Arquivo de informação no sistema SIOBE

A Figura 1.4 apresenta o esquema de armazenamento da informação relativa aos diferentes tipos de instrumentos no sistema SIOBE. Existem três classes de ficheiros

distintas. A primeira classe de ficheiros, denominada *tabela de apoio*, contém a definição dos instrumentos e grupos de instrumentos (para tipos de instrumentos com grupo). A segunda classe de ficheiros, denominada *ficheiro de leituras*, armazena as leituras registadas em cada campanha. A terceira classe de ficheiros, denominada *ficheiro de resultados*, contém os resultados calculados em cada campanha.

Em cada classe de ficheiros existe um tipo de ficheiro distinto para cada tipo de instrumento. Por exemplo, nas tabelas de apoio existe um tipo de ficheiro para armazenar informação sobre fios de prumo e outro tipo de ficheiro para os drenos. Por outro lado, no SIOBE a informação está armazenada por barragem, pelo que cada ficheiro contém apenas informação da barragem a que corresponde. Assim, por exemplo, existe um ficheiro com a tabela de apoio dos fios de prumo da barragem de Castelo do Bode e outro ficheiro com a tabela de apoio dos fios de prumo da barragem do Alqueva. Note-se que ambos os ficheiros apresentam a mesma estrutura, já que contêm informação sobre o mesmo tipo de instrumento (fio de prumo).

Em seguida, apresenta-se um exemplo da migração de dados realizada no âmbito do projecto *gestBarragens*. O cenário apresentado corresponde à migração da informação armazenada no sistema SIOBE, sobre instrumentos do tipo fio de prumo de uma determinada barragem.

A Tabela 1.1 apresenta um excerto do ficheiro com a definição dos fios de prumo (tabela de apoio). Note-se que os fios de prumo estão organizados em grupos. Neste caso, cada fio de prumo (grupo de instrumentos) é composto por um conjunto de bases de coordinómetro (instrumento).

FPD1	J-K	11	1-1	879.25	0.000	6.27	11.16	8.28
FPD1	J-K	21-1	1	810.50	0.000	12.12	14.86	12.37
FPD1	J-K	31-1	1	831.10	0.000	9.84	12.79	10.95
FPD1	J-K	41-1	1	865.60	0.000	5.97	8.44	7.82
FPD1	J-K	51-1	1	791.00	0.000	2.20	0.59	0.38
FPD2	M-N	61-1	1	879.25	0.000	13.30	15.90	13.20
FPD2	M-N	71-1	1	810.50	0.000	8.20	11.05	8.71

Tabela 1.1: Exemplo de tabela de apoio para fios de prumo

Cada linha do ficheiro corresponde a um instrumento (base de coordinómetro) e

contém a sua identificação, localização e parâmetros usados na validação de leituras e no cálculo de resultados.

Por exemplo, a primeira linha do ficheiro identifica a base de coordinómetro colocada no fio de prumo "FPD1", localizada no bloco[5] "J-K", com código "1", em observação expedita (a base também é observada em campanhas expeditas), factor radial "1", factor tangencial "-1", cota da base "879.25", ângulo da base "0.0", leitura inicial da radial ao fio "6.27", valor máximo da leitura da radial ao fio "11.16" e valor mínimo da leitura da radial ao fio "8.28".

A Tabela 1.2 apresenta um excerto do ficheiro de leituras para os fios de prumo.

```
321   1 311966 310 879.0  1 8.56 9.68 3.98 7.05 0
321   1 311966 310 879.0  212.86 5.23 5.52 8.01 0
321   1 311966 310 879.0 13 9.23 8.95 4.26 7.23 0
321   1 311966 310 879.0  4 7.9210.06 4.06 7.35 0
321   1 311966 310 879.0  5 2.05 5.21 7.49 1.66 0
321   1 311966 310 879.0  614.81 8.19 4.53 6.68 0
321   1 311966 310 879.0  7 9.17 9.60 3.40 6.93 0
```

Tabela 1.2: Exemplo de ficheiro de leituras para fios de prumo

Cada linha do ficheiro corresponde a uma leitura realizada num instrumento, definido na tabela de apoio, numa determinada data (no SIOBE não existe o conceito de campanha). Note-se que cada leitura corresponde ao registo de vários valores, que dependem do tipo de instrumento observado. No caso das bases de coordinómetro, uma leitura corresponde ao registo de valores para a distância radial ao cone, tangencial ao cone, radial ao fio e tangencial ao fio. Existe uma ocorrência associada a cada leitura, que permite classificar o estado da leitura (e.g. leitura duvidosa, instrumento avariado, sem acesso).

Por exemplo, a primeira linha do ficheiro apresentado na Tabela 1.2 contém o código da barragem "321", a identificação do tipo de instrumento "1" que corresponde às bases de coordinómetro, o número de instrumentos por campanha "31", a data "10/03/1966", o nível da albufeira aquando da leitura "879.0", o código da base de coordinómetro a que se refere a leitura "1", a leitura da radial ao fio "8.56", a leitura da radial ao cone

[5]Bloco é um elemento estrutural que compõe a estrutura da barragem. Os blocos encontram-se ligados através de juntas.

"9.68", a leitura da tangencial ao fio "3.98", a leitura da tangencial ao cone "7.05" e o código da ocorrência "0" que significa que não houve qualquer ocorrência a registar.

A Tabela 1.3 apresenta um excerto do ficheiro de resultados para os fios de prumo.

19660310	20.799999	-1.699998
19660310	7.099996	-5.999970E-01
19660310	14.099990	-1.600001
19660310	19.700001	-1.800001
19660310	2.900000	-2.384186E-06
19660310	15.700000	-8.000040E-01
19660310	4.600003	-4.000139E-01
19660310	10.300000	-4.000092E-01

Tabela 1.3: Exemplo de ficheiro de resultados para fios de prumo

Cada linha do ficheiro corresponde a um resultado associado a um instrumento, numa determinada data, que é calculado a partir das leituras desse instrumento nessa data. Tal como acontece para as leituras, cada resultado pode corresponder a vários valores, que dependem do tipo de instrumento. No caso das bases de coordinómetro, um resultado é constituído pelo deslocamento radial e pelo deslocamento tangencial.

Cada resultado está associado a uma leitura e a um instrumento. No entanto, essa relação não se encontra explicitada no ficheiro de resultados. É necessário inferir a identificação do instrumento a partir da ordenação do ficheiro de resultados. Assim, o primeiro resultado corresponde ao primeiro instrumento da tabela de apoio, o segundo resultado corresponde ao segundo instrumento da tabela de apoio, e assim sucessivamente. Neste caso, a primeira linha, com deslocamento radial de "20.799999" e deslocamento tangencial "-1.699998" corresponde ao resultado calculado em 10/03/1966 (primeiro campo "19660310"), na base de coordinómetro de código "1" (primeira linha da tabela de apoio).

A partir destes ficheiros de entrada, devem ser criados os procedimentos necessários para realizar a sua migração para a base de dados do sistema *gestBarragens*. O desenho e implementação desta base de dados foi realizado numa fase anterior à criação dos processos de migração. Este desenho resultou do levantamento de todos os requisitos do projecto *gestBarragens* que foi realizado, em conjunto, pelas equipas do LNEC, da EDP-Produção EM e do INESC-ID Lisboa [SGB+04].

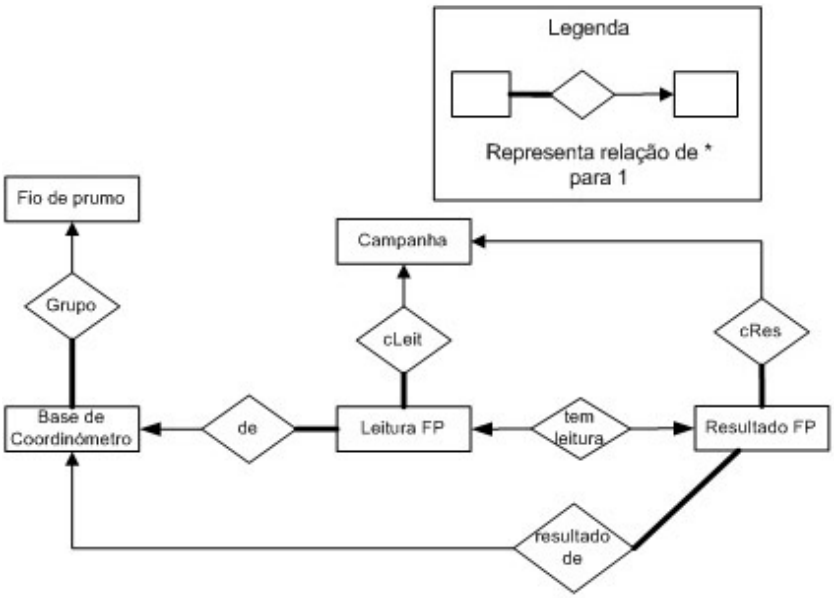

Figura 1.5: Modelo ER para o exemplo dos fios de prumo

A Figura 1.5 apresenta um excerto do modelo Entidade-Relação (ER), usando a nomenclatura definida em [SKS02], que representa algumas das entidades relacionadas com os fios de prumo. Este modelo ER é simplificado, na medida em que não apresenta os atributos das entidades, nem a listagem com as restrições de integridade que não podem ser expressas pelo modelo, pretendendo apenas ilustrar as relações entre as diferentes entidades. Na Tabela 1.4 apresenta-se o modelo relacional correspondente. Este modelo é também simplificado, uma vez que apresenta apenas um subconjunto dos atributos de cada relação. Segue a nomenclatura apresentada em [SKS02], onde os atributos que compõem as chaves primárias de cada relação estão sublinhados, os atributos que são chaves estrangeiras estão em itálico (e.g. o atributo *grupo* da relação *Base de Coordinómetro* referencia a relação *Fio de Prumo*) e os atributos não nulos estão a cheio (negrito).

A Figura 1.6 ilustra os mapeamentos entre os ficheiros do sistema SIOBE e as tabelas do modelo apresentado na Figura 1.5. Com a informação da tabela de apoio dos fios de prumo (exemplo na Tabela 1.1) é possível carregar as tabelas *Fio de prumo* e *Base de Coordinómetro*. Note-se que cada fio de prumo é composto por um conjunto de bases de coordinómetro.

Campanha (id, **datainicio**, **datafim**)
Fio de prumo (id, **identificacao**, **tipo**)
Base de coordinómetro (id, ***grupo***, **codigo**, RFInicial, RFMax, RFMin)
Leitura FP (id, ***campanha***, ***instrfixo***, data, RF)
Resultado FP (*instrfixo*, *campanha*, *leitura*, data, deslocRadial, deslocTangencial)

Tabela 1.4: Modelo relacional para o exemplo dos fios de prumo

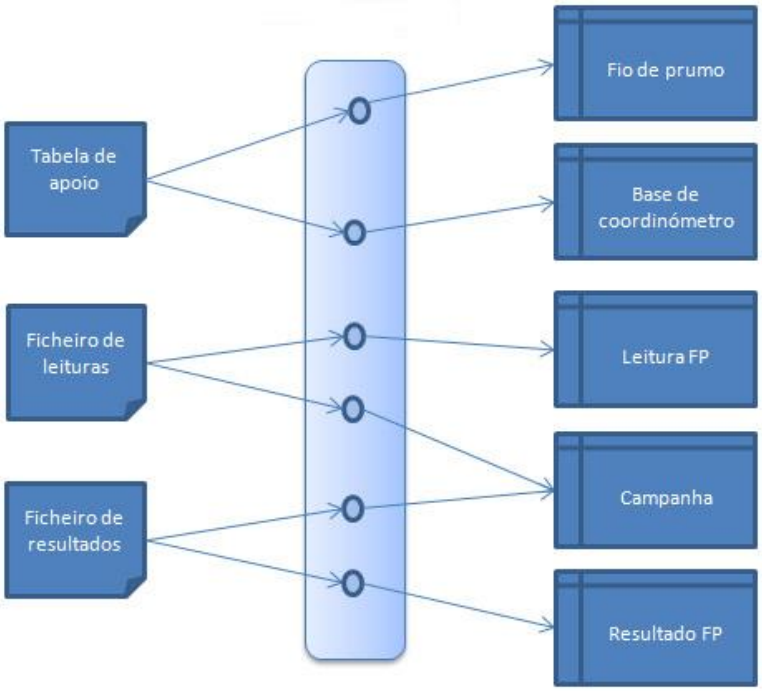

Figura 1.6: Mapeamentos para os dados do instrumento fio de prumo

A partir do ficheiro de leituras (exemplo da Tabela 1.2) é possível carregar as tabelas *Campanha*, com as datas incluídas nesse ficheiro e *Leitura FP*, que guarda as leituras realizadas nas bases de coordinómetro. Finalmente, a partir do ficheiro de resultados (exemplo da Tabela 1.3) é possível carregar as tabelas *Campanha*, com as datas dos resultados e *Resultado FP*, com os resultados calculados para cada base de coordinómetro.

A Figura 1.7 apresenta um exemplo da migração do excerto do ficheiro com a tabela de apoio dos fios de prumo, para as tabelas *Fio de Prumo* e *Base de coordinómetro*.

Em primeiro lugar, procede-se à criação dos fios de prumo. No exemplo apresentado existem dois fios de prumo, sendo o primeiro (FPD1) constituído pelas primeiras cinco

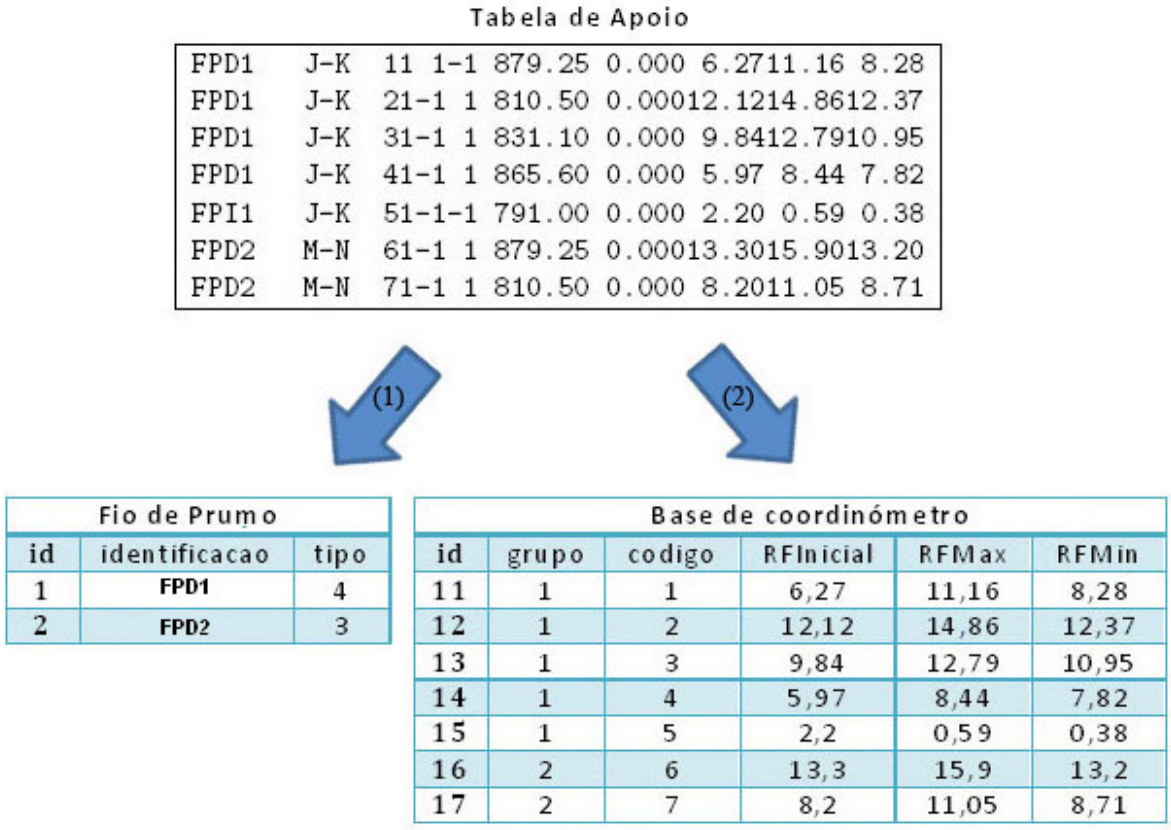

Figura 1.7: Exemplo da migração de fios de prumo e bases de coordinómetro

linhas do ficheiro (cinco bases de coordinómetro) e o segundo fio de prumo (FPD2) constituído pelas restantes linhas do ficheiro. Para identificar cada fio de prumo é necessário gerar o atributo numérico *id* que identifica univocamente cada fio de prumo.

Em segundo lugar, procede-se à inserção de registos na tabela *Base de coordinómetro*. Esta inserção tem que ser posterior à criação dos fios de prumo, já que é necessário estabelecer a relação entre cada base de coordinómetro e o respectivo fio de prumo, através da chave estrangeira *grupo* da tabela *Base de coordinómetro* que referencia a tabela *Fio de prumo*. Neste exemplo, não existe nenhum erro na definição dos instrumentos, pelo que todas as bases de coordinómetro existentes na tabela de apoio são migradas para a tabela *Base de coordinómetro*.

A migração das leituras dos fios de prumo só pode ser realizada depois de migrar todas as bases de coordinómetro, já que o atributo *instrfixo* da tabela *Leitura FP* referencia a tabela *Base de coordinómetro* (conforme o modelo relacional apresentado na

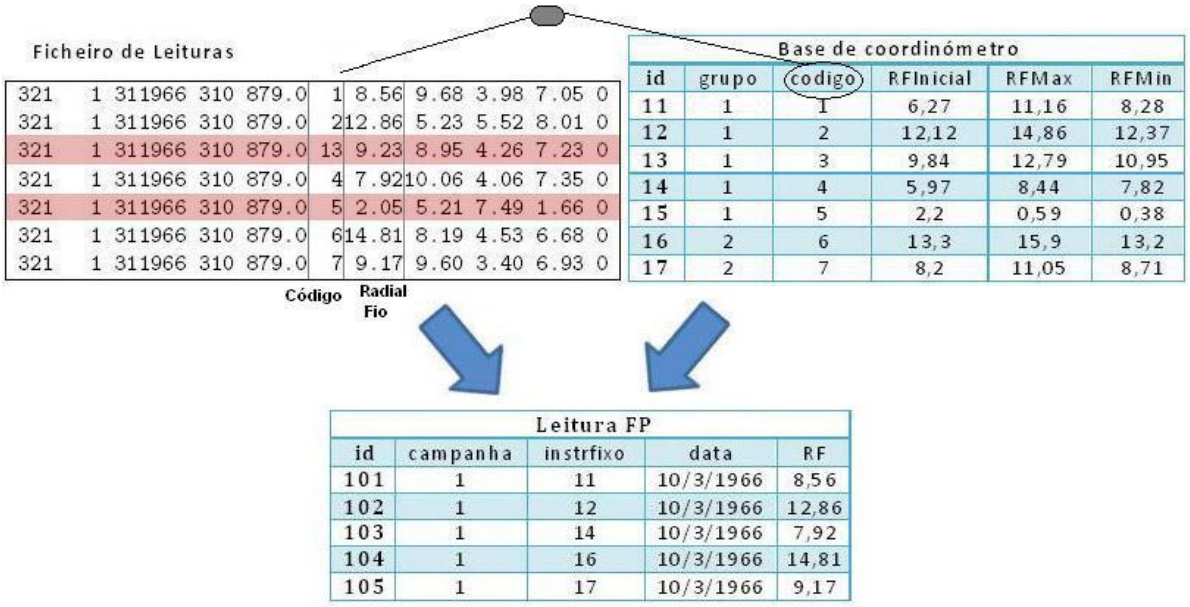

Figura 1.8: Exemplo da migração das leituras dos fios de prumos

Tabela 1.4). A Figura 1.8 ilustra a migração das leituras dos fios de prumo a partir do excerto do ficheiro de leituras apresentado, anteriormente, na Tabela 1.2. É necessário cruzar a informação proveniente do ficheiro de leituras com a informação da tabela *Base de coordinómetro*, a fim de estabelecer a relação entre cada leitura e a respectiva base de coordinómetro. Esta relação estava definida através do campo *código*, marcado no ficheiro de leituras da Figura 1.8, e do atributo *codigo* da tabela *Base de coordinómetro*. Por exemplo, a primeira leitura do ficheiro de leituras foi realizada na base de coordinómetro de código "1", que corresponde à primeira linha da tabela *Base de coordinómetro* ($id = 11$). Esta leitura dá origem à primeira linha da tabela *Leitura FP* ($id = 101$), que referencia a base de coordinómetro através da chave estrangeira *instrfixo*. Note-se que o valor da leitura radial ao fio (8.56) é validado pelos valores mínimo (8.28) e máximo (11.16) que estão definidos na tabela *Base de coordinómetro* pelos atributos *RFMin* e *RFMax*.

Existem alguns registos que não obedecem à lógica dos mapeamentos, pelo que não podem ser migrados para a base de dados final. Por exemplo, os dois registos marcados a vermelho no ficheiro de leituras não podem ser migrados. No primeiro registo, a leitura foi realizada numa base de coordinómetro com código "13". No entanto, não existe

nenhuma base de coordinómetro com código "13" na tabela *Base de coordinómetro*, pelo que não é possível determinar a relação entre esta leitura e a respectiva base. No segundo caso, o valor lido para a radial ao fio (2.05) na base de coordinómetro de código "5" não se encontra no intervalo de validação definido para a leitura da radial ao fio dessa base de coordinómetro ([0.35; 0.59]), o que viola uma restrição de integridade imposta pelo modelo da base de dados e, consequentemente, impede a migração da respectiva leitura.

1.4 O Problema a Resolver

Existem vários factores que aumentam a complexidade do processo de migração da informação de controlo de segurança de barragens de betão para o sistema *gestBarragens*. De facto, uma vez que a informação a migrar pertence a um domínio muito específico (barragens de betão), existem várias dificuldades na compreensão das fontes de dados, nomeadamente dos ficheiros ASCII mantidos pelo sistema SIOBE e dos ficheiros Excel com informação sobre as observações geodésicas. A falta de documentação actualizada também impede o fácil entendimento da estrutura dos dados fonte, que só é possível com a intervenção de um perito nos dados.

Após interpretar a estrutura dos dados fonte foi necessário desenhar e implementar o processo de migração. Esta tarefa colocou três tipos de problemas, que o trabalho desenvolvido nesta tese pretende resolver. Em primeiro lugar, os dados fonte podem apresentar problemas que impedem a sua migração. Por exemplo, a Figura 1.8 ilustra dois problemas durante a migração das leituras dos fios de prumo. Na primeira linha marcada a vermelho no ficheiro de leituras, a leitura foi realizada num instrumento que não existe, não sendo possível determinar a relação com a tabela do instrumento. Estes erros são bastante comuns, já que o armazenamento em ficheiros não garante a relação de *chave estrangeira* entre duas entidades. Na segunda linha marcada a vermelho, o valor registado não respeita os limites de validação estabelecidos para esse instrumento. Tendo em conta que durante o processo de migração não deve haver perda de informação, é importante que todos os registos errados sejam identificados e, eventualmente, corrigidos.

Em segundo lugar, existem vários ficheiros para cada tipo de fonte de dados, conforme ilustrado na Figura 1.4. Por exemplo, existe um ficheiro com a definição dos fios de prumo da barragem de Castelo do Bode e outro ficheiro com a definição dos fios de prumo da barragem do Alqueva. Quando é feita a migração dos fios de prumo da barragem de Castelo do Bode, todos os fios de prumo de outras barragens que tenham sido previamente migrados, devem ser mantidos na base de dados. Caso já existam fios de prumo da barragem de Castelo do Bode, então devem ser actualizados. O processo de migração deve ter em conta estes dois aspectos e não se limitar a adicionar os novos registos em cada migração. Pretende-se que o processo de migração de dados seja incremental, de modo a manter inalterados os dados já migrados e, se necessário, actualizar a informação existente.

Em terceiro lugar, pretende-se corrigir os problemas de dados identificados durante o processo de migração. Essa correcção só pode ser feita por peritos em Engenharia de Barragens que não têm, necessariamente, conhecimentos profundos em informática. Deste modo, é necessário criar uma aplicação autónoma que permita realizar a migração dos dados, sem ser necessária a intervenção e controlo constante de um programador. Esta aplicação deve dispor de mecanismos para identificar e corrigir os registos que não possam ser migrados. Para além disso, tendo em conta que o processo de migração da informação relativa ao controlo de segurança de barragens de betão envolve cerca de 1600 ficheiros de entrada, a aplicação deve disponibilizar uma interface simples para seleccionar as fontes de dados.

Tendo em conta os três requisitos identificados, a migração de dados para o sistema *gestBarragens* poderia ser realizada de várias formas. Uma primeira abordagem passaria pela criação de um programa de raíz, usando uma linguagem de programação como *C* ou *Java*. No entanto, a implementação do programa de migração de dados seria complexa e com elevado tempo de desenvolvimento, já que não existe nenhum mecanismo que permita apoiar a implementação do programa de migração. Por exemplo, o migração de dados incremental e a identificação de registos errados teriam que ser implementados de raíz. Para além disso, com esta solução não existe uma separação clara entre a lógica do programa de migração e a sua implementação, o que conduz a várias dificuldades na

manutenção do programa [Car07].

Uma segunda alternativa seria usar uma das ferramentas comerciais que apoiam o desenvolvimento de processos ETL, como o ETLQ [(SA], o Hummingbird ETL [tdl03b] ou o Informatica ETL [CFR03]. Estas ferramentas disponibilizam um conjunto de operadores que apoiam a criação de várias operações específicas (e.g. normalização de datas) e possuem mecanismos de geração de erros, que permitem identificar registos com problemas. No entanto, esta geração de erros é limitada, já que só ocorre num subconjunto de transformações. Com esta solução não seria possível, por exemplo, identificar os problemas na migração das leituras dos fios de prumo (por exemplo, falta de relação entre a leitura e o instrumento e valor fora dos limites estabelecidos para o instrumento). Para além disso, as ferramentas ETL não providenciam informação que auxilie a correcção de todos os tipos de erros ou inconsistências detectados. Por exemplo, as ferramentas comerciais não permitem determinar a causa/origem de todos os erros gerados. Finalmente, também não seria possível criar uma nova aplicação que executasse o programa de migração na ferramenta ETL e fornecesse uma nova interface para selecção dos ficheiros fonte e identificação/correcção de registos errados, já que estas ferramentas não disponibilizam uma API[6] documentada, que permita controlar a execução das transformações.

Finalmente, é importante referir que a adopção de uma ferramenta comercial no âmbito do projecto *gestBarragens*, implicaria elevados custos ao nível da aquisição da licença da ferramenta e do tempo de aprendizagem no seu funcionamento. Este factor contribuiu igualmente para que não fosse essa a solução tecnológica escolhida.

1.5 Contribuições

Esta tese descreve o processo de migração de dados, de natureza iterativa, que foi aplicado à informação legada de controlo de segurança de barragens de betão, no contexto do projecto *gestBarragens*. O processo de migração recorre à *framework* de limpeza e

[6]API (do inglês Application Programming Interface) é um conjunto de rotinas estabelecidas pelo *software* para utilização das suas funcionalidades noutras aplicações.

transformação de dados *Ajax* [GFS+01], que disponibiliza mecanismos e transformações adequados à migração de dados e, destaca-se por permitir detectar registos errados e determinar a sua proveniência.

O objectivo desta tese consiste na criação de um processo de migração iterativo para a informação de controlo de segurança de barragens de betão, no âmbito do projecto *gestBarragens*, usando a *framework* de limpeza e transformação de dados *Ajax*. Este processo inclui a migração dos dados armazenados no sistema de informação legado (SIOBE) e várias folhas Excel, para a base de dados do *gestBarragens*.

As principais contribuições desta tese são as seguintes:

- validação da *framework Ajax* num processo de migração de dados real. Da utilização do *Ajax* na migração do *gestBarragens*, destacam-se as seguintes extensões:
  - detecção de registos errados quando é violada uma restrição de integridade de uma tabela de saída (e.g. leituras de fios de prumo em bases de coordinómetro que não existem violam restrição de chave estrangeira).
  - mecanismos que controlam a migração de dados incremental, isto é, que permitem que a migração de dados actualize a informação já existente e mantenha os dados previamente migrados na base de dados.
- realização do estado de arte das ferramentas de qualidade de dados. Deste trabalho resultou a publicação de um artigo na revista internacional *Datenbank-Spektrum* [BG05]

No contexto do sistema *gestBarragens*, desenvolveu-se ainda uma aplicação *Java*, autónoma, designada *infoLegada2Gb*, que implementa o processo automático de migração de dados e disponibiliza uma interface gráfica para selecção dos ficheiros fonte. Esta aplicação foi utilizada exaustivamente pelos peritos do LNEC, permitindo migrar todos os dados legados. Desta utilização resultou a necessidade de melhorar a classificação dos erros gerados, com informação específica do domínio dos dados (e.g. fio de prumo de código "13" não está definido na tabela de apoio). Recorrendo às estruturas

internas da *framework Ajax* foi possível incluir nesta aplicação uma classificação detalhada dos erros. Note-se que esta classificação é específica para os dados relativos ao controlo de segurança de barragens de betão, pelo que não pode ser reutilizada noutro processo de migração.

Finalmente, no âmbito do projecto *gestBarragens* foram também publicados os artigos referenciados em [SGBP05],[PPS+05] e [SGB+06].

1.6 Organização

Esta tese encontra-se organizada em seis capítulos.

O Capítulo 2 apresenta o trabalho relacionado na área da limpeza e migração de dados. Descrevem-se as ferramentas comerciais e os trabalhos de investigação mais proeminentes.

No terceiro capítulo, descreve-se, em detalhe, o modelo conceptual usado (*framework Ajax*), sendo concedida maior relevância aos operadores e características mais significativas nos processos de migração de dados. Destacam-se as principais extensões efectuadas ao *Ajax* e alguns detalhes da sua implementação

No quarto capítulo, apresenta-se a implementação dos processos de migração dos dados legados no âmbito do projecto *gestBarragens*.

Finalmente, o quinto capítulo resume o trabalho desenvolvido nesta tese, sugerindo-se perspectivas de trabalho futuro.

Capítulo 2

Estado de Arte

A implementação de um processo de migração de dados é necessária quando determinada aplicação é descontinuada e os dados armazenados têm que ser convertidos para o esquema da nova aplicação. Nesse caso, os dados armazenados nos sistemas legados, que obedecem a um determinado esquema, devem ser convertidos para uma base de dados, que obedece a um esquema de dados distinto [CG04].

Normalmente, os sistemas legados apresentam anomalias nos dados. Aos dados que contêm anomalias dá-se, usualmente, o nome de *dados sujos*. Os dados sujos mais comuns em sistemas legados são: *(i)* falta de valores (no caso do sistema SIOBE, o código de instrumento pode não estar preenchido no ficheiro de leituras); *(ii)* dados não normalizados, como por exemplo a mesma pressão expressa em 1 atm e em 101325 Pa; *(iii)* registos duplicados, como por exemplo duas leituras iguais para o mesmo instrumento, na mesma data; *(iv)* inconsistências (e.g., duas leituras diferentes para o mesmo instrumento, na mesma data). O Apêndice C apresenta uma taxonomia de anomalias de dados que podem ocorrer nos dados armazenados pelos sistemas legados, ou em qualquer base de dados genérica [BG05].

A existência de problemas nos dados degrada significativamente a qualidade da informação que pode ser obtida pelas organizações. No caso particular do controlo de segurança de barragens podem, por exemplo, inviabilizar a detecção de anomalias graves no comportamento da estrutura da barragem.

A probabilidade de ocorrência de dados com problemas é maior quando a informação

legada se encontra armazenada em fontes de dados heterogéneas. Neste cenário, é comum existirem diferentes representações para a mesma informação, que podem dar origem a informação duplicada, como por exemplo dois registos com a definição do mesmo fio de prumo ou a informação inconsistente, isto é, informação que representa a mesma entidade mas que é contraditória (e.g., dois registos com a definição do mesmo fio de prumo, mas com cotas diferentes). Se o esquema de dados do sistema alvo definir entidades (e.g., definição de uma barragem) cuja informação provém de várias fontes de dados heterogéneas, torna-se necessário integrar tanto ao nível de esquema, já que as diferentes fontes de dados podem apresentar diferentes esquemas para representar a mesma entidade, como ao nível de instância, dado que as mesmas instâncias de dados podem ocorrer nas diferentes fontes de dados [CGL+98].

Normalmente, o esquema de dados do sistema alvo inclui várias restrições de integridade que não são verificadas no sistema legado. Assim, a existência de dados nas fontes de dados que violem uma dessas restrições de integridade pode impedir o carregamento dos dados para sistema alvo. Por exemplo, uma leitura num instrumento que não existe não pode ser migrada porque viola a restrição de chave estrangeira no sistema alvo. Deste modo, a qualidade dos dados não é apenas uma opção, mas sim um requisito impreterível num processo de migração de dados [VVS+00].

Nesta tese, define-se um processo de migração de dados como uma sequência de transformações, que devem ser aplicadas aos dados fonte, armazenados num determinado suporte de dados, como ficheiros ASCII ou bases de dados relacionais, de forma a produzirem dados correctos que obedeçam ao esquema do sistema alvo, que pode ser apoiado no mesmo ou num suporte de dados diferente do das fontes de dados. Normalmente, um processo de migração de dados pode ser modelado por um grafo de transformações de dados [GFS+01].

As ferramentas normalmente designadas por "ferramentas de migração de dados", como é o caso da *SwisSQL* [Swi] ou *SQLWays* [SQL], não permitem realizar um processo de migração de dados do modo que se define nesta tese. De facto, destinam-se, unicamente, a conversões entre SGBDs diferentes ou entre diferentes versões do mesmo SGBD. Estas ferramentas permitem, por exemplo, migrar os esquemas, procedimentos

e dados de uma instância de uma base de dados em *Oracle 8i* para uma instância em *Oracle 10g* ou para uma base de dados *SQL Server 2005*. Este tipo de migração não envolve transformações nas instâncias de dados nem nas relações entre os mesmos.

Existem duas soluções tecnológicas que permitem implementar processos de migração de dados, do modo que se define nesta tese:

- utilização de programas específicos desenvolvidos numa linguagem de programação como *C* ou *Java*;
- desenvolvimento de programas que utilizam a linguagem proprietária de um SGBD, como o *Oracle PL/SQL*, e primitivas em *SQL*;

Com a utilização de uma linguagem de programação genérica para desenvolver programas de migração específicos, não existe uma separação entre a lógica das transformações e a sua implementação. Deste modo, não é possível optimizar o programa de migração, já que o compilador de uma linguagem de programação genérica não detecta optimizações que dependem dos dados envolvidos [Car07]. Além disso, existem dificuldades na manutenção do programa de migração, uma vez que qualquer alteração na definição lógica de uma transformação implica reescrever o código que implementa o programa de migração.

O desenvolvimento de programas através da linguagem proprietária de um SGBD, como o *PL/SQL* de *Oracle*, utiliza um conjunto de procedimentos ou funções (e.g., funções para tratamento de *strings*), instruções condicionais e de controlo (e.g., *if*, *while*) e instruções *SQL*. Tal como acontece nas linguagens de programação genérica, também não existe uma separação entre a lógica das transformações e a sua implementação o que conduz a dificuldades na optimização e manutenção do programa de migração [CGLP06].

Em alternativa, pode recorrer-se a ferramentas de qualidade de dados para implementar processos de migração de dados. De facto, as ferramentas de qualidade de dados permitem extrair informação a partir de múltiplas fontes de dados, aplicar um conjunto de transformações e procedimentos para melhorar a qualidade dos dados (e.g., eliminação de registos duplicados) e, finalmente, inserir os dados limpos numa base de dados alvo (e.g., base de dados relacional ou *Data Warehouse*) [VVS+00].

Considera-se um processo de migração de dados como um caso particular de um processo genérico de qualidade de dados, no qual a estrutura de armazenamento e/ou o esquema dos dados fonte é diferente do esquema alvo.

Neste capítulo, analisa-se o conjunto de ferramentas de qualidade de dados (comerciais e de investigação) mais significativas, bem como os principais aspectos que as caracterizam. Na Secção 2.1, descreve-se um processo de qualidade de dados como uma sequência de etapas e apresenta-se uma classificação das ferramentas com base na etapa do processo que apoiam [BG05]. Na Secção 2.2, apresenta-se uma classificação das ferramentas de qualidade de dados, em função das funcionalidades que disponibilizam.

É importante salientar que a análise das ferramentas comerciais foi baseada nas páginas *web* e em artigos com informação reduzida, tipicamente *white papers* disponibilizados pelos respectivos fornecedores, não sendo possível obter informação detalhada sobre alguns aspectos. As ferramentas de investigação encontram-se bem documentadas pelos artigos científicos publicados.

2.1 Ferramentas de qualidade de dados

A metodologia genérica de um processo de qualidade de dados pode ser decomposta em cinco fases essenciais, conforme ilustra a Figura 2.1 [BG05]. Existem categorias de ferramentas de qualidade de dados que apoiam cada uma das cinco fases apresentadas. Saliente-se que existem várias ferramentas comerciais, como o *dfPower* [dDaSC], que abrangem as cinco categorias de ferramentas apresentadas. Note-se também que um processo de qualidade de dados é um processo de natureza iterativa, na medida em que, geralmente, não é possível atingir o nível de qualidade de dados pretendido numa única iteração. No caso particular da migração de dados, pretende-se iterar até realizar o carregamento completo dos dados no sistema alvo com a qualidade de dados pretendida.

As ferramentas de *data profiling* e de análise de dados disponibilizam mecanismos para detectar problemas nos dados (e.g. leitura de um instrumento fora do intervalo de validação) e fornecem conhecimento adicional sobre os dados, como a distribuição dos valores de um determinado atributo. Desta forma, estas ferramentas constituem os prin-

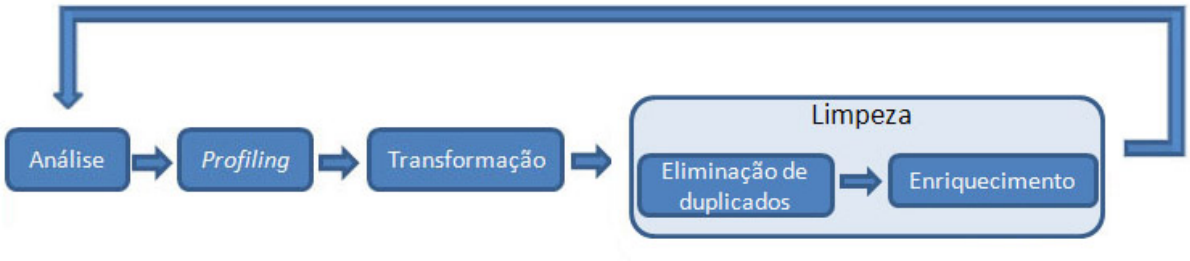

Figura 2.1: Categorias de ferramentas de qualidade de dados

cipais mecanismos de apoio na análise e interpretação dos dados fonte. Os restantes três grupos de ferramentas - de transformação, de eliminação de duplicados e de enriquecimento de dados - destinam-se à aplicação de transformações para satisfazer o esquema do sistema alvo e para corrigir os problemas de dados previamente identificados ou detectados nas fases de análise e *data profiling*. Note-se que as ferramentas de eliminação de duplicados e de enriquecimento de dados são casos particulares de ferramentas de limpeza de dados.

É importante salientar que as funcionalidades disponibilizadas por estes cinco grupos de ferramentas podem sobrepor-se. Por exemplo, não existe uma barreira completamente definida que separe os conceitos de transformação de dados e de limpeza de dados. De facto, uma simples normalização de uma data, como a transformação de "199821 2" para "21/02/1998", pode ser considerada uma transformação de dados, mas ao mesmo tempo é uma operação de limpeza de dados.

2.1.1 Análise de dados

As ferramentas de análise de dados permitem efectuar uma avaliação inicial aos dados através da aplicação de métodos de avaliação estatística (médias, desvios padrão, etc.), do estudo lógico dos valores dos dados (e.g. leitura da radial ao fio de um fio de prumo superior a cinco implica que a leitura da radial ao cone do fio de prumo é inferior a seis) e da aplicação de algoritmos de exploração de dados (*data mining*), com o objectivo de identificar regras e padrões de dados. A análise de dados deve ser acompanhada por um perito com conhecimentos específicos no domínio dos dados envolvidos, que deve

interpretar os resultados gerados. Desta forma, é possível garantir que os dados analisados respeitam determinadas restrições de integridade ou, caso contrário, identificar um conjunto de problemas nos dados fonte.

A fase de análise de dados permite determinar um conjunto de metadados que descreve as propriedades dos dados. Os metadados produzidos são muito importantes para apoiar a implementação das transformações de dados. Por exemplo, os problemas de dados que forem identificados durante a fase de análise de dados devem ser tratados/corrigidos nas fases seguintes do processo de qualidade de dados.

A ferramenta comercial *ETLQ* [Res04, (SA] é um exemplo interessante de uma ferramenta de análise de dados, já que a avaliação estatística dos dados é feita numa interface gráfica que permite seleccionar distribuições de valores entre colunas da mesma tabela (e.g. 10% dos fios de prumo localizados à cota 60 têm um valor para a radial do fio inicial inferior a 2) ou entre colunas de tabelas diferentes.

O trabalho de investigação desenvolvido em *Ken State University* [MM00] permite determinar o domínio, o intervalo de valores e a percentagem de valores nulos para cada atributo. Para além disso, inclui técnicas de aprendizagem não supervisionada com algoritmos de *clustering* e regras de associação.

As ferramentas comerciais *WizWhy* [(Wi], *Trillium* [Tri], *Migration Architect* [Sofc] e *dfPower* bem como o protótipo de investigação *Potter's Wheel* [RH01], constituem outros exemplos de ferramentas que disponibilizam mecanismos de análise de dados.

2.1.2 *Data profiling*

As ferramentas de *data profiling* permitem verificar se os dados pertencem a um determinado domínio de qualidade de dados[1] [QoT]. Determinar a qualidade, características e potenciais problemas dos dados na fase inicial de um processo de migração de dados, ou de um processo de qualidade de dados em geral, permite reduzir o tempo e recursos de implementação necessários, na medida em que se identifica um conjunto de "problemas"

[1]Domínio de qualidade de dados é definido como um conjunto de dados que respeita determinadas regras de qualidade de dados (especificação de um ou mais problemas que não devem existir no conjunto de dados).

que devem ser considerados/tratados nas fases de transformação e limpeza de dados. Caso os problemas de dados não sejam identificados na fase inicial, o processo de transformação e limpeza de dados pode falhar ou não atingir a qualidade de dados pretendida, o que implica um posterior refinamento e reprogramação das tarefas envolvidas.

Nas ferramentas de *data profiling* é possível especificar um conjunto de regras, mais ou menos complexas, que os dados devem respeitar, desde a simples verificação de uma chave estrangeira até à execução de um algoritmo de cálculo complexo. No caso dos dados relativos ao controlo de segurança de barragens de betão, os resultados são calculados a partir das leituras, através de um determinado algoritmo. Nas ferramentas de *data profiling* é possível especificar uma condição que verifica se os resultados correspondem à aplicação desse algoritmo às leituras que lhes deram origem.

Normalmente, as ferramentas de *data profiling* geram uma listagem de problemas organizados por importância, apresentam a sua distribuição em cada conjunto de dados e listam todos os valores em falta [Ols03].

A ferramenta comercial *dfPower* [dDaSC] possui um forte componente de *data profiling*. Em primeiro lugar, permite determinar um conjunto de metadados sobre os dados fonte (tipos de dados, número de registos, duplicados exactos, chaves estrangeiras, etc.). Em segundo lugar, determina padrões de representação para os valores de cada atributo (e.g. datas expressas em DD/MM/AAAA e AAAAMMDD), conforme ilustra a Tabela 2.1. Em terceiro lugar, possui uma funcionalidade denominada *Relationship Discovery*, que permite inferir chaves estrangeiras que não estavam definidas nas fontes de dados e verificar se existem "dados órfãos", isto é, registos que referenciam registos que não existem. Finalmente, permite especificar um conjunto de regras que devem ser verificadas pelos dados (*Business Rule Validation*). É possível definir regras básicas como regras de *look-up* ou fórmulas complexas que implementam um determinado algoritmo. O sistema disponibiliza uma interface gráfica para apresentação e exploração dos resultados determinados por cada técnica de *profiling*.

As ferramentas comerciais *WizWhy*, *Trillium*, *Migration Architect* e *ETLQ* bem como os trabalhos de investigação desenvolvidos em *Ken State University*, também disponibilizam mecanismos de *data profiling*.

Padrão	Contagem	Percentagem
DD/MM/AAAA	3166	98,29
DD-MM-AAAA	42	1,31
AAAA/MM/AA	12	0,37
AAAA-MM-AA	1	0,03

Tabela 2.1: Determinação de padrões para datas na ferramenta *dfPower*

2.1.3 Transformação de dados

A transformação de dados corresponde a um conjunto de operações, tais como integração e conversão de esquema, integração e conversão de dados, filtragem e agregação, que devem ser aplicadas sobre os dados, de forma a convertê-los para uma nova representação.

As transformações de dados requerem um conjunto de metadados, tais como a definição dos esquemas (atributos, tipos de dados, relações, etc.), as características dos dados (e.g. volume de dados) e os mapeamentos de cada transformação, com o objectivo de optimizar (e.g. particionamento de grandes volumes de dados) e validar (e.g. atributo do tipo *string* não pode ser somado com um atributo numérico) a execução de cada transformação.

A expressividade, isto é, a capacidade de implementar os vários tipos de transformações possíveis, é a característica mais relevante no apoio aos processos de migração e de qualidade de dados. As transformações devem ser genéricas e capazes de realizar grande parte dos processos de migração e de qualidade de dados, sem ser necessário recorrer à programação manual.

A Figura 2.2 identifica três classes de transformações [CW01], conforme o mapeamento entre os registos de entrada e os registos de saída, que representam: (*i*) mapeamentos de um-para-muitos; (*ii*) mapeamentos de muitos-para-um; (*iii*) mapeamentos de muitos-para-muitos.

Numa transformação de um-para-muitos, cada registo de entrada produz zero ou mais registos de saída, de forma independente. Considera-se uma transformação de um-para-um como um caso particular de uma transformação de um-para-muitos, na qual cada registo de entrada produz apenas um registo de saída. Uma operação de normalização corresponde a uma transformação de um-para-muitos ou ao caso particular

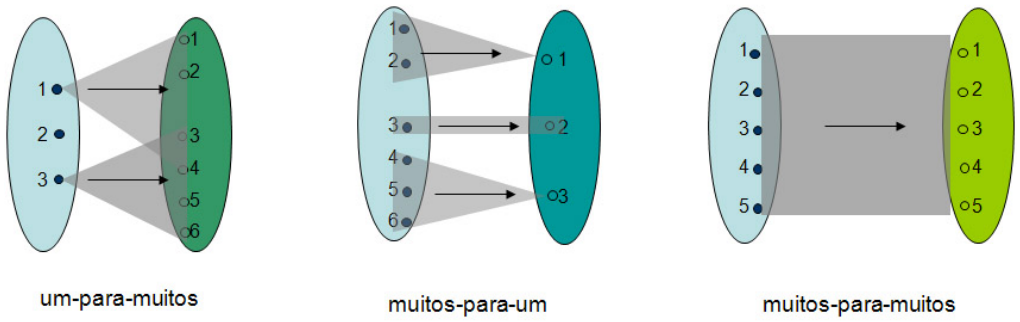

Figura 2.2: Classes de transformações

de uma transformação de um-para-um. Por exemplo, a normalização de uma data de "19980115" para "15/01/1998" corresponde ao caso particular de transformação de um-para-um. A normalização de uma morada portuguesa pode ser uma transformação de um-para-um ou de um-para-muitos, dependendo do esquema da base de dados alvo. Note-se que, uma transformação que a partir de um registo em que a morada está num único atributo, gera um registo em que a morada está na forma (rua, número, localidade) corresponde a uma transformação de um-para-um, uma vez que cada registo de entrada produz apenas um registo de saída. Caso o esquema da base de dados alvo normalize as moradas de alguma forma, separando, por exemplo, as localidades, um registo de entrada pode gerar mais do que um registo de saída (a morada e a localidade), o que corresponde a uma transformação de um-para-muitos.

Numa transformação de muitos-para-um, um conjunto de registos de entrada produz um único registo. Note-se que os conjuntos de registos de entrada têm que ser disjuntos, isto é, não pode existir nenhum registo que pertença a dois conjuntos e, consequentemente, que produza mais do que um registo. Uma transformação de muitos-para-um diz-se completa se todos os registos de entrada contribuírem para a produção de um registo. Uma operação de *Group-By* em *SQL* configura o exemplo de uma transformação de muitos-para-um completa.

Uma transformação de muitos-para-muitos corresponde a uma transformação atómica, implementada por um determinado algoritmo, que não é uma transformação de um-para-muitos nem uma transformação de muitos-para-um. Por exemplo, a ordenação dos registos de uma tabela é uma transformação de muitos-para-muitos, na qual todos os

registos da tabela de entrada foram usados para gerar todos os registos da tabela de saída.

Tipicamente, as ferramentas comerciais não implementam as transformações tendo em conta os mapeamentos que existem entre os registos de entrada e os registos produzidos. Com efeito, nas ferramentas comerciais definem-se vários tipos de operadores para cada domínio de transformações [Gal01]. Por exemplo, existe um operador específico para normalizar datas (e.g. "01011998" é normalizada para "01-01-1998"), outro operador para normalizar nomes, etc, não tendo em conta o tipo de mapeamento envolvido. Esta situação conduz à rápida proliferação de operadores, eventualmente redundantes entre si. De facto, com esta solução apenas se consegue lidar com problemas que tenham sido identificados previamente e os respectivos operadores tenham sido incluídos na ferramenta. Por outro lado, novos problemas que não sejam suportados pela ferramenta só podem ser resolvidos com recurso a programação manual, já que não existe nenhum operador que os suporte.

Na *framework* de transformação e limpeza de dados *Ajax* [Gal01] a abordagem é bastante diferente, uma vez que as transformações são baseadas nos mapeamentos entre os registos de entrada e os registos produzidos e não na lógica da transformação. Existem seis tipos de operadores que são baseados no tipo de mapeamento e não no tipo de transformação aplicada aos dados em si. Nomeadamente no caso da formatação de datas e de nomes, o operador utilizado pelo *Ajax* é o mesmo, diferindo apenas nas funções invocadas.

Uma abordagem bastante diferente é a seguida pelo protótipo *Potter's Wheel* que pretende ser completamente interactivo [RH01]. Em vez do utilizador definir a lógica da transformação, especifica os resultados que pretende obter a partir de um conjunto de valores de exemplo. Essa especificação é feita com recurso à utilização de uma interface gráfica tipo folha de cálculo, ilustrada na Figura 2.3. O sistema identifica, de forma automática, uma transformação que produz os resultados esperados. Em seguida, a transformação inferida é aplicada a outros registos, sendo o resultado imediatamente apresentado no ecrã, o que permite ao utilizador validar a execução da transformação inferida. Existe, deste modo, um processo iterativo e gradual na definição de cada

transformação.

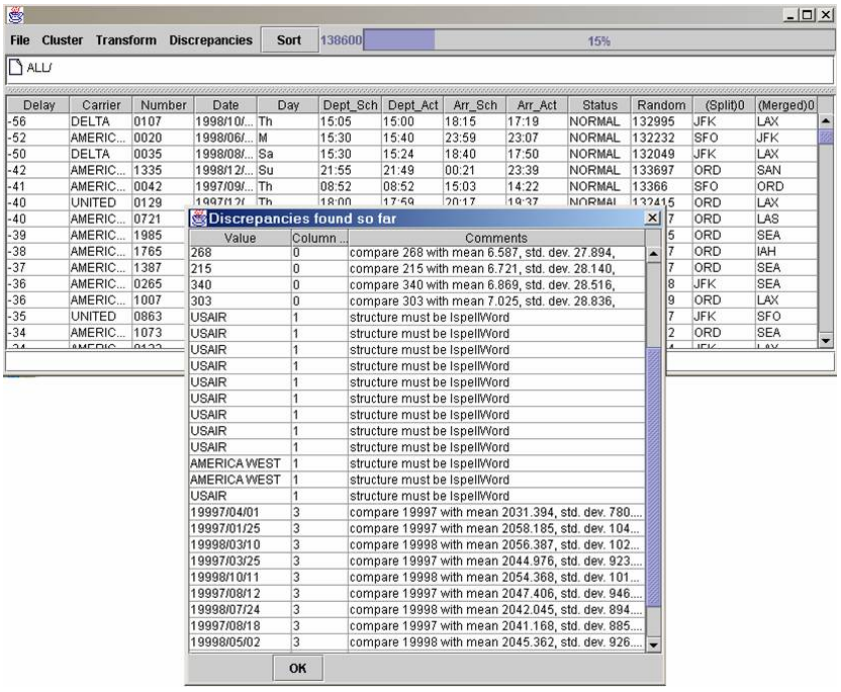

Figura 2.3: Interface do Potter's Wheel

As ferramentas comerciais *Data Integrator* [BLS03, Obj], *DataFusion* [(Ob], *Data Stage* [tdl03a, (As], *dfPower*, *ETLQ*, *Hummingbird ETL* [tdl03b, (Ge], *FirstLogic* [Fir05, Eva, Fir], *Informatica ETL* [CFR03, ETL], *Sagent* [TGV03, Sofd], *SQL Server DTS* [(Mia], *SQL Server 2005 Integration Services* [CKV+05, (Mib] e *Sunopsis* [MOME03, Sun], bem como os trabalhos de investigação desenvolvidos em *Ken State University* e os protótipos *Arktos* [VVS+00, Sim03], *Clio* [MHH+01, Mil98, MHH00], *FraQL* [SCS00, SS00] e *TransScm* [MZ98] constituem exemplos de ferramentas de transformação de dados.

2.1.4 Limpeza de dados

A tarefa de limpeza de dados consiste na detecção, remoção e/ou correcção de *dados sujos*, desde dados incorrectos, desactualizados, redundantes, inconsistentes, incompletos ou até mesmo mal formatados.

É importante corrigir os *dados sujos*, de forma a garantir a correcção e consistência da informação que representam, isto é, não só garantir a correcção dos dados de forma

isolada (um registo), como também garantir a correcção da informação como um todo (vários registos). As ferramentas mais sofisticadas conseguem efectuar a limpeza dos dados de forma automática, usando funções específicas, regras e tabelas de *look-up*.

Não existe uma barreira clara entre algumas tarefas de limpeza e de transformação de dados. Por exemplo, uma normalização é claramente uma transformação de dados, mas é ao mesmo tempo uma tarefa de limpeza de dados. Para além das tarefas tipicamente de transformação de dados, a limpeza de dados consiste na eliminação de duplicados e no enriquecimento de dados, que se descrevem na Secções 2.1.4.1 e 2.1.4.2, respectivamente.

2.1.4.1 Eliminação de duplicados

As ferramentas de eliminação de duplicados permitem identificar registos exacta ou aproximadamente duplicados isto é, registos que se referem à mesma entidade real e, posteriormente, consolidá-los num único registo. A eliminação de duplicados, se não for optimizada, é um processo que exige elevados recursos, pois implica a comparação de cada registo com todos os outros registos. Por exemplo, para determinar registos duplicados numa tabela com 1000 registos é necessário realizar $1000 * 1000$ comparações.

Normalmente, o processo de eliminação de duplicados integra os seguintes passos:

1. normalização dos formatos de dados, a fim de evitar discrepâncias. Por exemplo, normalizar todas as pressões para *Pascal*.

2. expansão de abreviaturas e códigos (e.g. expandir "V" para "Vértice").

3. determinação de registos duplicados (exactos e aproximados) através da execução de regras de junção de registos. Tipicamente, para determinar registos duplicados aproximados, determina-se a semelhança entre os valores de um subconjunto dos atributos de cada registo, que é calculada através de um determinado algoritmo (por exemplo, semelhanças entre cadeias de caracteres com algoritmos de distância de edição, entre datas pela diferença de dias). Calcula-se a semelhança entre dois registos, multiplicando o peso de cada atributo pela semelhança calculada entre os valores desse atributo nos dois registos, conforme ilustra a Figura 2.4. Consideram-se registos duplicados aproximados aqueles cuja semelhança total obedeça a um

critério previamente estabelecido, por exemplo a semelhança seja superior a um determinado valor.

4. consolidação dos registos duplicados num único registo.

5. actualização dos registos que referenciam os registos duplicados com a identificação do registo consolidado. Por exemplo, a tabela com as leituras dos fios de prumo referencia a tabela de bases de coordinómetro. Se a base de coordinómetro com identificação "10" for um duplicado da base de coordinómetro com identificação "12" e forem consolidadas para uma base de coordinómetro com identificação "15", então, todas as leituras que referenciavam as bases de coordinómetro com identificação "10" e "12" devem passar a referenciar a base de coordinómetro com identificação "15", sendo apagadas as bases "10" e "12".

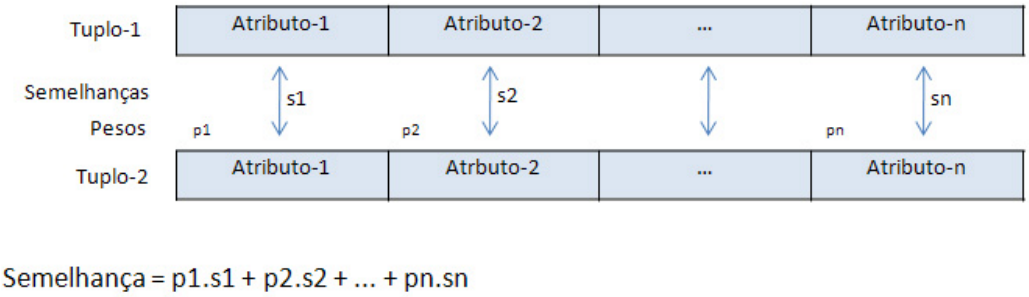

Figura 2.4: Semelhança entre dois registos

A ferramenta comercial *Informatica ETL* utiliza várias funções de semelhança em conjunto com *lógica difusa*[2]. Deste modo, a detecção de duplicados determina conjuntos de registos duplicados - *true* -, ou possivelmente duplicados - *maybe*, *sure*, *elbows* -. Posteriormente, para consolidar os registos duplicados num único registo, a ferramenta dispõe de uma interface gráfica que permite definir regras de consolidação (e.g. atributo textual mais longo, atributo numérico máximo) para cada registo, isoladamente, ou para cada conjunto de registos aproximadamente duplicados. Podem, por exemplo, definir-

[2]A lógica difusa (do inglês *fuzzy logic*) é uma generalização da lógica boleana (*true*, *false*) que admite valores lógicos entre a falsidade e a verdade, que representam uma determinada incerteza

se regras de consolidação para registos aproximadamente duplicados classificados como *maybe* e outras regras para duplicados *sure*.

A eliminação de duplicados na *famework* de limpeza e transformação de dados *Ajax* é realizada em três passos que são suportados pelos seus operadores. Em primeiro lugar, determinam-se os pares de registos cuja semelhança respeita um determinado critério, utilizando o operador *Match*. Em segundo lugar, cria-se um novo identificador para cada conjunto de duplicados, através do operador *Cluster*. Finalmente, a consolidação de cada *cluster* num único registo é feita com recurso ao operador *Merge* que inclui a invocação a funções agregadoras[3] que estejam definidas na biblioteca de funções do *Ajax*.

Existem várias ferramentas comerciais que permitem realizar eliminação de duplicados, como: *Centrus Merge/Purge* [Sofa], *ChoiceMaker* [Cho, BBWG03, BS04, Bor04], *DataBlade* [(IB], *DeDupe* [Sysa], *dfPower*, *DoubleTake* [(Pe], *ETLQ*, *ETI*DataCleanser* [(ET], *Firstlogic*, *Identity Search Server* [Sysb], *MatchIT* [Sysc], *Merge/Purge Plus* [Sofb], *NaDIS* [(MS], *QuickAddress Batch* [(QA], *Sagent*, *SQL Server 2005*, *Trillium* e *Wiz-Same* [(Wi]. Os protótipos de investigação *Flamingo Project* [JLM03, JKL04, JKLT05], *FraQL* e *IntelliClean* [LLL00] também permitem efectuar eliminação de duplicados.

2.1.4.2 Enriquecimento de dados

A tarefa de enriquecimento de dados consiste na utilização de informação adicional disponível na ferramenta ou obtida a partir de fontes de dados externas, com o objectivo de melhorar a qualidade dos dados que estão incompletos, indeterminados ou desactualizados. Este conjunto de informação adicional é específico para cada domínio de dados, como por exemplo informação sobre moradas portuguesas.

Os casos mais comuns de enriquecimento de dados estão relacionados com informação de moradas, de códigos geográficos e de dados demográficos. Por exemplo, na informação de moradas portuguesas é possível acrescentar informação sobre a localidade, ruas e números a partir de um campo com o código postal completo (e.g. o código postal 1000-055 corresponde aos números pares entre 188 e 202 da Avenida Almirante Reis em

[3]Função agregadora - função que aplicada a um conjunto de valores, retorna apenas um valor. Valor máximo, mínimo, médias, contadores, são exemplos de funções agregadoras.

Lisboa).

As ferramentas comerciais *Informatica ETL* [Inf06] e *QuickAddress Batch* permitem enriquecer dados de moradas de cerca de 200 países. No entanto, não dispõem de mecanismos para enriquecer dados de outros domínios. A ferramenta *dfPower* contém várias tabelas com informação que pode ser usada para enriquecimento de dados, desde dados demográficos a dados de criminosos e terroristas que podem ser usados, por exemplo, para comparar com uma lista de utilizadores.

Para realizar enriquecimento de dados na *framework Ajax* é necessário criar um conjunto de tabelas adicionais com a informação de detalhe, como por exemplo informação demográfica e de nomes portugueses. Posteriormente, recorrendo aos operadores do *Ajax* é possível especificar transformações de enriquecimento de dados.

As ferramentas comerciais *DataStage*, *ETLQ*, *Firstlogic*, *NaDIS*, *Sagent* e *Trillium* constituem exemplos de ferramentas de enriquecimento de dados.

2.2 Funcionalidades das Ferramentas de Qualidade de Dados

Nesta secção, apresentam-se as principais funcionalidades das ferramentas de qualidade de dados que foram analisadas. Estas funcionalidades podem agrupar-se em quatro categorias distintas: (*i*) manipulação de dados, que corresponde ao conjunto de funcionalidades que controlam a leitura e escrita de dados, isto é, extracção de dados, carregamento de dados e actualizações incrementais; (*ii*) especificação e execução do programa, isto é, o conjunto de funcionalidades que suportam a especificação e execução de um programa de qualidade de dados, ou seja, interface com o utilizador, repositório de metadados, técnicas de performance e controlo de versões; (*iii*) código externo, que inclui as funcionalidades de biblioteca de funções e suporte de linguagem de programação para o desenvolvimento de novas funções, que é necessário integrar nas ferramentas, uma vez que as primitivas predefinidas não são suficientes para resolver todos os problemas; (*iv*) análise e correcção, que apoiam o reinamento da lógica das transformações e a correcção

de anomalias nos dados fonte. A análise e correcção é composta pelas funcionalidades de detecção e tratamento de excepções, proveniência de dados e *debugging*.

Nas secções seguintes, analisa-se, em detalhe, cada uma funcionalidades e, em seguida, nas Tabelas 2.2 e 2.3, apresentam-se de forma resumida as funcionalidades observadas para cada uma das ferramentas comerciais e trabalhos de investigação.

2.2.1 Manipulação de dados

2.2.1.1 Extracção de dados

A extracção de dados dos sistemas fonte constitui o primeiro passo de qualquer processo de qualidade de dados. As ferramentas de qualidade de dados devem fornecer mecanismos que permitam extrair e reunir dados de múltiplas e heterogéneas fontes de dados, que podem variar desde bases de dados relacionais, ficheiros de texto, ficheiros *XML*, folhas de cálculo ou, até mesmo, pacotes aplicacionais, como o *SAP R/3*. Durante a fase de extracção, pode ser necessário efectuar filtragem de registos. Por outras palavras, deve ser possível definir várias regras para seleccionar apenas um subconjunto de dados a partir das fontes de dados.

Algumas ferramentas comerciais, como o *Sunopsis* [Sun], suportam apenas bases de dados relacionais. Outras ferramentas, como o *ETLQ* [Res04], suportam um enorme conjunto de tipos de fontes de dados, desde os conectores convencionais (e.g. *ODBC*, *JDBC*) até pacotes aplicacionais como o *SAP R/3*. Existem várias ferramentas comerciais, como o *Informatica ETL* [ETL] e o *Data Integrator* [Obj], que permitem extrair e reunir informação de múltiplas fontes de dados, bem como definir um conjunto de regras para filtrar os registos extraidos.

Na migração de sistemas de informação legados, como é o caso da migração efectuada no projecto *gestBarragens*, a extracção a partir de ficheiros *ASCII* assume maior importância, já que constitui o suporte de dados típico usado nas aplicações legadas.

2.2.1.2 Carregamento de dados

O processo de carregamento de dados para o sistema alvo constitui a fase final de um processo de qualidade de dados. Normalmente, o esquema alvo corresponde a uma base de dados relacional. As ferramentas de qualidade de dados devem disponibilizar, na fase de carregamento de dados, o seguinte conjunto de funcionalidades:

- carregamento de dados para diferentes tipos de sistemas alvo (e.g. ficheiro *ASCII* e base de dados relacional).
- possibilidade de acrescentar informação à que está contida no sistema alvo (*Append*).
- criação automática das estruturas de dados do sistema alvo, como, por exemplo, tabelas numa base de dados e respectivas restrições de integridade.

A ferramenta *Hummingbird ETL* [(Ge] optimiza o processo de carregamento para bases de dados relacionais armazenadas em *Oracle*, *Teradata*, *Microsoft SQL Server* ou *DB2*, recorrendo aos utilitários que os *SGBDs* disponibilizam para realizar carregamento em massa (e.g. *SQL*Loader* para *Oracle*).

As ferramentas comerciais *Sagent* [Sofd], *DataStage* [(As] e *dfPower* [dDaSC] consubstanciam exemplos de ferramentas que disponibilizam todas as funcionalidades enumeradas anteriormente.

2.2.1.3 Actualizações incrementais

As actualizações incrementais permitem modificar os dados do sistema alvo, de forma incremental, sem ser necessário actualizar todos os dados em cada carregamento. Por exemplo, se a mesma fonte de dados for usada em dois processos de qualidade de dados sucessivos, só os dados que foram actualizados, apagados ou inseridos na fonte de dados, após o primeiro carregamento, é que produzem alterações nos dados do sistema alvo.

O cenário ideal para as actualizações incrementais consiste em estabelecer uma política de actualização durante a fase de extracção de dados. Deste modo, apenas se faz a extracção dos registos novos e/ou actualizados, o que melhora significativamente a

eficiência e tempo de execução do processo de transformação. Uma estratégia alternativa consiste em estabelecer as actualizações incrementais apenas durante o processo de carregamento, o que conduz a um processo menos eficiente e de custo mais elevado.

A ferramenta *Hummingbird ETL* efectua actualizações incrementais, ao nível do processo de extracção de dados, fazendo uso do *Sybase Replication Server* para detectar modificações nas fontes de dados.

2.2.2 Especificação e execução do programa

2.2.2.1 Interface com o utilizador

Algumas ferramentas disponibilizam um ambiente de desenvolvimento gráfico integrado, o que as torna mais fáceis de utilizar. Estas ferramentas gráficas permitem ao utilizador definir processos de qualidade de dados através de grafos de transformações, usando uma interface *point-and-click* para definir as transformações. Naturalmente, existe informação específica, como a definição de uma fórmula matemática, que exige a introdução de dados de forma manual.

A maior parte das ferramentas comerciais, como *FirstLogic* [Fir, Fir05], *dfPower* [dDaSC, Cha05] e *Trillium* [Tri], providenciam uma interface gráfica para definição dos programas de qualidade de dados.

As ferramentas que não têm interface gráfica, como o *Ajax* [Gal01, GFS+01], disponibilizam, normalmente, uma linguagem de especificação para definir as transformações de dados. O protótipo *Arktos* [VVS+00, Sim03] disponibiliza duas linguagens de especificação. A linguagem *XADL* (*XML-based Activity Definition Language*) é uma linguagem baseada em *XML*, em que a estrutura dos ficheiros é validada por um DTD[4] bem definido. A linguagem *SADL* (*Simple Activity Definition Language*) é uma linguagem declarativa baseada na sintaxe do SQL. A escrita de um programa de transformação em XADL é mais complexa e exige que se escreva um número maior de palavras, mas é mais compreensível. No caso oposto, o uso de SADL é muito mais compacto, contudo é menos compreensível, adequando-se apenas a utilizadores treinados em programação.

[4]DTD (Document Type Definition): especifica o esquema e sintaxe de um ficheiro *XML* ou *SGML* [Def].

É importante salientar que o *Arktos* disponibiliza também uma interface gráfica de especificação das transformações, constituindo assim um exemplo interessante no qual se conjugam várias formas de especificação.

2.2.2.2 Repositório de metadados

O repositório de metadados consiste no armazenamento de informação sobre os dados e sobre a sequência de transformações de dados (grafo de transformações). Esta informação inclui os esquemas de dados, o volume de dados, ou as dependências entre as várias transformações. As estruturas de armazenamento do repositório de metadados podem ser estruturas internas da ferramenta, ficheiros controlados pela ferramenta ou uma base de dados relacional.

O repositório de metadados disponibiliza informação sobre os dados e transformações de dados ao motor de execução das transformações, de forma a validar (e.g. validação dos tipos de dados) ou a optimizar uma ou várias transformações em conjunto (e.g. duas transformações que não dependem uma da outra podem ser executadas em paralelo). A informação do repositório de metadados também pode ser fornecida a outras aplicações, por exemplo para produzir relatórios sobre os dados envolvidos em cada transformação.

Algumas ferramentas comerciais, como o *Informatica ETL* e o *Sagent* utilizam uma base de dados relacional para armazenar o repositório de metadados. O trabalho de investigação *Clio* também possui um repositório de metadados.

2.2.2.3 Técnicas de performance

As ferramentas de qualidade de dados disponibilizam um conjunto de funcionalidades que permitem acelerar o processo de qualidade de dados e garantir a sua escalabilidade. Algumas técnicas usadas para melhorar a performance são o particionamento de dados, o processamento paralelo de diferentes transformações com recurso a *threads* e o balanceamento de carga (e.g. de CPU, de armazenamento) em vários servidores (*cluster* de servidores). Algumas ferramentas disponibilizam técnicas de aumento de performance, como por exemplo o *DataStage* [(As], que efectua processamento paralelo através de particionamento e *pipelining*, e o *Hummingbird ETL*, que executa as transformações de

dados num ambiente multitarefa.

No protótipo de investigação *Ajax*, é possível optimizar o algoritmo que implementa cada transformação. Existe uma separação clara entre a definição lógica de cada transformação e a sua execução física, podendo a mesma transformação ser executada por diferentes algoritmos para diferentes conjuntos de dados [GFS+01].

2.2.2.4 Controlo de versões

O controlo de versões usado nas ferramentas de qualidade de dados é implementado através de um conjunto de operações básicas de controlo de versões (e.g. *check-in* e *check-out*), que permitem aos programadores manter diferentes versões dos programas de transformação de dados e dos dados envolvidos. Deste modo, para além de manter um histórico com as diferentes versões dos programas implementados, o controlo de versões facilita o trabalho colaborativo de vários programadores, que trabalham sempre sobre uma versão actualizada do programa criado.

A ferramenta comercial *ETLQ* disponibiliza mecanismos de *check-in* e *check-out* ao nível de um programa de transformações ou ao nível de um elemento do programa (e.g. transformação, tabela), sendo possível manter diferentes versões do código das transformações e dos dados. O sistema funciona com vários utilizadores, sendo necessário atribuir permissões para que possam alterar as versões de cada programa. Sempre que é feito um *check-in* o sistema regista num *log* as alterações efectuadas e o utilizador que as realizou [Com06].

Algumas ferramentas comerciais, como a *DataStage*, *Data Integrator*, e *Hummingbird ETL*, também suportam controlo de versões dos programas de transformação.

2.2.3 Funções externas

2.2.3.1 Biblioteca de funções

A biblioteca de funções engloba um conjunto de funções previamente definidas, como, por exemplo, funções de conversão de tipos de dados, que podem ser utilizadas na especificação das transformações. Assim, por exemplo, quando se pretende normalizar

datas, se existirem funções de normalização de datas na biblioteca de funções que se adaptem ao domínio em causa, não é necessário reprogramá-las.

Uma funcionalidade muito importante consiste na possibilidade de acrescentar novas funções à biblioteca de funções. Geralmente, as novas funções podem ser adicionadas através de uma linguagem de programação ou adicionando funções externas a partir de uma DLL (*Dynamic Link Library*).

Várias ferramentas comerciais, como o *Informatica ETL*, o *Data Integrator* e o *Hummingbird ETL* e trabalhos de investigação como o *Ajax*, o *TransScm* [MZ98] e o *Arktos* disponibilizam bibliotecas de funções com mecanismos adequados para adicionar novas funções definidas pelo utilizador.

2.2.3.2 Suporte de linguagem de programação

As ferramentas podem incluir mecanismos que permitam acrescentar novas definições de funções. Este suporte pode variar desde as linguagens de programação existentes, como *C* ou *Perl*, até uma linguagem de programação nativa. A ferramenta *DataStage* disponibiliza ambos os mecanismos, através da introdução de uma linguagem de programação nativa e suporte às linguagens genéricas *Perl* e *Basic*.

Nas ferramentas de investigação, como o *Ajax*, é possível acrescentar novas funções desenvolvidas em *Java*, definindo, na especificação do programa, o pacote de *Java* que contém essas funções, bem como a assinatura das mesmas, para que sejam validadas durante a compilação do programa.

2.2.4 Análise e correcção

2.2.4.1 Detecção e tratamento de excepções

Consideram-se excepções o conjunto de registos de entrada para os quais a execução de uma parte do processo de qualidade de dados falha. Um exemplo simples pode ser ilustrado no carregamento de uma tabela que contenha um atributo que é chave estrangeira para outra tabela. A geração de um registo para o qual o valor deste atributo não tem correspondência na tabela referenciada é considerado, necessariamente, uma

excepção, já que não pode ser inserido na tabela.

O tratamento de uma excepção, deve ser feito de forma manual ou automática. No tratamento de excepções de forma manual, o utilizador tem que decidir, excepção a excepção, qual a acção a tomar. Já na forma automática, não é necessária qualquer intervenção por parte do utilizador. O registo de excepção pode ser ignorado, apagado ou escrito num ficheiro ou tabela de excepções, dependendo da política implementada para o tratamento de excepções.

Algumas ferramentas comerciais (e.g. *Sunopsis* e *Informatica ETL*) e alguns trabalhos de investigação (e.g. *Ajax* e *Arktos*) disponibilizam mecanismos de detecção e tratamento de excepções. No *Ajax*, as excepções são geradas de forma automática, sendo colocadas numa tabela de excepções. Posteriormente, a correcção das excepções deve ser feita de forma manual.

O *Arktos* permite definir uma política de tratamento automático das excepções para cada uma das transformações. É possível adoptar uma das seguintes políticas: (*i*) *Ignore*, não há qualquer tratamento da excepção; (*ii*) *Delete*, o registo que deu origem à excepção (da fonte de dados da transformação) é apagado; (*iii*) *Contigency File*, o registo é copiado para um ficheiro de excepções; (*iv*) *Contigency Table*, o registo é copiado para uma tabela de excepções.

Na ferramenta *Sunopsis*, as excepções podem ser registadas numa única tabela (*Recycler*) ou numa tabela específica para as excepções de cada transformação. No primeiro caso, não é possível tratar as excepções geradas. Na ferramenta *Informática ETL*, as excepções são armazenadas em ficheiros de erros que também armazenam o código de execução da transformação. O objectivo desta solução é o de permitir re-executar a transformação apenas para o conjunto de registos de excepções, o que é realizado através de um componente denominado *Reject Loader*.

2.2.4.2 Proveniência de dados

As ferramentas que permitem determinar a proveniência dos dados permitem, igualmente, analisar os dados e a transformação que lhes deu origem (em particular os registos de excepções). A proveniência de dados pode ser determinada em qualquer passo

de execução do programa de qualidade de dados.

Existem ferramentas comerciais (e.g. *Data Integrator*) e de investigação (e.g. *Ajax*) que disponibilizam mecanismos avançados de análise da proveniência de dados.

Num caso genérico, uma transformação pode usar todos os registos de entrada para gerar cada um dos registos de saída da transformação. Contudo, na maior parte dos casos, cada registo produzido deriva de um subconjunto de registos de entrada e, muitas vezes, de apenas um registo de entrada.

Em [CW01], os autores apresentam uma definição formal para a proveniência de dados, em função dos tipos de transformação enunciados na Secção 2.1.3 (um-para-muitos, muito-para-um). É importante destacar que, para as transformações de muitos-para-muitos, não existe nenhum procedimento para determinar a proveniência dos dados. Uma vez que qualquer subconjunto dos registos de entrada pode ser usado na geração de um registo de saída através de uma transformação de muitos-para-muitos, apenas se pode afirmar que todo o conjunto de entrada deu origem a esse registo.

2.2.4.3 *Debugging*

As ferramentas de qualidade de dados devem providenciar mecanismos de análise dos programas de transformação de dados, com o objectivo de corrigir a lógica das transformações e os dados transformados.

As ferramentas comerciais disponibilizam mecanismos simples, como, por exemplo, um ficheiro de *log* detalhado (e.g. *DataStage* e *Data Integrator*). Mecanismos mais sofisticados podem ser encontrados nas ferramentas de investigação, como o *Ajax* e o *Arktos* [VVS+00, Sim03]. O *Ajax* disponibiliza uma interface gráfica que permite ao utilizador determinar a proveniência de cada registo gerado e corrigir, de forma interactiva, os dados errados ou refinar a lógica das transformações de dados especificadas. Dependendo das instruções fornecidas pelo utilizador, o sistema acede ou modifica os dados e invoca uma nova execução da transformação.

Na ferramenta de investigação *Potter's Wheel*, o utilizador constrói, de forma gradual, um programa de transformação de dados. As transformações são definidas de forma gráfica e a sua aplicação é visualizada imediatamente para todos os registos apresentados

no ecrã. Desta forma, o utilizador pode alterar a transformação rapidamente, caso os resultados produzidos não sejam os esperados.

2.2.5 Resumo da classificação

Com base nas características das ferramentas de qualidade de dados definidas anteriormente, apresentam-se, nesta secção, as Tabelas 2.2 e 2.3, que resumem as funcionalidades das ferramentas de investigação e comerciais, respectivamente.

	Manipulação de dados			Especificação e execução				Funções externas		Análise e correcção		
Ferramenta	Extracão de Dados	Carregamento de Dados	Actualizações Incrementais	Interface com o utilizador	Repositório de Metadados	Técnicas de Performance	Controlo de Versões	Biblioteca de Funções	Suporte de Linguagem	Excepções	Proveniência	Debugging
Ajax	DB,FF	DB	X	NG	X	Y	X	Y	JAVA	Y	Y	Y
Arktos	JDBC	JDBC	-	G	-	-	X	Y	JAVA	Y	-	Y
Clio	DB,XML	-	X	G	Y	Y	X	-	-	-	-	Y
Flamingo Project	DB	DB	-	NG	-	-	-	-	-	-	-	-
FraQL	-	-	-	NG	-	-	-	Y	N	-	-	-
IntelliClean	DB	DB	-	NG	-	-	-	-	-	X	-	Y
Ken State University	-	X	-	NG	-	-	-	-	-	Y	-	-
Potter's Wheel	Y	ODBC	-	G	Y	-	-	-	-	-	-	Y
Transscm [MZ98]	Y	-	-	G	-	-	-	Y	N	-	-	-

Tabela 2.2: Funcionalidades das ferramentas de investigação. Y: suportada; X: não suportda; -: desconhecido; DB: bases de dados relacionais; FF: ficheiros de texto; G: gráfica; NG: não gráfica

	Manipulação de dados			Especificação e execução				Funções externas		Análise e correcção		
Ferramenta	Extracção de Dados	Carregamento de Dados	Actualizações Incrementais	Interface com o utilizador	Repositório de Metadados	Técnicas de Performance	Controlo de Versões	Biblioteca de Funções	Suporte de Linguagem	Excepções	Proveniência	Debugging
Centrus Merge/Purge	DB	-	-	G	-	-	-	-	-	-	-	-
ChoiceMaker	DB, FF	-	-	G	Y	Y	-	Y	N	Y	Y	Y
Data Integrator	Vários	Y	Y	G	Y	Y	Y	Y	-	Y	Y	Y
DataBlade	Informix	Informix	-	G	-	-	-	-	-	X	X	Y
DataFusion	DB	DB	Y	G	-	Y	Y	Y	N	X	-	Y
DataStage	Vários	Y	-	G	Y	Y	Y	Y	Y	Y	Y	Y
DeDupe	DB	-	-	G	-	-	-	-	-	-	-	-
dfPower	Vários	Y	-	G	Y	Y	-	-	-	-	-	-
DoubleTake	ODBC	-	-	G	-	-	-	-	Y	-	-	-
ETI*Data Cleanser	Vários	-	-	G	Y	Y	-	Y	Y	Y	-	-
ETLQ	Vários	Y	-	G	Y	Y	Y	Y	N	-	-	-
Firstlogic	DB, FF	Y	-	G	Y	Y	-	Y	Y	-	-	-
Hummingbird ETL	Vários	Y	Y	G	Y	Y	Y	Y	N	M	Y	Y
Identity Search Server	DB	-	Y	G	Y	-	-	-	-	-	-	-
Informatica ETL	Vários	Y	Y	G	Y	Y	Y	Y	Y	Y	Y	Y
MatchIT	DB	-	-	G	-	-	-	-	-	-	-	Y
Merge/Purge Plus	-	-	-	-	-	-	-	-	-	-	-	-
Migration Architect	Vários	-	-	G	Y	-	-	-	-	-	-	-
NaDIS	-	-	X	G	X	-	-	X	X	X	X	-
QuickAddress Batch	ODBC	-	X	G	X	-	-	X	X	X	X	-
Sagent	Vários	Y	X	G	Y	Y	X	Y	N,SQL	-	-	-
SQL Server 2000 DTS	Vários	Y	X	G	X	-	-	Y	N	X	X	X
SQL Server 2005	Vários	Y	-	G	-	-	-	Y	N	-	-	-
Sunopsis	DB,FF	Y	Y	G	Y	Y	Y	X	SQL	Y	X	Y
Trillium	Vários	Y	-	G	Y	Y	Y	Y	N	-	Y	Y
WizRule	DB, FF	-	-	G	-	Y	-	-	-	-	-	-
WizSame	DB, FF	-	-	G	-	Y	-	-	-	-	-	-
WizWhy	DB, FF	-	-	G	-	Y	-	-	-	-	-	-

Tabela 2.3: Funcionalidades das ferramentas comerciais. Y: suportada; X: não suportada; -: desconhecido; N: nativa; DB: bases de dados relacionais; FF: ficheiros de texto; G: gráfica; M: manual

Capítulo 3

Aplicação e extensão da *framework Ajax* no âmbito do projecto *gestBarragens*

O principal objectivo das ferramentas de migração de dados consiste em suportar o desenho e execução de uma sequência de transformações de dados e de esquema, que permita gerar dados limpos e consistentes, a partir de um ou mais conjuntos de dados de entrada. Este fluxo de transformações de dados pode ser representado sob a forma de grafo [Gal01], como ilustrado na Figura 3.1. Cada *transformação* de dados, representada por Ti,j, no grafo tem como entrada um ou mais fluxos de dados (por exemplo, tabelas relacionais) e gera, como saída, um ou mais fluxos de dados. Os fluxos de saída de cada transformação podem ser usados como entrada de outras transformações.

As transformações representadas no grafo podem manipular instâncias de dados, denominando-se *transformações de dados*, bem como modificar a estrutura dos dados, sendo nesse caso denominadas de *transformações de esquema*. É importante destacar que, no caso específico da migração de dados, o esquema alvo já se encontra predefinido. Assim, o grafo que implementa um processo de migração de dados corresponde a um conjunto de transformações (de dados e/ou esquema) necessárias para converter os dados fonte para o esquema final. Embora seja um abuso de linguagem, no contexto desta tese designam-se, genericamente, as transformações envolvidas num processo de migração de

dados por "transformações de dados".

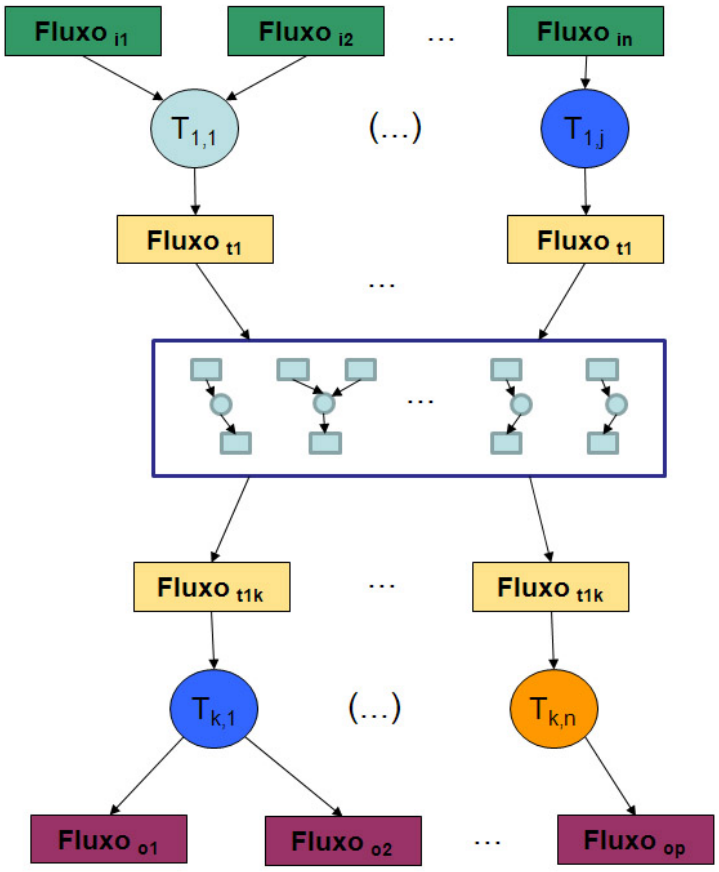

Figura 3.1: Grafo de Transformações

Existem várias ferramentas comerciais e trabalhos de investigação que permitem modelar um processo de migração de dados através de um grafo de transformações. O trabalho de investigação Arktos [VVS+00] e a ferramenta comercial Sunopsis [MOME03] constituem exemplos deste tipo de ferramentas. No entanto, existem duas dificuldades que não são facilmente suportadas nas ferramentas actualmente disponíveis no mercado. Em primeiro lugar, não existe uma separação entre a definição lógica de um programa de migração de dados e a sua implementação [GFSS00]. Deste modo, o algoritmo que executa cada transformação é sempre o mesmo, não tendo em conta o tipo e volume de dados envolvidos. Em segundo lugar, não existem mecanismos sofisticados que permitam efectuar o *debugging* de um programa de migração de dados [Gal01], o que dificulta o processo de refinamento e reformulação do mesmo.

O *Ajax* é uma *framework* de transformação e limpeza de dados [Gal01], que pretende suprir os problemas apresentados anteriormente. Existe uma separação bem definida entre o nível lógico e o nível físico. Ao nível lógico, um programa de transformação de dados é especificado de forma declarativa por uma extensão da linguagem *SQL*. A especificação traduz-se num grafo de transformações de dados, no qual cada nó corresponde a um operador de transformação de dados que aceita uma ou mais relações como entrada e produz uma ou mais relações como saída. No *Ajax* existem seis tipos diferentes de operadores, nomeadamente, *Map*, *View*, *Match*, *Cluster Merge* e *Apply*, que se encontram detalhados na Secção 3.2. Ao nível físico, existem alternativas de execução para cada operação lógica, que podem ser escolhidas de forma a optimizar o tempo de execução do programa. As optimizações de nível físico são feitas ao nível do algoritmo que implementa o operador, tornando a sua aplicação mais eficiente para um caso específico de dados.

Um exemplo de optimização da implementação física de um operador é a optimização da execução do operador *Match*. Este operador efectua uma junção aproximada entre duas relações. Para isso, calcula a semelhança entre quaisquer dois tuplos das relações de entrada, aplicando uma função de semelhança arbitrária [Gal01]. Todos os pares de tuplos que tenham uma semelhança superior a um valor previamente definido, são considerados duplicados aproximados. O algoritmo físico mais simples que corresponde à semântica do operador *Match* obtém-se, em primeiro lugar, executando o produto cartesiano entre as duas relações de entrada, em segundo lugar, aplicando a função de semelhança a todos os pares produzidos pelo produto cartesiano e, finalmente, seleccionando apenas os pares de tuplos cuja semelhança calculada seja superior a um determinado valor. Para relações com muitos tuplos, a aplicação deste algoritmo físico é incomportável. Para optimizar este operador, podem definir-se várias restrições. Por exemplo, num cenário em que os tuplos a comparar são compostos por um único atributo que corresponde a uma cadeia de caracteres, a implementação física pode definir que apenas se calcula a semelhança para os tuplos cuja diferença no tamanho das cadeias de caracteres seja inferior a um determinado valor, sendo descartados todos os outros pares de tuplos. Note-se que, para comparar cadeias de caracteres se usa, frequentemente,

uma função de distância de edição cujo valor é inversamente proporcional à semelhança, isto é, quanto menor for a distância de edição maior é a semelhança e vice-versa.

Normalmente, o volume dos dados a migrar para o sistema alvo é bastante elevado. Por outro lado, os dados apresentam, normalmente, vários casos especiais que não respeitam a lógica das transformações (e.g., registos errados, mal formatados), que podem impedir a correcta execução do programa de migração de dados. Assim, não se conseguem transformar todos os registos de entrada para o esquema da base de dados alvo, numa única iteração, sendo necessário que a ferramenta disponibilize mecanismos de *debugging* que permitam refinar o programa de transformação ou identificar e corrigir os registos que não podem ser transformados. No *Ajax*, designam-se de excepções todos os registos de entrada para os quais a lógica das transformações não se aplica. Nestes casos, o tuplo que origina a excepção é adicionado a uma relação de saída de excepções, que contém um campo descritivo da excepção gerada. De forma a suportar o *debugging* do programa de transformação de dados, o *Ajax* disponibiliza um mecanismo de explicação, que permite analisar os registos produzidos e a sua proveniência no grafo de transformações. Tendo em conta que também é possível analisar os registos de excepções, os mecanismos de *debugging* do *Ajax* permitem identificar facilmente a causa das excepções geradas, com o objectivo de refinar a lógica das transformações, ou corrigir erros nos registos de entrada.

No âmbito do projecto *gestBarragens* torna-se necessário migrar um conjunto de dados relativos ao controlo de segurança de barragens de betão. Pretende-se utilizar os mecanismos de geração de excepções e de *debugging* do *Ajax* para lidar com o volume dos dados envolvidos e os casos particulares que os dados possam apresentar. Para suportar o processo de migração do *gestBarragens* foi necessário dotar o *Ajax* de funcionalidades que ainda não estavam presentes. Destacam-se três extensões: (*i*) geração de excepções no operador *View* do *Ajax*, que corresponde a uma instrução *SQL*, a fim de identificar registos errados que violam restrições de integridade da base de dados alvo; (*ii*) suporte a carregamento incremental de dados; (*iii*) aumento da expressividade das transformações, através da extensão do *SQL* suportado, que é necessário para implementar as transformações de dados que permitem carregar a base de dados do *gestBarragens*.

Este capítulo está organizado do seguinte modo. Na Secção 3.1 apresenta-se a arquitectura da *framework Ajax*. Na Secção 3.2 descrevem-se os operadores do *Ajax*. Na Secção 3.3 descreve-se a utilização dos operadores do *Ajax* na migração dos dados relativos ao controlo de segurança de barragens de betão, no âmbito do projecto *gestBarragens*. Finalmente, na Secção 3.4 apresentam-se as principais extensões realizadas na *framework Ajax* de forma a suportar o processo de migração de dados do projecto *gestBarragens*.

3.1 Arquitectura

A Figura 3.2 apresenta, de forma simples, os principais componentes da *framework Ajax*. Nesta arquitectura destacam-se dois tipos de componentes: os repositórios e os componentes operacionais que constituem a base de execução do sistema.

Existem dois repositórios na arquitectura do sistema:

- Repositório de Dados: estrutura de armazenamento dos dados de entrada do sistema, dos dados intermédios produzidos pelas transformações e dos dados finais. É importante destacar que o sistema suporta repositórios de dados sob a forma de ficheiros ou de uma base de dados relacional acedida através de *Java Database Connectivity* (JDBC).

- Repositório de Funções e Algoritmos: estrutura que armazena um conjunto de funções e algoritmos programados utilizando a linguagem *Java*, que podem ser invocados pelos operadores de transformação. Normalmente, o domínio de cada problema de qualidade de dados exige a aplicação de diferentes lógicas de transformação e tipos de funções. Por exemplo, a normalização de moradas portuguesas é diferente da normalização de moradas americanas, o que implica a utilização de lógicas de transformação diferentes, apesar de se tratar de normalização de moradas em ambos os casos. Este repositório pode ser aumentado com a introdução de novas funções e algoritmos definidos pelo utilizador.

Os componentes operacionais que implementam os programas de transformação de

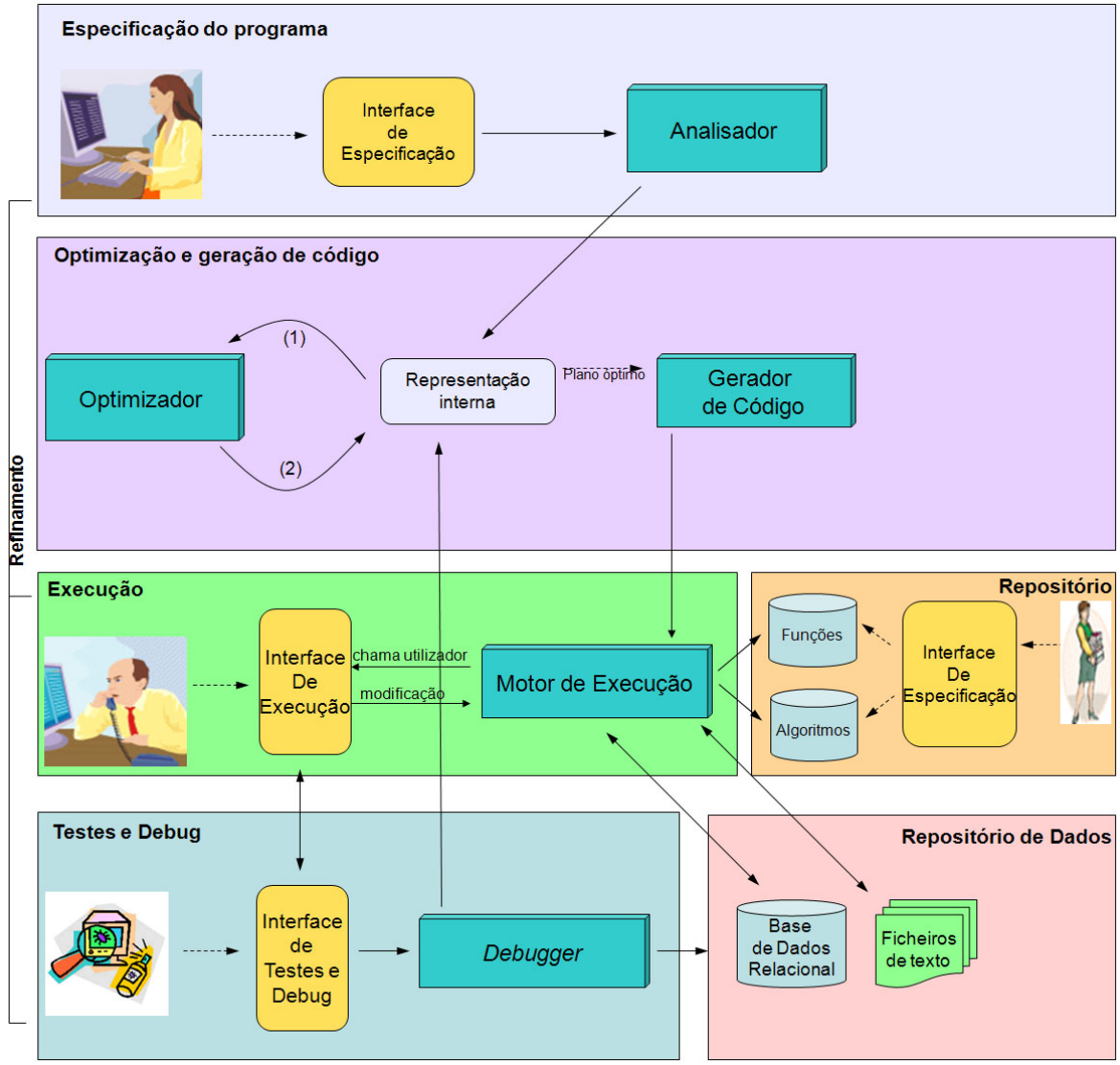

Figura 3.2: Arquitectura do *Ajax*

dados no *Ajax* são os seguintes:

- Analisador/Parser: analisa o programa de transformação de dados escrito usando a linguagem de especificação do *Ajax*, verifica a sua estrutura sintáctica e gera a sua representação interna, que é constituída por um conjunto de classes *Java*.
- Optimizador: analisa a estrutura de representação interna do programa de transformação de dados e determina o plano de execução óptimo para o conjunto de transformações especificadas.
- Gerador de código: gera código *Java* que implementa cada uma das transformações

definidas no programa de transformação.

- Motor de execução: executa as transformações de acordo com o código gerado pelo Gerador de código e segundo o plano concebido pelo Optimizador.

- *Debugger*: permite analisar os registos produzidos por cada transformação e a sua proveniência, com o objectivo de apoiar o refinamento e correcção do programa de transformação de dados definido.

Para ilustrar as relações existentes entre os diferentes componentes desta arquitectura, apresenta-se, de seguida, o cenário típico de execução de um programa de qualidade de dados.

Na fase de especificação, o utilizador escreve um programa de transformação, usando a linguagem declarativa suportada pelo *Ajax*, que consiste numa extensão da linguagem *SQL*. Através da interface de especificação, o utilizador indica o ficheiro que contém o programa definido. Todas as funções e algoritmos usados no programa especificado devem fazer parte dos repositórios de funções e de algoritmos, respectivamente. Se não fizerem parte dos repositórios, o utilizador deve implementar as funções e algoritmos pretendidos e adicioná-los ao repositório respectivo.

O programa especificado é então interpretado pelo analisador, que valida a sua sintaxe e cria as estruturas internas em *Java*, que representam o programa. As estruturas internas produzidas durante a fase de análise representam toda a informação definida no programa, como por exemplo, os nomes e tipos das transformações, e fluxos de entrada.

Após a fase de análise, é desencadeada uma fase de optimização do programa. A optimização é efectuada com base na informação contida nas estruturas internas, como, por exemplo, o tipo de transformação e os dados de entrada. O optimizador é responsável por determinar os algoritmos mais adequados para cada uma das transformações, bem como gerar um plano de execução tido como óptimo. Com base neste plano de execução, o gerador de código cria o código executável com os algoritmos seleccionados para cada uma das transformações.

Na fase de execução e correcção, o motor de execução executa o código gerado pelo gerador de código e disponibiliza uma interface gráfica que permite ao utilizador con-

trolar a execução de cada uma das transformações, assim como visualizar os resultados obtidos. É possível, por exemplo, iniciar uma transformação e visualizar os dados por ela produzidos.

Finalmente, o *debugger* permite ao utilizador determinar a proveniência de cada registo no grafo de transformações de dados. Este mecanismo pode ser usado em registos correctos ou em registos de excepções. Depois de determinar os registos de origem de um determinado registo, o sistema disponibiliza uma interface que permite ao utilizador corrigir, editar ou mesmo apagar os registos de origem e voltar a executar cada transformação. O utilizador pode navegar no grafo de transformações, sendo possível, por exemplo, executar uma transformação, analisar os resultados obtidos, corrigir os dados de entrada e voltar a executar a transformação com os dados corrigidos.

3.2 Operadores

A abordagem seguida por várias ferramentas comerciais consiste em definir um operador para cada tipo específico de transformação [Gal01]. Existem, por exemplo, operadores específicos para normalizar nomes e operadores específicos para normalizar moradas. Esta abordagem denota vários problemas, como a rápida proliferação de operadores, ou mesmo a redundância destes.

O *Ajax* segue uma abordagem alternativa, na qual se especificam os diversos tipos de transformações, em função do tipo de mapeamento existente entre os registos de entrada e os registos de saída. No *Ajax* definem-se seis operadores: *Map*, *View*, *Match*, *Cluster*, *Merge* e *Appply*. Cada operador do *Ajax*, excepto o operador *View* pode gerar excepções, que correspondem a registos de entrada que violam a lógica da transformação implementada pelo operador. As excepções podem ter origem na invocação de uma função externa implementada em *Java* ou na violação de uma restrição de integridade das tabelas de saída de um operador.

O operador *Map* tem como entrada uma única relação e produz, como saída, uma ou mais relações. Permite representar uma transformação de um-para-muitos, isto é, para cada tuplo da relação de entrada, gera zero ou mais tuplos nas relações de saída.

Existem três componentes fundamentais no operador *Map*:

- *let-clause* - consiste num conjunto de atribuições efectuadas a variáveis internas do operador. A cada variável é atribuído o valor de uma expressão, como, por exemplo, uma função registada na biblioteca de funções. A avaliação de uma expressão, nomeadamente a invocação de uma função *Java* pode gerar uma excepção, sendo o tuplo de entrada que a origina adicionado à tabela de excepções. Também existe cláusula *let* nos operadores *Match* e *Merge*.

- *where* - permite filtrar os registos da tabela de entrada através de uma determinada condição. Os operadores *Macth* e *Merge* também permitem filtrar os registos de entrada através da condição especificada na cláusula *where*.

- *constraint* - é constituído pelas restrições semelhantes às restrições de integridade da linguagem *SQL*, nomeadamente *not null*, *unique*, *foreign key* e *check*. Note-se que sempre que uma destas restrições é violada, o tuplo de entrada respectivo também é adicionado a uma tabela de excepções. No operador *View* também é possível especificar restrições nos tuplos produzidos, através da cláusula *constraint*. No entanto, no caso do operador *View* a violação de uma destas restrições interrompe a execução do operador, que é realizada em *SQL*.

O operador *View* corresponde a uma interrogação *SQL* aumentada com restrições de integridade, como *Not Null* e *Unique*, ao nível dos tuplos produzidos. A interrogação *SQL* permite efectuar junções e uniões. Assim, é possível fazer mapeamentos de muitos-para-um e de muitos-para-muitos. A execução deste operador é efectuada através de uma interrogação *SQL*. Tendo em conta que a existência de uma excepção em *SQL* interrompe a execução da interrogação *SQL*, o operador *View* não suporta a geração de excepções com a mesma semântica dos outros operadores.

O operador *Match* permite efectuar uma junção aproximada entre duas relações. O algoritmo de execução mais básico do operador *Match* efectua o produto cartesiano entre as duas relações de entrada e filtra apenas os pares de tuplos que apresentam uma semelhança superior a um determinado valor. A função (ou funções) de semelhança a utilizar

é arbitrária, podendo ser acrescentadas novas funções na biblioteca de funções. Este operador é indispensável em aplicações de limpeza de dados, para permitir a detecção de registos duplicados. O *Match* corresponde a um mapeamento de muitos-para-muitos, na medida em que um conjunto de tuplos da relação de entrada produz zero ou mais tuplos na relação de saída.

O operador *Cluster* corresponde a um caso especial de mapeamento de muitos-para-muitos aplicado a uma relação na qual os registos correspondem a distâncias entre pares de tuplos. O operador tem como entrada uma única relação e produz uma única relação de saída. Conceptualmente, os registos são organizados em grupos (*clusters*). A relação de saída contém o identificador do grupo e o identificador do registo que lhe pertence.

O operador *Cluster* surge, geralmente, na sequência da aplicação de um *Match*, sendo aplicado ao resultado de um *Match* para agrupar os registos semelhantes. Contudo, é importante salientar que o operador *Cluster* pode ser aplicado a relações que não sejam, necessariamente, o resultado de um operador *Match*.

O operador *Merge* tem como entrada uma única relação e produz uma relação de saída. Corresponde a um mapeamento de muitos-para-muitos, no qual os tuplos da relação de entrada são agrupados conforme os valores dos seus atributos. Deste modo, tuplos que tenham o mesmo valor no conjunto de atributos seleccionado pertencem à mesma partição. Por outro lado, também se podem usar funções agregadoras, desde que estejam definidas na biblioteca de funções do *Ajax*. Assim, é possível criar novas funções agregadoras e usá-las neste operador.

O operador *Apply* corresponde a um mapeamento de muitos-para-muitos, no qual o algoritmo aplicado é definido pelo utilizador. Desta forma, é possível aplicar transformações que não possam ser especificadas por nenhum dos operadores anteriores, bastando, para isso, definir o algoritmo pretendido. Os algoritmos usados pelo operador *Apply* podem ser aplicados a um tuplo ou a um conjunto de tuplos e gerar zero ou mais tuplos.

3.3 Migração de dados do *gestBarragens*

Um projecto de migração de dados pode ser visto como um conjunto de transformações de dados e de esquema que, aplicado aos dados fonte, os transforma em dados que satisfazem o esquema do sistema alvo. Um caso típico consiste na migração de dados armazenados em vários ficheiros ASCII para uma base de dados relacional, como é o caso da migração dos dados relativos ao controlo de segurança de barragens de betão para a base de dados do sistema *gestBarragens*.

É importante notar que o principal objectivo da migração de dados consiste em transformar os dados de forma a obedecerem ao esquema alvo. No entanto, é importante que os dados finais não sejam considerados "dados sujos" ou sem qualidade. Assim, o processo de migração deve incluir um forte componente de qualidade dos dados. O próprio esquema do sistema alvo pode "obrigar" a uma melhoria da qualidade dos dados, já que pode incluir novas restrições.

Considere-se o exemplo da migração da informação relativa aos fios de prumo no sistema *gestBarragens*. Os dados iniciais incluem um ficheiro com a definição dos fios de prumo (tabela de apoio), um ficheiro com as leituras realizadas nos fios de prumo (ficheiro de leituras) e um ficheiro com os resultados dos fios de prumo (ficheiro de resultados).

A Figura 3.3 apresenta um excerto do modelo ER (tal como o modelo introduzido na Secção 1.3.4), usando a nomenclatura definida em [SKS02], que modela a base de dados alvo. Este modelo é simplificado, na medida em que não apresenta os atributos das entidades nem a listagem das restrições de integridade que não podem ser modeladas no esquema. Na Tabela 3.1 apresenta-se o modelo relacional correspondente. Este modelo também é simplificado, dado que apresenta apenas um subconjunto dos atributos de cada relação. Segue a nomenclatura apresentada em [SKS02], onde os atributos que compõem as chaves primárias de cada relação estão sublinhados, os atributos que são chaves estrangeiras estão em itálico (e.g., o atributo grupo da relação *Base de Coordinómetro* referencia a relação *Fio de Prumo*) e os atributos não nulos estão a cheio (negrito).

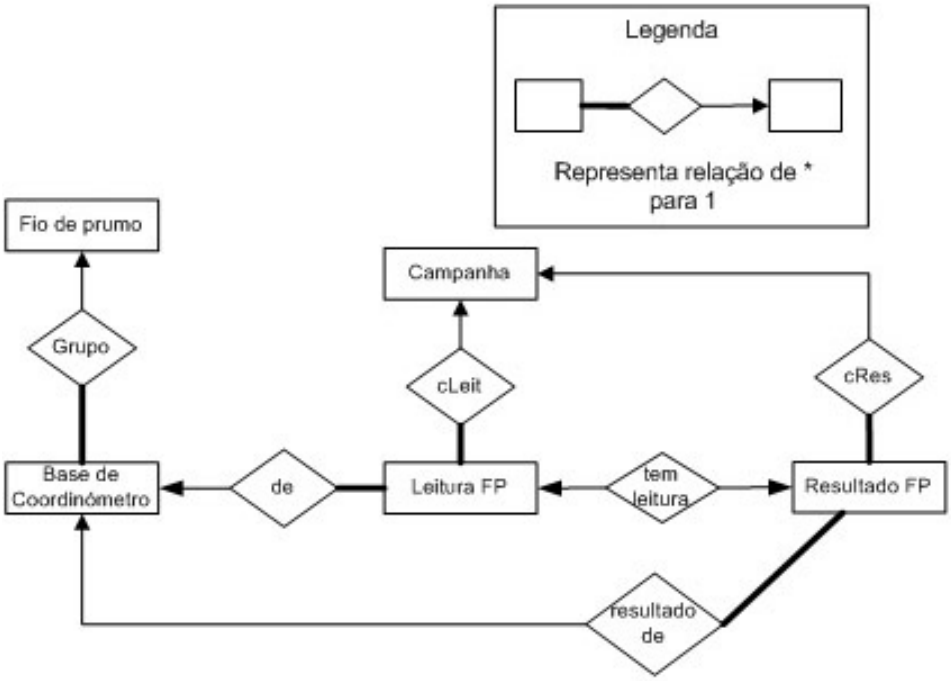

Figura 3.3: Modelo ER dos fios de prumo

Campanha (id, **datainicio**, **datafim**)
Fio de prumo (id, **identificacao**, **tipo**)
Base de coordinómetro (id, ***grupo***, **codigo**, RFInicial, RFMax, RFMin)
Leitura FP (id, ***campanha***, ***instrfixo***, data, RF)
Resultado FP (*instrfixo*, *campanha*, *leitura*, data, deslocRadial, deslocTangencial)

Tabela 3.1: Modelo relacional para o exemplo dos fios de prumo

Em complemento, o modelo relacional pode incluir restrições de integridade. No exemplo apresentado podem considerar-se as seguintes restrições:

- a data de fim de uma campanha tem que ser posterior à data de inicio.
- o valor máximo para a leitura radial ao fio ($RFMax$) tem que ser superior ao valor mínimo para a leitura radial ao fio ($RFMIN$).
- a leitura da radial ao fio (RF) tem que estar compreendida entre os valores mínimo ($RFMin$) e máximo ($RFMax$) da base de coordinómetro identificada pela chave estrangeira (*instrfixo*).

Desta forma, o objectivo consiste em transformar os dados dos ficheiros ASCII de entrada - tabela de apoio, ficheiro de leituras e ficheiro de resultado - para a base de dados do novo sistema. Como a base de dados impõe restrições de integridade que não existiam nos ficheiros ASCII, como por exemplo a definição das relações entre as bases de coordinómetro e as leituras dos fios de prumo (e.g., não pode existir uma leitura realizada numa base de coordinómetro que não existe) a execução do processo de migração de dados pode ser interrompido se alguma das restrições de integridade for violada. Assim, para que os dados iniciais possam ser migrados para o esquema alvo, pode ser necessário melhorar a qualidade dos dados. O processo de migração de dados inclui uma fase de limpeza de dados que é imposta pelo esquema final (normalmente mais evoluído). Essa fase de limpeza pode ser alargada de forma a melhorar a qualidade dos dados.

Considerando a migração de dados "apenas" como o conjunto de transformações aplicadas aos dados fonte, de forma a transformá-los para o esquema do sistema alvo, isto é, aplicando transformações de limpeza de dados e de esquema que garantam que os dados respeitam as restrições de integridade impostas pelo esquema final, os operadores do *Ajax* mais adequados para estas transformações são o *Map* e o *View*. No caso da migração dos dados relativos ao controlo de segurança de barragens de betão, não foram implementadas as tarefas exclusivamente de limpeza de dados, como por exemplo de eliminação de duplicados, já que o objectivo desta migração era unicamente garantir que os dados respeitavam o esquema definido pela base de dados alvo, que era mais evoluído do que o esquema dos dados fonte.

O operador *Map* permite realizar transformações de um-para-muitos com acesso a funções externas, em que cada tuplo de entrada pode gerar um ou mais tuplos de saída. É o operador ideal para operações sobre um registo, como normalizações, conversões ou geração de novos identificadores. No exemplo da migração dos fios de prumo, uma vez que é possível acrescentar novas funções, o *Map* também pode ser usado para extrair a informação de cada ficheiro. Por exemplo, a partir de cada linha da tabela de apoio é possível extrair a identificação, o código, o bloco, etc., recorrendo a uma função de *split* que tem como argumentos a origem, um *offset* e a cadeia de caracteres de cada linha do

ficheiro. Note-se que as funções utilizadas por uma transformação implementada pelo operador *Map* poderão ser reutilizadas noutras transformações.

O *View* permite fazer transformações de muitos-para-muitos (ou o caso especial de muitos-para-um), através das operações relacionais de união e junção.

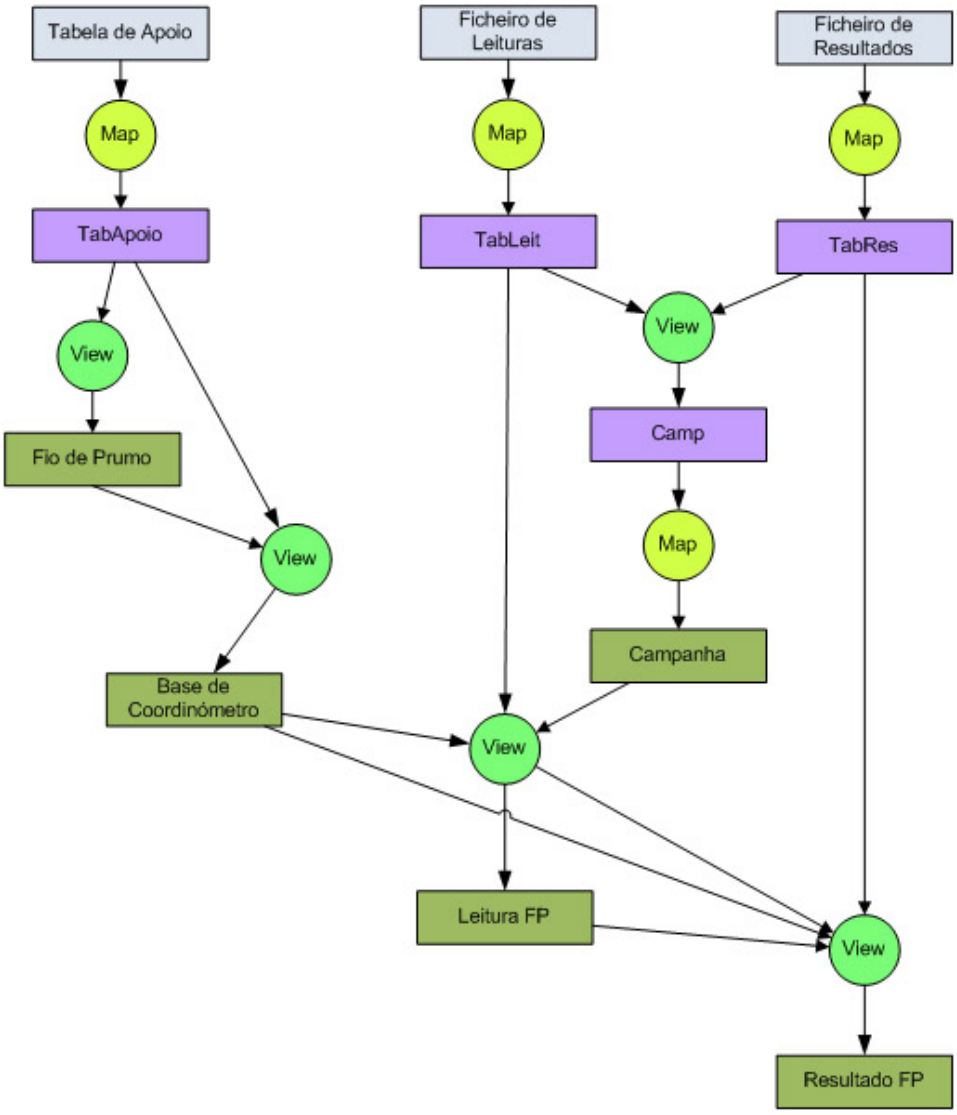

Figura 3.4: Grafo de transformações para os fios de prumo

A Figura 3.4 apresenta o grafo de transformações desenhado para migrar os dados relativos aos fios de prumo para o esquema da base de dados representado pelo modelo relacional da Tabela 3.1, que inclui as tabelas *Fio de Prumo*, *Base de Coordinómetro*, *Campanha*, *Leitura FP* e *Resultado FP*. Inicialmente, aplica-se um *Map* a cada um dos ficheiros de entrada. Neste mapeamento gera-se uma nova chave para identificar

cada registo e aplica-se um conjunto de funções para separar os diferentes campos da informação contida em cada registo inicial. Por exemplo, no mapeamento que produz a tabela *TabLeit*, cada linha do ficheiro da tabela de apoio, é separada em vários atributos, nomeadamente a identificação da base de coordinómetro, o bloco em que se situa, a cota e os limites de validação para cada leitura.

A tabela de apoio dos fios de prumo contém informação sobre as bases de coordinómetro e sobre os fios de prumo, que correspondem a grupos de bases de coordinómetro com a mesma identificação e bloco. A transformação que obtém os diferentes fios de prumo, isto é, os diferentes grupos de bases de coordinómetro com a mesma identificação e bloco, é implementada através do operador *View*, que produz a tabela *Fio de Prumo*. Do mesmo modo, no carregamento das campanhas é necessário obter as diferentes datas em que foram registadas leituras ou calculados resultados, que também é realizado pelo operador *View*, que produz a tabela *Camp*. Para finalizar o carregamento das campanhas é necessário gerar um novo identificador para cada campanha através do operador *Map* que produz a tabela *Campanha*.

O operador *View* também é utilizado para cruzar informação proveniente de várias relações. Por exemplo, para migrar as bases de coordinómetro para a tabela *Base de Coordinómetro* é necessário cruzar a informação da tabela *TabApoio*, que contém a informação de todas as bases de coordinómetro, com a informação da tabela *Fio de Prumo*, que contém a definição dos fios de prumo. Assim, é possível determinar o fio de prumo a que corresponde cada uma das bases de coordinómetro e estabelecer a relação de muitos-para-um representada no modelo ER da Figura 3.3.

É importante notar que, a migração dos dados para uma tabela como *Base de Coordinómetro* que contém uma chave estrangeira para a tabela *Fio de Prumo* tem que ser posterior ao carregamento da informação na tabela *Fio de Prumo*, já que é necessário estabelecer a relação da tabela *Base de Coordinómetro* com a tabela *Fio de Prumo*. Assim, a migração das leituras dos fios de prumo para a tabela *Leitura FP* depende da migração para as tabelas *Campanha*, *Base de Coordinómetro* e *TabLeit* que têm que ser cruzadas para obter os identificadores da campanha e da base de coordinómetro em que foi realizada a leitura. Do mesmo modo, a migração dos resultados dos fios de prumo

para a tabela *Resultado FP* depende da migração para as tabelas *Campanha*, *Base de Coordinómetro*, *Leitura FP* e *TabRes* que também têm que ser cruzadas para obter os identificadores da campanha, da base de coordinómetro e da leitura que deu origem a cada resultado.

3.4 Extensões

No *Ajax*, todos os operadores geram excepções, excepto o operador *View*, pois corresponde a uma interrogação *SQL*. Em *SQL*, as interrogações não geram excepções que permitam continuar a execução da interrogação. No exemplo da migração da informação relativa aos fios de prumo, é necessário considerar que uma leitura realizada num instrumento que não existe seja considerada uma excepção e, posteriormente, fornecer mecanismos para corrigir essa excepção. Assim, todos os registos de entrada que não são transformados pela lógica do operador devem ser detectados, através da geração de excepções. Esta necessidade implica dotar o operador *View* de mecanismos de geração de excepções.

Um dos desafios levantados pelos processos ETL utilizados no carregamento de *Data Warehouses* é o carregamento incremental dos dados. Este processo implica que um registo com a mesma informação não seja duplicado na base de dados final. Se o registo for alterado na fonte de dados, essa alteração deve reflectir-se nos registos produzidos. Este problema não atinge a mesma dimensão num processo de migração, já que este processo não é periódico, tendo apenas um "período de vida" limitado. Contudo, pode ser necessário re-migrar a mesma informação. Também é usual que a informação esteja dispersa em várias versões de ficheiros e, dessa forma, parte do processo de migração tenha que ser executada diversas vezes. No caso da migração dos dados relativos ao controlo de segurança de barragens de betão para o sistema *gestBarragens*, os dados fonte estão armazenados em vários ficheiros que contêm o mesmo tipo de informação. Por exemplo, existe um ficheiro com informação dos fios de prumo da barragem do Alto Rabagão e outro ficheiro com informação dos fios de prumo da barragem do Alqueva. Assim, por exemplo, aquando do carregamento da informação dos fios de prumo da

barragem do Alto Rabagão para as tabelas da base de dados final, todos os fios de prumo de outras barragens (e.g., Alqueva), que tenham sido previamente carregados, devem ser mantidos na base de dados. Por outro lado, se existirem registos da barragem do Alto Rabagão que estejam errados e sejam corrigidos, esses registos, numa segunda iteração, devem ser actualizados na base de dados final.

O *Ajax* não satisfaz estes requisitos para migração incremental, já que em cada transformação a tabela de saída é re-criada. No caso da tabela já existir e conter informação, essa tabela é eliminada e criada de novo sem qualquer registo.

Uma vez que o operador *View* equivale a uma cláusula *SQL* acrescida de algumas restrições, e tendo em consideração que a gramática deste operador só permitia junções e uniões em *SQL*, decidiu-se melhorar a gramática de forma a aumentar a expressividade do operador. As cláusulas acrescentadas na gramática de *SQL* suportada pelo *Ajax* estão relacionadas com as necessidades das transformações da migração de dados do projecto *gestBarragens*. As principais extensões ao *SQL* suportado são as seguintes:

- Union e Union All. É necessário distinguir entre *union* (com eliminação de registos duplicados) e *union all* (sem eliminação de duplicados). Na versão inicial só existia a cláusula *union*. No entanto, esta opção tinha o comportamento de um *union all*.

- Group by - having. É necessário disponibilizar o *group-by* de *SQL* e as funções agregadoras típicas (*count*, *max*, *min* e *avg*), bem como a cláusula *having* para controlar os registos produzidos.

- Outer join. É necessário permitir os três tipos de *outer join* existentes em *SQL*: *left outer join*, *right outer join* e *full outer join*. A gramática inicial não suportava nenhum tipo de *outer join*.

Em seguida, apresentam-se, em detalhe, as principais extensões implementadas, nomeadamente, as excepções no operador *View*, o suporte à migração incremental e as extensões da gramática do operador *View*.

3.4.1 Excepções no operador *View*

Como se referiu na Secção 3.2, o operador *View* corresponde a uma instrução *SQL* com restrições de integridade aplicadas aos registos produzidos. Nesta Secção analisa-se, em detalhe, o operador *View*. A partir de exemplos simples relacionados com a migração do projecto *gestBarragens*, pretende-se justificar a necessidade de geração de excepções no operador *View*.

A Tabela 3.2 apresenta um esquema simplificado com a sintaxe do operador *View*. A cláusula *create view* define a criação de uma transformação de nome *view-name* implementada pelo operador *View*. Esta transformação produz uma tabela com o mesmo nome da transformação, *view-name*. As cláusulas *from*, *where* e *select* possuem a mesma semântica que as respectivas cláusulas *SQL*, isto é, na cláusula *from* determinam-se as tabelas de entrada da transformação, na cláusula *where* definem-se os filtros aos registos de entrada (e.g., *valor* > 20), e na cláusula *select*, especificam-se os atributos de saída da transformação. A cláusula *constraint* permite definir restrições aos tuplos produzidos, como por exemplo *not null* e *unique*. Através da *key-clause* é possível determinar o conjunto de atributos que compõem a chave primária da tabela produzida pela transformação. Finalmente, é possível unir dois conjuntos de dados através da cláusula *union*.

<view-operator>:	**create view** <view-name>
	from <from-clause>
	[**where** <where-clause>]
	{ select <select-clause> **}**
	[**constraint** <constraint-clause>]
	[**union**
	from <from-clause>
	[**where** <where-clause>]
	{ select <select-clause> **}**
	[**constraint** <constraint-clause>]]∗
	[**key** <key-clause>]

Tabela 3.2: Sintaxe do operador View

Para ilustrar as funcionalidades deste operador, recorre-se ao exemplo da migração de dados relativos aos fios de prumo, que pode ser implementada pelo grafo de trans-

formações da Figura 3.4. Recorde-se que os dados fonte estão armazenados em três ficheiros: *tabela de apoio*, que contém a definição das bases de coordinómetro e fios de prumo; *ficheiro de leituras*, que contém os registos das leituras realizadas nas bases de coordinómetro de cada fio de prumo; e *ficheiro de resultados*, que contém os resultados calculados a partir das medições efectuadas nas bases de coordinómetro de cada fio de prumo.

O exemplo que se segue corresponde a uma transformação especificada pelo operador *View*, que produz a tabela *LeituraFP*.

```
CREATE VIEW LeituraFP
FROM TabLeit T, Campanha C
LEFT OUTER JOIN BaseCoordinometro B
ON B.codigo = T.codigo
WHERE T.Data = C.datainicio
{SELECT T.id, C.id as campanha, B.id AS instrfixo, T.data, T.rf
CONSTRAINT UNIQUE(campanha, instrfixo)
 NOT NULL instrfixo
 NOT NULL campanha
KEY id
}
```

Esta transformação corresponde à aplicação de um *outer join* entre as tabelas *TabLeit*, *BaseCoordinometro* e *Campanha*. Esta transformação é implementada por um *outer join*, para que todos os tuplos da tabela *TabLeit*, que corresponde ao ficheiro de leituras, sejam produzidos na saída da transformação. Assim, não se perde informação durante o processo de migração. A aplicação da transformação produz a tabela *LeituraFP*, que contém cinco atributos - *id*, *campanha*, *instrfixo*, *data* e *rf* -, tal como especificado na cláusula *select*. Os tipos de dados destes atributos correspondem aos tipos de dados dos atributos *id* da tabela *TabLeit*, *id* da tabela *Campanha*, *id* da tabela *BaseCoordinometro*, *data* e *rf* da tabela *TabLeit*. No exemplo do código apresentado, define-se a chave primária da tabela produzida (*LeituraFP*). Essa definição é feita através da

cláusula *key* que, neste caso, determina que o atributo *id* compõe a chave primária da tabela produzida. Para além disso, definem-se três restrições de integridade na cláusula *constraint* que indicam que o par de atributos (*campanha*, *instrfixo*) tem que ser único e que os atributos *instrfixo* e *campanha* não podem ser nulos[1].

É importante salientar que, tal como acontece numa interrogação em *SQL* do tipo `<insert> <into>(...)<select>(...)<from>(...)`, se existirem registos que violem uma restrição de integridade, a execução do operador falha. Num processo de migração de dados, não se pretende que a execução do operador falhe, mas sim que os registos correctos sejam migrados para o esquema final e que sejam fornecidos mecanismos para identificar e corrigir os registos que fazem falhar a execução do operador. Seguindo a nomenclatura usada no *Ajax*, pretende-se que os registos que falham na execução de um operador *View* sejam considerados excepções e que seja possível utilizar os mecanismos de *debugging* do *Ajax* para analisar e corrigir as excepções.

A Figura 3.5 apresenta um exemplo de execução da transformação representada no grafo da Figura 3.4 para gerar a tabela *LeituraFP*. Note-se que, nesta transformação, as tabelas *BaseCoordinometro* e *Campanha* funcionam como *look-up tables* para definir as relações estabelecidas, quer entre os registos de leituras e a base de coordinómetro, quer entre os registos de leituras e a campanha.

O registo produzido com o código "103" assume o valor *null* no atributo *instrfixo*, o que viola uma restrição de integridade na tabela de saída da transformação. Este erro deve-se ao facto de que o registo de código "13" da tabela *TabLeit* (marcado a vermelho na Figura 3.5) não está correcto, já que não existe nenhuma base de coordinómetro com código "13". Como o operador *View* é executado em *SQL*, a violação de uma restrição de integridade interrompe a execução do operador, não permitindo identificar os registos que violam restrições de integridade, nem tão pouco transformar os registos correctos. Note-se que, caso as três tabelas fossem cruzadas através de uma junção, em vez de um *outer join*, o registo irregular não passaria de todo para a saída da transformação, mas seria possível transformar os outros registos. Em todo o caso, o que se pretende é migrar

[1] As restrições de integridade que definem que o par de atributos (*campanha*, *instrfixo*) é único e não nulo significa que este par de atributos é uma chave candidata da relação *LeituraFP*[SKS02]

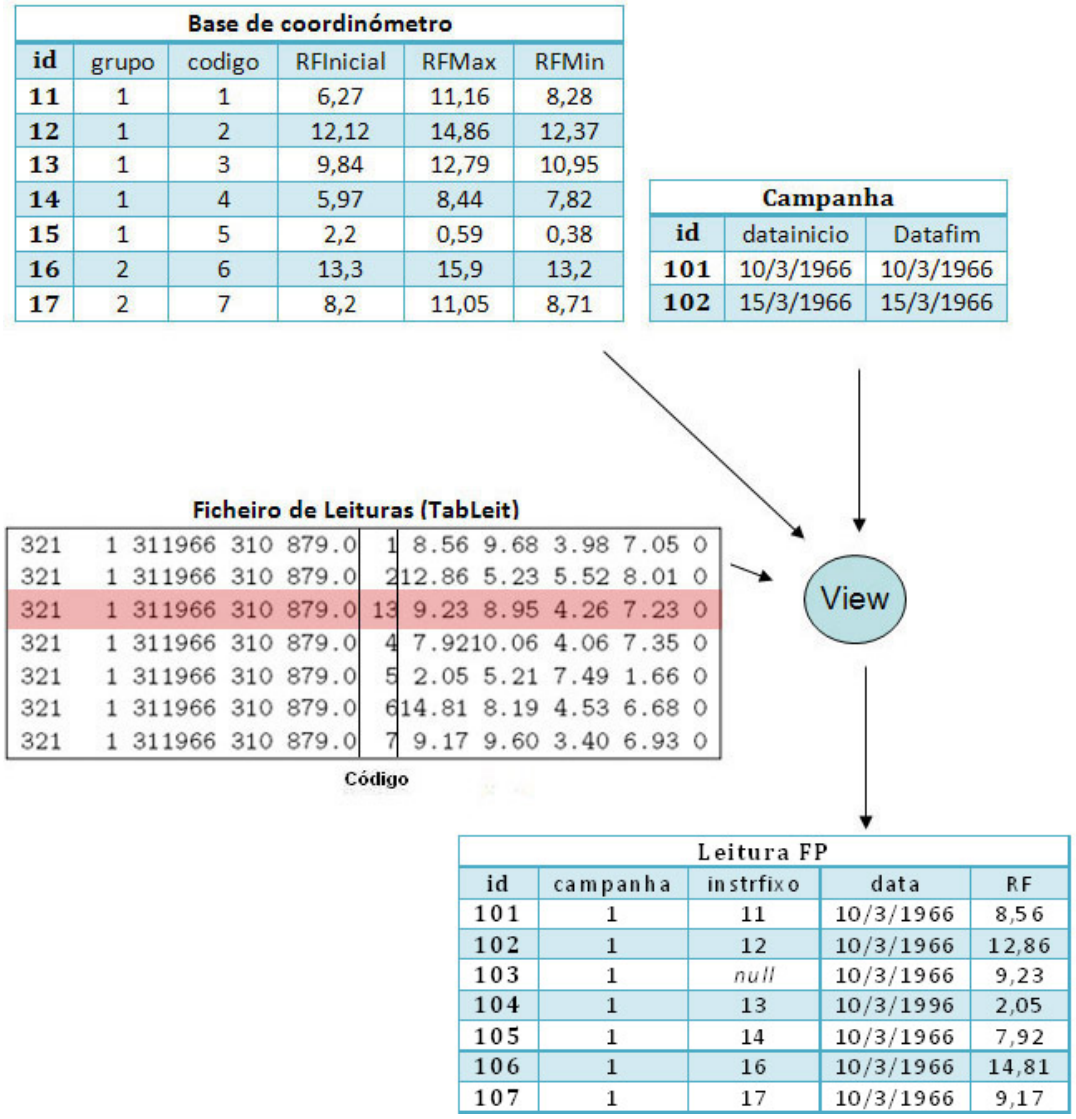

Figura 3.5: Registos errados no operador *View*

os registos correctos e identificar claramente os registos irregulares, para que possam ser corrigidos.

De seguida, descrevem-se duas possíveis soluções para a identificação de registos com problemas. A primeira solução implica a reescrita do código da transformação e a criação de uma nova transformação implementada pelo operador *View*, que pode ser executada em *SQL*, mas não é uma solução geral, aplicando-se apenas a cada problema em particular. Na segunda solução, procede-se à extensão do operador *View* com a inclusão de mecanismos de geração de excepções. Esta solução implica que o operador

View seja executado em *Java*.

3.4.1.1 Solução 1: sem geração de excepções

Esta solução baseia-se no exemplo apresentado na Figura 3.5, no qual só existem registos com anomalias, ou registos irregulares numa tabela de entrada da transformação (*TabLeit*). Neste caso, o registo de código "13" da tabela de entrada *TabLeit* não tem correspondência na tabela *Base de Coordinómetro*, o que implica a projecção do valor *null* no atributo *instrfixo* da tabela de saída.

A ideia consiste em identificar a causa da violação da restrição de integridade e, reescrever o código da transformação de forma a evitar violar essa restrição de integridade. Neste caso particular, se em vez de aplicar um *outer join* com a tabela *Base de Coordinómetro*, for realizada uma junção, o registo de código "103" não será produzido e, consequentemente, a execução do operador deixa de ser interrompida. No entanto, não existe nenhuma forma de identificar os registos de entrada que estavam errados, isto é, que davam origem a um registo que violava uma restrição de integridade na tabela de saída. Neste caso particular, com a reescrita da transformação, todos os registos de entrada da tabela *TabLeit* que não deram origem a nenhum registo, são considerados irregulares. O registo de código "13" da tabela *TabLeit* no exemplo da Figura 3.5 é o único registo irregular.

A identificação de registos irregulares só pode ser realizada após a execução da transformação e através da análise dos registos gerados. Deste modo, é necessário inspeccionar os registos gerados e localizar os registos da tabela de entrada que não deram origem a qualquer registo produzido. Esta análise recorre ao mecanismo de proveniência de dados disponibilizado pelo *Ajax*, que se baseia na propagação dos identificadores dos registos. Assim, é possível criar uma nova transformação, implementada pelo operador *View* que verifica quais os tuplos de entrada que não produziram nenhum tuplo. A Figura 3.6 ilustra a criação dessa nova transformação, representada por *T'*. Neste caso, a transformação *T'* determina os registos da tabela *TabLeit* que não deram origem a nenhum registo da tabela *Leitura FP*, o que pode ser realizado através da seguinte interrogação *SQL*:

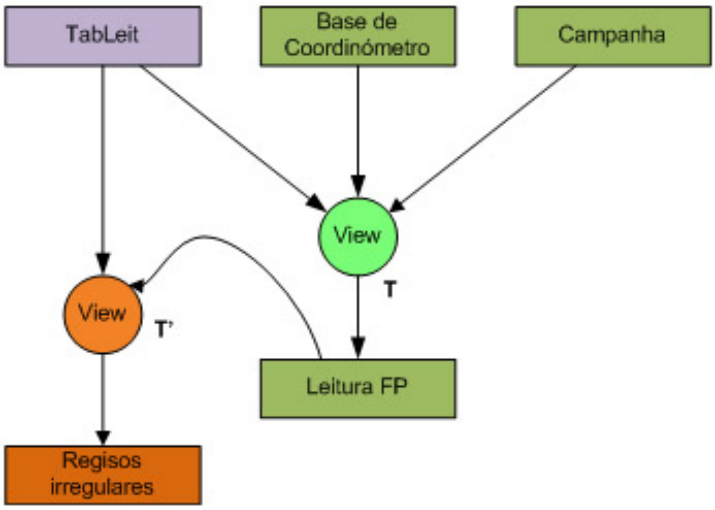

Figura 3.6: Identificação de registos errados com nova transformação

```
SELECT *
FROM TabLeit
WHERE id NOT IN
( SELECT id
 FROM LeituraFP )
```

Num caso genérico, se *K1* for o conjunto dos atributos que compõem a chave da tabela base e *KF1* o mesmo conjunto de atributos na tabela produzida, a seguinte interrogação *SQL* determina os registos irregulares na transformação:

```
SELECT *
FROM TABELA-BASE
WHERE <K1> NOT IN
( SELECT <KF1>
 FROM TABELA-PRODUZIDA )
```

Esta solução permite determinar registos irregulares como o registo de código "13" da tabela *TabLeit* no exemplo da Figura 3.5, mas é muito ineficiente. Em transformações que envolvam tabelas com alguns milhares de registos, esta implementação funciona razoavelmente, mas, com centenas de milhar ou milhões de registos, como é o caso da

informação envolvida na migração para o projecto *gestBarragens*, esta solução torna-se incomportável, não dando resposta em tempo útil.

Mantendo a mesma lógica na identificação de registos irregulares, pode resolver-se o problema da ineficiência e escalabilidade através da reescrita da interrogação *SQL*, com recurso a *outer joins*. Note-se que o resultado lógico é exactamente o mesmo da interrogação implementada com a cláusula *NOT IN*.

No exemplo apresentado, a interrogação *SQL* pode ser reescrita para:

```
SELECT TL.*   -- lista de colunas da tabela TabLeit
FROM (
  SELECT TL.*, LFP.id AS LFPID
  FROM TabLeit TL
     LEFT OUTER JOIN
     LeituraFP LFP
     ON TL.id = LFP.id
     )
WHERE LFPID IS NULL
```

Num caso genérico, representando novamente por *K1* o conjunto dos atributos que compõem a chave da tabela base e *KF1* o mesmo conjunto de atributos na tabela produzida, a seguinte interrogação *SQL* determina os registos irregulares da transformação:

```
SELECT TB.*   -- lista de colunas da tabela-base
FROM (
  SELECT TB.*, TP.<KF1> AS <TPID>
  FROM TABELA-BASE TB
     LEFT OUTER JOIN
     TABELA-PRODUZIDA TP
     ON TB.<K1> = TP.<KF1>
     )
WHERE <TPID> IS NULL
```

Esta solução apresenta várias dificuldades. De facto, com este mecanismo é possível identificar registos irregulares, como o registo com código "13" no exemplo da Figura 3.5, isto é, registos da tabela origem que não têm correspondência numa das tabelas envolvidas na junção. No entanto, este mecanismo não é automático. É necessário reescrever o código que implementa a transformação, de forma a evitar a violação de restrições de integridade e é necessário escrever uma nova transformação que identifique os registos irregulares. Esta nova transformação pode ser criada automaticamente, sem ser necessário a intervenção do utilizador, mas é impossível prever todos os cenários de erros. Por exemplo, no caso da migração das leituras dos fios de prumo, não é possível ter duas leituras no mesmo instrumento e na mesma data. Registos nestas condições também são irregulares, já que violam a restrição de *unique* na tabela de saída, e devem ser corrigidos. Pode seguir-se a mesma abordagem para identificar estes registos, mas não existe nenhuma forma automática de prever e identificar todas as irregularidades nos dados envolvidos num processo de migração de dados. Além disso, é sempre necessário reescrever a transformação para evitar violar as várias restrições de integridade e criar uma nova transformação para identificar cada tipo de irregularidades nos dados.

3.4.1.2 Solução 2: com geração de excepções

Esta solução visa suprir os problemas apresentados pela solução anterior, na qual é necessário reescrever a transformação e criar uma nova transformação para identificar registos irregulares num operador *View*. Pretende-se incluir o conceito de excepções no operador *View*, que devem permitir identificar registos de entrada errados ou que não respeitem a lógica da transformação, isto é, que violem uma restrição de integridade imposta na tabela de saída.

A geração de excepções no operador *View* não existe por omissão. De facto, uma *view* corresponde exactamente a uma interrogação *SQL* e o mecanismo de excepções suportado pela linguagem *SQL* não permite que a execução da interrogação continue após a ocorrência de uma excepção. Sempre que se pretender que uma transformação do tipo *View* gere excepções é necessário torná-lo explícito na sintaxe do operador. A Tabela 3.3 apresenta a sintaxe do operador *View* com a possibilidade de geração de

excepções.

<view-operator>:	**create view** <view-name> [**with exceptions**] **from** <from-clause> [**where** <where-clause>] **{ select** <select-clause> **}** [**constraint** <constraint-clause>] [**union** **from** <from-clause> [**where** <where-clause>] **{ select** <select-clause> **}** [**constraint** <constraint-clause>]]∗ [**key** <key-clause>]

Tabela 3.3: Sintaxe do operador View com geração de excepções

Assim, a execução das *Views* que tenham a geração de excepções activa tem que ser efectuada necessariamente em *Java*, o que a torna menos eficiente, daí a possibilidade de opção entre a geração ou não de excepções, através da cláusula opcional *with exceptions*. Os dados de entrada de uma transformação implementada pelo operador *View* são obtidos através de uma interrogação *SQL*. Posteriormente, os registos produzidos são inseridos, um a um, na tabela de saída. Todos os registos que violem uma restrição de integridade são colocados na tabela de excepções.

Ao contrário da implementação através da criação de novas transformações, nesta solução a identificação de registos irregulares ocorre durante a execução da transformação, já que a geração de excepções ocorre registo a registo, isto é, ou o registo é inserido na tabela final ou é inserido na tabela de excepções.

3.4.2 Migração incremental

Em [Gal01], define-se o modo como a computação de alterações de forma incremental pode ser aplicada na *framework Ajax*. De facto, pretende-se determinar o impacto que uma alteração nos registos de entrada de um operador pode ter nos registos produzidos. Por outras palavras, pretende-se aferir se é possível aplicar a transformação apenas para o conjunto de registos alterados, inseridos ou apagados. Os operadores *Map*, *View*, *Match* e *Merge* podem, geralmente, ser executados de forma incremental.

Na Figura 3.7 apresenta-se um exemplo de execução incremental para o operador *Map* quando o tuplo de entrada *t* é alterado para o tuplo *t'*. Recorde-se que num operador *Map*, cada tuplo de entrada produz um ou mais tuplos de saída, independentemente dos outros tuplos de entrada. Esta característica permite a execução incremental deste operador. Assim, a execução do operador *Map* de forma incremental, pode ser feita em dois passos simples. Em primeiro lugar, é necessário determinar os tuplos que foram produzidos ou as excepções geradas, a partir do tuplo que foi alterado ou apagado na fonte de dados e eliminar esses tuplos produzidos ou de excepções. Em segundo lugar, deve aplicar-se novamente a transformação para o novo tuplo, no caso de ser alterado ou inserido um novo tuplo numa das tabelas de entrada e adicionar os tuplos produzidos nas tabelas de saída.

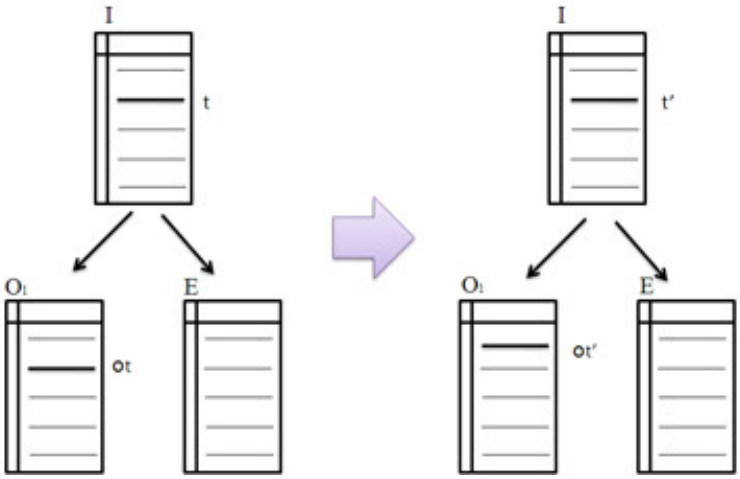

Figura 3.7: Execução incremental do operador Map numa actualização de t para t'

O operador *View* também pode ser executado de forma incremental se a transformação implementada obedecer a determinados critérios. A Figura 3.8 apresenta a execução incremental para o operador *View* numa transformação do tipo *Select, Project, Join, Union* (SPJU) quando o tuplo *t* de uma das relações de entrada é alterado para o tuplo *t'*. Numa interrogação *SQL* do tipo SPJU, cada tuplo de uma relação de entrada produz tuplos de forma independente dos outros tuplos da mesma relação de entrada. Deste modo, a execução de um operador *View* do tipo SPJU pode ser incremental quando se alteram tuplos numa única relação de entrada. Assim, quando o tuplo t é

alterado para t' numa relação de entrada da transformação, basta eliminar da relação de saída da transformação todos os tuplos que foram produzidos a partir do tuplo t e, aplicar a transformação para o tuplo t' em conjunto com todos os tuplos das outras relações de entrada.

No caso de transformações que envolvam agregação, do tipo *Aggregate, Project, Select, Join, Union* (APSJU) também é possível executar o operador *View* de forma incremental. Nestes casos, é necessário ter em conta o critério de criação de cada grupo, isto é, os atributos que definem cada partição. Assim, quando um tuplo t é actualizado para t' podem ocorrer três cenários: (*i*) o tuplo t' mantém-se na mesma partição em que estava o tuplo t; (*ii*) o tuplo t' muda de partição; (*iii*) o tuplo t' corresponde a uma nova partição. No primeiro caso, basta identificar o tuplo de saída que representa a partição a que pertencia o tuplo t e, recalcular as funções agregadoras para todos os tuplos da partição, juntamente com o tuplo t'. No segundo caso, é necessário recalcular as funções agregadoras para a partição a que pertencia o tuplo t e para a partição a que pertence o tuplo t'. Se o tuplo t era o único tuplo da partição antiga, é eliminado da relação de saída o tuplo produzido para essa partição. No terceiro caso, é necessário adicionar um novo tuplo na relação de saída com base na informação do tuplo t' e recalcular as funções agregadoras para a partição a que pertencia o tuplo t. Novamente, se o tuplo t era o único tuplo da partição antiga, é eliminado da relação de saída o tuplo produzido para essa partição.

O estudo da execução incremental dos operadores do *Ajax* realizado em [Gal01] determina a possibilidade de execução incremental dos operadores quando as fontes de dados são as mesmas e sofrem alterações num subconjunto de tuplos. No entanto, os dados legados provêm muitas vezes de ficheiros ASCII em que a informação pode estar partida em vários ficheiros com informação do mesmo tipo, como é o caso do *gestBarragens*. No fundo, esta divisão corresponde a uma partição horizontal dos dados. Além disso, a mesma informação pode estar duplicada em diferentes fontes de dados, como por exemplo em diferentes ficheiros. Assim, na fase de carregamento para o esquema final da base de dados do *gestBarragens*, é necessário verificar se cada um dos registos produzidos já se encontrava ou não na base de dados. Esta análise tem que ser feita

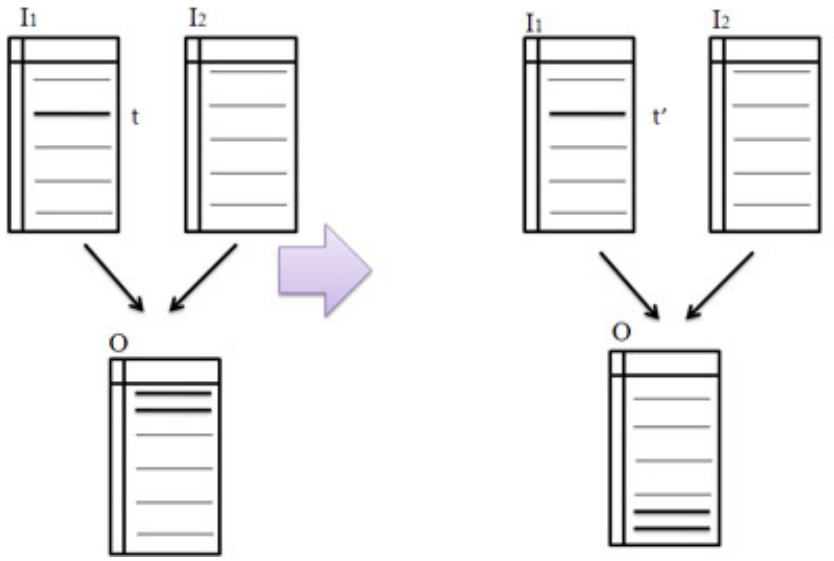

Figura 3.8: Execução incremental do operador View do tipo SPJU numa actualização de t para t'

registo a registo e de forma independente, já que no resultado de uma transformação para um determinado conjunto de dados de entrada, é possível produzir registos que já estavam na base de dados e registos que são novos.

Note-se que, no *Ajax*, a execução de cada operador conduz à re-criação das tabelas de saída. Assim, quando a mesma transformação é executada várias vezes, toda a informação existente na tabela de saída é removida. Desta forma, quando existem vários ficheiros (e.g., vários ficheiros com as leituras dos fios de prumo) que devem ser migrados em várias iterações, isto é, em cada iteração é migrada a informação de um ficheiro, só a informação do último é que ficará na tabela produzida. Torna-se então necessário dotar o *Ajax* de mecanismos que permitam efectuar uma migração incremental que, nesta tese, se define como um processo de migração no qual os registos só são introduzidos na tabela final se forem registos novos e são actualizados se forem registos alterados. Deste modo, é possível migrar várias vezes a mesma informação, ou informação que se encontra particionada pelas várias fontes de dados, sem perder a informação previamente migrada.

Note-se que, a tarefa de verificar se um registo novo já estava ou não na base de dados é independente dos outros registos produzidos e tem que ser feita registo a registo.

Deste modo, a solução encontrada passa por utilizar o operador *Map* para executar esta verificação, registo a registo. A escolha do operador *Map* deve-se ao facto de que este operador implementa uma transformação de um-para-muitos, que pode ser vista como um iterador nos registos de entrada. Assim, numa iteração registo a registo, implementada em *Java*, é possível verificar se cada um dos registos gerados já pertence ao conjunto de registos previamente armazenado nas tabelas de saída, de forma independente dos outros registos produzidos.

No exemplo da migração dos fios de prumo, apresentado no grafo da Figura 3.4, as tabelas *Fio de Prumo*, *Base de Coordinómetro*, *Leitura FP*, *Campanha* e *Resultado FP* são tabelas finais. Por exemplo, para gerar os tuplos da relação *Leitura FP*, que são carregados através de uma transformação implementada pelo operador *View*, é necessário acrescentar uma nova transformação implementada pelo operador *Map*. Nessa transformação, determina-se que os registos são actualizados por campanha e código de instrumento (*instrfixo*). Assim, considerando a geração do registo com campanha = "1" , instrfixo = "11", data = "10/03/1966" e rf = "9.2", se já existir um registo na tabela *LeituraFP* com o par (campanha, instrfixo) igual a ("1", "11"), então esse registo será actualizado pelo novo registo, independentemente dos valores dos restantes atributos (não chave). Na situação inversa, se o par (campanha, instrfixo) não tiver correspondência na tabela *LeituraFP*, então o resultado da transformação para este registo corresponde a uma inserção na tabela *LeituraFP*. No carregamento da tabela *Campanha*, não é necessário acrescentar nenhuma nova transformação, já que a última transformação no grafo de transformações já era implementada por um *Map*. Assim, basta determinar as condições de actualização dos registos nessa transformação.

<map-operator>:	create mapping <map-name> from <predicate> [<alias>] [let <let-clause>] [where <where-clause>] (<output-clause> [**increment by** <increment-atributes>]) +

Tabela 3.4: Sintaxe do operador *Map* com migração incremental

A Tabela 3.4 apresenta a sintaxe do operador *Map* com a inclusão da cláusula *increment by* que é opcional e permite definir as condições de actualização de cada registo.

Note-se que o operador *Map* tem como entrada uma única tabela especificada na cláusula *from*. Na cláusula *let* é possível atribuir expressões a um conjunto de variáveis internas do operador (e.g., função da biblioteca de funções, valor de um atributo da tabela de entrada). Na cláusula *where* especificam-se as condições de filtragem dos registos de entrada e, finalmente, na cláusula *output* definem-se os atributos das tabelas de saída. A cláusula *increment by* especifica os atributos que determinam se um registo deve ser actualizado ou inserido na tabela produzida. Uma vez que este operador efectua um mapeamento de um-para-muitos, podendo gerar vários tuplos em diferentes tabelas, é necessário especificar os atributos que controlam a actualização dos registos para as diferentes projecções que se pretende que sejam incrementais.

No código seguinte apresenta-se o exemplo de aplicação deste operador de forma incremental.

```
CREATE MAPPING LeituraFPIncr
FROM LeituraFP L
LET y = f(x) -- atribuição de variáveis
{ SELECT L.id, L.campanha, L.instrfixo, L.data, L.rf
KEY id
INCREMENT BY campanha, instrfixo
}
```

Nestes termos, a aplicação repetida desta transformação acumula os registos na tabela *LeituraFPIncr*, não apagando a informação previamente armazenada. Assim, considerando novamente a geração do registo com campanha = “1” , instrfixo = “11”, data = “10/03/1966” e rf = “9.2”, se já existir um registo na tabela *LeituraFPIncr* com o par (campanha, instrfixo) igual a (“1”, “11”), então esse registo será actualizado pelo novo registo, independentemente dos valores dos restantes atributos (não chave). Se os valores para o par (campanha, instrfixo) não existirem na tabela *LeituraFP* então, o novo registo será inserido na tabela *LeituraFP*.

3.4.3 Expressividade do operador *View*

Nesta Secção descrevem-se, em detalhe, as extensões da gramática *SQL* suportada pelo operador *View* que permitem aumentar a expressividade deste operador. Apresenta-se, nomeadamente, a introdução da distinção entre *union* e *union all* para união de registos, agregação de informação através das cláusulas *distinct*, *group-by* e *having* e, junção através de *outer join*.

3.4.3.1 *Union*

A cláusula **union** equivale à união de *SQL*, ou seja, permite unir dois conjuntos de dados. As restrições aplicadas à união no *Ajax* são as mesmas que existem no caso do *SQL*, pelo que os dois conjuntos têm obrigatoriamente o mesmo número de atributos e os mesmos domínios.

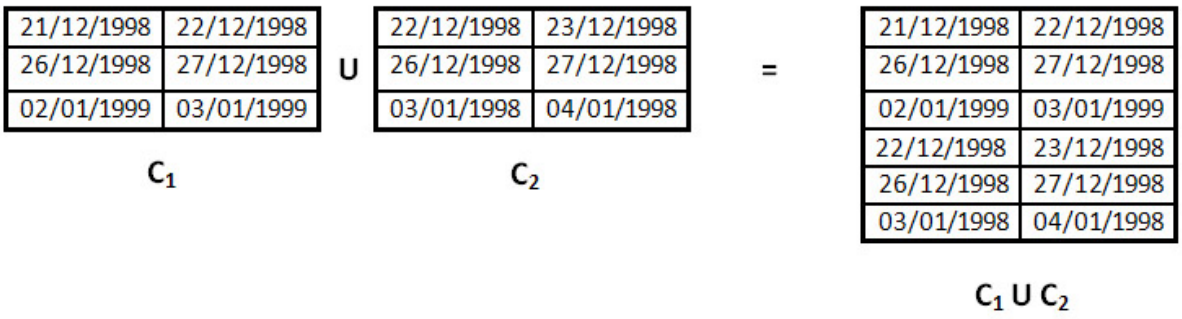

Figura 3.9: União no operador *View* do *Ajax*

A Figura 3.9 ilustra a aplicação da união na versão corrente do *Ajax*. Note-se que não existe qualquer eliminação de duplicados, pelo que o registo ("26/12/1998", "27/12/1998") aparece duas vezes na relação produzida pela união dos dois conjuntos de registos. Deste modo, a união no *Ajax* corresponde a um *union all* em *SQL*, já que não é feita a eliminação de duplicados.

É comum existirem múltiplas fontes de dados com informação sobre as mesmas entidades. Por exemplo, a definição das campanhas provém do ficheiro de leituras e do ficheiro de resultados. Para consolidar esta informação numa tabela de campanhas, é necessário unir os registos contidos nos dois ficheiros. Esta transformação pode ser im-

plementada através da cláusula *union* do operador *View* que, no grafo da Figura 3.4 produz a tabela *Camp* a partir das tabelas *TabLeit* e *TabRes*. Deste modo, procedeu-se à extensão do operador *View*, com a introdução da distinção entre *union* e *union all*. A Tabela 3.5 apresenta a sintaxe do operador *View* com a extensão da cláusula *union*.

O exemplo que se segue corresponde a uma especificação possível para o carregamento das campanhas, representado pelo operador *view* que produz a tabela *Camp* a partir das tabelas *TabLeit* e *TabRes*, conforme o grafo da Figura 3.4.

```
CREATE VIEW Camp
FROM TabLeit TL
{SELECT TL.Data as datainicio, TL.Data as datafim
}
UNION
FROM TabRes TR
{SELECT TL.Data as datainicio, TL.Data as datafim
}
```

Com a extensão implementada, a aplicação da cláusula *union* corresponde à união dos dois conjuntos, com eliminação de registos duplicados. Caso se pretenda a união completa dos dois conjuntos, independentemente da ocorrência de registos duplicados, especifica-se a união como *union all*. Note-se que a *union all* tem o mesmo comportamento que a *union* na versão inicial do operador, que se encontra ilustrada na Figura 3.9.

3.4.3.2 Agrupar informação com *distinct*, *group by* e *having*

Por vezes, é necessário extrair informação de tabelas cuja granularidade é diferente daquela que existe nos registos fonte. Considere-se o exemplo do ficheiro de entrada com a definição dos fios de prumo e bases de coordinómetro, denominado tabela de apoio. A Tabela 3.6 contém um excerto da tabela de apoio dos fios de prumo de uma determinada barragem.

Cada registo contém a identificação do fio de prumo a que pertence cada base de coordinómetro. Para criar uma tabela com os diferentes fios de prumo (tabela *Fio de*

<view-operator>:	create view <view-name> [**with exceptions**] from <from-elements> [(**left** | **full** | **right**) **outer join** <**from-element**> **on** <**on-clause**>] [where <where-clause>] [**group by** <group-by-clause> [**having** <having-clause>]] { select [**distinct**] <**select-clause-with-aggregate-function**> } [constraint <constraint-clause>] [**union** | **union all** from <from-elements> [(**left** | **full** | **right**) **outer join** <**from-element**> **on** <**on-clause**>] [where <where-clause>] [**group by** <group-by-clause> [**having** <having-clause>]] { select [**distinct**] <**select-clause-with-aggregate-function**> } [constraint <constraint-clause>]]* [key <key-clause>]

Tabela 3.5: Sintaxe do operador View com as extensões implementadas

Prumo no grafo da Figura 3.4), é necessário extrair os valores distintos para as colunas "identificacao" e bloco ("FPD1", "J-K" e "FPD2", "M-N" no exemplo da Tabela 3.6). Para que esta funcionalidade possa ser obtida, acrescentou-se a possibilidade de introduzir a cláusula *distinct* na projecção dos atributos de saída da transformação.

O exemplo que se segue corresponde à aplicação da cláusula *distinct* numa transformação que efectua a selecção dos fios de prumo a partir da definição dos fios de prumo e bases de coordinómetro.

```
CREATE VIEW FioPrumo
FROM TabApoio T
{SELECT DISTINCT T.identificacao as nome, T.bloco
}
```

Deste modo, com a introdução da cláusula *distinct*, são inseridos na tabela *FioPrumo* os valores distintos dos atributos *identificacao* e *bloco* da tabela *TabApoio*.

A aplicação da cláusula *distinct* agrupa, no mesmo conjunto, os registos de entrada

FPD1	J-K	11	1-1	879.25	0.000	6.27	11.16	8.28
FPD1	J-K	21-1	1	810.50	0.000	12.12	14.86	12.37
FPD1	J-K	31-1	1	831.10	0.000	9.84	12.79	10.95
FPD1	J-K	41-1	1	865.60	0.000	5.97	8.44	7.82
FPD1	J-K	51-1	1	791.00	0.000	2.20	0.59	0.38
FPD2	M-N	61-1	1	879.25	0.000	13.30	15.90	13.20
FPD2	M-N	71-1	1	810.50	0.000	8.20	11.05	8.71

Tabela 3.6: Exemplo de tabela de apoio para fios de prumo

que têm valores comuns para os atributos seleccionados, produzindo apenas um registo de saída para cada conjunto. Desta forma, as cinco bases de coordinómetro que têm o valor "FPD1" no campo *identificacao* e "J-K" no campo bloco, produzem apenas um registo com nome "FPD1" e bloco "J-K".

Agrupar registos através da cláusula *distinct* torna-se bastante limitado quando se pretende obter informação adicional sobre cada um dos conjuntos. Essa informação pode corresponder, por exemplo, ao número de elementos de cada grupo. No exemplo da criação dos diferentes fios de prumo, pretende-se obter o número de bases de coordinómetro que compõem cada fio de prumo. Em *SQL* é possível fazer este tipo de interrogação, recorrendo à cláusula *group-by* e às funções agregadoras[2].

Procedeu-se à extensão do operador *View*, com a introdução da cláusula *group-by* e das funções agregadoras *count* (contagem de valores), *avg* (média aritmética), *max* (valor máximo), *min* (valor mínimo) e *sum* (soma de valores). A Tabela 3.5 apresenta a sintaxe do operador *View* com a extensão das cláusulas *distinct*, *group by* e funções agregadoras.

O exemplo que se segue corresponde ao operador *View*, utilizado para efectuar a selecção dos fios de prumo e do respectivo número de bases de cordinómetro.

```
CREATE VIEW FioPrumo
FROM TabApoio T
GROUP BY T.identificacao
```

[2]Uma função agregadora opera num conjunto de valores e retorna um único valor escalar. O *SQL* disponibiliza as seguintes funções agregadoras: *COUNT* conta o número de elementos (os elementos podem pertencer a qualquer domínio), *AVG* efectua a média aritmética de um conjunto de valores numéricos, *MAX* devolve o valor máximo de um conjunto de valores numéricos, *MIN* devolve o valor mínimo de um conjunto de valores numéricos, *SUM* efectua a soma de um conjunto de valores numéricos.

```
{SELECT T.identificacao as nome, T.bloco, COUNT(*) AS numBases
}
```

Com a possibilidade de agrupar registos através da cláusula *group-by*, a filtragem de informação com a cláusula *where* pode ser insuficiente. Na cláusula *where*, os registos são filtrados de forma isolada, independentemente do grupo a que pertencem, não sendo possível testar valores retornados por funções agregadoras. A cláusula *having* de *SQL* permite filtrar os grupos gerados por um *group-by*, através de uma expressão que envolva funções agregadoras como, por exemplo, `having count(*) > 1`. Deste modo, procedeu-se à extensão do operador *View* com a inclusão da cláusula *having*, conforme a sintaxe apresentada na Tabela 3.5.

De facto, no sistema SIOBE podiam ser registadas duas leituras para o mesmo instrumento no mesmo dia. A base de dados do *gestBarragens* não permite a existência de várias leituras do mesmo instrumento no mesmo dia. Usando a cláusula *having* define-se que apenas os grupos com um único registo (sem duplicados) sejam migrados para a tabela de leituras. O código que se segue implementa essa transformação.

```
CREATE VIEW LeituraFP
FROM TabLeit T, Camp C, BaseCoordinometro B
WHERE T.data = C.datainicio AND
T.codigo = B.codigo
GROUP BY C.id, B.id, T.data
HAVING COUNT(*) = 1
{SELECT MIN(T.id), C.id as campanha, B.id as instrfixo, T.data, AVG(T.rf)
}
```

3.4.3.3 *Outer join*

A junção de tabelas em *SQL*, com recurso a *outer joins* [LZ05], permite seleccionar registos que não tenham correspondência noutra tabela. Todos os atributos provenientes da tabela que não têm correspondência assumem o valor *null*. O *outer join* pode ser de

três tipos: *left outer join*[3], *right outer join*[4] e *full outer join*[5]. Este tipo de transformação assume crucial importância na migração de dados, já que não se deve perder informação em nenhuma das transformações aplicadas.

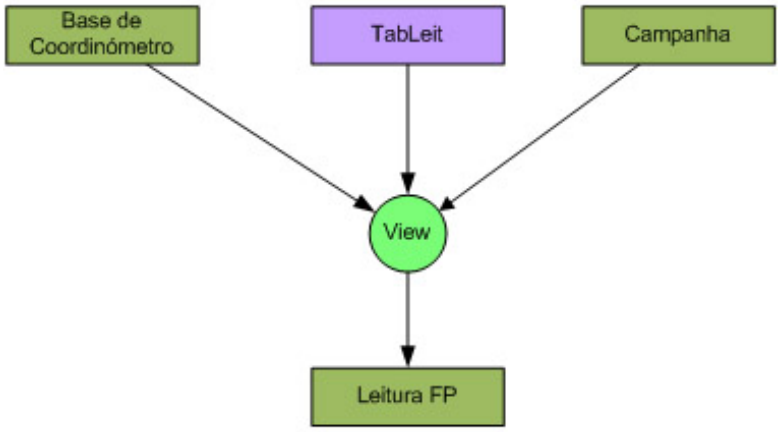

Figura 3.10: *View* com junção de tabelas

A Figura 3.10 ilustra uma transformação que junta informação proveniente de várias tabelas. Neste caso a junção é feita entre as bases de coordinómetro, as campanhas e os registos de leituras armazenados na tabela *TabLeit*.

Na aplicação de um *join* com a cláusula *where* do operador *View*, na forma `BaseCoordinometro.codigo = TabLeit.codigo AND Campanha.datainicio = TabLeit.data`, os registos de leituras para os quais o código da base de coordinómetro não existe na tabela *BaseCoordinometro* não seriam seleccionados. Deste modo, a sintaxe do operador *View* foi extendida de forma a permitir efectuar *outer joins*. A Tabela 3.5 apresenta a sintaxe do operador *View* com a extensão do *outer join*.

[3]Sendo A e B duas relações, em *A left outer join B*, os registos da relação A que não têm correspondência na relação B também fazem parte do resultado

[4]Sendo A e B duas relações, em *A right outer join B*, os registos da relação B que não têm correspondência na relação A também fazem parte do resultado

[5]Sendo A e B duas relações, em *A full outer join B*, os registos da relação A e da relação B que não têm correspondência nas relações B e A, respectivamente, também fazem parte do resultado

Capítulo 4

Migração do *gestBarragens* passo a passo

Neste Capítulo descreve-se, passo a passo, o desenvolvimento do processo de migração dos dados legados para a base de dados do sistema *gestBarragens*. Em primeiro lugar, na Secção 4.1, descreve-se o estado dos sistemas de informação legados. Na Secção 4.2, apresenta-se o sistema *gestBarragens* e a base de dados alvo. Na Secção 4.3, ilustra-se a estratégia de migração adoptada neste projecto. De seguida, na Secção 4.4 apresenta-se, em detalhe, a execução e correcção de dados na migração do sistema de observações e, finalmente, na Secção 4.5 apresenta-se a aplicação *infoLegada2Gb*.

4.1 Sistemas de Informação Legados

Dada a diversidade das actividades envolvidas no controlo de segurança das barragens, o LNEC disponibiliza vários Sistemas de Informação Legados. Estes sistemas têm como principais objectivos, por um lado, constituir um arquivo da informação relativa ao controlo de segurança das barragens, conforme legislado pelo Regulamento de Segurança em Barragens [RdSdB90] e, por outro lado, disponibilizar mecanismos de análise que permitam garantir o controlo de segurança das barragens.

Os sistemas legados existentes são peças de *Software* isoladas, isto é, programas que não comunicam entre si. Por seu turno, os arquivos de informação estão armazenados em

ficheiros que dependem de cada aplicação legada e apresentam formatos heterogéneos. Existe informação replicada, possivelmente inconsistente, entre as várias fontes de dados.

Nas secções seguintes descrevem-se, o sistema de gestão de observações - SIOBE e o sistema de gestão de observações geodésicas. É de referir que existem no LNEC outros sistemas simples, como por exemplo o sistema de gestão de modelos matemáticos e físicos apresentado no Apêndice D.3, que têm objectivos distintos dos sistemas de gestão de observações. Note-se que, no âmbito do projecto *gestBarragens*, só é necessário migrar a informação armazenada no SIOBE e nos sistemas de gestão de observações geodésicas.

4.1.1 Sistema de Gestão de Observações - SIOBE

Por iniciativa do LNEC foi desenvolvido, há mais de duas décadas, um sistema de gestão de observações designado por SIOBE (Sistema de Informação para Observação de Barragens de Betão). O SIOBE tem sido utilizado pelo LNEC e por alguns donos de obra, nomeadamente a EDP-Produção EM e o INAG. A sua plataforma de funcionamento é um computador com MS-DOS.

O SIOBE é um sistema que suporta o registo e gestão de informação produzida por sistemas de observação colocados em pontos estratégicos das barragens e permite explorar os dados recolhidos através da produção de tabelas e de gráficos.

Ao longo dos anos, o SIOBE foi objecto de diversos desenvolvimentos, sem que, no entanto, se tenha alterado a estrutura tecnológica que lhe está subjacente. De facto, as aplicações são desenvolvidas em Fortran e a informação é mantida em ficheiros binários e ASCII com modelos de representação de baixo nível, que não incluem qualquer restrição de integridade.

O Apêndice D.1 descreve, com maior detalhe, os componentes do sistema SIOBE e os problemas que este apresenta. Dos problemas apresentados pelo SIOBE, destaca-se a baixa qualidade dos dados armazenados, que se deve a uma estrutura de armazenamento rudimentar em ficheiros ASCCI e à falta de validação adequada nas aplicações de introdução de dados no sistema e, várias dificuldades na extensão do sistema devido à tecnologia de desenvolvimento em *Fortran* e à estrutura de armazenamento de dados.

4.1.1.1 Arquivo de informação

Os dados do SIOBE são armazenados num conjunto de ficheiros ASCII e ficheiros binários, que contêm dados relativos ao período de tempo que vai desde a década de 40 até à actualidade. A Figura 4.1 apresenta o esquema de armazenamento para a informação dos 21 tipos de instrumentos suportados pelo SIOBE. Existem três classes de ficheiros distintas. As tabelas de apoio contêm a definição dos instrumentos. Os ficheiros de leituras armazenam as leituras realizadas em cada instrumento ao longo do tempo. Os ficheiros de resultados contêm as grandezas calculadas a partir das leituras realizadas em cada instrumento.

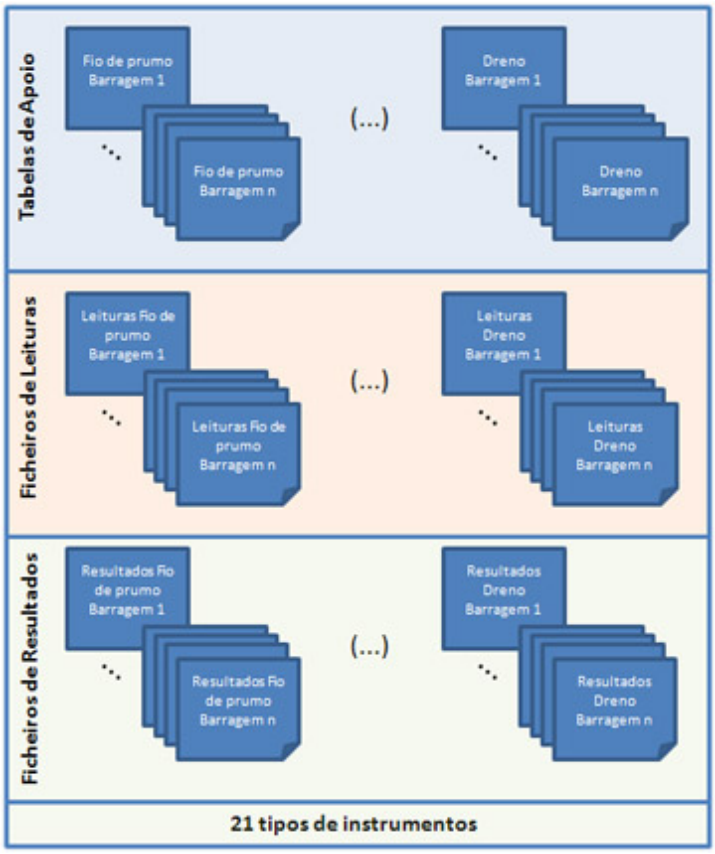

Figura 4.1: Arquivo de informação do SIOBE

Em cada uma das três classes de ficheiros existe um tipo de ficheiro diferente para cada um dos 21 tipos de instrumentos suportados pelo SIOBE. Assim, por exemplo, existe um tipo de ficheiro para a tabela de apoio que armazena a definição dos instrumentos do tipo *fio de prumo* e outro tipo de ficheiro para a tabela de apoio para os

instrumentos do tipo *dreno*. Da mesma forma, os ficheiros de leituras e de resultados também dependem do tipo de instrumento. Por outro lado, no SIOBE a informação está organizada por barragem, pelo que cada ficheiro contém apenas informação da barragem que lhe corresponde. Assim, seguindo o exemplo das tabelas de apoio para os fios de prumo, existe um ficheiro com a definição dos fios de prumo da barragem de Castelo do Bode e outro ficheiro distinto, mas com a mesma estrutura, com a definição dos fios de prumo da barragem do Alqueva.

31 6 14 3 0.0 ** 321 1 **
FPD1 J-K 11 1-1 879.25 0.000 6.2711.16 8.28 9.27 8.33 8.06 3.88 6.25 6.20 6.81 5.56 5.49
FPD1 J-K 21-1 1 810.50 0.00012.1214.8612.37 5.13 3.75 3.63 5.41 7.82 7.74 7.80 6.56 6.49
FPD1 J-K 31-1 1 831.10 0.000 9.8412.7910.95 8.78 7.40 7.32 4.20 6.52 6.45 7.05 5.76 5.68

Tabela 4.1: Exemplo de tabela de apoio

A Tabela 4.1 apresenta um excerto de uma tabela de apoio com informação dos fios de prumo de uma determinada barragem. As tabelas de apoio são compostas por três tipos de informação:

- primeira linha do ficheiro que contém informação utilizada para o dimensionamento da tabela de apoio isto é, metadados sobre a informação contida na tabela de apoio. No exemplo da Tabela 4.1 a primeira linha do ficheiro indica que existem 31 instrumentos, 6 colunas são números inteiros, 14 colunas são números reais, não existe validação de resultados (código 3) e o nível da base usada no modelo estatístico é zero.

- cada linha do segundo bloco de informação corresponde a um instrumento, com a sua identificação, valores necessários para o cálculo dos resultados relativos a esse tipo de instrumento e informação para a validação de dados (limite inferior e superior para cada leitura desse instrumento).

- bloco do ficheiro, que é opcional e contém os coeficientes estabelecidos pelos métodos estatísticos que permitem fazer a validação dos resultados. O exemplo apresentado na Tabela 4.1 não inclui este bloco do ficheiro.

Cada ficheiro de leituras contém as leituras de vários instrumentos de um determinado tipo, realizadas no âmbito de uma ou mais campanhas de observação. Cada bloco de dados do ficheiro corresponde à informação recolhida numa campanha e é delimitado pelo nível da albufeira. A Tabela 4.2 apresenta um excerto de um ficheiro de leituras de fios de prumo. Cada linha do ficheiro de leituras corresponde a uma leitura realizada num instrumento, numa determinada data. Assim, cada linha contém a identificação do instrumento, a data da leitura, a cota da albufeira e os valores medidos, que dependem do tipo de instrumento.

11321 1 311966 310 879.0 1 8.16 9.68 3.98 7.05 0
11321 1 311966 310 879.0 210.86 5.23 5.52 8.01 0
11321 1 311966 310 879.0 3 9.23 8.95 4.26 7.23 0

Tabela 4.2: Exemplo de ficheiro de leituras

Os ficheiros de resultados armazenados pelo sistema SIOBE são ficheiros binários. Cada ficheiro contém informação recolhida numa ou mais campanhas para um determinado tipo de instrumento numa barragem. O SIOBE disponibiliza um mecanismo de geração de ficheiros ASCII a partir dos ficheiros binários. Deste modo, foi gerado o conjunto dos ficheiros ASCII que devem ser migrados para o novo sistema.

19640115 0.000000E+00 -2.099993
19640115 -2.000046E-01 -2.099993
19640115 -2.000046E-01 -2.099998

Tabela 4.3: Exemplo de ficheiro de resultados

A Tabela 4.3 apresenta um excerto de um ficheiro de resultados relativo aos fios de prumo. Cada linha do ficheiro corresponde aos resultados de cada instrumento numa determinada data, para a barragem a que corresponde o ficheiro. No entanto, a identificação do instrumento a que corresponde cada resultado não é explicita. É necessário inferir a identificação do instrumento a partir da ordenação do ficheiro de resultados. Assim, o primeiro resultado numa determinada data corresponde ao resultado do primeiro instrumento da tabela de apoio, o segundo resultado corresponde ao segundo instrumento, e assim sucessivamente.

4.1.1.2 Resumo da informação armazenada no SIOBE

A Tabela D.1 do Apêndice D.1.3 apresenta um quadro que resume a informação armazenada pelo sistema SIOBE e que deve ser migrada para a base de dados alvo do sistema *gestBarragens*.

Por exemplo, na linha relativa ao Fio de Prumo, indica-se que têm que ser migradas 40 tabelas de apoio, que correspondem a 40 barragens e têm um total de 493 instrumentos definidos, 38 ficheiros de leituras com um total de 276.676 linhas e 39 ficheiros de resultados com um total de 314.124 linhas.

Em resumo, a migração do sistema envolve o carregamento de 1558 ficheiros ASCII que totalizam 16.533.247 registos. Note-se que o arquivo de informação do SIOBE também contém outro tipo de informação, como a definição da estrutura de cada barragem ou a definição dos gráficos de exploração de dados, No entanto, esta informação não deve ser migrada para a base de dados do novo sistema. No Apêndice D.1.2 descrevem-se todos os tipos de ficheiros armazenados pelo SIOBE.

4.1.2 Sistema de Gestão de Observações Geodésicas

O núcleo de geodesia aplicada do LNEC é responsável pela recolha de um conjunto de observações que recorrem à utilização de métodos e instrumentos geodésicos específicos. Estas observações permitem calcular deslocamentos de vários pontos da barragem ao longo do tempo. Os deslocamentos calculados são posteriormente registados no sistema SIOBE. No entanto, o sistema SIOBE não fornece quaisquer mecanismos que permitam gerir a informação relacionada com as observações geodésicas, como a definição das redes, ou o registo das leituras em cada campanha. Os cálculos dos deslocamentos a partir das observações geodésicas são realizados por aplicações isoladas. O Apêndice D.2 apresenta algumas aplicações envolvidas nas Observações Geodésicas.

4.1.2.1 Estrutura de armazenamento

A informação relativa às observações geodésicas encontra-se armazenada em ficheiros *Excel*, que incluem as observações realizadas desde a década de 60 até à actualidade.

A Figura 4.2 apresenta o esquema de armazenamento dos ficheiros *Excel*. Existe um ficheiro *Excel* para cada barragem. Por exemplo, o ficheiro *Alqueva.xls* contém apenas informação acerca da Barragem do Alqueva. Cada ficheiro *Excel* é composto por um conjunto de folhas, com nome e estrutura predefinidos.

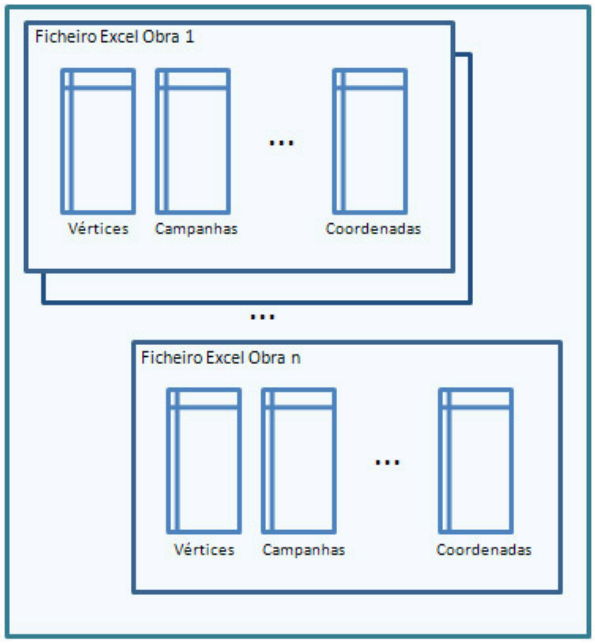

Figura 4.2: Estrutura de armazenamento das observações geodésicas

No Apêndice D.2.4 detalha-se a estrutura de cada uma das folhas *Excel* que compõem o arquivo de informação das observações geodésicas. A Tabela D.2 deste Apêndice apresenta um resumo da estrutura de cada folha *Excel*. Por exemplo, a folha *Camp* com a definição de campanhas é composta por seis atributos (CampNum, CampNome, Data, CotaAgua, Outras, Comments).

4.2 O sistema gestBarragens

O sistema *gestBarragens* é um sistema de informação que permite gerir toda a informação relacionada com o controlo de segurança de barragens. No âmbito desta tese, interessa referir que o *gestBarragens* inclui um módulo de gestão de observações, que substitui o sistema SIOBE e, um módulo de gestão de observações geodésicas que gere toda a informação relacionada com as observações geodésicas, inicialmente armazenada em

ficheiros *Excel*.

O sistema *gestBarragens* inclui todas as funcionalidades do sistema SIOBE, que também podem ser aplicadas aos dados resultantes das observações geodésicas. Sendo uma evolução dos sistemas legados, o novo sistema inclui novas funcionalidades como a criação automática de relatórios de exploração de dados, ou a apresentação de gráficos nos modelos de representação das barragens, através de um sistema de informação geográfico.

O desenho e implementação da base de dados do sistema *gestBarragens* resultou de uma fase anterior ao processo de migração de dados. A base de dados resultou do levantamento de requisitos do projecto *gestBarragens*, desenvolvido, em conjunto, pelas equipas do INESC-ID, do LNEC e da EDP-Produção EM [SGB+04]. O SGBD que armazena a base de dados do *gestBarragens* é o *Oracle*.

A Figura 4.3 apresenta um excerto do modelo ER, usando a nomenclatura definida em [SKS02], que representa as entidades envolvidas nas observações. Este modelo é simplificado na medida em que só inclui os tipos de instrumentos fios de prumo e dreno e não representa os atributos das entidades, nem a listagem com as restrições de integridade que não podem ser expressas pelo modelo, pretendendo apenas ilustrar as relações entre as diferentes entidades.

As leituras são realizadas num único instrumento no âmbito de uma campanha. Note-se que uma leitura de um fio de prumo é uma entidade independente de uma leitura de um dreno. Os resultados correspondem a grandezas calculadas a partir de uma leitura e estão associados ao instrumento a que corresponde o resultado e a uma determinada campanha. Adicionalmente, cada resultado pode ter associada a leitura que lhe deu origem.

É importante notar que as entidades que armazenam a informação dos drenos (*Dreno*, *Leitura Dreno* e *Resultado Dreno*) não dependem das entidades que armazenam informação dos fios de prumo (*Fio de prumo*, *Base de Coordinómetro*, *Leitura FP* e *Resultado FP*), pelo que é possível considerar este modelo como a composição de vários modelos simples, em que cada modelo corresponde a um tipo de instrumento. Na Figura E.1 do Apêndice E apresenta-se, com maior detalhe, o modelo ER que modela as entidades

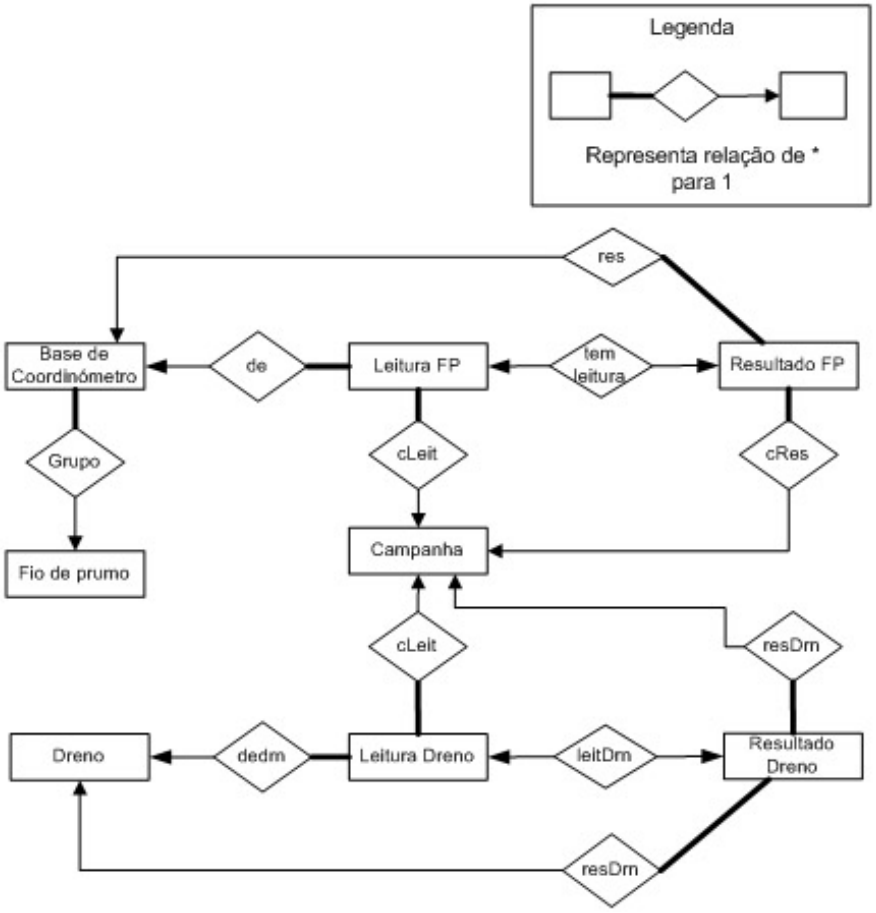

Figura 4.3: Modelo ER para fios de prumo e drenos

relacionadas com as observações.

A Figura 4.4 apresenta um excerto do modelo ER, usando a nomenclatura definida em [SKS02], para as observações geodésicas.

Cada rede geodésica é composta por um conjunto de giros e por um conjunto de ligações, que correspondem à associação entre dois vértices. Cada giro é composto por um conjunto de ligações e tem associados dois vértices que assumem os papéis de *vértice-estação* e de *vértice-origem* do giro. Cada rede tem também associado um conjunto de vértices considerados fixos. A observação das redes geodésicas é feita através do registo de leituras associadas a uma ligação da rede, numa determinada campanha. As leituras são realizadas com o auxilio de um ou mais instrumentos de leitura. Em cada campanha de leituras, são calculados os resultados de cada vértice com base numa campanha de referência. A Figura E.2 do Apêndice E apresenta, o modelo ER completo que modela

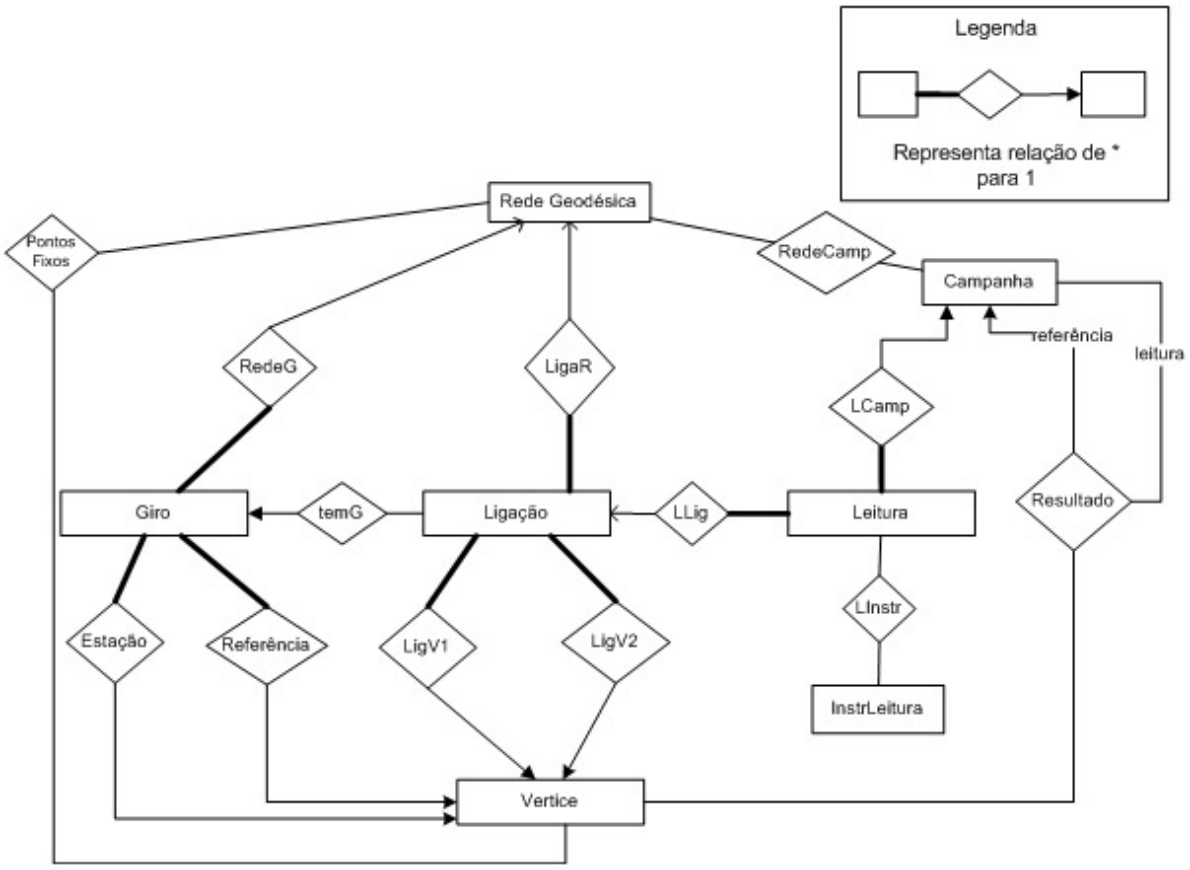

Figura 4.4: Modelo ER das observações geodésicas

as entidades relacionadas com as observações geodésicas.

É importante verificar que os modelos das observações e das observações geodésicas, representados nas Figuras 4.3 e 4.4 são completamente disjuntos, não existindo entidades partilhadas pelas observações e pelas observações geodésicas.

4.3 Desenho e implementação do grafo de transformações

A ideia básica inerente à estratégia de migração consiste em decompor o processo de migração de dados em transformações conceptualmente mais simples, que devem ser aplicadas numa ordem específica de forma a compor o processo de migração.

No caso da migração de dados para o sistema *gestBarragens*, conforme se referiu na Secção 4.2, a base de dados alvo pode ser decomposta em vários conjuntos de entidades que são independentes entre si. Deste modo, o processo de migração pode ser decomposto

em vários sub-processos mais simples. Nas observações, cada sub-processo corresponde à migração da informação relativa a um tipo de instrumento, como por exemplo, a definição de fios de prumo, as leituras dos fios de prumo e os resultados dos fios de prumo. Por outro lado, a migração da informação relativa às observações geodésicas corresponde a um processo de migração distinto da migração das observações. Assim, a migração de dados para a base de dados do sistema *gestbarragens* inclui um processo de migração das observações, composto por vários sub-processos (um para cada um dos 21 tipos de instrumentos) e um processo de migração para as observações geodésicas, conforme ilustra o esquema da Figura 4.5.

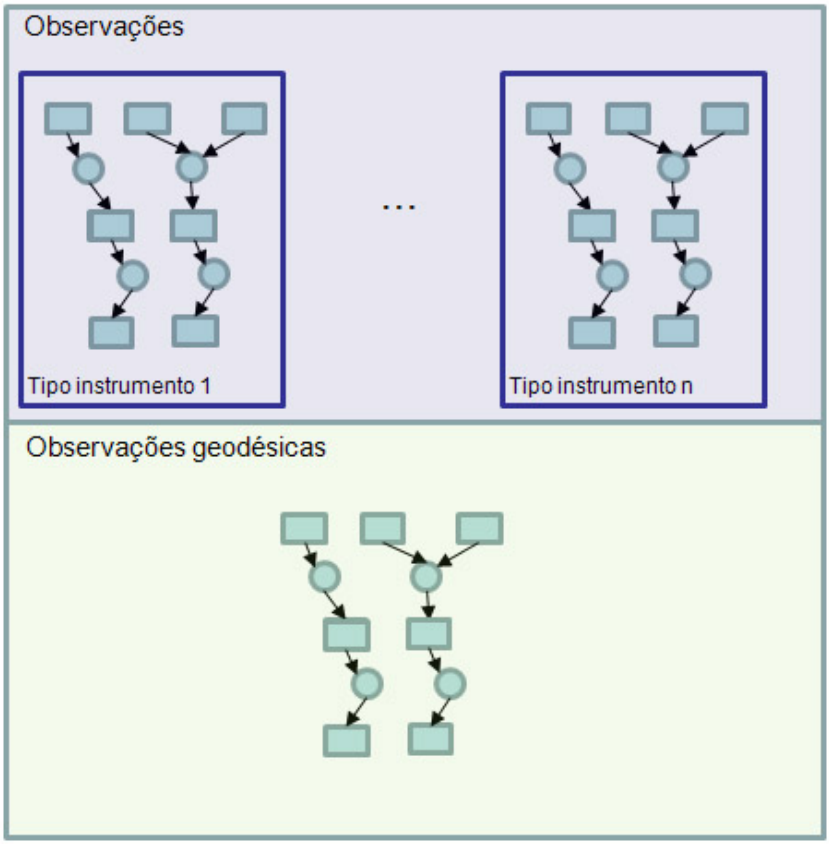

Figura 4.5: Decomposição do processo de migração do *gestBarragens*

Em ambos os processos de migração, a estratégia consiste na aplicação de uma sequência de transformações de dados. Em primeiro lugar, é gerado um novo identificador para cada um dos registos de entrada, através de uma transformação implementada pelo operador *Map*. Deste modo, é possível aplicar os mecanismos de *debugging*

do *Ajax*, sendo possível "navegar" no grafo de transformações. O segundo passo nas transformações consiste na extracção da informação relevante a partir de cada uma das fontes de dados. Este passo consiste em "partir" e normalizar a informação em vários atributos com domínios específicos. Por exemplo, a extracção de uma data é feita com recurso a uma função que a valida e, se a data não for válida gera uma excepção. Esta transformação é implementada pelo operador *Map*.

Em seguida, após ser feita a extracção e normalização dos dados, implementam-se, com recurso ao operador *view*, as transformações que cruzam informação entre as várias relações. Finalmente, o carregamento dos dados para as tabelas finais tem que ser feito através de uma transformação implementada pelo operador *Map*, de forma garantir a migração incremental dos dados, do modo em que foi referida no Capítulo 3.

A Figura 4.6 apresenta o excerto do grafo de transformações para a informação relacionada com os instrumentos do tipo dreno, que segue a estratégia enunciada anteriormente. Note-se que existem várias dependências no fluxo de transformações. Em primeiro lugar, o carregamento dos instrumentos para a tabela *Dreno* só depende do ficheiro de entrada *MgrApoioDrn*. Em segundo lugar, para realizar o carregamento das leituras para a tabela final *DrenoLeitura*, é necessário cruzar os dados de entrada com os drenos carregados na tabela *Dreno*, pelo que o carregamento das leituras depende do carregamento dos instrumentos. Finalmente, note-se que o carregamento dos resultados também depende dos instrumentos e das leituras, pelo que se pode definir a seguinte ordem de dependências: instrumentos > leituras > resultados.

No Apêndice F descreve-se com maior detalhe os grafos de transformações para os fios de prumo e para os drenos. O Apêndice G apresenta o grafo de transformações para as observações geodésicas.

4.4 Execução/correcção do grafo de transformações

O volume de dados a migrar é bastante elevado, pelo que é de esperar que as transformações rejeitem muitos registos, o que significa que o critério implementado não consegue suportá-los. A detecção dos registos que correspondem a excepções permite

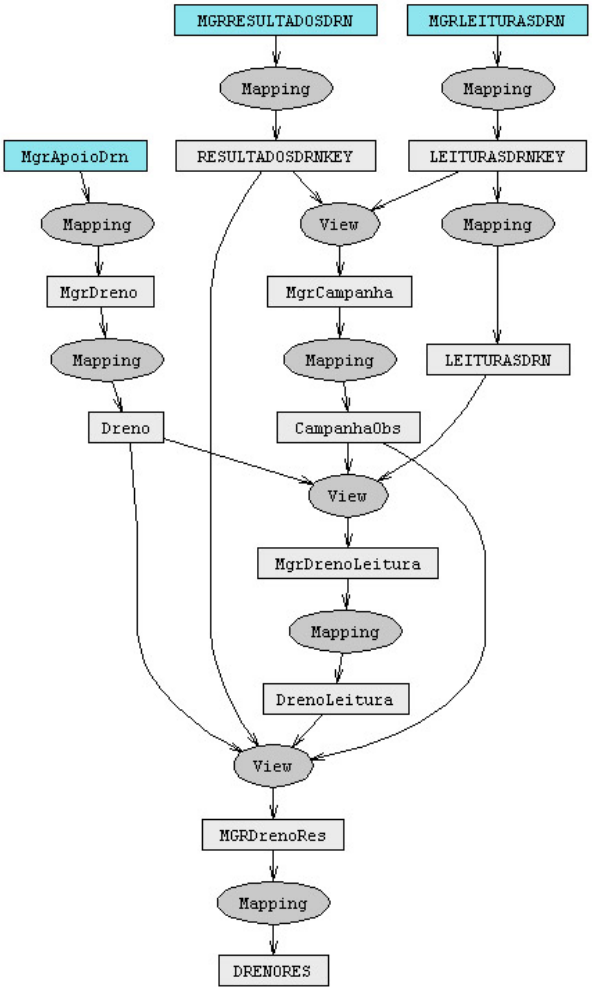

Figura 4.6: Grafo de transformações para os drenos

desenvolver um processo iterativo de refinamento da lógica das transformações. Após a execução de cada transformação, o seu resultado é analisado com vista ao refinamento da definição lógica da transformação.

Depois de se ter refinado a lógica das transformações, continuam a existir registos de excepções que correspondem a erros nos dados de entrada, os quais só podem ser corrigidos pelo utilizador, de forma manual. O *Explicador* do *Ajax* suporta a pesquisa da proveniência dos registos errados, permitindo ao utilizador corrigi-los através da edição, remoção ou inserção de novos registos em qualquer relação do grafo de transformações.

A Figura 4.7 apresenta a interface que o *Ajax* disponibiliza para apresentar o grafo de transformações. Cada rectângulo representa uma relação e as elipses representam os operadores do *Ajax*. As associações entre os operadores e as relações são estabelecidas

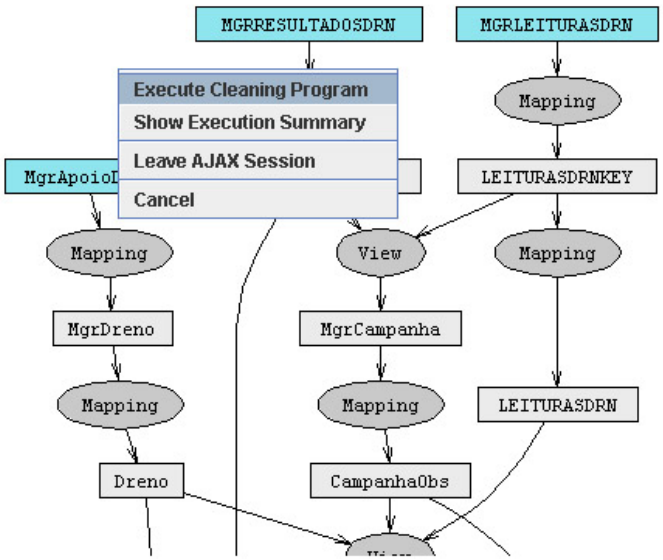

Figura 4.7: Execução do grafo de transformações para os drenos

por setas, cujo sentido determina o fluxo de propagação dos dados. É possível executar todo o programa, através da opção *Execute Cleaning Program* ilustrada na Figura 4.7, ou executar uma transformação de cada vez.

A Figura 4.8 apresenta um excerto do grafo após a execução das duas transformações que permitem carregar a tabela *Dreno*. A primeira transformação fica a verde, o que significa que não houve geração de excepções. Na segunda transformação, existem registos de excepções, o que é indicado pelo facto da transformação ficar a vermelho.

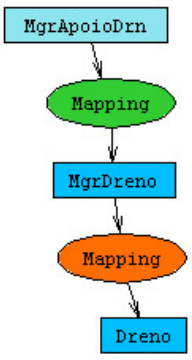

Figura 4.8: Transformação com geração de excepções no operador *Map*

Para além de informar de forma visual se a transformação gerou excepções, pode

analisar-se o sumário de execução de cada transformação, que reporta o número de registos produzidos, o número de excepções geradas e o tempo de execução. A Figura 4.9 apresenta o sumário de execução para o *Map* que gera a tabela *Dreno*. Note-se que a partir de 271 registos de entrada, foram produzidos 269 registos regulares e dois registos de excepções.

Summary of execution for transformation **Dreno**:

Exceptions Number	**2**
Execution Time	**0h 0m 0s 844ms**
Input Data Flows:	
MgrDreno	**271** tuples
Output Data Flows:	
Dreno	**269** tuples

Figura 4.9: Sumário de execução da transformação *Dreno*

Os mecanismos de *debugging* do *Ajax* permitem analisar os registos produzidos e as excepções geradas, de forma interactiva no grafo de transformações, recorrendo ao *Explicador* do *Ajax*. O utilizador pode obter a proveniência de cada um dos tuplos, conforme ilustra a Figura 4.10. Neste caso, o operador *Map* cuja execução tem como objectivo carregar a tabela *Dreno* gerou dois registos de excepções. Cada registo de excepção tem um campo com a descrição da excepção gerada, denominado *exceptionInfo*. No caso do operador *Map*, as excepções correspondem a excepções *Java* lançadas na invocação a funções externas. Assim, é necessário conhecer a lógica das funções e o significado das excepções lançadas por cada função *Java*. Por exemplo, no segundo registo de excepção indica-se que houve um erro na colocação do caracter '.' na invocação da função *getDouble20262*. Esta função tenta extrair um número com duas casas decimais, entre as posições 20 e 26 de uma cadeia de caracteres.

O utilizador pode apagar (*Delete*), actualizar (*Update*) ou analisar a proveniência do registo de excepção (*Explain*). No caso da migração de dados, o objectivo consiste em analisar os registos de entrada que provocam as excepções e corrigi-los de forma a que possam ser correctamente migrados.

Quando o utilizador selecciona a opção *Explain*, para analisar a proveniência do

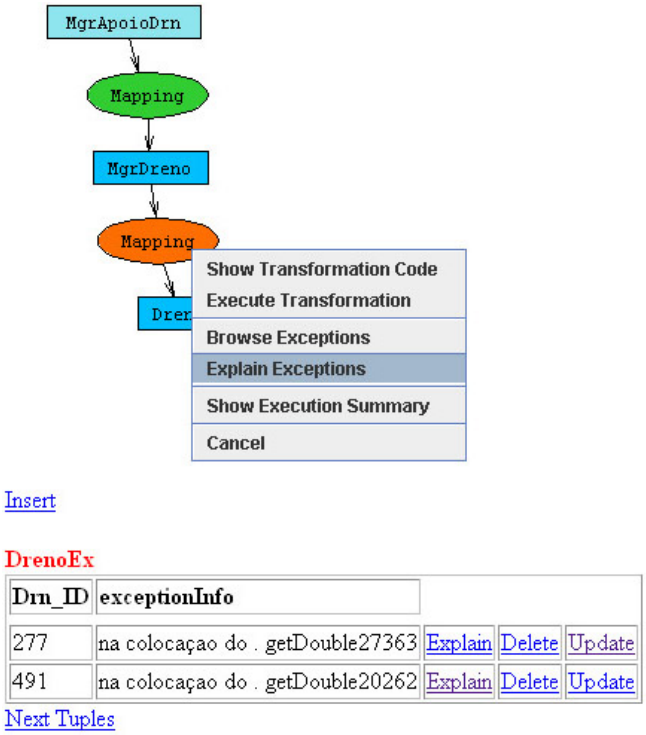

Figura 4.10: Análise dos registos de excepções geradas pelo operador *Map*

registo, obtém uma nova interface com o registo que produziu o registo analisado (ver Figura 4.11). Neste caso, como se trata de uma transformação implementada por um operador *Map* (um-para-muitos), só um registo de entrada é que está na origem de cada registo produzido. Na interface apresentada na Figura 4.11 o utilizador pode corrigir ou apagar o registo que está na origem do registo de excepção e, posteriormente, re-executar a transformação com os registos de entrada corrigidos. Como se referiu anteriormente, tendo em conta a mensagem de erro apresentada para este registo de excepção, um utilizador que conheça o domínio dos dados verifica que entre as posições 20 e 26 está a cadeia de caracteres "84.850", que corresponde à cota da albufeira e tem três casas decimais em vez duas. O utilizador pode corrigir este valor para "848.50" e voltar a executar a transformação.

A Figura 4.13 apresenta um exemplo de geração de excepções no operador *View*, que corresponde à transformação ilustrada na Figura 4.12 que cruza informação proveniente das tabelas *Dreno*, *CampanhaObs* e *LeiturasDrn* para produzir a tabela *MgrDrenoLeitura*. Analisando o valor do atributo de informação da excepção (*exception-*

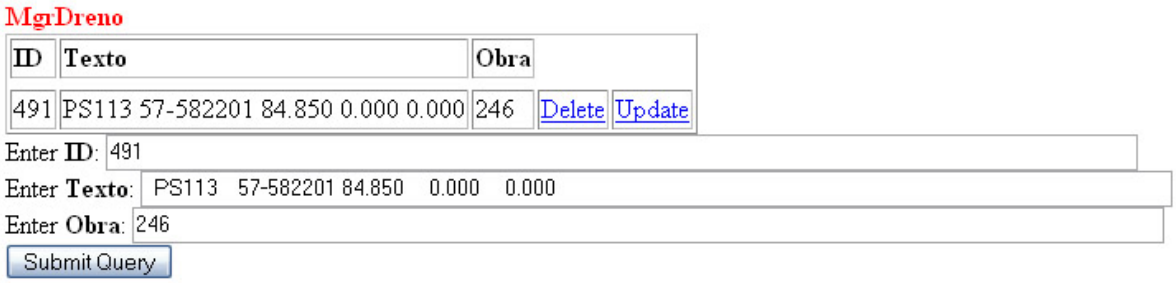

Figura 4.11: correcção de excepções no operador *Map*

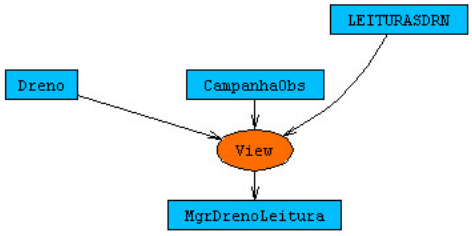

Figura 4.12: Geração de excepções no operador *View*

Info) é possível verificar que as excepções são geradas porque os registos produzidos violam a restrição de integridade de *not null* no atributo *Drn* da tabela de saída da transformação. Esta excepção significa que foram registadas leituras em drenos que não existem.

Tal como acontece no operador *Map*, para cada um dos registos de saída ou de excepções, também é possível analisar os registos que lhe deram origem. No caso do operador *View*, como implementa uma transformação de muitos-para-muitos, podem existir vários registos na origem de cada registo de saída ou de excepção. A Figura 4.14 apresenta a interface para corrigir os registos que estão na origem de um dos registos de excepções. Apesar da transformação ter três tabelas de entrada, só é possível corrigir os registos nas tabelas *CampanhaObs* e *LeiturasDrn*, já que nenhum registo da tabela *Dreno* deu origem ao registo de excepção (valor *null* na projecção do atributo *Drn*).

Este mecanismo iterativo de análise e correcção de excepções, permite corrigir facilmente os registos que não podem ser migrados. No entanto, essa correcção é realizada registo a registo, isto é, para corrigir cada uma das excepções é necessário determinar a sua proveniência, analisar o erro e corrigir o registo (no caso do operador *Map*) ou

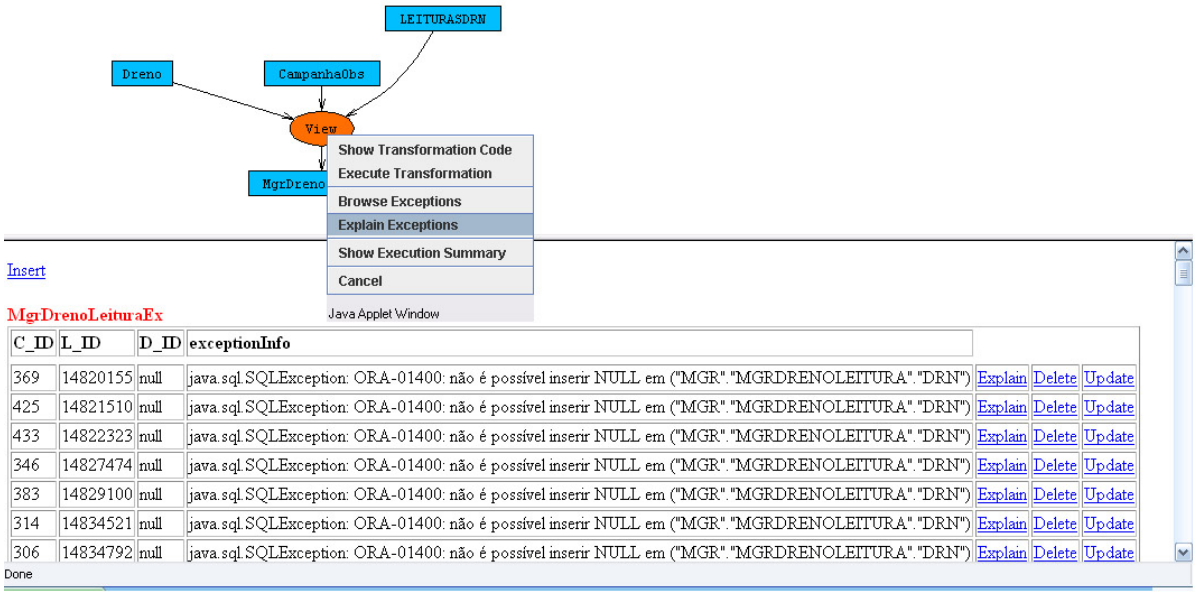

Figura 4.13: Análise de excepções geradas pelo operador *View*

registos (no caso do operador *View*), que estiveram na origem da excepção. De facto, quando o número de excepções gerado é elevado, este mecanismo de análise e correcção registo a registo não é viável. Por exemplo, na Figura 4.15 apresenta-se o sumário de execução do operador *View* para gerar a tabela *MgrDrenoLeitura*. Neste caso, foram gerados 15686 registos de excepção que, desta forma, exigem um elevado esforço manual para serem corrigidos. Para facilitar a correcção de registos, desenvolveu-se uma aplicação à medida, denominada *infoLegada2Gb* que inclui um módulo de classificação de excepções baseado nos mecanismos de proveniência de dados disponibilizados pelos operadores do *Ajax*

Figura 4.14: Correcção de excepções no operador *View*

Summary of execution for transformation **MgrDrenoLeitura**:

Exceptions Number	**15686**
Execution Time	**0**h **5**m **14**s **94**ms
Input Data Flows:	
Dreno	**209** tuples
LEITURASDRN	**68563** tuples
CampanhaObs	**253** tuples
Output Data Flows:	
MgrDrenoLeitura	**52877** tuples

Figura 4.15: Sumário de execução da transformação *MgrDrenoLeitura*

4.5 Aplicação *infLegada2Gb*

A aplicação *infoLegada2Gb* é uma aplicação desenvolvida em *Java* à medida do projecto de migração para o sistema *gestBarragens*. Esta aplicação tem como principal objectivo ultrapassar duas dificuldades. Em primeiro lugar, no *Ajax* a especificação dos fluxos de entrada das transformações, neste caso, os ficheiros ASCII e ficheiros *Excel* que incluem os dados das observações e das observações geodésicas, tem que ser efectuada no programa de transformação e limpeza de dados. Assim, cada vez que se pretende executar o grafo de transformações para outros dados de entrada, é necessário compilar um novo programa de transformações. Além disso, a especificação dos fluxos de entrada é manual. Tendo em conta que, só a migração das observações inclui 1558 ficheiros distintos deve ser fornecida uma interface que facilite a selecção dos ficheiros de entrada.

Em segundo lugar, a interface de análise e correcção de excepções do *Ajax* não se adapta a elevados número de registos, exigindo um elevado esforço manual para correcção dos registos. Na migração do *gestBarragens*, é comum existirem vários erros do mesmo tipo, como por exemplo, duas campanhas completamente duplicadas, ou código de um determinado instrumento sempre errado num ficheiro de leituras. A ideia consiste em classificar as excepções geradas, com base na lógica da transformação que as produziu, fornecendo informação ao utilizador para as corrigir facilmente. Posteriormente, o utilizador corrige directamente os ficheiros de entrada. Por exemplo, um erro na formatação de um ficheiro de leituras, no qual o código de identificação do instrumento não aparece no campo correcto, faz com que todos os registos sejam excepções. Indicando

claramente ao utilizador que o código não está no campo correcto, este pode utilizar um editor de texto que suporte *Macros* e corrigir facilmente esta situação.

A Figura 4.16 apresenta a interface da ferramenta *infoLegada2Gb*. O utilizador dispõe de um botão de *browse* no qual pode especificar o ficheiro ou pasta com os ficheiros de entrada do programa de migração. Existe uma área de *log* que informa o utilizador do estado de execução no grafo de transformações. A aplicação é composta por quatro separadores: (*i*) Tabelas de apoio, para migrar a definição de todos os instrumentos; (*ii*) Leituras, que permite migrar as leituras dos instrumentos; (*iii*) Resultados, para migrar os resultados dos instrumentos; (*iv*) Consolidar, que implementa um algoritmo para agrupar várias campanhas.

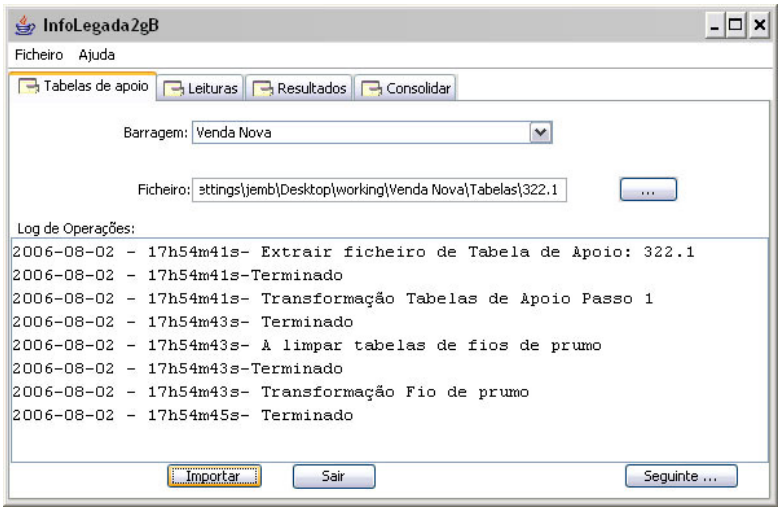

Figura 4.16: Interface da aplicação infoLegada2Gb

Quando a execução do programa termina, é produzido um ficheiro de erros que contém todas as excepções geradas. Estas excepções estão classificadas e organizadas por classes de excepções. Além disso, as classes de excepções também estão ordenadas, na medida em que a correcção das excepções da primeira classe pode conduzir à correcção automática das classes de excepções seguintes. A classificação das excepções é feita à medida de cada transformação. Para isso, desenvolveram-se um conjunto de procedimentos que analisam as excepções geradas e, usando os mecanismos de proveniência de dados disponibilizados pelo *Ajax*, analisam os registos das tabelas de entrada para classificar com exactidão o problema de dados.

A Figura 4.17 apresenta um excerto de um ficheiro de erros gerado durante a execução da migração das observações geodésicas. Os erros encontram-se agrupados e ordenados em classes. Assim, a correcção dos erros da primeira classe pode resolver os erros das classes posteriores. A mensagem de erro é explicita e facilmente interpretável por uma pessoa sem conhecimentos de informática. Por exemplo, na primeira classe de erros, em vez da mensagem de erro indicar a restrição de integridade que foi violada, indica que a rede geodésica não está definida na folha *DefPlan*.

```
----Ficheiro de Erros -----
2007-03-16 - 10h53m48s

Tabela MGR_RESCEN_VBADS (EXP, ID, ID_BARRAGEM, NOMEREDE, CAMPNOME, PONTONOME, PESOX, DX, PESOY, DY, PESOZ, DZ)
A rede LNIV_Crmt não está definida na folha DefPlan, 3495, 248, LNIV_Crmt, A03, R1, null, null, null, null, null, 0
A rede LNIV_Crmt não está definida na folha DefPlan, 3501, 248, LNIV_Crmt, A04, R1, null, null, null, null, null, 0
A rede LNIV_G2 não está definida na folha DefPlan, 3508, 248, LNIV_G2, A05, PF-G2, null, null, null, null, null, 0


Tabela MGR_RESALT_VBADS (EXP, ID, ID_BARRAGEM, CTRLCAMPREF, PONTONOME, CAMPNOME, VALOR)
Folha ResCampRef não tem intervalo para ctrl = 1001 na campanha A05, 1, 248, 1001, PD1-Niv, A05, 14998.2
Folha ResCampRef não tem intervalo para ctrl = 1001 na campanha A06, 2, 248, 1001, PD1-Niv, A06, 14998.3
Folha ResCampRef não tem intervalo para ctrl = 1001 na campanha A07, 3, 248, 1001, PD1-Niv, A07, 14998.2
Folha ResCampRef não tem intervalo para ctrl = 1001 na campanha A08, 4, 248, 1001, PD1-Niv, A08, 14998.2
O vértice: 06-07A-G3 tem duas definições (em defPontos e em defPontosHist), 501, 248, 1090, 06-07A-G3, A03, 30033
O vértice: 06-07A-G3 tem duas definições (em defPontos e em defPontosHist), 502, 248, 1090, 06-07A-G3, A05, 30032.8
```

Figura 4.17: Ficheiro de erros da aplicação *infoLegada2Gb*

Capítulo 5

Conclusões

Neste Capítulo, enunciam-se as principais conclusões resultantes do trabalho desenvolvido nesta tese e apresentam-se várias perspectivas de continuação do trabalho desenvolvido.

5.1 Sumário

Em primeiro lugar, cumpre salientar que a evolução dos sistemas de informação requer a migração dos dados para bases de dados mais estruturadas, a fim de simplificar e potenciar a extracção de informação.

Os processos de migração de dados têm várias semelhanças com os processos *ETL* aplicados ao carregamento de *Data Warehouses*. De facto, abstraindo das respectivas particularidades, tanto os processos de migração de dados como os processos *ETL* consistem, essencialmente, na extracção de dados de uma ou várias fontes de dados, na aplicação de um conjunto de transformações de dados e/ou de esquema e, finalmente, no carregamento dos dados transformados para uma base de dados alvo. A complexidade inerente à criação de processos de migração específicos, torna necessária a utilização de ferramentas que apoiem o desenvolvimento destes processos.

O projecto *gestBarragens*, que esteve subjacente à elaboração desta tese, inclui uma fase de migração de dados provenientes de fontes de dados heterogéneas e pouco estruturadas. Para suportar a migração de dados necessária neste projecto, procedeu-se,

com recurso à ferramenta *Ajax*, à implementação de um conjunto de transformações. Contudo, foi necessário efectuar a extensão dos operadores do *Ajax*, de forma a suportar o correcto funcionamento do processo de migração. As extensões mais significativas consistiram na possibilidade de geração de excepções no operador *View*, que implementa uma interrogação *SQL*, na definição de condições de actualização de registos através do operador *Map*, que permite realizar migração incremental e na extensão da gramática do *SQL* suportada pelo operador *View*.

Foi delineada uma estratégia de migração, suportada pelo *Ajax*, que veio a revelar-se bastante adequada para este tipo de migração. Com efeito, numa primeira fase desenvolveram-se, de forma iterativa, as transformações. Numa segunda fase, foram executadas as transformações e, no caso de gerarem registos de excepção, os dados que lhes deram origem foram corrigidos, sendo a transformação novamente executada. Para facilitar a correcção de registos e a selecção dos ficheiros fonte, desenvolveu-se uma aplicação à medida, denominada *infoLegada2Gb*. Esta aplicação foi utilizada exaustivamente, iterando-se entre a fase de execução das transformações e a correcção dos dados errados, até que não fossem geradas excepções. Em consequência, foi possível migrar todos os dados armazenados pelos sistemas legados, o que permite usufruir de todas as funcionalidades do novo sistema (*gestBarragens*) aplicadas sobre os dados antigos.

5.2 Trabalho Futuro

Antes de mais, importa referir que existem várias possibilidades de desenvolvimento do trabalho realizado nesta tese.

Em primeiro lugar, deve implementar-se a execução incremental dos operadores do *Ajax* quando existem alterações nos tuplos de entrada de cada transformação. Além disso, deve ser avaliada a execução de vários algoritmos de execução incremental, de forma a determinar as condições em que o custo da execução incremental é inferior ao custo de refazer a transformação para todo o conjunto de dados de entrada.

Pode aplicar-se a execução incremental dos operadores do *Ajax* à implementação de um processo *ETL* para refrescamento de um *DataWarehouse*. De facto, existem duas

aproximações possíveis para modificar os dados do sistema alvo de forma incremental. A primeira aproximação consiste em estabelecer as actualizações incrementais durante a fase de carregamento de dados, tal como acontece na solução apresentada na Secção 3.4.2 com a extensão do operador *Map* para suportar migração incremental. Assim, deve estudar-se uma evolução da solução apresentada nesta tese, de forma a suportar os requisitos do carregamento de processos *ETL* cíclicos. A segunda aproximação consiste em estabelecer o critério de actualizações incrementais durante a fase de extracção de dados, que só é possível registando as alterações que foram realizadas nas fontes de dados desde o último carregamento. Neste caso, é necessário definir o impacto de cada alteração, remoção ou inserção de registos nas fontes de dados, que pode ser tratado através da execução incremental dos operadores do *Ajax*. As duas soluções apresentadas para o carregamento incremental de um *DataWarehouse* devem ser devidamente avaliadas e comparadas, a fim de determinar os cenários óptimos para a sua aplicação.

Em segundo lugar, baseando-se na necessidade de geração de excepções no operador *View*, deve equacionar-se a possibilidade de geração de excepções em *SQL*, de forma automática, usando as estruturas geridas pelos sistemas de gestão de bases de dados. Para fundamentar as excepções em *SQL*, é necessário, em primeiro lugar, formalizar o conceito de excepções nos operadores da *Álgebra Relacional*. De seguida, deve seleccionar-se um sistema de gestão de base de dados "*open source*", como o *PostgreSQL* [Pos] ou o *MySQL* [MyS] e implementar a geração de excepções, seguindo a formalização definida na *Álgebra Relacional*.

Em terceiro lugar, conforme apresentado no Capítulo 4, quando existe geração de excepções no operador *View* pode ser necessário desenvolver mecanismos de classificação das excepções que facilitem a sua correcção. De facto, as excepções no operador *View* correspondem a registos que violam uma restrição de integridade. Dependendo da lógica da interrogação *SQL* que implementa a transformação, a mesma restrição de integridade pode ter sido violada por diferentes motivos. Um trabalho bastante interessante será desenvolver um mecanismo de classificação automática de excepções que, analisa a lógica da interrogação *SQL* e, determina a causa dos registos de excepção.

Finalmente, a linguagem declarativa do *Ajax* adequa-se apenas a utilizadores ex-

perientes e com conhecimentos de programação, podendo, portanto, ser estudada uma alternativa a esta linguagem, eventualmente baseada em *XML* ou o desenvolvimento de uma interface gráfica para definição das transformações. Além disso, seria interessante seguir a abordagem adoptada no *Potter's Wheel* [RH01], no qual não se especificam as transformações, apenas se indicando os dados que devem ser produzidos a partir de um subconjunto dos dados de entrada. A possibilidade de inferir a lógica de um operador do *Ajax*, a partir de um conjunto de exemplos de mapeamentos, revela-se um potencial trabalho a realizar.

Apêndice A

Listagem de instrumentos do *gestBarragens*

Equipamento	**Leitura**	**Resultado**
Ancoragem	Leitura 1, Leitura 2 e Leitura 3	Força (KN)
Base de Alongâmetro	Invar (mm), Lado 1 (mm), Lado 2 (mm) e Lado 3 (mm)	Abertura (mm) e Deslizamento (mm)
Base Planimétrica	Leitura (mm)	Diferença (mm)
Base Tridimensional	Invar (mm), Lado 1 (mm), Lado 2 (mm) e Lado 3 (mm)	Abertura (mm), Deslocamento horizontal (mm) e Deslocamento vertical (mm)
Continua na próxima página		

Equipamento	**Leitura**	**Resultado**
Célula de Fluência	Pressão na célula (bar), Resistência total (Ω) e Relação de resistências (%)	Pressão na célula (bar), Temperatura (ºC), Extensão (m×10-6)';
Clinómetro de Resistência	Resistência total (Ω) e Relação de resistências (%)	Temperatura (ºC) e Rotação (º)
Convergenciómetro	Padrão (mm), Número do furo da fita, Fita (mm) e Temperatura ambiente (ºC)	Temperatura ambiente (ºC) e Convergência (mm)
Dreno	Volume (l) e Tempo (s)	Caudal (l/min)
Escala de Nível	Nível da albufeira (m) e Nível de jusante (m)	Nível da albufeira (m) e Nível de jusante (m)
Extensómetro Acústico	Frequência (Hz) ou período de vibração da corda (s)	Extensão (m×10-6)
Extensómetro de Fundação	Invar (mm) e Vara (mm)	Deslocamento (mm)
Extensómetro de Grande Base	Resistência total (Ω) e Relação de resistências (%)	Temperatura (ºC) e Extensão real (m×10-6)
Extensómetro de Resistência	Resistência total (Ω) e Relação de resistências (%)	Temperatura (ºC) e Extensão real (m×10-6)

Continua na próxima página

Equipamento	**Leitura**	**Resultado**
Extensómetro Incremental	Profundidade do anel (m), Temperatura (^{o}C), Posição (mm) e Distância entre anéis (mm)	Profundidade do anel (m), Temperatura (^{o}C) e Distância entre anéis (mm)
Extensómetro Mecânico M.P.B.X.	1$^{\underline{a}}$ leitura do invar A (´´), 1$^{\underline{a}}$ leitura do invar B (´´), Fio (´´), 2$^{\underline{a}}$ leitura do invar A (´´) e 2$^{\underline{a}}$ leitura do invar B (´´)	Deslocamento (mm)
Fio de Prumo (Base)	Leitura radial ao fio (cm), Leitura tangencial ao fio (cm), Leitura radial ao cone (cm) e Leitura tangencial ao cone (cm)	Deslocamento radial relativo (mm), Deslocamento tangencial relativo (mm), Deslocamento radial absoluto (mm) e Deslocamento tangencial absoluto (mm)
Fissurómetro/Testemunho	Abertura (mm), Deslizamento vertical (mm) e Deslizamento horizontal (mm)	Abertura (mm), Deslizamento vertical (mm) e Deslizamento horizontal (mm)
Higrómetro ou Higrógrafo	Humidade (%)	Humidade (%)
Inclinómetro	Profundidade (m), Leitura no sentido positivo do eixo A, Leitura no sentido negativo do eixo A, Leitura no sentido positivo do eixo B e Leitura no sentido negativo do eixo B	Profundidade (m), Desvio lateral na direcção A (mm) e Desvio lateral na direcção B (mm)
Continua na próxima página		

Equipamento	**Leitura**	**Resultado**
Medidor de Juntas Acústico	Frequência (Hz) ou período de vibração da corda (s)	Abertura
Medidor de Juntas de Resistência	Resistência total Ω e Relação de resistências (%)	Temperatura (º C) e Abertura (mm)
Medidor de Pressão de Resistência	Resistência total Ω e Relação de resistências (%)	Temperatura (ºC) e Pressão nos poros (MPa)
Par Termoeléctrico	Temperatura (ºC)	Temperatura (ºC)
Piezómetro	Volume (l), Tempo (s) e Pressão (bar) ou altura de água (m)	Altura de água (m) e Caudal (l/min)
Piezómetro de Corda Vibrante	Temperatura (ºC) e Subpressão (bar)	Temperatura (ºC) e Subpressão (bar)
Pluviómetro	Precipitação (mm)	Precipitação (mm)
Tensómetro de Resistência	Temperatura (ºC) e Tensão (MPa)	Temperatura máxima (ºC)
Termómetro de Máxima e Mínima do Ar	Temperatura máxima (ºC) e Temperatura mínima (ºC)	Temperatura máxima (ºC) e Temperatura mínima (ºC)
Termómetro de Resistência	Resistência Total (Ω)	Temperatura (ºC)

Tabela A.1: Equipamentos

Apêndice B

Fio de Prumo

Este apêndice apresenta os detalhes dos instrumentos do tipo Fio de Prumo.

Através da utilização de Fios de Prumo é possível determinar, como resultados, deslocamentos horizontais em barragens.

O fio de prumo materializa uma vertical que atravessa a barragem, através de um fio de aço de alta resistência. Este fio pode estar fixo num ponto a cota elevada da estrutura, sendo designado por fio de prumo direito. Se o fio se encontrar fixo num ponto profundo da fundação, então o mesmo designa-se por fio de prumo invertido.

Nos fios de prumo direitos, o fio é tenso por acção de um peso de cerca de 600 N. Em todos os pontos em que o fio de prumo se encontra acessível, em diferentes cotas da barragem é possível determinar os deslocamentos do ponto de fixação em relação a esses pontos, isto é, deslocamentos relativos.

Nos fios de prumo invertidos, o fio é tenso por acção da impulsão aplicada pela água contida num reservatório sobre um flutuador nela mergulhado. Neste caso, em todos os pontos que estiverem acessíveis, é possível obter deslocamentos em relação ao ponto profundo da fundação. Uma vez que este ponto pode ser considerado fixo, os deslocamentos obtidos através de fios de prumo invertidos são deslocamentos absolutos.

Nos locais de passagem do fio de prumo que estão acessíveis, nomeadamente nas galerias de visita, nos nichos, nas plataformas ou nos passadiços solidarizados com a estrutura, são colocadas peças solidarizadas à estrutura, bases de coordinómetro, onde é possível estacionar o dispositivo de medida utilizado - coordinómetro óptico - que

permite obter as coordenadas rectangulares do fio no plano de medida.

Quando a forma da barragem não permite a colocação de um tubo vertical desde a fundação até a cota elevada da barragem, utiliza-se um conjunto fio de prumo invertido + fio de prumo direito. Para isso montam-se em dois poços com a mesma verticalidade dois fios de prumo, um direito e outro invertido, e faz-se com que a base de coordinómetro inferior do fio de prumo direito fique o mais próximo possível da base superior do fio de prumo invertido. Com este dispositivo pode-se relacionar os deslocamentos relativos fornecidos pelo fio de prumo direito, com os dados pelo fio de prumo invertido e assim obter as componentes absolutas dos deslocamentos.

Para manter um controlo sobre as condições de funcionamento do coordinómetro ou possibilitar o uso de outro aparelho, instala-se junto à base de coordinómetro uma referência, designada cone de referência, com a qual se materializa um ponto que tem os mesmos movimentos que a base vizinha. As leituras correspondentes ao cone têm assim o mesmo valor para o mesmo aparelho, excepto os erros de observação.

As leituras fornecidas por este tipo de aparelho numa determinada época n, são as seguintes:

RFn - Leitura radial ao fio na época n

RCn - Leitura radial ao fio na época n

TFn - Leitura tangencial ao fio na época n

TCn - Leitura tangencial ao fio na época n

As expressões para o cálculo dos resultados, isto é, dos deslocamentos radial (DR) e tangencial (DT), em cada um dos pontos de medida são as seguintes:

$$DR = ([(RF_n - RF_i) - (RC_n - RC_i)] .Fr. \cos(\alpha) - [(TF_n - TF_i) - (TC_n - TC_i)] .Ft. \sin(\alpha)) .10$$

$$DT = ([(RF_n - RF_i) - (RC_n - RC_i)] .Fr. \sin(\alpha) + [(TF_n - TF_i) - (TC_n - TC_i)] .Ft. \cos(\alpha)) .10$$

Apêndice C

Problemas de Qualidade de Dados

Existem vários trabalhos significativos que, apresentam uma taxonomia de classificação de problemas de dados [KCH+02, RD00, ORHG05, MF03]. Uma possível abordagem consiste em separar estes problemas em duas categorias distintas: (*i*) problemas de esquema; (*ii*) problemas de instância (registo).

Os problemas de esquema de dados, não estão relacionados com os dados armazenados, mas sim com a sua estrutura. Este tipo de problemas pode ser evitado através da optimização da estrutura (e.g. esquema da base de dados) sob a qual os dados estão organizados.

Os problemas de instância, estão relacionados com os dados armazenados e não com a sua estrutura, não podendo ser evitados através de um melhoramento no esquema de armazenamento. Os problemas de instância incluem todos os problemas de dados que não são problemas de esquema, pelo que, esta separação conduz a uma taxonomia completa.

C.1 Problemas de Esquema

Os problemas de esquema podem ser evitados através de um melhoramento no desenho da base de dados, tradução e integração de esquemas. Os Sistemas de Gestão de Bases de Dados (SGBD) actuais, disponibilizam um conjunto significativo de mecanismos para garantir definições e restrições de esquema. Assim, distinguem-se problemas de esquema

que podem ser evitados pelos SGBD actuais, dos problemas que não podem ser evitados através dos mecanismos providenciados pelos SGBD.

C.1.1 Evitados pelos SGBD Actuais

Os SGBD actuais disponibilizam, essencialmente, dois mecanismos para garantir a integridade de uma base de dados: restrições de integridade e controlo de transacções.
Restrições de Integridade são definidas no desenho da base de dados para especificar um conjunto de condições que os objectos da base de dados devem respeitar [TG01]. O SQL:1999[fSIA99] suporta uma linguagem para definir o seguinte conjunto de restrições de integridade:

- NOT NULL: previne que um registo tenha um valor null (não esteja preenchido) numa determinada coluna, isto é, obriga a que todos os registos tenham um valor específico para essa coluna.

- Default: especifica o valor por omissão para uma determinada coluna. Assim, essa coluna assume o valor determinado por omissão, para todos os registos que não tenham qualquer valor nessa coluna.

- UNIQUE: especifica que uma coluna ou conjunto de colunas deve ter valor único numa tabela específica, isto é, não podem haver dois registos nessa tabela com o(s) mesmo(s) valor(es) para a(s) coluna(s) que estão definidas como UNIQUE.

- PRIMARY KEY: define a chave primária de uma tabela que é, obrigatoriamente, não nula e única. Cada tabela só pode ter uma chave primária.

- FOREIGN KEY: especifica uma chave estrangeira, na qual o valor de um atributo ou conjunto de atributos corresponde à chave primária de outra tabela.

- CHECK: define uma restrição baseada numa condição definida pelo utilizador, numa determinada tabela.

- DOMAIN: define um domínio de dados restrito para uma determinada coluna.

- ASSERTION: define uma restrição que é global à base de dados, isto é, não depende de uma tabela específica e é verificada em todas as operações realizadas na base de dados.

- TRIGGERS: O SQL:1999 disponibiliza uma linguagem procedimental para definição de restrições que são verificadas sempre que ocorre um determinado evento.

Com este conjunto de restrições de integridade, existe um conjunto significativo de problemas que pode ser evitado:

- Dados omissos: Problema que ocorre quando os dados não foram armazenados. Uma restrição *Not Null* evita este problema, obrigando um conjunto de colunas a ser preenchidas.

- Tipo de dados errado: Os dados guardados não correspondem ao seu domínio real. Por exemplo, a idade do empregado é '2o'. As restrições de domínio (*Domain*) evitam estes problemas de dados.

- Valor fora do limite espectável: Acontece quando o valor de uma coluna não está na gama de valores espectável. Se, por exemplo, a idade espectável de um empregado estiver no intervalo [18, 65], então, uma idade de 14 é um *Valor fora do limite espectável*. As restrições de domínio (*Domain*) e de verificação (*Check*) evitam este problema de dados.

- *Dangling data*: Os dados numa tabela não têm correspondência na outra tabela. Por exemplo, o identificador de um departamento não existe na tabela de Departamentos, mas existe uma referência para esse valor na tabela de Empregados, isto é, o empregado referencia um departamento que não existe. Este problema é evitado pelas restrições de integridade referencial (*Foreign Keys*).

- Duplicados exactos: Este problema acontece quando registos diferentes têm o mesmo valor num atributo (ou conjunto de atributos), para os quais não deveria ser permitido. Por exemplo, o número da segurança social não pode ser o mesmo para empregados diferentes. As restrições *Unique* e *Primary Key* evitam duplicados exactos.

- Restrições de Domínio: Registos que violam uma determinada restrição de domínio, como por exemplo, dependências entre atributos, número máximo de linhas permitida. O SQL: 1999 disponibiliza as restrições *DOMAIN* e *ASSERTION* para evitar erros de domínio. Contudo, estas funcionalidades não são implementadas pela maior parte dos SGBDs (como o Oracle 9i), nos quais, este tipo de problema de dados só pode ser evitado através da codificação de *TRIGGERS*.

Controlo de Transacções Através dos mecanismos de controlo de transacções, os SGBDs actuais permitem evitar três tipos de erros de dados: (*i*) *Lost Update*, (*ii*) *Dirty Read*, (*iii*) *Unrepeatable Read* [SKS02] Os erros de perda de transacções são evitados através dos mecanismos de recuperação da Base de Dados. Quando duas ou mais transacções lêem e escrevem os mesmo dados, simultaneamente, podem ocorrer dois tipo de anomalia. Um *Lost Update* ocorre quando, por exemplo, a transacção T1 Lê o saldo de uma conta como 50€e, por uma operação de levantamento de 50€, decrementa o saldo, actualizando-o para 0€. Se, ao mesmo tempo, a transacção T2 ler o mesmo saldo de 50€e, por uma operação efectua (por exemplo uma transferência de 50€), actualizar o saldo para 0€, uma das duas actualizações perde-se. Um *Dirty Read* ocorre quando, por exemplo, uma transacção T1 aumenta o saldo de 50€para 100€e, a transacção T2 lê esse saldo de 100€, permitindo fazer um levantamento de, por exemplo, 90€e, devido a uma falha ou cancelamento, a transacção T1 aborta. Neste caso, a transacção T2 leu um valor errado, já que, a transacção T1 ainda não tinha terminado com sucesso. O problema de *Unrepeatable Read* acontece quando, por exemplo, a transacção T1 lê o saldo da conta como 50€, em seguida, a transacção T2 actualiza o mesmo saldo para 100€. Se a transacção T1 ler novamente o saldo da conta, pode ler um valor diferente (a primeira leitura 50€e a segunda 100€) o que implica um erro, já que, leituras diferentes significam dados alterados que poderão mudar novamente. Um erro de perda de transacções ocorre num cenário em que, por exemplo, uma transacção transfere um valor de uma conta para outra. Se o sistema falhar quando a transacção debitou o valor de uma conta e, antes de ter creditado na outra, esse valor "evapora-se". As propriedades de durabilidade e atomicidade das transacções evitam este problema.

C.1.2 Não evitados pelos Sistemas de Gestão de Bases de Dados Actuais

O seguinte conjunto de problemas de dados não pode ser evitado pelos SGBDs actuais:

- Categorias de Dados Errada: Um determinado valor, não pertence à categoria de valores permitidos para esse atributo. Um exemplo comum é o das Categorias de Distritos, Concelhos e Freguesias. Por exemplo, o concelho de Castelo Branco no Distrito de Lisboa é um erro de categorias de dados.

- Dados desactualizados: Dados que violam uma determinada restrição temporal, isto é, o intervalo de tempo ou instante no qual os dados são válidos. Por exemplo, o salário de um empregado não é válido, para efeitos de contabilidade assim que o empregado for despedido.

- Dados espaciais errados: Inconsistência em dados espaciais, como por exemplo coordenadas e figuras geométricas, quando são armazenadas em atributos múltiplos. Por exemplo, as quatro coordenadas de um quadrado, têm que ser combinadas de forma a gerar um quadrado fechado com todos os lados iguais.

- Conflito de Nomes: O mesmo nome de atributo é usado para objectos diferentes (homónimos) ou diferentes nomes atributos são usados para o mesmo objecto.

- Conflitos estruturais: O mesmo objecto é representado por esquemas diferentes em diferentes tabelas e/ou diferentes bases de dados. Por exemplo, um morada pode ser representada por um único campo, ou separada em diferentes atributos como: cidade, rua, número, etc.

C.2 Problemas de instância

Os problemas de instância são todos os erros e inconsistências de dados que não podem ser visíveis ou evitados ao nível do desenho da base de dados. Note-se, no entanto, que as instâncias também reflectem problemas de esquema, como por exemplo, um registo

com um valor vazio num campo obrigatório. Os problemas de instância separam-se em problemas que envolvem um único registo dos problemas que envolvem múltiplos registos.

C.2.1 Único registo

Os problemas de único registo surgem de um ou vários atributos num registo de uma tabela de uma base de dados. Por outras palavras, este tipo de problema está relacionado, unicamente, com uma entidade e não depende da restante informação armazenada na base de dados.

- Falta de valores num campo *NOT NULL*: Um atributo encontra-se preenchido com um código de erro usado para ultrapassar a restrição de integridade imposta. Por exemplo, o número -99999999 num campo destinado ao número do bilhete de identidade é um valor indefinido usado para ultrapassar a restrição de integridade.

- Dados erróneos: Os dados são válidos mas não estão conforme a entidade real. Um exemplo de dados errados, pode ser o nome de um empregado ser "João" e, a entidade real a que corresponde esse registo chamar-se "Afonso".

- Erros ortográficos: Palavras mal escritas nos registos da base de dados, como por exemplo, João Maira" em vez de "João Maria".

- Valores embebidos: Existência de informação adicional num determinado campo. Um exemplo comum de valores embebidos é a introdução de um título num campo destinado ao nome, por exemplo, "Presidente João Maria".

- Valores no campo errado: Dados armazenados no campo errado. Para um exemplo simples, considere-se o valor "Portugal" num campo destinado à cidade.

- Dados ambíguos: Dados que podem ser interpretados de mais do que uma forma, tendo significados diferentes. Os dados ambíguos ocorrem devido a abreviaturas ou contexto incompleto, da seguinte forma:

  - Abreviatura: O nome abreviado "J. Maria" pode ser expandido de diferentes formas, tais como: "João Maria", "José Maria", "Joaquim Maria", etc.
  - Contexto incompleto: A falta de contexto (informação adicional) não permite identificar a entidade representada. Por exemplo, o nome de cidade "Miami" pode pertencer ao estado da Florida ou ao estado de Ohio.

C.2.2 Múltiplos registos

Os erros de múltiplos registos são erros de dados que não podem ser detectados considerando apenas um registo isoladamente, já que, este tipo de erros depende da sua relação com outros registos. Note-se que os erros de múltiplos registos podem acontecer apenas num conjunto de entidade (tabela) ou, através da relação entre diferentes conjuntos de entidades (tabelas diferentes ou até mesmo bases de dados diferentes).

- Registos duplicados: Registos que correspondem à mesma entidade real e que não contêm informação contraditória. Por exemplo, os seguintes registos correspondem ao mesmo empregado e são considerados duplicados: Emp1(Nome= "João Maria", Morada= "23, Avenida da Liberdade, Lisboa", Data de Nascimento= "01/01/1975"); Emp2(Nome= "J. Maria", Morada= "23, Av. Liberdade, Lisboa", Data de Nascimento= "01/01/1975").

- Registos contraditórios: Registos que correspondem à mesma entidade real, mas que contêm algum tipo de informação que é contraditória. Considere-se novamente os registos de empregados Emp1 e Emp2, mas com a seguinte informação: Emp1(Nome= "John Stevens", Morada= "23, Avenida da Liberdade, Lisboa", Data de Nascimento= "01/01/1975"); Emp2(Nome= "John Stevens" Morada= "23, Avenida da Liberdade, Lisboa", Data de Nascimento= "01/01/1965")

- Dados não normalizados: Os problema ocorre quando registos diferentes não usam a mesma representação dos dados, inviabilizando a sua comparação:

  - Transposição de palavras: Num campo único, as palavras podem aparecer com diferentes ordenação. Por exemplo, os nomes "João Silva" e "Silva,

José", não usam a mesma regra de ordenação.

- Formato de codificação: Uso de diferentes formatos em diferentes registos, como por exemplo, ASCII, UTF-8.
- Diferente representação: Um exemplo de diferentes representações está nos valores monetários que podem ser, por exemplo, \$10.5, 10.5€, etc.
- Unidade de Medida: Diferentes unidades de medidas usadas em vários registos, como por exemplo, distâncias medidas em centímetros e em polegadas.

Apêndice D

Sistemas Legados

D.1 Sistema SIOBE

O SIOBE [LNE80, HS93, Ama04] baseia-se na gestão de um conjunto de ficheiros binários e ASCII que permitem controlar o sistema e armazenar os resultados das observações em barragens. Note-se também, o uso de ficheiros para a introdução de dados no sistema. Em seguida descrevem-se os módulos que constituem o SIOBE, a estrutura e utilidade dos tipos de ficheiros usados pelo SIOBE e, finalmente, os principais problemas identificados no SIOBE.

D.1.1 Módulos do sistema

O SIOBE é composto por um conjunto de programas interactivos que, apesar de estarem relacionados, não se encontram integrados numa única aplicação. Em seguida descrevem-se os módulos que agrupam estes programas.

D.1.1.1 OBSERV

Constituído por um conjunto de programas que efectuam o tratamento dos dados recolhidos nos sistemas de observação das barragens. De uma forma geral, permite validar as leituras inseridas através de um ficheiro de dados, transformando-as em grandezas de observação (resultados) que também podem ser validadas. Posteriormente, arquiva as

grandezas em ficheiros binários.

D.1.1.2 LISTAR

Permite explorar os resultados contidos na base de dados (sistema de ficheiros) respeitantes a uma determinada barragem e tipo de instrumento, num período de tempo indicado pelo utilizador. Os resultados podem ser visualizados no ecrã ou impressos em papel, sob a forma de tabelas.

D.1.1.3 DIAGR

O módulo DIAGR é semelhante ao módulo LISTAR, uma vez que também permite explorar os resultados presentes na base de dados. Elabora gráficos de evolução temporal das grandezas de observação (resultados).

A execução deste programa encontra-se condicionada pela existência de ficheiros de definição dos diagramas (.LIG) previamente estabelecidos para os vários tipos de instrumentos colocados em cada barragem. Caso se pretenda incluir uma folha de rosto com o esquema de localização dos instrumentos, torna-se necessária a existência de um ficheiro que contém a definição dos desenhos (.FLR).

D.1.1.4 INTQUA

As respostas dos sistemas barragem-fundação resultam da actuação simultânea de diversas acções, entre as quais se destacam a variação do nível de água na albufeira e a variação da temperatura ambiente. Do ponto de vista da interpretação do comportamento observado e do controlo de segurança é importante separar nas respostas observadas as parcelas relativas a cada uma dessas solicitações.

Embora cada parcela possa ser individualmente caracterizada quando se verificar, entre duas épocas, a variação predominante da respectiva solicitação, a forma mais geral de separar os efeitos de cada solicitação consiste na utilização de métodos estatísticos geralmente designados por análises quantitativas de resultados. Estes métodos têm por objectivo identificar um modelo de comportamento com base na história da barragem - modelo de interpretação quantitativa - que se traduz no estabelecimento de uma relação

estatística entre as principais variáveis ambientais (nível da água na albufeira e temperatura ambiente), efeitos associados ao tempo, e as variáveis de controlo (grandezas observadas).

Nos modelos de interpretação quantitativa estabelecem-se assim relações funcionais entre os efeitos observados num determinado instante e as acções que os originam. Habitualmente utilizam-se polinómios para caracterizar o efeito da pressão hidrostática, funções sinuosidade para caracterizar a influência da onda térmica anual e polinómios e/ou logaritmos para os efeitos do tempo.

O módulo INTQUA permite explorar a base de dados para um determinado período de tempo, elaborando a interpretação quantitativa de uma série de resultados, de acordo com o modelo estabelecido. Os resultados da interpretação quantitativa podem ser apresentados sob a forma de tabela ou graficamente.

D.1.1.5 EXTENS

Este módulo permite explorar os resultados dos grupos de extensómetros Carlson, para um determinado período de tempo, calculando as extensões corrigidas das variações autogéneas de volume, do efeito de Poisson e de compatibilidade, bem como o estado de tensão. Os resultados deste módulo podem ser apresentados sob a forma de tabelas ou graficamente. Para além dos gráficos de evolução temporal, este módulo também permite a representação gráfica da variação do estado de tensão entre duas datas definidas pelo utilizador.

D.1.2 Dados armazenados pelo SIOBE

O sistema de armazenamento do SIOBE é constituído por um conjunto de ficheiros ASCII e binários que se descrevem em seguida.

D.1.2.1 Ficheiro Geral de Configuração (HEAD.DAT)

O ficheiro HEAD.DAT é um ficheiro de controlo do SIOBE, sem o qual não é possível executar o sistema. É um ficheiro binário que contém dois tipos de informação: (i)

informação de todas as barragens registadas no sistema, e (ii) informação de todos os tipos de instrumentos suportados.

D.1.2.2 Ficheiro de Apoio por Tipo de Instrumento (Tabela de Apoio)

As tabelas de apoio são ficheiros ASCII, que contêm informação sobre a identificação e elementos de cálculo, para cada instrumento. Para cada tipo de instrumento em cada barragem, tem que existir uma tabela de apoio, correspondendo cada linha dessa tabela a um instrumento (instância) desse tipo de instrumento.

As tabelas de apoio são compostas por três tipos de informação:

- TABELA APOIO INFO-1: Primeira parte da tabela de apoio que contém informação usada para o dimensionamento da leitura da tabela e para a validação dos resultados (informa o sistema se deve ou não fazer validação dos resultados).
- TABELA APOIO INFO-2: Cada linha deste bloco de informação corresponde a um instrumento (instância), com a sua identificação, a informação para o cálculo dos resultados relativos a esse tipo de instrumento e a informação para a validação de dados (limite inferior e superior para cada leitura).
- TABELA APOIO INFO-3: Bloco da tabela que é opcional e contém os coeficientes estabelecidos pelos métodos estatísticos que permitem fazer a validação dos resultados.

D.1.2.3 Ficheiro de Leituras (Ficheiro de Dados)

Os ficheiros de dados são ficheiros ASCII que registam as leituras feitas nas campanhas de observação. Os ficheiros de dados no SIOBE são apenas fontes da informação correspondente às leituras que o sistema irá usar para calcular os resultados. O SIOBE não armazena a informação das leituras guardada nos ficheiros de dados, sendo necessário guardá-los num directório à parte que não pode interagir com o sistema.

Cada ficheiro de dados pode conter informação de várias campanhas e de vários instrumentos, sendo constituído por vários blocos de dados. Cada bloco de dados cor-

responde à informação de uma campanha para um tipo de instrumento e é delimitado pela informação principal eventual (nível da albufeira).

Comporta dois tipos de informação:

- INFOCOMUM: Informação comum (o número de instrumentos lidos para esse tipo, o tipo, a barragem, etc.) a todos os instrumentos do mesmo tipo.
- INFOINDIVIDUAL: Informação que diz respeito a cada instrumento (instância). A informação é relativa a uma leitura feita por esse instrumento (data da leitura, valores lidos e identificação do instrumento).

D.1.2.4 Ficheiro de Erros (Ocorrências)

Durante a execução do programa é feita a validação das leituras apresentadas num ficheiro de dados. Caso exista um erro numa qualquer linha do ficheiro de dados, essa linha é copiada para o ficheiro de erros, sendo apresentada uma mensagem adicional que informa sobre a natureza do erro (e.g., código de barragem inexistente, valor lido fora dos limites de validação, etc.).

D.1.2.5 Ficheiro de Resultados

Os ficheiros de resultados são ficheiros binários. Existe um ficheiro de resultados para cada tipo de instrumento implementado numa barragem. Cada registo (linha do ficheiro) corresponde aos resultados (todas as grandezas calculadas para o tipo de instrumento) de cada instrumento desse tipo, numa campanha.

D.1.2.6 Ficheiro de Controlo dos Arquivos de Resultados

O ficheiro de controlo dos arquivos de resultados é um ficheiro binário que contém informação indispensável ao controlo do armazenamento dos resultados.

Existe um ficheiro de controlo do arquivo de resultados dos diversos tipos de instrumentos para cada barragem. Assim, este ficheiro permite identificar se um dado tipo de instrumento está ou não implementado numa barragem. Em acréscimo, define o dimensionamento do ficheiro de arquivo de resultados para cada tipo de instrumento, isto

é estabelece um limite máximo de campanhas que corresponde ao número máximo de "linhas"que cada ficheiro de resultados poderá ter. Por outro lado, mantém a informação se o ficheiro de resultados se encontra ou não ordenado por datas. A reordenação dos resultados é necessária para alguns tipos de exploração.

D.1.2.7 Ficheiro com a Geometria da Barragem (Folha de Rosto)

As folhas de rosto são ficheiros binários que contêm informação geométrica da barragem, e têm como objectivo servir de base para alguns desenhos efectuados pelo SIOBE (por exemplo, esquema de localização dos instrumentos).

A constituição deste tipo de ficheiros é bastante simples, sendo compostos por vários grupos de coordenadas de pontos, cujo desenho deverá representar uma "imagem" da barragem.

D.1.2.8 Ficheiro com Informação para Elaboração de Diagramas e Interpretações Quantitativas (.LIG)

Os ficheiros LIG são ficheiros no formato ASCII, que contêm informação utilizada na elaboração de desenhos (DESENHOINFO) e informação utilizada pelos programas de interpretação quantitativa (INTQUAINFO).

Para a elaboração dos desenhos, o ficheiro .LIG define quais os tipos de instrumentos para os quais existem diagramas estruturados, bem como o número de diagramas definidos para cada tipo de instrumento. Esta informação designa-se por DESENHOSINFO e é acrescida de um número de linhas que permite fazer a separação entre o final do bloco relativo aos desenhos e o início do bloco da interpretação quantitativa. A seguir ao bloco DESENHOINFO, cada linha do ficheiro contém informação relativa à estrutura de cada diagrama (DIAGRAMAINFO). Esta informação inclui o factor de escala, os limites máximos e mínimos dos eixos, etc.

D.1.3 Resumo dos dados armazenados no SIOBE

A Tabela D.1 resume a informação armazenada pelo sistema SIOBE.

Para cada um dos 21 tipos de instrumento que o SIOBE armazena, apresenta-se, para cada tipo de ficheiro (Tabela de Apoio com a definição dos instrumentos, Leituras e Resultados), o número de ficheiros que têm que ser migrados, bem como o número de registos acumulados nesses ficheiros.

Por exemplo, na linha relativa ao Fio de Prumo, indica-se que têm que ser migradas 40 tabelas de apoio, que correspondem a 40 barragens e têm um total de 493 instrumentos definidos, 38 ficheiros de leituras com um total de 276.676 linhas e 39 ficheiros de resultados com um total de 314.124 linhas.

Tipo	**Instrumento**		**Leitura**		**Resultado**	
	Ficheiros	Registos	Ficheiros	Registos	Ficheiros	Registos
Fio de Prumo	40	493	38	276.676	39	314.124
Extensómetro de Resistência	35	4.532	34	1.200.564	34	1.543.912
Base de Alongâmetro	49	4.606	46	1.614.830	49	1.749.858
Medidor de Juntas	36	1.335	34	396.914	35	420.386
Termómetro de Resistência	32	1.849	30	451.495	31	618.551
Extensómetro de Fundação	38	562	35	207.133	37	227.907
Dreno	58	5.498	53	1.441.986	56	1.761.310
Piezómetro	53	1.759	49	631.172	53	797.219

Continua na próxima página

Tipo	**Instrumento**		**Leitura**		**Resultado**	
	Ficheiros	Registos	Ficheiros	Registos	Ficheiros	Registos
Temperatura do Ar e Níveis	60	60	52	407.800	42	407.319
Extensómetro de Grande Base	7	186	7	45.416	7	52.109
Extensómetro Acústico	2	175	2	2.559	2	4.895
Medidor de Juntas Acústico	1	5	1	75	1	129
Tensómetro de Resistência	32	1.849	30	451.495	31	618.551
Medidor de Pressão	10	82	9	17.389	10	29.190
Par Termoeléctrico	10	1.011	10	279.048	10	300.809
Clinómetro de Resistência	3	33	3	19.089	3	20.278
Células de Fluência	8	210	3	32.784	7	34.049
Nivelamento Geodésico	43	1.176	-	-	41	23.274
Deslocamento Geodésico	42	1.023	-	-	38	21.907
Convergenciómetro	3	9	1	8	1	725
Base Tridimensional	11	299	10	35.728	11	47.832

Continua na próxima página

Tipo	Instrumento		Leitura		Resultado	
	Ficheiros	Registos	Ficheiros	Registos	Ficheiros	Registos
Totais	573	26.752	447	7.512.161	538	8.994.334

Tabela D.1: Resumo dos ficheiros fonte

D.1.4 Problemas identificados no SIOBE

Durante o levantamento da informação armazenada no SIOBE, foi identificado um conjunto de problemas que o sistema apresenta.

O primeiro grupo de lacunas deve-se principalmente à estrutura baseada em ficheiros e à utilização da linguagem FORTRAN, que limita fortemente a evolução da informação guardada e das aplicações oferecidas aos utilizadores. O segundo grupo de lacunas corresponde a novos requisitos de exploração que surgiram ao longo dos anos.

Enumeram-se em seguida alguns pontos fracos do SIOBE:

- Os ficheiros de entrada de dados (que correspondem a leituras) não são armazenados pelo SIOBE, pelo que não é possível relacionar directamente os dados com os respectivos resultados

- A solução de ficheiros de resultados implica uma dimensão fixa. Assim, para todos os registos (linhas), isto é, para todas as campanhas, deve ser guardada informação sobre os resultados de todos os instrumentos desse tipo, mesmo que não tenham sido observados.

- A entrada de dados no programa através da linha de comandos é bastante rudimentar, não existindo validação adequada e dando origem à introdução de erros.

- Os dados são introduzidos unicamente através da linha de comandos, utilizando uma interface MS-DOS interactiva, o que implica a necessidade de conhecimento de códigos por parte do utilizador. Por exemplo, o utilizador é obrigado a conhecer os códigos dos instrumentos que pretende analisar.

- A tecnologia FORTRAN é também apontada como um ponto fraco, dada a dificuldade de expansão e integração de novos módulos.

- Limitações de algumas aplicações, nomeadamente o programa INTQUA.

- Limitações de representação gráfica dos resultados (permite apenas representação temporal).

D.2 Observações Geodésicas

A informação relativa às observações geodésicas pode dividir-se em três grupos distintos: (i) informação utilizada para planeamento da rede de observação, (ii) observações registadas e (iii) deslocamentos calculados. De forma sumária, descreve-se em seguida a forma como esta informação é armazenada actualmente e quais as transformações que lhe são aplicadas. O grande problema identificado na gestão da informação relativa às observações geodésicas é a sua quase total separação do sistema SIOBE. Apenas são guardados no SIOBE os deslocamentos calculados no fim do processo de transformação das observações.

D.2.1 Planeamento da Rede de Observação

Para efectuar observações em redes geodésicas, é necessário um estudo inicial de planeamento das redes de observação. Este planeamento consiste em definir as coordenadas dos vértices da rede, de forma a que essa configuração permita optimizar a precisão e fiabilidade da rede, bem como o custo e duração das campanhas de observação. Para apoiar o projecto de redes geodésicas, desenvolveram-se no LNEC vários "programas de desenho", nomeadamente, os programas: 1DNETD, 2DNETD e 3DNETD. Estes programas permitem validar a precisão e fiabilidade das linhas de nivelamento, redes bidimensionais e tridimensionais, calculando os respectivos índices de precisão (elipses de erro) e fiabilidade (NRL).

D.2.2 Observações Geodésicas

Após a definição e validação das redes geodésicas, iniciam-se as observações. Actualmente, as leituras realizadas são registadas em cadernetas de campo (de forma manual) ou em dispositivos do tipo PCM/CIA (de forma automática pelo instrumento de leitura). Posteriormente, após uma verificação sumária das leituras, estas são arquivadas em folhas Excel. Para cada barragem, existem várias folhas Excel (tipicamente uma para cada rede) que contêm informação sobre as observações feitas em linhas de nivelamento (desníveis) e em redes planimétricas (ângulos e distâncias). Nos casos em que o registo foi feito de forma manual sobre as cadernetas de campo é necessário introduzir toda a informação das leituras nos ficheiros Excel. Caso o registo seja feito de forma automática pelo instrumento, existe uma aplicação desenvolvida em Visual Basic que carrega as leituras nos ficheiros Excel.

D.2.3 Deslocamentos

Existem actualmente no LNEC vários programas de ajustamento que calculam resultados (deslocamentos) à custa das observações feitas, quer em linhas de nivelamento, quer em redes planimétricas. Os programas implementados são denominados 1DNETA, 2DNETA e 3DNETA e calculam os deslocamentos nos pontos à custa das leituras efectuadas nas linhas de nivelamento, redes bidimensionais e tridimensionais, respectivamente. O carregamento dos ficheiros de entrada para estes programas é feito em gabinete, de forma manual, após o registo das leituras em folhas Excel. Após a execução dos programas de ajustamento, os deslocamentos calculados são adicionados às coordenadas dos pontos na época de referência, que é a época especial utilizada no cálculo dos deslocamentos (na época de referência os deslocamentos são zero). Estas coordenadas são também arquivadas nas folhas Excel, constituindo assim o arquivo de resultados das observações geodésicas. Actualmente, para visualização e exploração gráfica de resultados é usado o SmartSketch, que é a ferramenta CAD da Intergraf.

D.2.4 Dados das observações geodésicas

As observações geodésicas encontram-se armazenadas num conjunto de ficheiros *Excel*, que respeitam uma estrutura bem definida. Existe um ficheiro *Excel*, que inclui um conjunto de folhas com informação sobre as observações geodésicas de cada barragem.

Na folha *Camp*, definem-se as campanhas que foram realizadas naquela barragem. A folha *DefPontos* regista todos os vértices que compõem as redes geodésicas da barragem. Na folha *DefNiv*, definem-se as várias ligações que compõem cada uma das linhas de nivelamento instaladas na barragem. Na folha *DefPlan* definem-se as ligações e giros que compõem as redes planimétricas. Na folha *LeitDesn* registam-se os desníveis verificados nas ligações das linhas de nivelamento. Na folha *LeitHz* registam-se os ângulos horizontais observados nas redes planimétricas. Na folha *LeitEst* registam-se os parâmetros meteorológicos que podem ser observados durante cada giro de uma rede planimétrica. Na folha *LeitDist* registam-se as distâncias observadas nas redes planimétricas. Na folha *LeitV* definem-se as ligações que compõem as redes tridimensionais e os ângulos verticais observados nas várias campanhas. Na folha *ResCampRef* determina-se a campanha de referência usada no cálculo dos resultados. Na folha *ResCen* registam-se os deslocamentos observados para os pontos considerados fixos em cada rede geodésica. Na folha *ResAlt* registam-se os resultados calculados a partir das linhas de nivelamento, isto é, coordenadas na direcção *dz*. Na folha *ResPlan* registam-se as coordenadas calculadas a partir das redes planimétricas e tridimensionais, isto é, coordenadas nas direcções *dx*, *dy* e *dz*. A folha *DefPontosHist* guarda o histórico de identificação de vértices e, finalmente, a folha *Folha de Controlo* inclui o dimensionamento das em número de linhas e de colunas das restantes folhas.

A Tabela D.2 apresenta a estrutura detalhada de cada uma das folhas *Excel* que compõem o arquivo de informação das observações geodésicas. Por exemplo, a folha *Camp* com a definição de campanhas é composta por seis atributos (CampNum, CampNome, Data, CotaAgua, Outras, Comments).

Folha	**Atributo**	**Descrição**
Camp	CampNum	identificador numérico da época
	CampNome	nome única para a época
	Data	data da campanha
	CotaAgua	cota da água a montante
	Outras	outras designações para a campanha
	Comments	comentários relativos à campanha
DefPontos	PontoNome	nome do vértice
	X	coordenada do vértice em xx
	Y	coordenada do vértice em yy
	Z	coordenada do vértice em zz
	Fi	ângulo de rotação do vértice
	TipoPonto	tipo de vértice
	FuncaoRede	função do vértice na rede
	Centragem	tipo de centragem do vértice
	HistNum	código de controlo de histórico
DefNiv	NomeRede	nome da linha de nivelamento
	PontoAnter	identificação do vértice atrás
	PontoPoster	identificação do vértice à frente
	NumOrdem	posição da ligação na rede
DefPlan	NomeRede	nome da rede planimétrica
	ORIG	identificação do vértice origem
	EST	identificação do vértice estação
	PV	identificação do vértice visado
	CodGiro	código do giro a que pertence a ligação

Continua na próxima página

Folha	**Atributo**	**Descrição**
	outrosCodGiro	designações alternativas para o giro
LeitDesn	NomeRede	nome da rede altimétrica
	PontoAnter	vértice atrás na ligação da rede *NomeRede*
	PontoPoster	Vértice à frente na ligação da rede *NomeRede*
	CampNome	nome da campanha em que é feita a leitura
	ValorDesnivel	valor do desnível entre os vértices *PontoAnter* e *PontoPoster*
	NumOrdem	posição da ligação
LeitHz	OR	identificação do vértice origem da ligação
	EST	identificação do vértice estação da ligação
	PV	identificação do vértice visado da ligação
	CodGiro	código do giro da ligação
	Campanhas	uma coluna para cada campanha observada. Na coluna de cada campanha é registado o valor do ângulo horizontal
LeitEst	CamNome	nome da campanha em que foram registados os parâmetro meteorológicos
	NomeRede	nome da rede planimétrica
	CodGiro	identificação do giro da rede *NomeRede*
	Temperatura	valor da temperatura ambiente
	Pressão	valor da pressão atmosférica
	Humidade	valor da humidade relativa
LeitDist	HorizEspac	distância horizontal / espacial
	NomeRede	nome da rede geodésica em que é medida a distância
	EST	identificação do vértice estação
	PV	identificação do vértice visado
	NumOrdem	número da posição da ligação na leitura de distâncias

Continua na próxima página

Folha	**Atributo**	**Descrição**
	Campanhas	uma coluna para cada campanha observada. Em cada linha regista-se o valor da distância entre *EST* e *PV* na campanha de cada coluna
LeitV	NomeRede	nome da rede tridimensional
	EST	identificação do vértice estação
	PV	identificação do vértice visado
	NumOrdem	posição da ligação na rede tridimensional
	Alvo	identificação do alvo usado na leitura
	Instrumento	identificação do instrumento usado na leitura
	Campanhas	uma coluna para cada campanha observada. Em cada linha regista-se o valor do ângulo vertical registado na campanha de cada coluna.
ResCampRef	CtrlInicial	valor inicial do intervalo de controlo
	CtrlFinal	Valor final do intervalo de controlo
	CampNome	identificação da campanha em que foram realizadas as leituras
	NomeRede	identificação da rede geodésica
	CampRef	identificação da campanha de referência usada no cálculo de resultados
ResCen	NomeRede	nome da rede geodésica na qual o vértice é fixo
	CampNome	identificação da campanha
	PontoNome	identificação do vértice fixo
	PesoX	peso do deslocamento na direcção xx
	dX	deslocamento na direcção xx
	PesoY	peso do deslocamento na direcção yy
	dY	deslocamento na direcção yy
	PesoZ	peso do deslocamento na direcção zz
	dZ	deslocamento na direcção zz

Continua na próxima página

Folha	**Atributo**	**Descrição**
ResAlt	CtrlCampRef	valor de controlo para a campanha de referência
	PontoNome	identificação do vértice
	Campanhas	uma coluna para cada campanha observada. Em cada linha regista-se o valor da cota de cada vértice na respectiva campanha
ResPlan	CtrlCampRef	valor de controlo para a campanha de referência
	PontoNome	identificação do vértice
	Campanhas	acrescentam-se três colunas para as coordenadas x, y e z de cada campanha observada. Em cada linha regista-se o valor das coordenadas x, y e z de cada vértice na respectiva campanha
DefPontoHist	HistNum	código de identificação de histórico na folha *DefPontos*
	PontoNome	designação antiga do vértice
	CampInic	identificação da campanha a partir da qual a designação do vértice é válida
	CampFim	identificação da campanha a partir da qual esta designação foi descontinuada
	Motivo	motivo da alteração do vértice
Folha de Controlo	Nome Folha	nome da folha *Excel*
	Nº de Linhas	número de linha do ficheiro *Excel*
	Nº de Colunas	número de colunas do ficheiro *Excel*

Tabela D.2: Arquivo de informação das observações geodésicas: estrutura da folha *Excel*

Em seguida, descreve-se em detalhe a estrutura de cada uma das folhas que compõem os ficheiros *Excel*.

D.2.4.1 Folha Camp

Contém informação sobre todas as campanhas de observações geodésicas realizadas na barragem.

A Figura D.1 apresenta um exemplo da Folha Camp, que contém as seguintes colunas:

- CampNum: número inteiro único que identifica as épocas para uma barragem.

	A	B	C	D	E	F
1	**CampNum**	**CampNome**	**Data**	**CotaAgua**	**Outras**	**Coments**
2	1	A00_Niv	2002-02-14	0,00		**Campanha artificial para o cálculo dos desníveis**
3	2	A01	2002-02-14	86,5		**Poligonal do coroamento**
4	3	A02	2002-02-19	89		**Nivelamento no coroamento**

Figura D.1: Folha Camp

- CampNome: nome único que identifica as épocas. CampNome não pode estar vazio, dado que todas as referências a campanhas, nas outras folhas Excel do ficheiro, são feitas usando este campo.
- Data: data em que a campanha foi realizada.
- CotaAgua: cota da Água a montante.
- Outras: outras designações para a campanha. Corresponde a um nome único que identifica as épocas para uma barragem, segundo a designação antiga (apenas para o caso de existir mais do que uma designação).
- Coments: comentários relativos à campanha.

D.2.4.2 Folha DefPontos

Contém informação sobre todos os vértices usados em redes geodésicas da barragem. Um vértice corresponde a um ponto com determinadas coordenadas.

	A	B	C	D	E	F	G	H
1	**PontoNome**	**X**	**Y**	**Z**	**Fi**	**TipoPonto**	**FuncaoRede**	**Centragem**
2	PD1	-49369,6	541157,4	0,0	500,00	Pilar		W
3	PD	57967,2	504793,2	0,0	500,00	Pilar		W
4	P1	100874,4	546504,3	0,0	-42,81	Pilar		W

Figura D.2: Folha DefPontos

A Figura D.2 apresenta um exemplo da Folha DefPontos, que contém as seguintes colunas:

- PontoNome: nome único que identifica um vértice da rede. Este campo não pode estar vazio, dado que todas as referências para os vértices são feitas através do nome dos mesmos.

- X: valor da coordenada segundo o eixo dos xx.
- Y: valor da coordenada segundo o eixo dos yy.
- Z: valor da coordenada segundo o eixo dos zz, isto é, cota do vértice.
- Fi: ângulo de rotação do vértice.
- TipoPonto: tipo do vértice, como por exemplo, taco ou pilar.
- FuncaoRede: função exercida pelo vértice, como por exemplo, objecto ou auxiliar.
- Centragem: tipo de centragem do vértice (Kern ou Wild).
- HistNum: código usada para controlo do histórico de designações do vértice.

D.2.4.3 Folha DefNiv

Contém a definição das linhas de nivelamento (redes altimétricas) da barragem. Cada linha de nivelamento é composta por um conjunto de ligações entre dois vértices, que assumem o papel de Ponto Anterior (ou ponto atrás) e Ponto Posterior (ou ponto à frente). É importante garantir a ordem entre as ligações de cada rede.

	A	B	C	D
1	NomeRede	PontoAnter	PontoPoster	NumOrdem
2	LN_Crmt	PD1-niv	s1	1
3	LN_Crmt	s1	p1-niv	2
4	LN_Crmt	p1-niv	p2-niv	3

Figura D.3: Folha DefNiv

A Figura D.3 apresenta um exemplo da Folha DefNiv. Cada linha do ficheiro corresponde a uma ligação da linha de nivelamento e contém as seguintes colunas:

- NomeRede: nome que identifica a linha de nivelamento. Todas as ligações de uma linha de nivelamento têm o mesmo nome nesta coluna.
- PontoAnter: identificação do vértice que assume o papel de ponto atrás em cada ligação. Este valor tem que existir na coluna PontoNome da folha DefPontos.

- PontoPoster: identificação do vértice que assume o papel de ponto à frente em cada ligação. Este valor tem que existir na coluna PontoNome da folha DefPontos.

- NumOrdem: número inteiro que determina a posição de cada ligação na rede. Não podem existir duas ligações na mesma rede com um NumOrdem igual.

D.2.4.4 Folha DefPlan

Contém a definição das redes planimétricas (redes bidimensionais) da barragem. Cada rede planimétrica é composta por um conjunto de ligações em vários giros.

	A	B	C	D	E	F
1	**NomeRede**	**ORIG**	**EST**	**PV**	**CodGiro**	**OutrosCodGiro**
2	Pol_Cormt	PD1	PD	P1	1	
3	Pol_Cormt	PD1	PD	P2	1	
4	Pol_Cormt	PD1	PD	P3	1	

Figura D.4: Folha DefPlan

A Figura D.4 apresenta um exemplo da Folha DefPlan, que contém as seguintes colunas:

- NomeRede: nome que identifica a rede planimétrica. Todas as ligações de uma rede planimétrica têm o mesmo nome nesta coluna.

- ORIG: identificação do vértice que assume o papel de ponto origem em cada ligação. Este valor tem que existir na coluna PontoNome da folha DefPontos.

- EST: identificação do vértice que assume o papel de ponto estação em cada ligação. Este valor tem que existir na coluna PontoNome da folha DefPontos.

- PV: identificação do vértice que assume o papel de ponto visado em cada ligação. Este valor tem que existir na coluna PontoNome da folha DefPontos.

- CodGiro: código que identifica o giro a que pertence a ligação.

- OutrosCodGiro: outras designações que possam identificar o giro.

D.2.4.5 Folha LeitDesn

Contém todas as leituras de desníveis registados nas redes altimétricas. Para cada ligação definida na folha DefNiv e para cada campanha, contém o valor do desnível verificado entre os vértices ponto atrás e ponto à frente.

	A	B	C	D	E	F
1	**NomeRede**	**PontoAnter**	**PontoPoster**	**CampNome**	**ValorDesnivel**	**NumOrdem**
2	LN_Crmt	PD1-niv	s1	A06	-1522,75	1
3	LN_Crmt	s1	p1-niv	A06	-1005,3	2
4	LN_Crmt	p1-niv	p2-niv	A06	-1575,3	3

Figura D.5: Folha LeitDesn

A Figura D.5 apresenta um exemplo da Folha LeitDesn, que contém as seguintes colunas:

- NomeRede: nome da rede altimétrica, definida na folha DefNiv, a que pertence a ligação.
- PontoAnter: nome do vértice que assume o papel de ponto anterior na ligação. Este valor tem que existir na coluna PontoNome da folha DefPontos.
- PontoPoster: nome do vértice que assume o papel de ponto posterior na ligação. Este valor tem que existir na coluna PontoNome da folha DefPontos.
- CampNome: Nome da campanha geodésica na qual é feita a leitura do desnível. Este valor tem que existir na coluna nampNome da folha Camp.
- ValorDesnivel: valor do desnível entre os vértices PontoAnter e PontoPoster na Campanha CampNome.
- NumOrdem: número de ordem da ligação.

É importante destacar que as colunas (NomeRede, PontoAnter, PontoPoster, NumOrdem) devem estar definidas na folha DefNiv.

D.2.4.6 Folha LeitHz

Contém todas as leituras de ângulos horizontais registados nas redes planimétricas. Para cada ligação definida na folha DefPlan existe uma linha nesta folhas. Para além disso, acrescenta-se uma coluna com o valor do ângulo registado em cada campanha.

	A	B	C	D	E	F	G
1	**NomeRede**	**OR**	**EST**	**PV**	**CodGiro**	**A01**	**A02**
2	Pol_Cormt	PD1	PD	P1	1	1301063,433	1301075,9
3	Pol_Cormt	PD1	PD	P2	1	1331728,25	1331735,3
4	Pol_Cormt	PD1	PD	P3	1	1374729,067	1374762,417
5	Pol_Cormt	PD	P1	P2	2	2063550,533	2063552,183
6	Pol_Cormt	PD	P1	P3	2	2105325,367	2105359,4
7	Pol_Cormt	P1	P2	P3	3	2069715	2069773,125
8	Pol_Cormt	P1	P2	PD	3	3967122,45	3967117,558

Figura D.6: Folha LeitHz

A Figura D.6 apresenta um exemplo da Folha LeitHz que contém as seguintes colunas:

- NomeRede: nome da rede planimétrica, definida na folha DefPlan, a que pertence a ligação.
- OR: nome do vértice que assume o papel de ponto origem na ligação. Este valor tem que existir na coluna PontoNome da folha DefPontos.
- EST: nome do vértice que assume o papel de ponto estação na ligação. Este valor tem que existir na coluna PontoNome da folha DefPontos.
- PV: nome do vértice que assume o papel de ponto visado na ligação. Este valor tem que existir na coluna PontoNome da folha DefPontos.
- CodGiro: código que identifica o giro a que pertence a ligação.
- Nomes das campanhas: para cada campanha geodésica em que foram observadas redes planimétricas, acrescenta-se uma coluna com o nome da campanha. Em cada linha, o valor para essa coluna corresponde ao ângulo horizontal registado na campanha respectiva.

É importante destacar que as colunas (NomeRede, OR, EST, PV, CodGiro) devem estar definidas na folha DefPlan.

D.2.4.7 Folha LeitEst

Esta folha contém informação acerca das estações meteorológicas (temperatura, pressão e humidade). Cada linha corresponde ao valor registado na estação meteorológica durante a observação de cada giro das redes planimétricas.

	A	B	C	D	E	F
1	CampNome	NomeRede	CodGiro	Temperatura	Pressão	Humidade
2	**A01**	Pol_Cormt	1	14,2		
3	**A02**	Pol_Cormt	2	15		
4	**A03**	Pol_Cormt	3	13,8		

Figura D.7: Folha LeitEst

A Figura D.7 apresenta um exemplo da Folha LeitEst, que contém as seguintes colunas:

- CampNome: nome da campanha geodésica na qual foi feito o registo dos parâmetros meteorológicos. Este valor tem que existir na coluna CampNome da folha Camp.
- NomeRede: nome da rede planimétrica definida na folha DefPlan a que pertence o giro.
- CodGiro: identificação do giro.
- Temperatura: valor da temperatura ambiente.
- Pressão: valor da pressão atmosférica.
- Humidade: valor da humidade relativa.

D.2.4.8 Folha LeitDist

Esta folha contém informação sobre as distâncias calculadas entre um ponto estação e um ponto visado numa determinada rede geodésica.

	A	B	C	D	E	F	G
1	**HorizEspac**	**NomeRede**	**EST**	**PV**	**NumOrdem**	**A01**	**A02**
2	**TipoDist**					1	1
3	H	Pol_Cormt	PD1	PD	1	113329,0	113329,2
4	H	Pol_Cormt	PD	PD1	2	113329,7	113329,2
5	H	Pol_Cormt	PD	P1	3	59838,3	59838,8
6	H	Pol_Cormt	PD	P2	4	115492,4	115492,8
7	H	Pol_Cormt	PD	P3	5	198261,4	198259,1
8	H	Pol_Cormt	P1	PD	6	59838,4	59838,8

Figura D.8: Folha LeitDist

A Figura D.8 apresenta um exemplo da Folha LeitDist, que contém as seguintes colunas:

- HorizEspac: informa se a distância é horizontal ou espacial.
- NomeRede: nome da rede geodésica na qual foi registada a distância.
- EST: vértice que assume o papel de ponto estação. Este valor tem que existir na coluna PontoNome da folha DefPontos.
- PV: vértice que assume o papel de ponto visado. Este valor tem que existir na coluna PontoNome da folha DefPontos.
- NumOrdem: número de ordem que permite determinar a posição da ligação na observação de distâncias.
- Nomes das campanhas: para cada campanha geodésica em que foram observadas distâncias, acrescenta-se uma coluna com o nome da campanha. Em cada linha, o valor para esta coluna corresponde à distância entre os vértices EST e PV.

Na segunda linha desta folha (TipoDist), determina-se o tipo de distância verificada em cada época, que pode ser corrigida ou não corrigida.

D.2.4.9 Folha LeitV

A folha LeitV contém informação sobre ângulos verticais (direcções zenitais) registados em redes tridimensionais. Neste caso, ao contrário do que sucede com as linhas de

nivelamento e as redes planimétricas, não existe nenhuma folha com a definição das redes tridimensionais.

Cada linha do ficheiro contém uma ligação entre um ponto estação e um ponto visado numa rede tridimensional, bem como as correspondentes leituras efectuadas nas várias campanhas.

	A	B	C	D	E	F	G	H
1	NomeRede	EST	PV	NumOrdem	Alvo	Instrumento	A01	A02
2	TRIA	PRCE	PRCD	1	AK162	E2	999895,8	999860,5
3	TRIA	PRCE	14J72	2	AK16	E2	997162,1	996732,0
4	TRIA	PRCE	14M72	3	AK16	E2	997830,0	997421,5
5	TRIA	PRCE	PRJE	4	AK173	E2		
6	TRIA	PRCE	PRJD	5	AK173	E2		
7	TRIA	PRCE	PRCD	6	AK162	E2		
8	TRIA	PRCD	PRCE	7	AK162	E2	1000270,8	1000259,5

Figura D.9: Folha LeitV

A Figura D.9 apresenta um exemplo da Folha LeitV, que contém as seguintes colunas:

- NomeRede: nome que identifica a rede tridimensional. Todas as ligações de uma rede tridimensional têm o mesmo nome nesta coluna.
- EST: identificação do vértice que assume o papel de ponto estação em cada ligação. Este valor tem que existir na coluna PontoNome da folha DefPontos.
- PV: identificação do vértice que assume o papel de ponto visado em cada ligação. Este valor tem que existir na coluna PontoNome da folha DefPontos.
- NumOrdem: número que determina a ordem da ligação na rede tridimensional.
- Alvo: identificação/nome do alvo usado no ponto visado para a determinação do ângulo vertical.
- Instrumento: identificação do instrumento usado na medição.
- Nomes das campanhas: para cada campanha geodésica em que foram observadas redes tridimensionais, acrescenta-se uma coluna com o nome da campanha. Em cada linha, o valor para esta coluna corresponde ao ângulo vertical registado nessa campanha.

D.2.4.10 Folha ResCampRef

Os resultados geodésicos (coordenadas) são calculados a partir das leituras. Para fazer esse cálculo, é necessário usar os resultados armazenados numa outra campanha, designada por campanha de referência. Esta folha contém informação relativa às campanhas de referência usadas no cálculo de cada resultado.

	A	B	C	D	E
1	CtrlInicial	CtrlFinal	**CampNome**	**NomeRede**	**CampRef**
2	1	14	A01	Pol_Cormt	A01
3	1001	1032	A02	LNIV_Crmt	A00_Niv
4	1	14	A03	Pol_Cormt	A01
5	15	28	A03	Pol_G4	A01
6	1001	1032	A03	LNIV_Crmt	A00_Niv
7	1033	1066	A03	LNIV_G2	A00_Niv
8	1067	1097	A03	LNIV_G3	A00_Niv
9	1098	1127	A03	LNIV_G4	A00_Niv

Figura D.10: Folha ResCampRef

A Figura D.10 apresenta um exemplo da Folha ResCampRef, que contém as seguintes colunas:

- CtrlInicial: número de controlo que é referenciado pelos registos de resultados nas folhas ResPlan e ResAlt.
- CtrlFinal: número de controlo que é referenciado pelos registos de resultados nas folhas ResPlan e ResAlt. A informação constante de cada linha corresponde a um intervalo de resultados compreendido entre os valores CtrlInicial e CtrlFinal.
- CampNome: nome da campanha em que foram observadas as leituras. Este valor tem que existir na coluna CampNome da folha Camp.
- NomeRede: nome da rede geodésica. Este campo é apenas informativo.
- CampRef: nome da campanha de referência usada no cálculo dos resultados para a campanha definida em CampNome.

D.2.4.11 Folha ResCen

A folha ResCen contém informação sobre os deslocamentos das direcções x, y e z, bem como sobre os pesos atribuídos aos vértices fixos (ou centros) de cada rede em cada campanha. Cada linha do ficheiro corresponde a um vértice fixo de uma rede geodésica numa campanha.

	A	B	C	D	E	F	G	H	I
1	**NomeRede**	**CampNome**	**PontoNome**	**PesoX**	**dX**	**PesoY**	**dY**	**PesoZ**	**dZ**
2	Pol_Cormt	A03	PD					0,0	0,0
3	Pol_Cormt	A03	PE					0,0	0,0
4	LNIV_Crmt	A03	R1					.1	0,0
5	Pol_Cormt	A04	PD					0,0	0,0
6	Pol_Cormt	A04	PE					0,0	0,0
7	Pol G4	A04	19-20					0,0	0,0

Figura D.11: Folha ResCen

A Figura D.11 apresenta um exemplo da Folha ResCen, que contém as seguintes colunas:

- NomeRede: nome da rede geodésica na qual o vértice é fixo.
- CampNome: identificação da campanha geodésica.
- PontoNome: identificação do vértice ou ponto fixo.
- PesoX: peso do deslocamento do vértice segundo a direcção do eixo dos xx.
- dX: deslocamento do vértice fixo segundo a direcção do eixo dos xx.
- PesoY: peso do deslocamento do vértice segundo a direcção do eixo dos yy.
- dY: deslocamento do vértice fixo segundo a direcção do eixo dos yy.
- PesoZ: peso do deslocamento do vértice segundo a direcção do eixo dos zz.
- dZ: deslocamento do vértice fixo segundo a direcção do eixo dos zz.

D.2.4.12 Folha ResAlt

Contém informação sobre todos as coordenadas calculadas nos vértices das linhas de nivelamento. Uma vez que as redes são altimétricas, só é possível determinar a cota de cada vértice.

	A	B	C	D	E	F
1	CtrlCampRef	PontoNome	A02	A03	A04	A05
2	1001	PD1-Niv	14998,2	14998,3	14998,2	14998,2
3	1002	s1	13477,3	13475,4	13475,3	13474,9
4	1003	P1-Niv	12470,9	12470,1	12470,1	12469,7
5	1004	P2-Niv	10896,8	10894,7	10894,7	10894,0
6	1005	R1	10000,0	10000,0	10000,3	10000,0
7	1006	PD-Niv	8359,2	8359,1	8359,4	8359,2

Figura D.12: Folha ResAlt

A Figura D.12 apresenta um exemplo da Folha ResAlt, que contém as seguintes colunas:

- CtrlCampRef: valor de controlo para a determinação da campanha de referência. Este valor, em conjunto com a identificação da campanha, permite referenciar a campanha de referência na folha ResCampRef. Note-se que o valor desta coluna é mapeado num intervalo entre as colunas CtrlInicial e CtrlFinal da folha ResCampRef.

- PontoNome: identificação do vértice a que corresponde a cota determinada em cada campanha.

- Nomes das campanhas: para cada campanha geodésica em que foram observadas linhas de nivelamento, acrescenta-se uma coluna com o nome da campanha. Em cada linha, o valor para esta coluna corresponde à cota do vértice calculada nessa campanha.

D.2.4.13 Folha ResPlan

Contém informação sobre todas as coordenadas calculadas nos vértices das redes planimétricas e tridimensionais. Com as redes planimétricas é possível determinar as coordenadas x e y de cada vértice. A coordenada z (cota) só é calculada caso existam leituras verticais.

	A	B	C	D	E	F	G	H
1	**CtrlCampRef**	**PontoNome**	A01X	A01Y	A01Z	A03X	A03Y	A03Z
2	1	PD1						
3	2	PD	158058,5	504625,3		158058,5	504625,3	
4	3	P1	200898	546406		200896,2	546404,8	
5	4	P2	244530,4	581188,8		244525,4	581188	
6	5	P3	315034	625729,2		315024,1	625745,6	
7	6	P4				400180,3	647314,9	
8	7	P5				476027,7	636710,4	

Figura D.13: Folha ResPlan

A Figura D.13 apresenta um exemplo da Folha ResPlan, que contém as seguintes colunas:

- CtrlCampRef: valor de controlo para a determinação da campanha de referência. Este valor, em conjunto com a identificação da campanha, permite referenciar a campanha de referência na folha ResCampRef. Note-se que o valor desta coluna é mapeado num intervalo entre as colunas CtrlInicial e CtrlFinal da folha ResCampRef.

- PontoNome: identificação do vértice a que correspondem as coordenadas determinadas em cada campanha.

- Coordenadas nas campanhas: para cada campanha geodésica acrescentam-se três colunas para registo das coordenadas x, y e z. A designação corresponde ao nome da Campanha seguido da identificação da coordenada. No exemplo apresentado, A01X, A01Y e A01Z correspondem, respectivamente, às coordenadas x, y e z na campanha de nome A01.

D.2.4.14 Folha DefPontosHist

O mesmo vértice pode ser identificado por diferentes designações em épocas distintas, devido a reformulações aplicadas nas redes geodésicas. Em cada registo do ficheiro é feito o mapeamento entre o nome antigo do vértice e a sua designação actual, bem como o intervalo de campanhas em que essa designação foi usada.

	A	B	C	D	E
1	**HistNum**	**PontoNome**	**CampInic**	**CampFim**	**Motivo**
2	1	06-07A-G3	2002-10-01	2003-01-20	ponto coberto por betão
3	1	06-07-G3	2004-01-19		ponto original foi danificado quando foi retirado betão que o cobria
4	2	21-22A-G4	2002-10-01	2002-10-01	ponto danificado
5	2	21-22-G4	2003-01-20		ponto original foi danificado

Figura D.14: Folha DefPontosHist

A Figura D.14 apresenta um exemplo da Folha DefPontosHist, que contém as seguintes colunas:

- HistNum: código numérico de controlo e identificação do registo. Este valor identifica o vértice na folha DefPontos.
- PontoNome: designação antiga do vértice. Esta designação também pode ser referenciada nas outras folhas.
- CampInic: campanha a partir da qual esta designação é válida.
- CampFim: campanha a partir da qual esta designação foi descontinuada.
- Motivo: informação sobre o motivo que esteve na origem da alteração da designação do vértice.

D.2.4.15 Folha de Controlo

A folha de controlo contém informação sobre a dimensão das restantes folhas existentes no ficheiro Excel.

A Figura D.15 apresenta um exemplo da Folha de Controlo. Note-se que para qualquer folha do ficheiro Excel existe uma linha com o nome da mesma e o número de linhas

	A	B	C
1	Nome Folha	N° de Linhas	N° de colunas
2	Folha de Controlo	16	3
3	Camp	12	6
4	DefPontos	180	8
5	DefNiv	320	4
6	DefPlan	47	6
7	LeitDesn	946	6
8	LeitHz	47	11
9	LeitEst	1	6
10	LeitDist	78	12
11	LeitV	1	6
12	ResCampRef	40	5
13	ResCen	42	9
14	ResAlt	152	10
15	ResPlan	29	23
16	DefPontosHist	5	5

Figura D.15: Folha de Controlo

e colunas nela usados. Desta forma, é possível controlar a informação lida durante a fase de extracção.

D.3 Gestão de Modelos Matemáticos e Físicos

No controlo de segurança de barragens de betão e alvenaria, o Núcleo de Modelação Matemática e Física (NMMF) centra a sua actividade no desenvolvimento e aplicação de modelos matemáticos e físicos para análise do comportamento estrutural das barragens para cenários correntes e de rotura.

A actividade experimental do NMMF consiste na execução de ensaios em modelos físicos a escala reduzida de barragens nos quais são simuladas as diferentes grandezas intervenientes num determinado cenário em estudo (geometria, propriedades dos materiais, acções, etc.). Estes ensaios têm como objectivo principal a determinação do comportamento do protótipo para diferentes cenários. Estes resultados podem ser utilizados para a validação dos modelos matemáticos, essencialmente, para situações de

colapso da barragem ou da fundação.

O NMMF é responsável pela manutenção de vários pacotes de programas (alguns desenhados e desenvolvidos no LNEC e outros comerciais), que utilizam técnicas de modelação matemática (e.g., modelos de elementos finitos). Os resultados obtidos com a aplicação destes programas podem ser visualizados tanto numérica como graficamente. Estes modelos destinam-se a prever e a interpretar o comportamento de uma barragem quando submetida a um determinado conjunto de solicitações. Para o controlo de segurança das barragens, os resultados dos modelos são comparados com os obtidos a partir das observações registadas no campo (que se encontram armazenadas no SIOBE). Esta comparação permite calibrar os modelos matemáticos, que podem ser utilizados para previsão do comportamento e, desta forma, verificar se a estrutura da barragem se está a comportar de acordo com o expectável. A informação utilizada como entrada e os resultados produzidos pelos modelos matemáticos fica armazenada em ficheiros ASCII.

Apêndice E

Base de dados do gestBarragens

E.1 Observações

A Figura E.1 ilustra o modelo ER[1] simplificado da base de dados alvo para o sistema de observações. Este modelo é simplificado por não incluir todas as entidades envolvidas e também por as entidades apresentarem apenas um subconjunto dos seus atributos.

Cada barragem (entidade obra) é constituída por um conjunto de elementos de obra. Por sua vez, cada elemento de obra pode ser composto por um conjunto de subelementos de obra. Existem vários tipos de elementos, como Fundações, Centrais e Tomadas de Água, bem como vários tipos de subelementos, como Pilares, Galerias e Blocos. No total, existem 16 tipos de elementos e 22 tipos de subelementos possíveis.

No modelo apresentado, optou-se por não representar os vários tipos de elementos e subelementos de obra, já que a sua representação é complexa e desnecessária para a compreensão do processo de migração.

Uma campanha corresponde a um período de tempo durante o qual os instrumentos de uma determinada barragem são observados. Deste modo, cada campanha está necessariamente associada a uma barragem (no modelo ER, poderia considerar-se a campanha como uma entidade fraca de obra).

As leituras correspondem à observação de instrumentos numa determinada campanha. No total, existem 21 tipos de instrumentos instalados numa barragem e, con-

[1]Modelo Entidade-Relação, usando a nomenclatura definida em [SKS02].

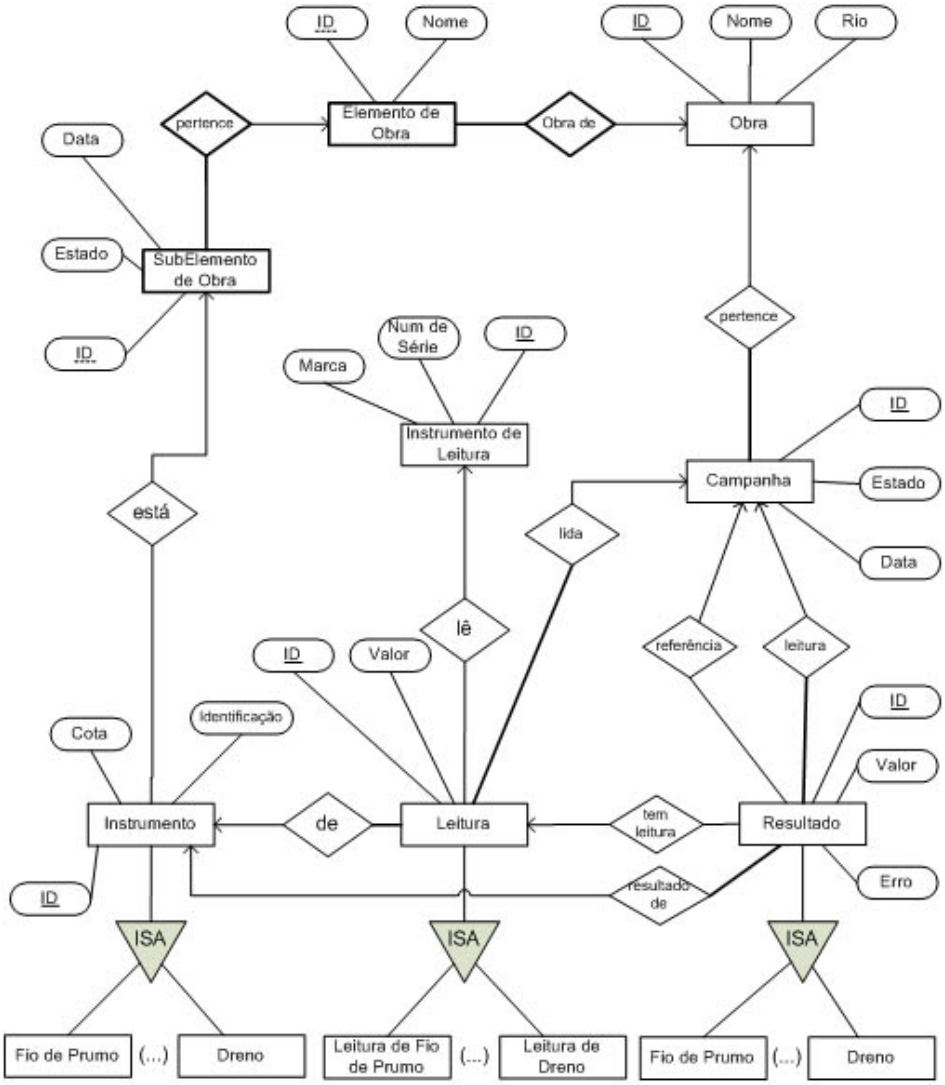

Figura E.1: Modelo ER do Sistema de Observação

sequentemente, 21 tipos de leituras diferentes. Por uma questão de simplificação, no modelo ER apenas estão representados os instrumentos *Fio de Prumo* e *Dreno*.

Cada leitura é efectuada por um determinado instrumento que, no modelo se designa por *Instrumento de Leitura*. Contudo, esta informação não consta do arquivo de dados do SIOBE, pelo que não faz parte do processo de migração.

Conceptualmente, os resultados são grandezas geradas a partir das leituras. No entanto, uma vez que as leituras não eram armazenadas directamente no SIOBE, é possível existirem resultados sem qualquer leitura associada.

A cada resultado está associada a campanha na qual foi feita a leitura que lhe deu origem, bem como a campanha usada para o cálculo do mesmo. Note-se que também existem 21 tipos de resultados diferentes, conforme o tipo de instrumento a que pertencem.

Cada instrumento tem uma associação com o subelemento de obra, o que permite determinar a localização do instrumento na estrutura da barragem.

É importante salientar que existem alguns tipos de instrumento, como é o caso dos *Extensómetros de Resistência*, que se encontram associados num grupo. Deste modo, cada *Extensómetro de Resistência* pertence a um grupo de extensómetros que apresenta determinadas características. Note-se que este conceito não se encontra modelado no ER apresentado na Figura E.1.

E.2 Observações Geodésicas

A Figura E.2 ilustra o modelo ER[2] simplificado da base de dados alvo para o sistema de observações geodésicas. Este modelo é simplificado por não incluir todas as entidades envolvidas e por as entidades dele constantes apresentarem apenas um subconjunto dos seus atributos.

Em cada barragem está instalado um conjunto de redes geodésicas que podem ser linhas de nivelamento, redes bidimensionais e redes tridimensionais.

Cada rede geodésica é composta por um conjunto de vértices que correspondem a pontos materializados na barragem (entidade Obra). A associação entre vértices constitui uma ligação de uma rede geodésica. As ligações que compõem as redes geodésicas podem ser (i)*Ângulos azimutais*, (ii)*Ângulos verticais*, (iii)*Distâncias* e (iv)*Desníveis*. As redes podem ter giros, que constituem um conjunto de ligações ordenadas.

Por uma questão de simplificação, o modelo ER não detalha as relações existentes entre os diferentes tipos de rede e os tipos de ligações. No entanto, enquanto uma linha de nivelamento é composta exclusivamente por ligações de *desníveis*, as redes bidimensionais são compostas por ligações de *ângulos azimutais* e *distâncias* e, finalmente, as redes

[2]Modelo Entidade-Relação, usando a nomenclatura definida em [SKS02]

tridimensionais são redes bidimensionais com ligações de *ângulos verticais*.

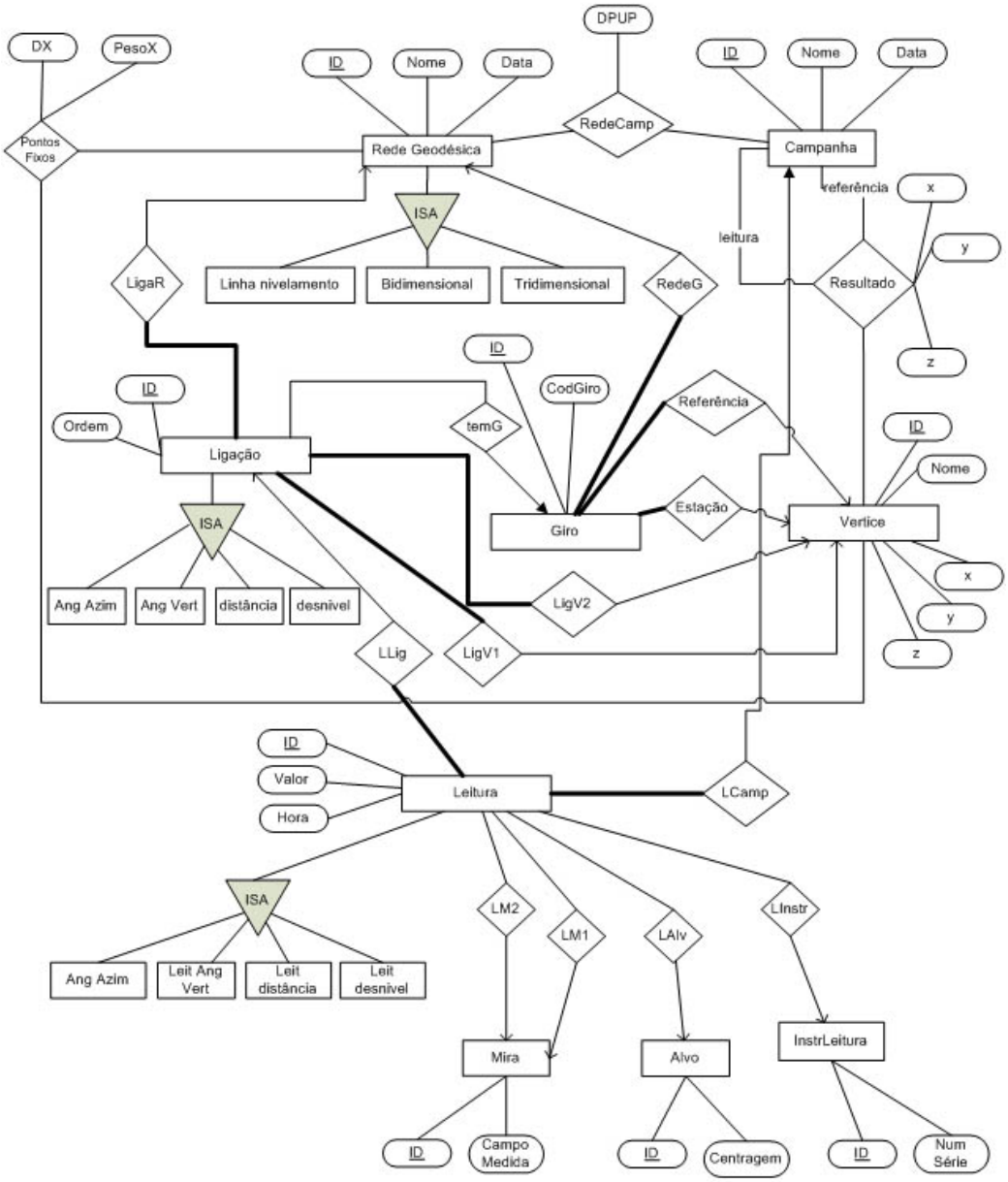

Figura E.2: Modelo ER do Sistema de Observações Geodésicas

Uma campanha corresponde a um período de tempo durante o qual as redes geodésicas são observadas. A observação das redes corresponde à realização de uma leitura para cada uma das ligações que as compõem, sendo que a leitura depende do tipo de ligação (aspecto não modelado no ER). Por outro lado, cada leitura é efectuada por um determinado instrumento de leitura e com recurso a duas miras, no caso da medição de

desníveis, ou a um alvo, nas restantes leituras. Por exemplo, uma distância é medida por um *distanciómetro*, com uso de um alvo reflector.

Como resultado da observação das redes geodésicas, determinam-se, para cada campanha, as coordenadas de cada vértice. A entidade *Resultado* representa as coordenadas calculadas para cada vértice, numa determinada campanha, com recurso aos valores de uma campanha de referência.

Apêndice F

Grafo de transformações para as observações

Nesta Apêndice, descreve-se a sequência de transformações, implementadas no *Ajax*, que permitem fazer a migração dos dados fonte relativos aos *drenos* e *fios de prumo*, para o esquema da base de dados alvo.

A Figura F.1 ilustra o grafo de transformações para efectuar a migração dos dados relativos aos instrumentos do tipo *Dreno* e *Fio de Prumo*. A migração dos restantes 19 tipos de instrumento não se encontra detalhada. No entanto, tendo em conta que a lógica das transformações é semelhante, considera-se que a sequência de transformações relativa aos instrumentos do tipo *Drenos* e *Fios de Prumo* ilustra as transformações efectuadas para os restantes instrumentos.

Como se pode verificar através da observação da Figura F.1, existe uma dependência na sequência das transformações. De forma resumida, dir-se-à que a migração começa pela definição dos instrumentos provenientes das tabelas de apoio, em seguida tem lugar a transformação dos ficheiros de leitura e, finalmente, a geração dos resultados.

Na migração dos instrumentos existem duas classes: instrumentos sem grupo e instrumentos com grupo. Os *Drenos* representam a migração de instrumentos sem grupo, enquanto os *Fios de Prumo* representam a migração de instrumentos com grupo.

A Figura F.2 ilustra o grafo de transformações para os *drenos*.

Basta um operador *Map* para efectuar o carregamento para a tabela final de drenos

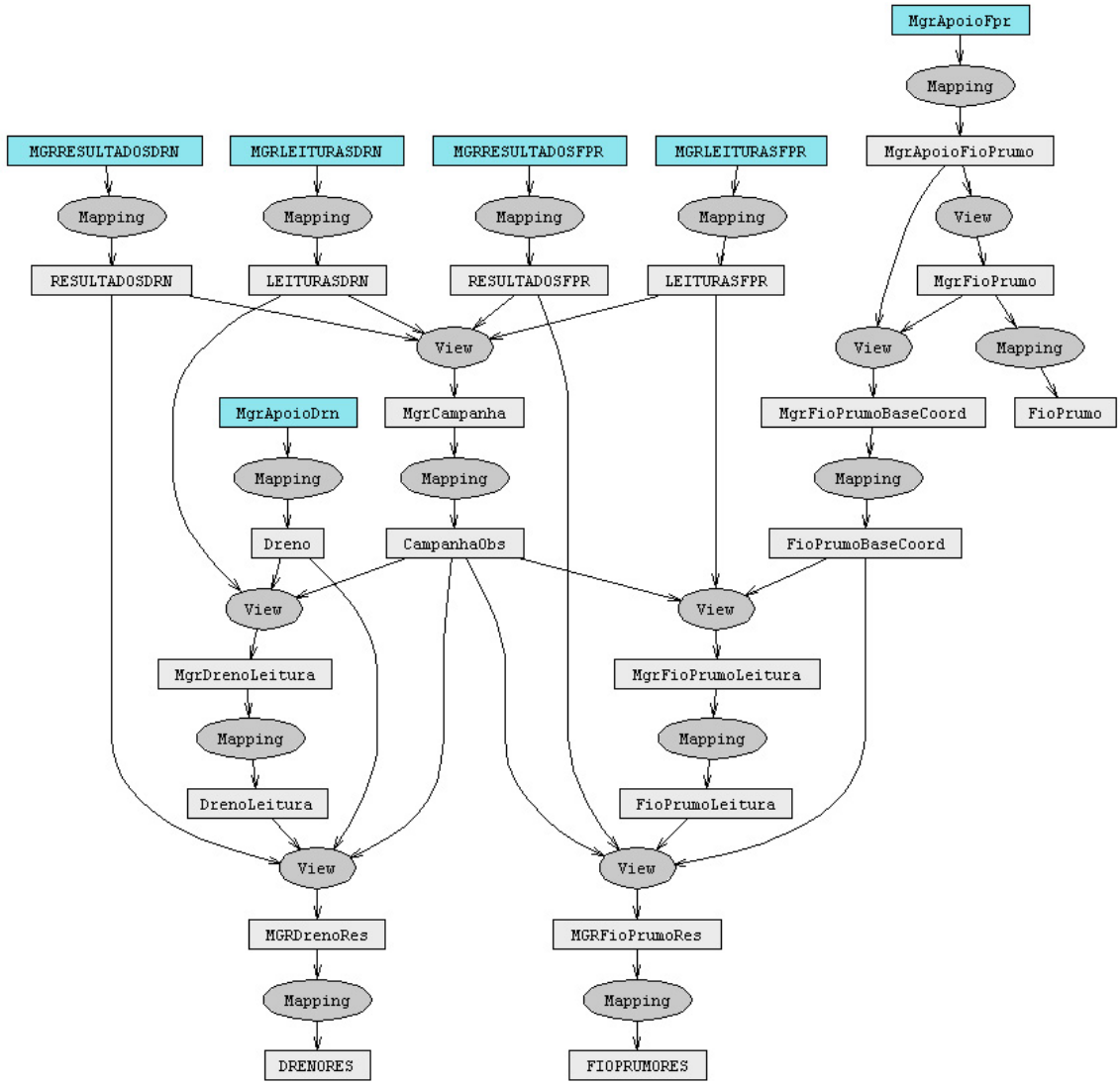

Figura F.1: Grafo Completo para *Dreno* e *Fio de Prumo*

designada por *Dreno*.

Este operador é responsável por extrair a informação da relação de entrada através de um conjunto de funções invocadas na cláusula *let*. É gerado um novo identificador para cada registo, com recurso à função *Java* `generateId(int)`.

Uma vez que a tabela produzida por este operador é uma tabela do esquema alvo, é necessário determinar as condições para assegurar a actualização de registos, que, neste caso, é feita pelos atributos *Obra*, que identifica univocamente uma barragem, e *codigosiobe* que, para cada barragem, identifica univocamente o instrumento.

O código que se segue corresponde à definição desta transformação.

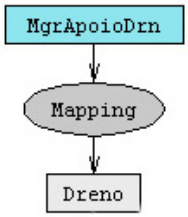

Figura F.2: Migração de instrumentos do tipo *Dreno*

```
CREATE MAPPING Dreno
FROM MgrApoioDrn Drn
LET keyId = generateId(2),
 dreno = getDreno(Drn.Texto, 0, 10),
 bloco = getBloco(Drn.Texto, 11, 15),
 codigosiobe = getCodigo(Drn.Texto, 16, 18),
 oe = getOe(Drn.Texto, 19, 19),
 cota = getCota(Drn.Texto, 20, 26, 2),
 caudali = getCaudalInit(Drn.Texto, 27, 36, 3),
 caudalmax = getCaudalMax(Drn.Texto, 37, 46, 3)
{SELECT keyId AS ID, Drn.Obra, dreno, bloco, codigosiobe, oe,
        cota, caudali, caudalmax
KEY ID
INCREMENT BY Obra, codigosiobe
}
```

A Figura F.3 ilustra o grafo de transformações para os *Fios de Prumo*, que representam os instrumentos que têm grupo.

São necessárias cinco transformações para efectuar o carregamento dos *Fios de Prumo*.

A primeira transformação, que é semelhante ao mapeamento dos *Drenos*, consiste na extracção de informação da tabela fonte através da aplicação de um conjunto de funções. Também é gerado um novo identificador para cada registo.

O código a seguir indicado implementa esta transformação.

```
CREATE MAPPING MgrApoioFioPrumo
```

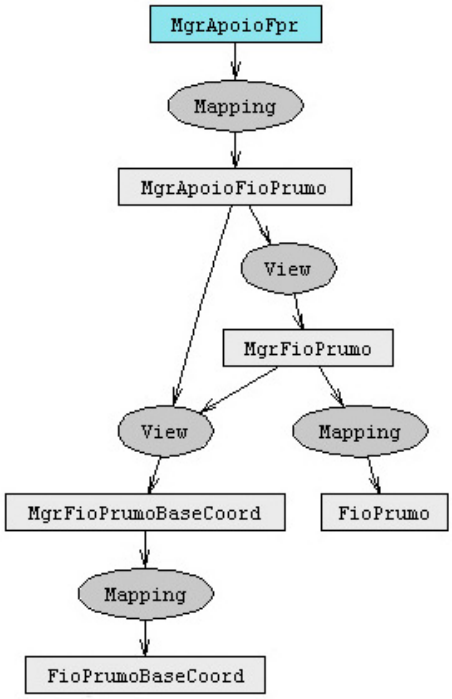

Figura F.3: Fio de Prumo

```
FROM MgrApoioFpr Fpr
LET keyId = generateId(1),
 designacao = getString(Fpr.Texto, 0, 4),
 grupo = getGrupoFpr(Fpr.Texto),
 bloco = getString(Fpr.Texto, 5, 9),
 codigo = getInt(Fpr.Texto, 10, 12),
 oe = getInt(Fpr.Texto, 13, 13),
 fr = getInt(Fpr.Texto, 14, 15),
 ft = getInt(Fpr.Texto, 16, 17),
 cota = getDouble(Fpr.Texto, 18, 24, 2),
 angtangcorrecttanginc = getDouble(Fpr.Texto, 25, 30, 3),
 rfi = getDouble(Fpr.Texto, 31, 35, 2),
 rfmax = getDouble(Fpr.Texto, 36, 40, 2),
 rfmin = getDouble(Fpr.Texto, 41, 45, 2),
 tfi = getDouble(Fpr.Texto, 46, 50, 2),
```

```
 tfmax = getDouble(Fpr.Texto, 51, 55, 2),
 tfmin = getDouble(Fpr.Texto, 56, 60, 2),
 rci = getDouble(Fpr.Texto, 61, 65, 2),
 rcmax = getDouble(Fpr.Texto, 66, 70, 2),
 rcmin = getDouble(Fpr.Texto, 71, 75, 2),
 tci = getDouble(Fpr.Texto, 76, 80, 2),
 tcmax = getDouble(Fpr.Texto, 81, 85, 2),
 tcmin = getDouble(Fpr.Texto, 86, 90, 2)
{SELECT keyId AS ID, Fpr.Obra,  designacao, grupo, bloco, codigo,
        oe, fr, ft, cota, rfi, rfmax, rfmin, tfi, tfmax, tfmin,
        rci, rcmax, rcmin, tci, tcmax, tcmin
KEY ID
}
```

Neste caso não se define nenhuma condição de actualização, já que a tabela gerada é temporária. Nos passos seguintes, é necessário determinar os grupos de instrumentos (fios de prumo) e, em seguida, gerar, com base nos grupos criados, os instrumentos (base de coordinómetro do fio de prumo) e a respectiva relação com o grupo.

O código seguinte corresponde à definição dos grupos de instrumentos. Cada grupo tem um nome específico e está num único bloco da barragem. Em vez de se gerar um novo identificador, utiliza-se o identificador do primeiro instrumento que pertence a esse grupo que é, necessariamente, único.

```
CREATE VIEW MgrFioPrumo
FROM MgrApoioFioPrumo V
{SELECT MIN(V.ID) AS ID, V.Obra, V.grupo AS nome, V.bloco
 GROUP BY V.Obra, V.grupo, V.bloco
}
CONSTRAINT UNIQUE (Obra, nome)
KEY ID
```

Efectuada a criação dos grupos, é feita a junção entre a tabela com a definição dos

grupos e a tabela inicial, a fim de gerar os instrumentos (base de coordinómetro do fio de prumo) e a respectiva relação com o grupo.

O código que se segue implementa essa transformação.

```
CREATE VIEW MgrFioPrumoBaseCoord WITH EXCEPTIONS
FROM MgrApoioFioPrumo V LEFT OUTER JOIN MgrFioPrumo G
ON V.Obra = G.Obra AND V.grupo = G.nome
{SELECT V.ID, V.Obra,  V.designacao, G.ID AS GrupoId, V.bloco,
        V.codigo, V.oe, V.fr, V.ft, V.cota, V.rfi, V.rfmax,
        V.rfmin, V.tfi, V.tfmax, V.tfmin,  V.rci, V.rcmax,
        V.rcmin, V.tci, V.tcmax, V.tcmin
}
CONSTRAINT NOT NULL GrupoId
           UNIQUE (Obra, designacao)
KEY ID
```

A particularidade deste operador é a geração de excepções. Por um lado, é feito um *left outer join*, pelo que, caso não exista correspondência com o grupo, esse registo será considerado uma excepção já que a tabela final não permite que o identificador do grupo esteja vazio. Por outro lado, será considerada uma excepção a circunstância de um registo juntar com mais do que um grupo, já que o atributo *ID* corresponde à chave primária, não podendo, por isso, ser duplicado.

Os dois mapeamentos que se executam a seguir são feitos apenas para determinar as condições de actualização das tabelas finais, que são (*Obra*, *Grupo*) para o grupo e (*Obra*, *codigo*) para o instrumento.

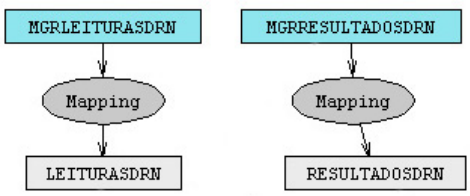

Figura F.4: Leituras e Resultados

A Figura F.4 ilustra o primeiro passo da transformação para as leituras e resultados

dos *Drenos*. Em ambos os casos a transformação é implementada por um *map* que extrai a informação de cada uma das entradas e gera um novo identificador para cada registo.

O código que se segue representa o código que implementa os mapeamentos para as leituras e para os resultados dos *Drenos*, respectivamente.

```
CREATE MAPPING LeiturasDrn
FROM MgrLeiturasDrn Drn
LET keyId = generateId(3),
 codTratamento = getInt(Drn.Texto, 0, 2),
 dataLeit = getData(Drn.Texto, 8, 15),
 nivelAlbufeita = getDouble(Drn.Texto, 16, 22, 2),
 codigosiobe = getInt(Drn.Texto, 23, 25),
 volume = getDouble(Drn.Texto, 26, 32, 2),
 tempo = getDouble(Drn.Texto, 33, 37, 1),
 ocorrencia = getInt(Drn.Texto, 38, 39)
{SELECT keyId AS ID, Drn.Obra, codTratamento, dataLeit, nivelAlbufeita,
        codigosiobe, volume, tempo, ocorrencia
KEY ID
}

CREATE MAPPING ResultadosDrn
FROM MgrResultadosDrn Drn
LET keyId = generateId(4),
 dataRes = getData(Drn.Texto, 3, 10),
 codigosiobe = getInt(Drn.Texto, 11, 13),
 caudal = getDouble(Drn.Texto, 14, 19, 2)
{SELECT keyId AS ID, Drn.Obra, dataRes, codigosiobe, caudal
KEY ID
}
```

Após separar a informação contida nas leituras e nos resultados, é necessário criar a entidade *campanha*, que não existe no sistema legado.

A Figura F.5 apresenta o grafo desta transformação.

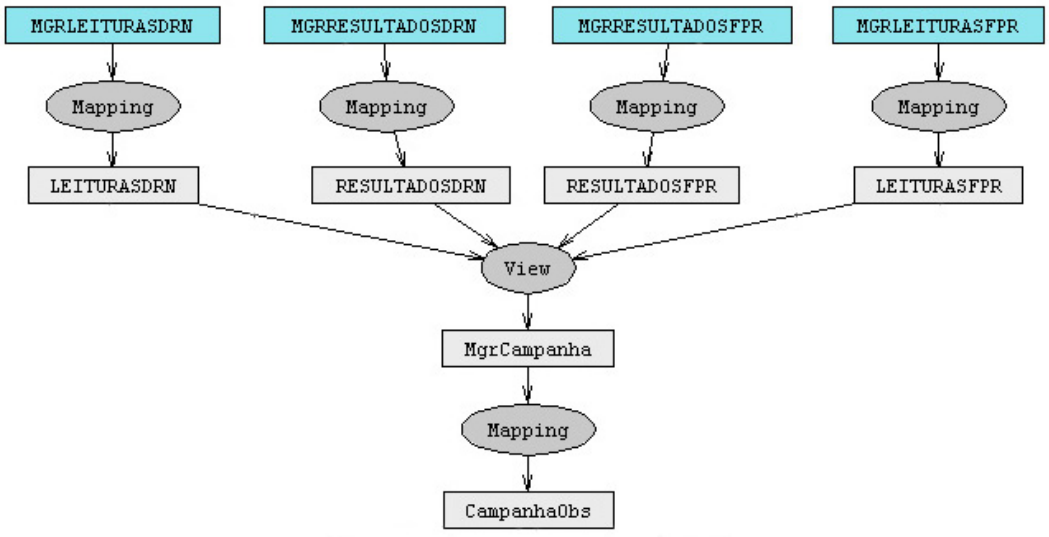

Figura F.5: Carregamento de campanhas de observação

A criação das campanhas é feita a partir das leituras e resultados de todos os instrumentos (no exemplo apenas estão representados os *Drenos* e os *Fios de Prumo*).

A transformação é implementada por um operador *View*, que extrai das tabelas de leituras e de resultados todas as diferentes datas, unindo-as numa única tabela. Não existem registos duplicados, já que a união efectuada procede à sua eliminação.

O código seguinte representa esta transformação.

```
CREATE VIEW MgrCampanha
FROM LEITURASDRN L
{SELECT DISTINCT L.Obra, 2 AS Tipo, L.DATALEIT AS DataInicio,
        L.DATALEIT AS DataFim, 1 AS Estado
}
UNION
FROM LEITURASFPR L
{SELECT DISTINCT L.Obra, 2 AS Tipo, L.DATALEIT AS DataInicio,
        L.DATALEIT AS DataFim, 1 AS Estado
}
UNION
```

```
FROM RESULTADOSDRN L
{SELECT DISTINCT L.Obra, 2 AS Tipo, L.DATARES AS DataInicio,
        L.DATARES AS DataFim, 1 AS Estado
}
UNION
FROM RESULTADOSFPR L
{SELECT DISTINCT L.Obra, 2 AS Tipo, L.DATARES AS DataInicio,
        L.DATARES AS DataFim, 1 AS Estado
}
KEY Obra, DataInicio
```

O segundo passo, no carregamento das campanhas, consiste na execução de um mapeamento que gera um identificador para cada campanha. Por outro lado, tendo em conta que a tabela produzida é uma tabela do esquema alvo, é necessário definir a condição de actualização dos registos que, neste caso, é composta pelo par (*Obra*, *DataInicio*).

O código que se segue implementa este mapeamento.

```
CREATE MAPPING Campanha
FROM MgrCampanha T
LET keyId = generateId(5),
{SELECT keyId AS ID, T.Obra, T.Tipo, T.DataInicio, T.DataFim, T.Estado

CONSTRAINT UNIQUE (Obra, DataInicio)
           NOT NULL (Obra)
           NOT NULL (DataInicio)
KEY ID
INCREMENT BY Obra, DataInicio
}
```

Finalmente, a migração dos dados do Sistema de Observação termina com o carregamento das leituras e dos resultados. As Figuras F.6 e F.7 apresentam o grafo de

transformações para as leituras e resultados dos *Drenos*.

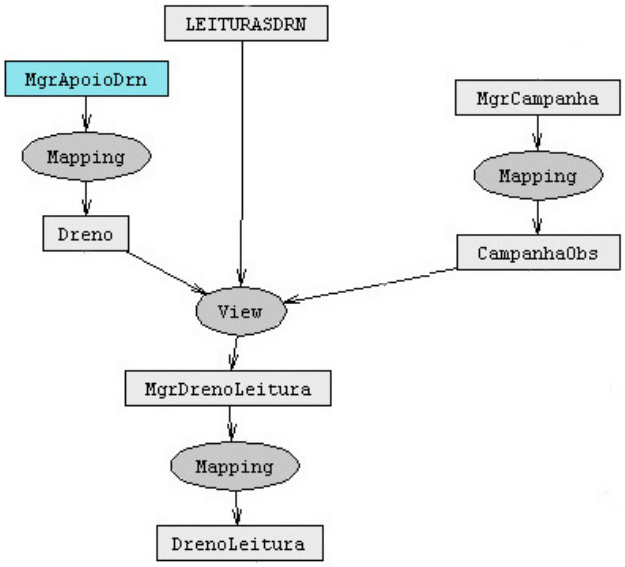

Figura F.6: Leituras

O carregamento das leituras é efectuado através da união entre a definição do instrumento, os dados extraídos do ficheiro de leituras e as *Campanhas*. Deste modo, é possível determinar quais são as relações entre as leituras, as campanhas e o instrumento. Esta junção é feita, pelo operador *View*, com determinação de excepções.

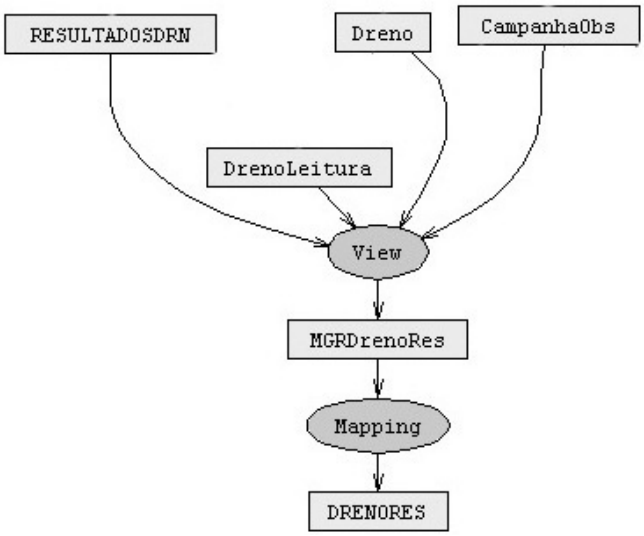

Figura F.7: Resultados

Uma vez que a tabela de leituras é uma tabela alvo, é necessário efectuar um mapeamento no qual se define a condição de actualização que, neste caso, é o par (*idCampanha*,

idInstrumento).

O carregamento dos resultados é semelhante ao das leituras, sendo, portanto, efectuado através da união entre a definição do instrumento, os dados extraídos do ficheiro de resultado, as leituras desse instrumento e as *Campanhas*. Deste modo, é possível determinar quais são as relações entre o resultado, a leitura que lhe deu origem, a campanha e o instrumento. Esta junção também é feita, pelo operador *View*, com determinação de excepções.

Tendo em conta que a tabela de resultados é uma tabela alvo, é necessário efectuar um mapeamento no qual se define a condição de actualização que, neste caso, é o par (*idCampanha*, *idInstrumento*).

Importa referir ainda que o grafo que representa a migração completa deste sistema é composto por 186 transformações.

Apêndice G

Grafo de transformações para as observações geodésicas

Neste Apêndice, resume-se a sequência das transformações mais importantes, implementadas no *Ajax*, que permitem fazer a migração dos dados relativos às observações geodésicas para o esquema da base de dados alvo.

A Figura G.1 representa o grafo das transformações aplicadas. Uma vez que as leituras são efectuadas em redes geodésicas, antes de efectuar o carregamento das leituras e posteriores resultados é necessário migrar a definição das redes. Cada rede é composta por um conjunto de ligações e giros que, por sua vez, se definem através dos vértices materializados na barragem.

O primeiro passo na migração, consiste em determinar uma chave única para cada registo em todas as tabelas de entrada. Deste modo, para cada uma das tabelas de entrada define-se um operador *map*, que recorre à função *Java* `generateId(int)`.

A entidade *Campanha* é obtida directamente a partir da folha *Excel Camp*, não dependendo de informação contida nas outras folhas.

A Figura G.2 apresenta um excerto do grafo de transformações para a geração da entidade Vértice.

É necessário proceder à junção da tabela com os dados fonte (*MgrDefPontosKey*) com as tabelas *Centragem*, *FuncaoRede* e *TipoVertice*. Assim, é possível estabelecer as relações entre os vértices e os respectivos tipos, centragens e funções na rede. Esta

junção é feita através de um operador *view* com geração de excepções.

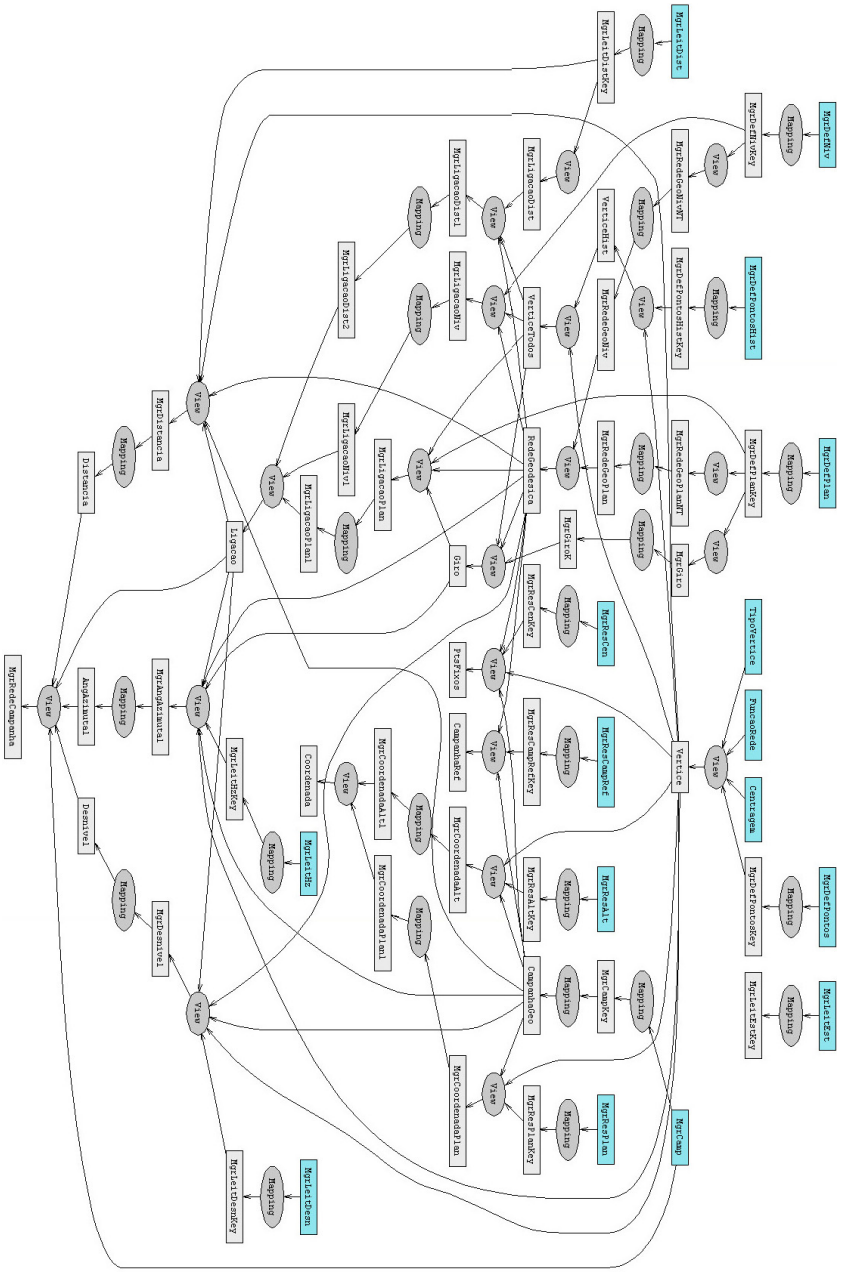

Figura G.1: Grafo de transformações das observaçõe geodésicas

Note-se que, na migração da informação geodésica, não é necessário efectuar um mapeamento com definição de actualização para as tabelas finais. Isto sucede porque a migração é feita para um esquema temporário da base de dados. Existe um processo posterior, desenvolvido em *Java*, que faz o carregamento para o esquema da base de dados final.

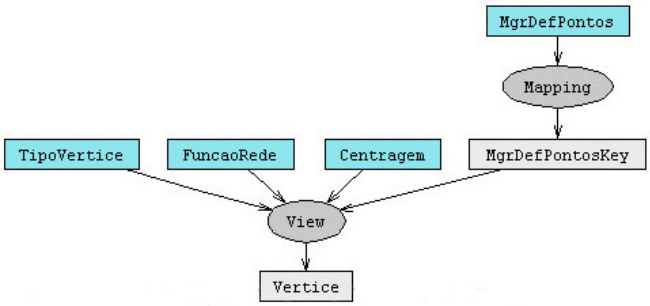

Figura G.2: Vértice

Em seguida, procede-se ao carregamento das redes geodésicas. Esta informação provém das folhas *DefNiv* e *DefPlan*, que contêm a definição das linhas de nivelamento e das redes bi ou tridimensionais, respectivamente.

A Figura G.3 ilustra o carregamento das redes geodésicas.

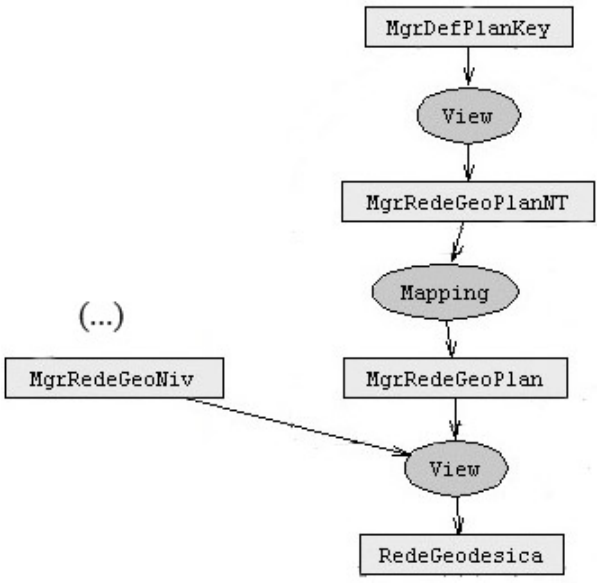

Figura G.3: Rede Geodésica

Existe uma nova rede para cada *NomeRede* definido nas folhas *DefNiv* e *DefPlan*. Assim, cria-se uma nova rede para cada nome de rede distinto, com recurso à cláusula *distinct* do operador *view*. Em seguida, gera-se um novo identificador para cada rede, através de um operador *map*, com recurso à função de geração de identificadores.

Finalmente, tendo em conta que as redes são produzidas a partir de dois fluxos, faz-se a união de todas as redes, através da cláusula *union* do operador *view*.

Para completar a definição das redes geodésicas, é necessário criar o conjunto das ligações que as compõem. A Figura G.4 ilustra o carregamento das ligações de desníveis.

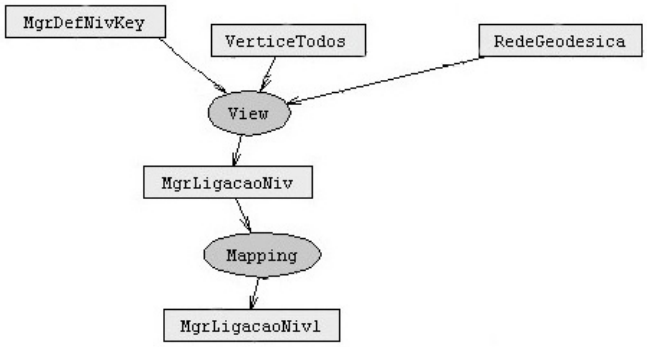

Figura G.4: Ligação de uma linha de nivelamento

Cada ligação é composta por dois vértices e pertence a uma única rede geodésica. Desta forma, é necessário cruzar a definição das linhas de nivelamento (folha *DefNiv*) com as tabelas de vértice e de redes geodésicas, a fim de obter os identificadores das respectivas entidades. Esta transformação é implementada por um operador *view* com geração de excepções.

A criação das ligações de distâncias, de ângulos verticais e de ângulos horizontais é semelhante à criação das ligações das linhas de nivelamento.

Depois de definir as redes geodésicas e as respectivas ligações, é possível migrar as leituras efectuadas em cada ligação, numa determinada campanha.

A Figura G.4 ilustra o carregamento dos desníveis.

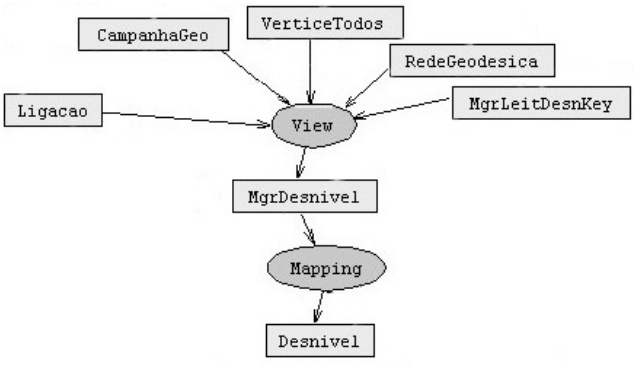

Figura G.5: Leitura de um desnível

A folha *LeitDesn* contém os desníveis verificados em cada ligação numa determinada campanha. Para carregar esta informação, é necessário determinar em que ligação foi feita cada leitura. Para isso, implementa-se um operador *view*, que acede às tabelas de ligações, de redes, de campanhas, de vértices e a informação proveniente da folha *LeitDesn*. Este operador também é executado com geração de excepções.

Finalmente, os resultados (coordenadas) estão armazenados em duas folhas (*ResAlt* e *ResPlan*).

A Figura G.6 ilustra o carregamento dos resultados.

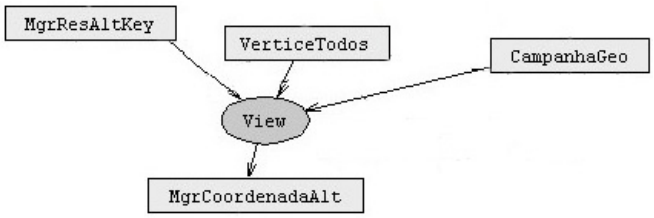

Figura G.6: Resultado - Coordenada altimétrica

Cada resultado corresponde às coordenadas de um vértice numa campanha. Uma vez que não existe informação sobre a leitura que deu origem a cada resultado, a migração destes só envolve as campanhas e os vértices. Desta forma, o carregamento das coordenadas é efectuado através de um *view* que acede à tabela de coordenadas, campanhas e vértices, com geração de excepções.

Bibliografia

[Ama04] J. Amante. *Manual SIOBE*, 2004.

[(As] DataStage (Ascential). http://www.ascential.com.

[BBWG03] M. Buechi, A. Borthwick, A. Winkel, and A. Goldberg. Cluemaker: A language for approximate record matching. In *8th International Conference on Data Quality*, Boston, November 2003.

[BG05] J. Barateiro and H. Galhardas. A survey of data quality tools. In *Datebank Spektrum*, pages 15–21, August 2005.

[BLS03] T. Biasusse, J. Léraillé, and B. Silva. Étude de l'etl businness objects data integrator, 2003.

[BLW+97] J. Bisbal, D. Lawless, B. Wu, J. Grimson, R. Richardson, and V. Wade. An overview of legacy information system migration, 1997.

[Bor04] A. Borthwick. The ChoiceMaker 2 record matching system. In *White Paper*, November 2004.

[BS04] A. Borthwick and M. Soffer. Business requirements of a record matching system. In *Ninth International Conference on Information Quality (MIT ICIQ)*, Cambridge, September 2004.

[Car07] Paulo Carreira. *Mapper: An Efficient Data Transformation Operator*. PhD thesis, Faculdade de Ciências da Univesidade de Lisboa, 2007.

[CD97] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1):65–74, 1997.

[CFR03] V. Chhoa, F. Fievet, and A. Riffard. Étude de l'etl informatica, 2003.

[CG04] P. Carreira and H. Galhardas. Efficient development of data migration transformations. Demo paper. In *ACM SIGMOD Int'l Conf. on the Managment of Data*, Paris, France, June 2004.

[CGL+98] D. Calavanese, G. Giacomo, M. Lenzerini, D.Nardi, and R. Rosati. Information integration: Conceptual modeling and reasoning support. In *CoopIS'98*, pages 280–291, 1998.

[CGLP06] P. Carreira, H. Galhardas, A. Lopes, and J. Pereira. One-to-many transformation through data mappers. *Data and Knowledge Engineering Journal*, 2006.

[Cha05] R. Chandras. A smart response to bad data, 2005.

[Cho] Choicemaker. http://www.choicemaker.com.

[CKV+05] S. Chaudhuri, K.Ganjam, V.Ganti, R. Kapoor, V. Narasayya, and T.Vassilakis. Data Cleaning in Microsoft SQL Server 2005. In *ACM SIGMOD/PODS Conference*, 2005.

[Com06] SAS Company. Etl studio - simplify etl processes with powerful transformation wizards and standardized, reusable metadata - fact sheet, 2006.

[CW01] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. In *27th VLDB Conference*, Rome, Italy, 2001.

[dDaSC] dfPower (DataFlux a SAS Company). http://www.dataflux.com.

[Def] Document Type Definition. http://www.w3schools.com/dtd/.

[dQ96] Instituto Português da Qualidade. Vocabulário internacional de meteorologia. termos fundamentais e gerais, 2ª edição, 1996.

[(ET] ETI*Data Cleanser (ETI). http://www.eti.com.

[ETL] Informatica ETL. http://www.informatica.com.

[Eva] Data Quality Tools METAspectrum Evaluation. www.firstlogic.com/pdfs/metaspectrumdqt.pdf.

[Fir] Firstlogic. http://www.firstlogic.com.

[Fir05] Firstlogic. Firstlogic introduces next-generation data quality platform, 2005.

[fSIA99] International Organization for Standardization (ISO) and American National Standards Institute (ANSI). Iso international standard. In *database language sql - part 2*, September 1999.

[Gal01] Helena Galhardas. *Nettoyage de données: Modèle, langage déclaratif et algorithmes*. PhD thesis, Université de Versailles Saint-Quentin-en-Yvelines, September 2001.

[(Ge] Hummingbird ETL (Genio). http://www.hummingbird.com.

[GFS+01] H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C. A. Saita. Declarative data cleaning: Language, model, and algorithms. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB'01)*, Rome, Italy, September 2001.

[GFSS00] H. Galhardas, D. Florescu, D. Shasha, and E. Simon. Ajax: An extensible data cleaning tool. In *SIGMOD*, 2000.

[Gom99] A. Gomes. Legislação portuguesa sobre segurança de barragens, 1999.

[HS93] A. Tavares de Castro H. Silva, J. Amante. Sistema de informação para observação de barragens de betão (siobe), manual de utilização, 1993.

[(IB] DataBlade (IBM). http://www.informix.com.

[Inf06] Informatica. Addressing data quality at the enterprise level. six questions your organization should ask to ensure high-quality data. White Paper, 2006.

[JKL04] Liang Jin, Nick Koudas, and Chen Li. NNH: Improving performance of nearest-neighbor searches using histograms. In *Extending Database Technology (EDBT)*, Crete, Greece, 2004.

[JKLT05] Liang Jin, Nick Koudas, Chen Li, and Anthony Tung. Indexing mixed types for approximate retrieval. In *VLDB*, 2005.

[JLM03] Liang Jin, Chen Li, and Shared Mehrotra. Efficient record linkage in large data sets. In *Eighth International Conference on Database Systems for Advanced Applications*, Kyoto, Japan, 2003.

[KCH+02] Won Kim, Byoung-Ju Choi, Eui-Kyeong Hong, Soo-Kyung Kim, and Doheon Lee. A taxonomy of dirty data. *Data Mining and Knowledge Discovery*, 2559:19–34, December 2002.

[LLL00] Mong Lee, Tok Ling, and Wai Low. Intelliclean: A knowledge-based intelligent data cleaner. In *ACM SIGKDD*, Boston, 2000.

[LNE80] LNEC. Validação de dados e de resultados da observação de barragens de betão. Relatório interno, 1980.

[LZ05] P. Larson and J. Zhou. View matching for outer-join views. In *31st VLDB Conference*, 2005.

[MF03] Heiko Müller and Johann-Christoph Freytag. Problems, methods, and challenges in comprehensive data cleansing. *Technical Report HUB-IB-164, Humboldt University Berlin*, 2003.

[MHH00] R. J. Miller, L. M. Haas, and M. Hernandéz. Schema Mapping as Query Discovery. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB'00)*, pages 77–78, Cairo, Egypt, September 2000.

[MHH+01] R. J. Miller, L. M. Haas, M. Hernandéz, C. T. H. Ho, R. Fagin, and L. Popa. The Clio Project: Managing Heterogeneity. *SIGMOD Record*, 1(30), March 2001.

[(Mia] SQL Server 2000 DTS (Microsoft). http://www.microsoft.com.

[(Mib] SQL Server 2005 (Microsoft). http://www.microsoft.com.

[Mil98] R. J. Miller. Using Schematically Heterogeneous Structures. *Proc. of ACM SIGMOD Int'l Conf. on the Managment of Data*, 2(22):189–200, June 1998.

[MM00] J. Maletic and A. Marcus. Automated identification of errors in data sets. *IQ2000*, 2000.

[MOME03] P. Marchandeau, O.Crovella, V. Miekisiak, and S. Esiliv. Étude de l'etl sunopsis, 2003.

[(MS] NaDIS (MSI). http://www.msi.com.au.

[MyS] MySQL. http://dev.mysql.com/.

[MZ98] T. Milo and S. Zhoar. Using schema matching to simplify heterogeneous data translation. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB'98)*, New York, USA, August 1998.

[(Ob] DataFusion (Oblog). http://www.oblog.pt.

[Obj] Data Integrator (Business Objects). http://www.businessobjects.com.

[Ols03] Jack Olson. *Data Quality - The Accuracy Dimension.* Morgan Kaufman Publishers, 2003.

[oPAG96] Commission on Preservation Access and Research Library Group. Report of the task force on archiving digital information. In *Preserving Digital Information*, 1996.

[ORHG05] P. Oliveira, F. Rodrigues, P. Henriques, and H. Galhardas. A taxonomy of data quality problems. *International Workshop on Data Quality (in conjunction with CAISE'05)*, 2005.

[(Pe] DoubleTake (Peoplesmith). http://www.peoplesmith.com.

[Por01] Eliane Portela. *Novas Metodologias de Apoio ao Controlo de Segurança de Barragens de Betão - Uma abordagem através de sistemas periciais.* PhD thesis, Instituto Superior Técnico - Universidade Técnica de Lisboa, 2001.

[Pos] PostgreSQL. http://www.postgresql.org/.

[PPS+05] E. Portela, C. Pina, A. Silva, H. Galhardas, and J. Barateiro. A modernização dos sistemas de informação de barragens: o sistema gestbarragens. In *Barragens: Tecnologia, Segurança e Interacção com a Sociedade*, Outubro 2005.

[(QA] QuickAddress Batch (QAS). http://www.qas.com.

[QoT] Brian Marshall. Data Quality and Data Profiling A Glossary of Terms. http://www.agt.net/public/bmarshal/dataqual.htm.

[RD00] Erhard Rahm and Hong Do. Data cleaning: Problems and current approaches. In *IEEE Techn. Bulletin on Data Engineering*, December 2000.

[RdSdB90] Decreto-lei N.11/90 Regulamento de Segurança de Barragens. Diário da república, 6 de Janeiro 1990.

[Res04] Bloor Research. ETLQ data quality report, 2004.

[RH01] V. Raman and J. Hellerstein. Potter's wheel: An interactive data cleaning system. *VLDB*, 27:381–390, 2001.

[(SA] ETLQ (SAS). http://www.sas.com.

[SCS00] Kay-Uwe Sattler, Stefan Conrad, and Gunter Saake. Adding conflict resolution features to a query language for database federations. *Int. Workshop on Engineering Federated Information Systems, EFIS'00*, pages 42–52, June 2000.

[SGB+04] Alberto Silva, Helena Galhardas, José Barateiro, Hugo Matos, Jorge Gonçalves, Tiago Martins, Hélder Soares, and Marco Custódio. Especificação de requisitos técnicos do "gestbarragens", 2004.

[SGB+06] A. Silva, H. Galhardas, J. Barateiro, H. Matos, and J. Gonçalves end E. Portela. Data analysis features of the gestbarragens system. In *Second International Conference of Innovative Views of .NET Technologies*, Outubro 2006.

[SGBP05] A. Silva, H. Galhardas, J. Barateiro, and E. Portela. O sistema de informação gestbarragens. In *Barragens: Tecnologia, Segurança e Interacção com a Sociedade*, Outubro 2005.

[SGP02] A. Silva, H. Galhardas, and E. Portela. gestbarragens: Sistema integrado de gestão da informação para o controlo de segurança de barragens, versão 1.3, Fevereiro 2002.

[Sim03] A. Simitsis. Modeling and managing ETL processes. In *VLDB PhD Workshop*, Berlin, 2003.

[SKS02] Silberchatz, Korth, and Sudarshan. *Database System Concepts*. McGraw Hill, 2002.

[Sofa] Centrus Merge/Purge Library (Group 1 Software). http://www.centrus.com.

[Sofb] Merge/Puge Plus (Group 1 Software). http://www.g1.com.

[Sofc] Migration Architect (Evoke Software). http://www.evokesoft.com.

[Sofd] Sagent (Group 1 Software). http://www.sagent.com.

[Sol] Software Labs Advanced Data Management Solutions. http://www.software-labs.net/migration/datamigration.aspx.

[SQL] SQLWays. http://www.ispirer.com/products/.

[SS00] Kay-Uwe Sattler and Eike Schallehn. A data preparation framework based on a multidatabase language. *Proceedings of the International Database Engineering and Applications Symposium*, pages 219 – 228, 2000.

[Sun] Sunopsis. http://www.sunopsis.com.

[Swi] SwisSQL. http://www.swissql.com.

[Sysa] DeDupe (HelpIT Systems). http://www.dedupeit.com.

[Sysb] Identity Search Server (Identity Systems). http://www.identitysystems.com.

[Sysc] MatchIT (HelpIT Systems). http://www.helpit.com.

[tdl03a] Étude de l'ETL. Data stage, 2003.

[tdl03b] Étude de l'ETL. Hummingbird etl (genio), 2003.

[TG01] Can Turker and Michael Gertz. Semantic integrity constraint support in sql:1999 and commercial (object-)relational database management systems. In *The VLDB Journal 10*, pages 241–269, 2001.

[TGV03] F. Tchuente, M. Guen, and E. Vezin. Étude de l'etl "sagent", 2003.

[Tri] Trillium. http://www.trilliumsoft.com.

[VVS+00] P. Vassiliadis, Z. Vagena, S. Skiadopoulos, N. Karayannidis, and T. Sellis. Arktos: A tool for data cleaning and transformation in data warehouse environments. *IEEE Data Engineering Bulletin*, 23(4):42–47, December 2000.

[WDB+97] B. Wu, D.Lawless, J. Bisbal, J. Grimson, V. Wade, D. O'Sullivan, and R. Richardson. Legacy systems migration - a method and its tool-kit framework. In *Joint 1997 Asia Pacific Software Engineering Conference, and International Copmuter Science Conference*, Dezembro 1997.

[(Wi] WhizWhy (WizSoft). http://www.wizsoft.com/.

[WLB+97] B. Wu, D. Lawless, J. Bisbal, J. Grimson, R. Richardson, and D. O'Sullivan. Legacy system migration : A legacy data migration engine, 1997.