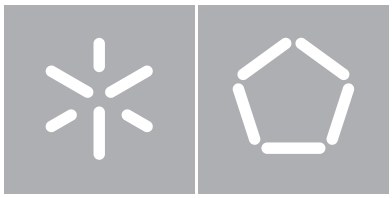
Universidade do Minho
Escola de Engenharia

Miguel Gonçalves Dias

Captura de Dados em Tempo Real em Sistemas de Data Warehousing

Outubro de 2013



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Miguel Gonçalves Dias

Captura de Dados em Tempo Real em Sistemas de Data Warehousing

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho realizado sob orientação de
Professor Doutor Orlando Manuel Oliveira Belo

Outubro de 2013

Anexo 3

DECLARAÇÃO

Nome

_____

Endereço electrónico: _____ Telefone: _____ / _____

Número do Bilhete de Identidade: _____

Título dissertação □/tese □

_____

_____

_____

Orientador(es):

_____

_____ Ano de conclusão: _____

Designação do Mestrado ou do Ramo de Conhecimento do Doutoramento:

_____

Nos exemplares das teses de doutoramento ou de mestrado ou de outros trabalhos entregues para prestação de provas públicas nas universidades ou outros estabelecimentos de ensino, e dos quais é obrigatoriamente enviado um exemplar para depósito legal na Biblioteca Nacional e, pelo menos outro para a biblioteca da universidade respectiva, deve constar uma das seguintes declarações:

1. É AUTORIZADA A REPRODUÇÃO INTEGRAL DESTA TESE/TRABALHO APENAS PARA EFEITOS DE INVESTIGAÇÃO, MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

2. É AUTORIZADA A REPRODUÇÃO PARCIAL DESTA TESE/TRABALHO (indicar, caso tal seja necessário, nº máximo de páginas, ilustrações, gráficos, etc.), APENAS PARA EFEITOS DE INVESTIGAÇÃO, , MEDIANTE DECLARAÇÃO ESCRITA DO INTERESSADO, QUE A TAL SE COMPROMETE;

3. DE ACORDO COM A LEGISLAÇÃO EM VIGOR, NÃO É PERMITIDA A REPRODUÇÃO DE QUALQUER PARTE DESTA TESE/TRABALHO

Universidade do Minho, ___/___/_____

Assinatura: _____

Aos meus Pais e à minha Irmã

Agradecimentos

Quero deixar aqui expressos os meus agradecimentos a todos aqueles que me acompanharam e apoiaram nesta jornada,de crescimento pessoal, intelectual e que contribuíram para que fosse possível, concluir este trabalho de dissertaçãocom sucesso:

- Ao meu orientador Professor Doutor Orlando Manuel Oliveira Belo, pela disponibilidade, incentivo e criticas ao longo deste trabalho, que permitiram levar este trabalho fosse mais além do proposto, mas também pela confiança transmitida.

- Ao André Nogueira, pela disponibilidade em todos os momentos para me ajudar com a implementação da ferramenta, e pela facilidade que atribuía aos problemas.

- Ao Eduardo Pessoa, à Ana Oliveira, ao Eduardo Fonseca, ao João Martins, ao Francisco Dourado, ao André Carvalho, ao Nélio Guimarães, à Ana Sofia Duarte e ao Dave Moderno pelo companheirismo, pelas brincadeiras, pelo apoio e pela paciência que tiveram comigo.

- À minha Irmã Ana Sofia Costa e ao Hélio Pereira pelo apoio incondicional, pela força que sempre me deram, pela disponibilidade para me ouvirem mesmo quando não era uma companhia agradável e pela confiança que me deram.

- À Eduarda Silva pelo apoio incondicional, por me conseguir animar sempre, por me aturar quando eu era insuportável, pela força e incentivo. E ainda pela preciosa ajuda nas leituras e correções do documento.

- A todos os meus colegas e amigos que sempre me apoiaram, incentivaram e me permitiram momentos de descontração nas horas boas e nas mais difíceis.

- Aos meus pais, Ana Maria Costa e Miguel Costa pela oportunidade que me proporcionaram, pelo incentivo e pela compreensão que sempre me deram.

Resumo

Captura de Dados em Tempo Real em Sistemas de *Data Warehousing*

A massificação dos sistemas de informação tem contribuído significativamente para a forma como os utilizadores interagem com as empresas e seus sistemas. Esta nova relação entre cliente e fornecedor tem aumentado significativamente o volume de dados gerados pelas organizações, criando novas necessidades de como manter e gerir toda esta informação. Assim, as empresas têm investido cada vez mais em soluções que permitam manter toda a informação tratada e consolidada num repositório único de dados. Estes sistemas são vulgarmente designados por sistemas de *data warehousing*. Tradicionalmente, estes sistemas são refrescados em modo *offline*, em períodos de tempo que podem ser diários ou semanais. Contudo, o aumento da competitividade no mundo empresarial torna este tipo de refrescamentos desadequados, originando uma reação atrasada à ação que despoletou essa informação. Na realidade, períodos longos de refrescamento tornam a informação desatualizada, diminuído consequentemente a sua importância e valor para a organização em causa. Assim sendo, é cada vez mais necessário que a informação armazenada num sistema de *data warehousing*, seja a mais recente possível, evitando interrupções na disponibilização da informação. A necessidade de obter a informação em tempo real, coloca alguns desafios, tais como manter os dados acessíveis 24 horas por dia, 7 dias por semana, 365 dias por ano, reduzir o período de latência dos dados ou evitar estrangulamentos operacionais nos sistemas transacionais. Assim, é imperativo a utilização

de técnicas de coleta de dados não intrusivas, que atuem no momento em que determinado evento ocorreu num sistema operacional e reflitam a sua informação de forma imediata (ou quase imediata) num sistema de *data warehousing*. Neste trabalho de dissertação pretende-se estudar a problemática relacionada com a captura de dados em tempo real e conceber um componente que capaz de suportar um sistema de extração de dados em tempo real universal, que capture as mudanças ocorridas nos sistemas transacionais, de forma não intrusiva, e as comunique na altura certa ao seu sistema de *data warehousing*.

Palavras-chave: Sistemas de *Data Warehousing*, ETL, Integração em Tempo Real, Captura de Dados Novos ou Alterados, *Data Warehousing* em Tempo Real.

Abstract

Real Time Change Data Capture in Data Warehousing Systems

The mass of information systems has contributed significantly to the way users interact with companies and their systems. This new relation between customer and supplier hassignificantly increased the amount of data generated by organizations, creating new needs to maintain and manage all this information. Thus, companies haveincreasingly invested in solutions that allow them to maintain all the information processed and consolidated on a unique data repository. These systems are commonly called Data Warehousing Systems. Traditionally, these systems are refreshed in offline mode in periods of time that can be daily or weekly. Although, the increase of the competitively in the business world, makes this kind of refreshments unsustainable, resulting in a delayed reaction to the action that triggered this information. In truth, long periods between refreshments make the information out-dated, consequently decreasing his importance and the value of the organization. . In that case, it is increasingly necessary that the information stored on the data warehousing systems, is the more recent possible, taking back interruption on the share of that information. The need of obtain information in real time, puts some challenges, as keep all the data accessible 24 hours a day, 7 day a week, 365 days a year, reducingthe periods of data latency or avoiding operational strangulations in transactional systems. Thus, it is imperative the usage of techniques of data collection non-intrusive that can act when some particular event occurred on operational systems and

reflect that information immediately (or almost immediately) on the data warehousing system.In this dissertation, we intend to study all the problematic related to real time change data capture, and conceiving a component capable to support an universal real time data extraction system, capable of capture the changes occurred on a transactional system, in a non-intrusive way and communicate with the data warehousing system in the right time.

KeyWords: Data Warehousing Systems, ETL, Real Time Integration, Change Data Capture, Real Time Data Warehousing

Índice

Índice de Figuras

Índice de Tabelas

Lista de Siglas e Acrónimos

API	– *Application Programming Interface*
ASCII	– *American Standard Code for Information Interchange*
BD	– Base de Dados
CDC	– *Change Data Capture*
DW	– *Data Warehouse*
ETL	– *Extract, Transform and Load*
IDE	– *Integrated Development Environment*
OLAP	– *On-Line Analytical Processing*
ODBC	– *Open Database Connectivity*
SDW	– Sistemas de *Data Warehousing*
SGBD	– Sistema de Gestão de Base de Dados
SQL	– *Structured Query Language*

1 Introdução

1.1 Contextualização

Com o evoluir do mercado empresarial e com a crescente interação das empresas com os sistemas de informação, os volumes de dados armazenados por estas têm vindo a aumentar de forma exponencial. Com o aumento significativo da informação por parte das organizações no seu dia-a-dia, aumentou também a necessidade de armazenar essa informação num "único" repositório. Esta necessidade de organização por parte das empresas, levou a um forte investimento em sistemas que permitissem armazenar toda a informação num único repositório de dados, devidamente tratada e consolidada, e que refletisse as regras de negócio e estruturas organizacionais. Estes sistemas são usualmente designados por Sistemas de *Data Warehousing* (SDW) e têm como objetivo facilitar o papel dos gestores das empresas, providenciando técnicas, métodos e ferramentas, que permitem analisar a informação armazenada, e ajudar no processo de tomada de decisão (Golfarelli & Rizzi 2009). Os SDW foram criados com o objetivo de armazenar num único repositório dados históricos, devidamente tratados, integrados, não-voláteis e orientados a vertentes específicas de negócio da empresa, de maneira a representar fielmente toda a organização de uma empresa e suportar os seus processos de tomada de decisão. Estes sistemas permitem aos gestores fazer consultas com elevado grau de desempenho, gerar relatórios detalhados, facilitando a extração informação necessária

aos processos de análise da empresa. São, por isso, sistemas que trazem grandes vantagens aos agentes de decisão.

No centro destes sistemas, estão os *Data Warehouses* (DW), que são bases de dados integradas, orientadas ao assunto, históricas e não voláteis, especialmente concebidas para o suporte do processo de tomada de decisão (Inmon 2005). Estas bases de dados, permitem armazenar grandes volumes de dados num único local, de maneira a satisfazer as necessidades dos processos de tomada de decisão empresariais. O facto de serem integrados permite armazenar informação proveniente de vários sistemas e aplicações principais ao negócio das organizações. Por serem históricos e não voláteis, garantem que a informação não se perde ao longo do tempo, possibilitando a análise histórica de dados.
Um dos pontos mais importantes no desenho e implementação de um DW é o desenho do fluxo de dados, desde as fontes de informação do sistema até ao seu destino (Skoutas & Simitsis, 2006). Este processo, designado vulgarmente por ETL – *Extract*, *Transform* and *Load* -, cria a necessidade de utilização de ferramentas que permitam extrair a informação pertinente dessas fontes, transformando-a e carregando-a no DW destino de acordo com as necessidades dos seus agentes de decisão e estruturas de armazenamento. De acordo com Kimball *et al.* (2008), 70% do esforço de implementação e manutenção de um DW é gasto com o processo de ETL. Tal como o nome indica, este processo está dividido em três componentes principais. A saber:

- Extração, que de acordo com um plano pré determinado, extrai a informação pertinente das diversas fontes de dados;
- Transformação, no qual se trata a informação extraída, verificando-se os dados e eliminando possíveis erros ou indefinições que possam deixar o DW num estado inconsistente ou incorreto;
- Carregamento, que introduz a informação já tratada no DW.

Habitualmente os processos de ETL são processos extremamente complexos e de processamento demorado. Como tal, o refrescamento do DW nem sempre é feito de acordo com a necessidade dos utilizadores finais. Em muitas situações, o refrescamento é periódico e durante esse tempo o sistema encontra-se inativo ou com baixo desempenho, gerando frequentemente tempos de resposta demasiado longos. Adicionalmente, o refrescamento periódico leva a que muitas vezes a atualidade dos dados num DW não seja coincidente com a atualidade dos dados nos sistemas fonte. Isto significa que as tradicionais janelas *batch*, na sua maioria definidas em períodos noturnos, estão a tornar-se cada vez mais difíceis de acomodar. Por outro lado, interromper ou influenciar a performance das fontes de dados nem sempre torna aceitável que se realizem processos de extração de dados a qualquer momento do dia (Oracle & Paper, 2012). Motivos como este tem contribuído para que os métodos de integração de dados em tempo real tenham vindo a ganhar cada vez mais preponderância. Estes métodos têm como base o principio da ação-reação, isto é, sempre que uma nova transação é executada sobre qualquer uma das fontes que suportam o DW, os novos dados são de imediato identificados, extraídos e carregados para o DW, fazendo com que a informação reflita o estado atual do negócio sobre o qual assenta o DW.

As empresas que promovem ambientes de trabalho com necessidades de processamento em tempo real devem eliminar os obstáculos que podem ser levantados no processo de disponibilização dos dados, bem como em eventuais estrangulamentos ou perdas de desempenho nos sistemas operacionais. Contudo, estas imposições levantam alguns desafios sérios, que precisam de ser devidamente solucionados para que tais condições operacionais possam ser satisfeitas. Entre eles salientam-se os seguintes:

- Disponibilidade – os serviços críticos de uma empresa devem manter-se acessíveis 24 horas por dia, 7 dias por semana, 365 dias por ano, sem que haja interrupções ou perdas de desempenho.

- Redução de latência – os dados devem ser os mais recentes possível. É vulgar verificar-se que, no mundo empresarial, os dados vão perdendo importância à medida que o tempo passa.
- Heterogeneidade e flexibilidade –a integração dos dados deve ser feita facilmente a partir de qualquer sistema de informação onde se trabalhe.
- Integridade transacional – deve ser garantida a exatidão e integridade dos dados quando estes são extraídos de uma fonte e consequentemente carregados para o repositório final.

1.2 Motivação e Objetivos

As necessidades atuais de mercado levam a que os gestores tomem decisões com base na informação mais recente acerca dos desenvolvimentos que ocorrem no quotidiano das suas empresas. Num mundo em que se privilegia informação atual e consistente, os *data warehouses* tornaram-se as ferramentas de exceção no suporte aos processos de tomada de decisão (Skoutas & Simitsis 2006). Há muito que os gestores deixaram de estar satisfeitos com o tempo de validade da informação que têm disponível e manipulam. Nesse sentido, é necessário fazer um refrescamento da informação em tempo útil (ou real) de forma a que os dados não atinjam a situação de ficarem obsoletos. Uma das melhores soluções passa por disponibilizar a informação necessária, se possível, em tempo real de forma a que os dados estejam acessíveis no momento em que se pretenda utilizá-los. Contudo a necessidade de manter a informação atualizada no momento, levanta várias questões que precisam de ser respondidas. Uma dessas questões tem que ver com o facto do processo de ETL lidar com fontes de dados heterogéneas, que vão desde simples ficheiros de cálculo, até aplicações de gestão com repositórios de dados proprietários. É necessário conseguir aceder à informação de forma fácil e rápida, de maneira a integrá-la no sistema de destino, na altura certa. Tradicionalmente, este tipo de processos é realizado durante a noite, com os sistemas inativos. Porém, as necessidades de processamento em tempo real, obrigam a

que o refrescamento da informação seja feito em períodos de tempo inferiores a um dia ou mesmo horas, ou que seja feito imediatamente a seguir a esta surgir nas fontes.

Geralmente os sistemas de ETL adotam uma estratégia de extração de dados que passa por extrair toda a informação existente nas fontes até ao momento em que se pretende capturar e, dessa forma, identificar apenas aquela que ainda não foi carregada. Contudo o volume de dados a tratar pode crescer gravosamente, o que degrada seriamente o desempenho do processo (Jörg & Deßloch 2008). Por outro lado, o contínuo refrescamento da informação pode levar a determinados picos de atividade, quer ao nível do processo de ETL quer do sistema final. Esta situação pode levar a que os períodos de maior atividade de um DW possam coincidir com os períodos em que os novos dados possam estar a ser carregados (Langseth 2004). Isto levará a que o desempenho do sistema se degrade nesses momentos e ainda a que os resultados das consultas retornem resultados já obsoletos. Esta situação obriga a que a arquitetura de um sistema de ETL*,* para cenários de processamento em tempo real, seja devidamente pensada e criada para suportar as necessidades desse tipo de processamento, sempre muito exigente e crítico. O grande ponto passa, naturalmente, pela extração de dados, e pelas técnicas que vão permitir dar o suporte adequado ao sistema de ETL nesse tipo de cenários.

Uma forma de contrariar o habitual processo de extração de dados, via cálculo diferencial de dados, passa pela utilização de mecanismos de captura de modificações efetuadas sobre os dados das fontes –*Change Data Capture* (CDC). Esses mecanismos são uma forma não intrusiva, que permitem identificar quais as alterações que foram feitas no sistema, desde o último refrescamento, à medida que estas vão surgindo no sistema, e guardar essa nova informação numa estrutura de dados especialmente criada para o efeito. Uma estratégia de extração de dados, utilizando ficheiros de *log* dos SGBD (Shi *et al.* 2008), pode melhorar seriamente o desempenho de um sistema de ETL, uma vez que é uma forma não intrusiva, permitindo lidar com esses ficheiros e extrair a informação mais recente, ou seja, apenas aquela que ainda não está no *data warehouse*, desde que este esteja preparado para esse

efeito.

Com base nos pressupostos referidos anteriormente, definiram-se como objetivos principais para este trabalho de dissertação os seguintes pontos:

- Estudar e analisar as principais técnicas de captura de dados;
- Estudar e analisar a forma como os principais fornecedores implementam essas técnicas;
- Estudar a problemática inerente a este processo e a forma como os principais fornecedores os resolvem;
- Desenvolver uma ferramenta capaz de efetuar captura de dados em tempo real a partir de sistemas heterogéneos.

1.3 Estrutura do Documento

Além do presente capítulo, esta dissertação integra um conjunto de mais quatro capítulos que estão organizados da seguinte forma:

- **Capítulo 2** – Métodos de angariação de dados. Neste capítulo faz-se uma introdução aos mecanismos de captura de dados, nomeadamente ao nível da sua caraterização, vantagens e desvantagens deste tipo de mecanismos e formas de implementação prática.

- **Capítulo 3**–Desafios inerentes ao processo de captura de dados. Neste capítulo apresenta-se um caso de estudo que envolve um processo típico de captura de dados. Adicionalmente, apresentam-se os desafios mais importantes que uma implementação de um sistema deste género levanta, qual a forma ou formas de os resolver, e que tipo de abordagens utilizam os principais fabricantes na sua implementação.

- **Capítulo 4** – Uma ferramenta de captura de dados. Nesta parte aborda-se o projeto e implementação de uma ferramenta para processos de captura de dados, apresentando os vários métodos comummente utilizados para resolver cada um dos problemas expressos no capítulo anterior, bem como as características mais pertinentes, próprias da ferramenta implementada.

- **Capítulo 5** – Conclusões e Trabalho Futuro. Este capítulo final apresenta um breve comentário final ao trabalho realizado, bem como uma apreciação crítica à abordagem desenvolvida. Por fim, apresentam-se algumas linhas de trabalho para futuro, essencialmente com o objetivo de estender o estudo que aqui está apresentado e aplicá-lo de forma mais efetiva a um cenário real.

2 Métodos de Angariação de Dados

A extração de dados é um dos principais passos do processo ETL de um de sistema de *Data Warehousing*. É nesta fase que são extraídos os dados considerados pertinentes e importantes para satisfazer os requisitos de negócio associados com um DW, sendo também necessário identificar desses dados aqueles que não foram ainda inseridos no seu sistema de destino. Nos sistemas de ETL convencionais a identificação dos novos dados é realizada efetuando um processo de cálculo ou de comparação de tabelas. Este processo passa por identificar a totalidade de informação nas fontes e calcular a diferença entre essa informação e aquela que está no sistema destino. Contudo esta técnica de identificação e angariação de novos dados apenas é praticada quando falamos em tabelas com alguns milhares de registos (Barkaway 2009), pois quando ultrapassamos essa fasquia, o impacto na performance do sistema é muito alto (Jörg & Deßloch 2008).

Atualmente os agentes de decisão requerem que a informação que lhes é disponibilizada reflita o estado mais recente do negócio, desta forma os períodos entre cada refrescamento da informação devem ser definidos de acordo as necessidades dos agentes. Nesse momento, o volume de informação a tratar pode ser da ordem dos milhões de registos, o que vai tornar incomportável a disponibilização em tempo útil da informação em cada refrescamento. O mecanismo de captura de dados novos ou alterados, vulgarmente conhecido por *Change Data Capture* (CDC) veio alterar a forma como os dados são

identificados nas fontes. Um processo expedito típico de CDC captura as modificações ocorridas numa dada fonte de informação logo após as transações ocorrerem e gravarem os dados na base de dados, identificando-os e transferindo-os de imediato para estruturas auxiliares. Desta forma, garante-se que o volume de informação capturado em cada momento é aquele que ainda não foi carregado para o sistema destino. Desta forma contraria-se o problema do seu impacto em termos de desempenho, uma vez que os dados são identificados e capturados de forma continua e diferenciada.

Ao longo deste capitulo identificar-se-á e caracterizar-se-á este mecanismo, quais as principais estratégias para captura de dados novos ou alterados, e por fim, alguns dos principais SGBD que suportam este mecanismo e as estratégias por eles implementadas para o fazerem.

2.1 Captura de Novos Dados ou Alterados

O mecanismo de captura de dados novos ou alterados em bases de dados relacionais, vulgarmente conhecido por *Change Data Capture,* é um mecanismo responsável por identificar de forma contínua apenas as alterações que são feitas numa base de dados (ou numa dada tabela), não abrangendo dados que não foram modificados, e enviar em seguida esses dados para outros sistemas ou simplesmente guardando-os em estruturas auxiliares. Ao contrário da estratégia de identificação dos dados que utiliza um processo de análise diferencial de tabelas, neste mecanismo são apenas identificados e capturados os dados cuja existência é nova no sistema. Desta forma, um processo de captura que siga esta abordagem pode revelar-se menos intrusivo, o que poderá causar um impacto reduzido no desempenho do sistema onde os dados são capturados (Eccles *et al.* 2010).

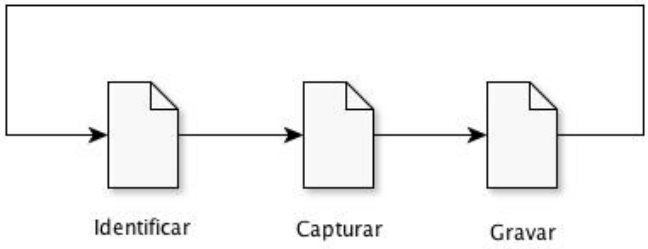


Figura 2-1: Os três passos de um processo típico de captura de dados.

Os mecanismos de captura de novos dados baseiam-se num princípio ativo de ação-reação. O princípio de funcionamento do processo divide-se em três momentos: identificar, capturar e gravar, funcionando numa forma cíclica (Figura 2-1). No primeiro momento – identificar – o processo pesquisa a ocorrência de uma determinada operação sobre os registos de uma ou mais tabelas das quais se pretende capturar os dados, seja essa operação uma inserção, remoção ou atualização. No segundo momento – capturar - é efetuada a captura dos registos identificados no primeiro momento, para que, no último momento – gravar –, esses registos sejam guardados em estruturas ou tabelas para que outras partes os possam utilizar. Este processo é repetido tantas vezes quanto o número de registos que estejam a ser inseridos no intervalo em que o mesmo está ativo. Desta forma, garante-se que são capturados apenas os dados cuja existência é mais recente, minimizando assim o volume de informação a ser tratado pelas aplicações que necessitem dessa informação e, consequentemente, aumentando o desempenho do sistema (Oracle & Paper 2012). Assim, esta estratégia, por reduzir ao estritamente necessário o volume de informação a ser propagada, a latência – como sabemos, um fator chave (Ankorion 2005) – pode ser reduzida a segundos ou microssegundos,entre o momento em que a informação é identificada e o momento em que chega ao destino, se for feita, obviamente, de forma correta (Tank *et al.* 2010). Esta é, também, uma das grandes vantagens da estratégia referida, uma vez que algumas das suas implementações permitem mesmo fazer a captura de dados em tempo real, o que satisfaz as necessidades de alguns sistemas, bastante críticos e exigentes, na identificação da informação alterada. Esta é uma das vantagens que fazem

deste mecanismo o mais utilizado em sistemas de *data warehousing* em tempo real(Kimball & Caserta 2004). Em suma, podemos reduzir as suas vantagens a questões como:

- Completude, uma vez quesão capturados todos os efeitos das operações de inserção, remoção e atualização, guardando informação histórica no caso desta última operação;
- Captura assíncrona, que pode ser configurada de forma a ter um impacto mínimo na operacionalidade do sistema;
- Redução dos custos e aumento da performance do sistema ao possibilitar menos transferências de dados, resultando consequentemente numa redução de custos em termos de *hardware* e de *software* (Ankorion 2005);
- Os sistemas de ETL não necessitam de carregar os dados a partir de sistemas *offline*, uma vez que a informação é transferida enquanto os sistemas estão em funcionamento;
- Garantia de que a informação enviada para outros sistemas é sempre a mais recente, refletindo adequadamente a informação envolvida nas atividades de negócio realizadas.

Existem várias formas de implementar este mecanismo de CDC, tais como a leitura de ficheiros *log* de transações dos SGBD, a utilização de gatilhos dentro dos objetos da bases de dados geridas pelos SGBD ou a colocação de atributos específicos para acolhimento de etiquetas temporais relativas, por exemplo, à data de criação ou de atualização de um dado registo. Nas próximas secções serão abordados cada um destes métodos, expondo-se sempre que oportuno as suas vantagens e desvantagens.

2.2 Estratégias para Identificação de Novos Dados

2.2.1 Utilizando de Atributos com Etiquetas Temporais

A aplicação desta estratégia incide diretamente sobre os sistemas fonte. É frequente que os sistemas fonte guardem atributos (colunas) nas estruturas das tabelas para guardar alguma informação temporal sobre quando um registo é inserido ou atualizado. Estas colunas, designadas por Kimball & Caserta (2004) por colunas de auditoria, guardam usualmente a data e hora em que um dado registo é inserido. Quando os valores dos atributos também são atualizados, as colunas de auditoria são igualmente atualizadas com a data da realização dessa operação, garantindo assim um registo de informação temporal sobre o registo .Desta forma, é possível identificar quais os registos que foram inseridos ou atualizados na fonte, através da seleção de todos os registos cujo valor da coluna de auditoria seja superior à data pretendida, neste caso do último ciclo de captura de dados. De forma a aumentar o desempenho das pesquisas sobre as colunas de auditoria – que podem envolver milhões de registos – é aconselhável a introdução de índices nessas mesmas colunas, de forma a tornar mais eficaz qualquer pesquisa que sobre elas seja efetuada (Eccles *et al*. 2010).

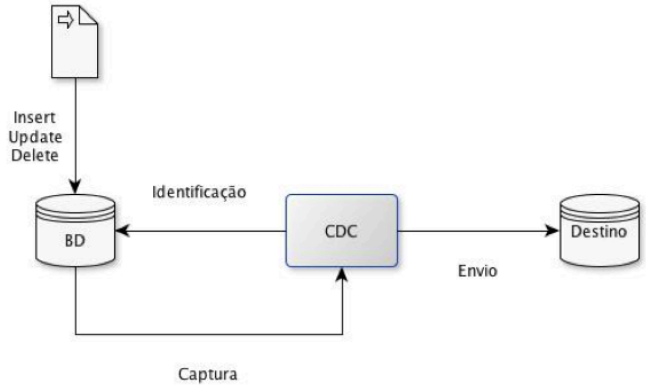


Figura 2-2: Ilustração da captura de dados baseada em colunas de auditoria.

Na Figura 2-2 podemos ver uma ilustração do modo de funcionamento do processo de captura baseada em colunas de auditoria. No processo ilustrado, os dados são inseridos na fonte, ao mesmo tempo que é inserido/atualizado o valor da coluna de auditoria correspondente. Quando o processo entra em ação é efetuada uma pesquisa pelos dados mais recentes e, de seguida, esses dados são enviados para o sistema de destino. Existem, contudo, algumas desvantagens neste tipo de estratégia, sendo a principal o facto de não ser possível identificar quais os registos que são removidos nas fontes. Tal acontece, porque ao remover um registo a sua informação temporal é igualmente removida (pelo menos numa versão mais simplificada da estratégia implementada) (Vassiliadis & Simitsis 2009).Uma alternativa passa por adicionar algumas colunas com informação do registo, ou seja, devemos indicar também qual o tipo de operação efetuada sobre o registo, e no caso de este ser removido, indicar a operação realizada e desconsiderar esses casos em futuros processos de pesquisa. Porém, isso irá também aumentar o impacto dessa alteração sobre os registos e os processos de pesquisa. Apesar das vantagens da inserção de índices nas colunas como forma para aumentar o desempenho da pesquisa, esta ação também conduz a um aumento da carga de trabalho do sistema no momento de inserir ou alterar registos, uma vez que tem mais estruturas de dados para atualizar. Por fim, mas não menos importante, nesta estratégia de CDC os dados permanecem armazenados nos próprios sistemas fonte. Quando os mecanismos de CDC são ativados, a seleção dos dados será realizada diretamente da fonte, circunstância que vai aumentar a latência entre a identificação dos dados e a sua captura, bem como o consequente carregamento para o sistema destino (Chen et al. 2010), pelo que a frequência dos carregamentos deve ser adaptada conforme as necessidades do sistema. De referir que esta estratégia não deverá ser uma primeira opção nos casos em que se pretende uma solução de captura de dados em tempo real.

2.2.2 **Utilização de Gatilhos**

Os gatilhos são funções bastante conhecidas nos SGBD, que se baseiam num princípio do tipo ação-reação. Tipicamente, são "blocos" de código armazenados na base de dados, definidos sobre objetos de uma base de dados (tabelas na sua maioria), que reagem a eventos particulares de modificação de dados que ocorrem na base de dados (Connolly & Begg 2004). Quando um determinado evento satisfaz as condições definidas para ativação do gatilho, este é ativado "disparando" a execução das ações definidas na sua estrutura. Os efeitos da execução das ações despoletadas por um gatilho são irreversíveis e, por isso, deve-se ter particular cuidado com a sua configuração e com a definição das ações que o despoletam.

Os métodos de captura de novos dados que utilizam gatilhos são, normalmente, métodos sofisticados, pois tiram proveito das vantagens inerentes aos próprios gatilhos para responder a eventos que ocorrem no sistema. Ao introduzir-se gatilhos nas tabelas sobre as quais se pretende capturar informação, assegura-se que, quando um determinado evento de inserção, de remoção ou de atualização de um registo ocorrer, o gatilho "dispara"o seu conjunto de ações, nas quais poderão estar instruções para guardar a nova informação no sistema de destino (Corp, S. 2004). Ao contrário do método baseado em colunas de auditoria, este permite identificar qualquer tipo de alterações, em particular as relacionadas com a remoção de registos, sendo, por isso, um método mais completo e vantajoso.Por isso, na nossa opinião, este método revela-se mais completoe vantajoso.

A Figura 2-3 ilustra um processo de CDC suportado por gatilhos. Quando as transações que manipulam a informação do sistema são executadas, os gatilhos "disparam" e enviam a informação para o sistema de destino.

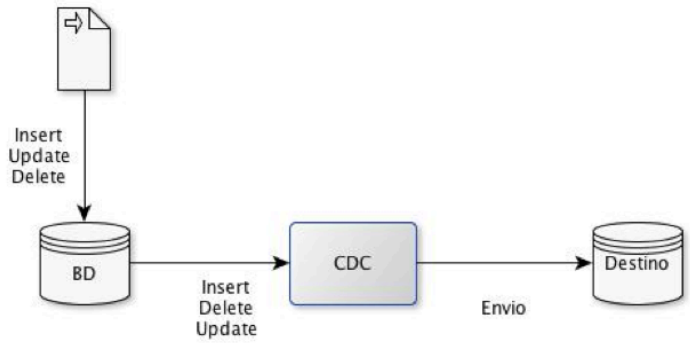


Figura 2-3: Ilustração da captura de dados baseada em gatilhos.

Este seria o método mais natural para a implementação de mecanismos de CDC, já que reflete na prática de forma efetiva o princípio de ação-reação. Todavia, a natureza intrusiva dos gatilhos (Lane *et al.* 2009) contraria esta tese. A sua utilização em SGBD deve ser devidamente ponderada, uma vez que os gatilhos causam algum impacto no desempenho do sistema no qual são utilizados(Eccles *et al.* 2010). Apesar disso, os gatilhos permitem reduzir os custos de captura de dados, quando comparados com a estratégia baseada em colunas de auditoria. Mas, ao mesmo tempo, obriga a uma manutenção constante dos gatilhos, pois quando a estrutura de uma tabela é alterada, o gatilho deve ser igualmente alterado de forma a manter a consistência da solução e, consequentemente, dos dados a enviar.

2.2.3 **Utilização de Ficheiros *Log* Transacionais**

Por forma a precaverem-se contra problemas operacionais que possam interromper o processamento de um sistema operacional ou deixando-o num estado inconsistente, alguns fabricantes de SGBD optaram por manter um jornal com toda a atividade transacional realizada no sistema. Este ficheiro é reconhecido por *log* transacional e permite identificar todos os tipos de transações que ocorrem no sistema e, com essa informação, analisar eventuais situações de inconsistência de dados no sistema. Caso esta situação aconteça é

possível anular o efeito de (partes de) transações que, por algum motivo, conduziram o sistema a um estado de inconsistência e reconduzi-lo a um estado anterior consistente. Este ficheiro de *log* é guardado em um ou mais ficheiros, sendo a sua estrutura normalmente configurável, apesar de variar de fabricante para fabricante.

A estratégia de captura de dados baseada em ficheiros de *log* transacionais tira proveito da capacidade que os SGBD têm de armazenar toda a informação transacional nesses ficheiros *log* (JÄorg & Dessloch n.d.). Assim, podemos identificar as transações que acrescentaram informação, por exemplo, logo após uma transação ocorrer, mantendo basicamente um processo de leitura ativo sobre os ficheiros de *log*. Ao contrário das abordagens anteriores, que afetam usualmente a performance do sistema, este tipo de abordagem revela-se muito menos intrusiva e consequentemente menos propícia para afetar o desempenho do sistema (Eccles *et al.* 2010). Além disso, permite ainda a captura de todas as operações que foram realizadas – inserção, remoção ou atualização de registos – sem que seja necessário realizar qualquer tipo de trabalho adicional no sistema (Thomas & Dessloch 2009). Este processo, ao manter uma leitura ativa sobre os ficheiros de *log*, afasta a necessidade de efetuar qualquer configuração adicional sobre o SGBD onde os dados estão a ser capturados, tornando o impacto no desempenho do sistema fonte quase inexistente.

Na Figura 2-4, podemos ver uma ilustração do procedimento relativo à estratégia que apresentámos. Nele verificamos que quando é inserida nova informação no SGBD, essa informação é guardada no ficheiro de *log*. No momento em que essa nova informação é escrita no ficheiro *log*, o processo de CDC que mantém uma leitura ativa desse ficheiro, vai identificar essa informação como sendo nova, capturá-la e enviá-la para o sistema de destino.

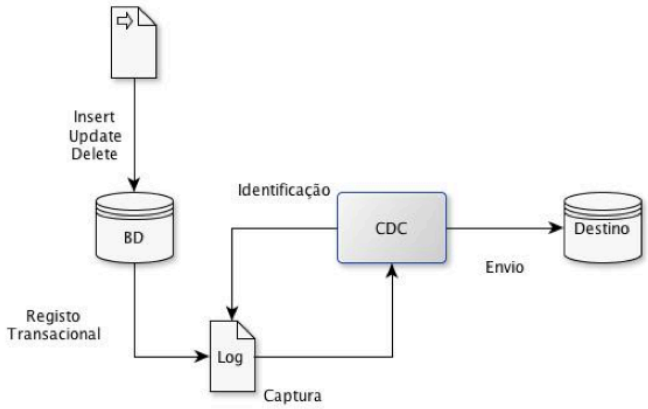


Figura 2-4: Ilustração de um processo de captura de dados baseada em ficheiros *log* transacionais.

As vantagens inerentes a este método de identificar e capturar as alterações realizadas sobre os dados de uma fonte são claras. O facto de ser a menos intrusiva e a que tem um menor impacto na performance do sistema torna-a, frequentemente, a técnica favorita. Porém, é um facto que, nem todos os SGBD são capazes de produzir ficheiros de *log* transacionais ou que as suas estruturas nem sempre são conhecidas ou documentadas. Isso leva a que seja necessário criar algoritmos específicos que consigam trabalhar as estruturas dos diferentes ficheiros de *log* mantidos pelos diversos fabricantes de SGBD. Além disso, os SGBD podem diferir em muito na forma como mantêm esses ficheiros. Por exemplo, alguns deles mantêm o registo de todas as transações, mesmo quando acontece uma operação de *ROLLBACK* que as anula, sendo esta uma desvantagem inerente a esta técnica (Barkaway 2009). Apesar disso, esta técnica não deixa de ser aquela que mais adeptos tem, especialmente devido às vantagens que traz quando comparada com as outras abordagens que atuam com base em colunas de auditoria e em gatilhos.

2.3 Ferramentas para a Captura de Novos Dados

Os mecanismos de captura de dados novos ou alterados em fontes de informação foram algo que não passou ao lado das empresas que fornecem SGBD, e serviços integrados. Ao longo deste capítulo apresentam-sealgumas das principais propostas, bem como as diversas estratégias de captura de novos dados que foram implementadas nos seus serviços.

2.3.1 O Sistema da *Oracle*

A *Oracle* introduziu o mecanismo de captura de dados novos ou alterados no seu SGBD na versão *9i*. Atualmente, já na sua versão *11g,* o sistema oferece modos de captura síncrona e assíncrona, configuráveis conforme as necessidades dos utilizadores. A captura síncrona utiliza a técnica de captura baseada em gatilhos. A captura assíncrona utiliza a estratégia baseada em *logs*, capturando apenas as transações que foram terminadas com sucesso. Contudo, existem alguns cuidados a ter em conta neste tipo de processo, uma vez que é muito dependente de um nível adicional de registo no *redo log*. Este deve ser cuidadosamente balanceado para não sobrecarregar o processo de registo das transações nos *redo logs*. A *Oracle* apresenta-nos duas formas de configurar este modo de captura, utilizando *hot logs* ou *auto logs*, configurações estas que analisaremos de seguida.

Captura Síncrona

Este modo utiliza a estratégia de captura síncrona baseada em gatilhos. Possibilita uma captura de dados em tempo real, já que permite a identificação dos dados a capturar imediatamente após estes terem sido introduzidos na base de dados fonte (JBSAC n.d.) .Neste modo, os dados são guardados em tabelas criadas para esse efeito dentro da base de

dados fonte, o que causa, naturalmente, um elevado impacto na performance do sistema operacional em processos de captura de dados de forma contínua.

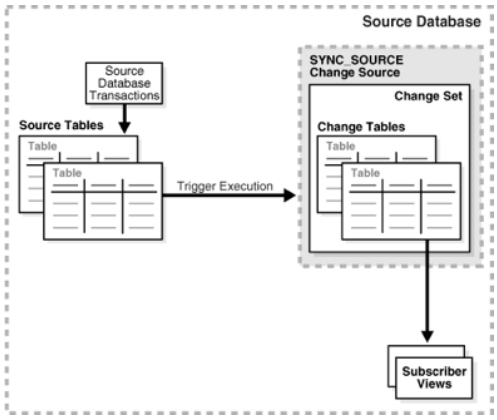


Figura 2-5: Ilustração de um processo de captura de dados síncrona em *Oracle* – figura extraída de (Lane *et al.* 2011).

A Figura 2-5 representa o modo de captura de dados síncrona do sistema Oracle. Nela podemos ver que as transações são executadas nas tabelas fonte, que ao serem executadas fazem disparar os gatilhos que enviam para as tabelas os dados relativos às alterações realizadas.

Captura AssíncronaUtilizando *HotLogs*

Neste tipo de captura os dados são recolhidos utilizando o *redolog* que está a ser usado nomomento do registo das transações ocorridas na base de dados na qual se pretende identificar as alterações realizadas. As tabelas de destino, nas quais vão ser guardados os dados capturados, são preenchidas à medida que as transações vão terminando com sucesso (Lane *et al.* 2011). Por isso, existe uma pequena latência entre o momento em que as transações são executadas e o momento em que os dados são carregados para as tabelas de destino, uma vez que os dados só são identificados quando as transações são dadas como terminadas com sucesso. As tabelas de destino são guardadas dentro da base de dados da qual são extraídas as alterações.

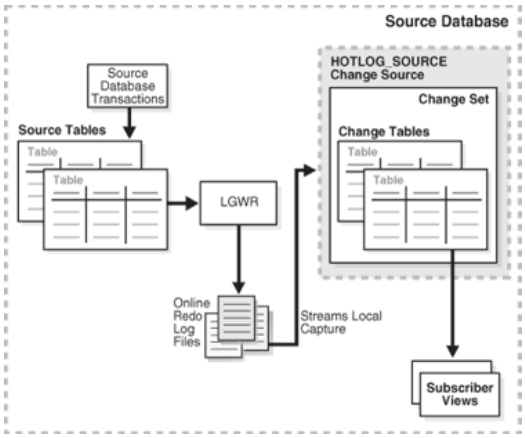


Figura 2-6: Ilustração da captura de dados assíncrona com *HotLogs* em *Oracle*– figura extraída de (Lane *et al.* 2011).

A Figura 2-6 ilustra o processo de captura de dados utilizando omodo assíncrono com HotLogs. Neste método, as transações que ocorrem são registadas no *RedoLog* da base de dados fonte. Em seguida, todas as transações completas são identificadas e os dados que elas afetaram são carregados para as tabelas de destino correspondentes.

Captura Assíncrona Utilizando *AutoLogs*

No processo de captura de dados assíncrona usando *autologs* os dados são capturados a partir dos *redologs*, que, neste modo, podem estar a ser utilizados pelo sistema para guardar as transações, ou podem estar arquivados em localizações externas ao sistema. Distingue-se do modo anterior por permitir que os dados capturados sejam enviados para tabelas que não estejam localizadas na base de dados origem, na qual foi feita a captura (Lane *et al.* 2011). Para isso, este método utiliza um sistema de transporte de *redo logs* – conhecido por *Redo Transport Service*-, que permite o envio de *redo logs* de uma base de dados para outra, desde que as especificações de *hardware* sejam as mesmas, bem comodo sistema operativo e da versão do sistema Oracle instalado. Deste modo, os ficheiros *redolog* são enviados para a base de dados de destino, sendo aí identificadas as alterações através das transações terminadas e depois guardadas nas tabelas dessa mesma base de dados. Desta forma, não haverá qualquer impacto operacional no sistema sobre o qual trabalha na captura dos dados.

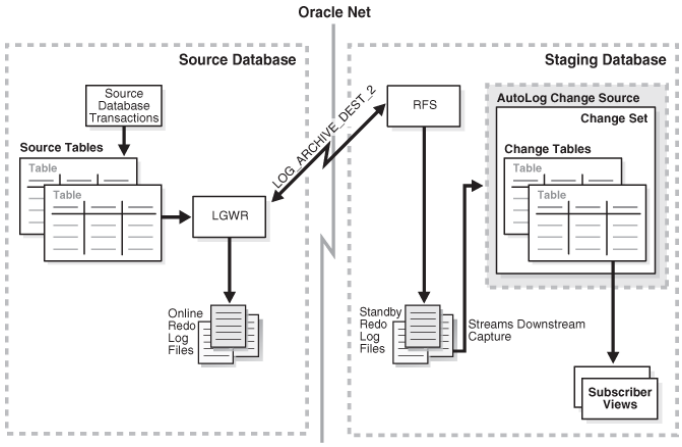


Figura 2-7: Ilustração do processo de captura de dados assíncrona usando *autologs* (Lane *et al.* 2011).

A Figura 2-7 mostra uma generalização do processo de dados assíncrona usando *autologs* em Oracle. Aqui as transações são registadas nos *redo logs* à medida que ocorrem na base de dados fonte, sendo posteriormente enviados para a base de dados destino na qualos dados serão armazenados para futura integração.

2.3.2 **O Sistema do *Microsoft SQL Server***

A Microsoft introduziu o conceito de captura de dados novos e alterados na versão 2008 do seu sistema *SQLServer*, atualmente na versão 2012. Antes desta versão a captura de novos dados teria que ser feita recorrendo a ferramentas que trabalhassem com o sistema, implicando, com isso, a necessidade de fazer alterações aos esquemas das tabelas em que se pretendia fazer a captura das alterações ou introduzir gatilhos nessas mesmas tabelas com as desvantagens inerentes a essa estratégia. Uma das grandes novidades do sistema *SQLServer*, a partir da versão 2008, foi a introdução de um mecanismo de captura de dados interno, com a capacidade de fazer uma leitura assíncrona na procura de alterações no sistema, sem que para isso seja necessário fazer qualquer alteração aos esquemas das tabelas ou recorrer a gatilhos (Sack 2008). Para isso ser possível, o *SQL Server* utiliza um agente interno ao sistema, que mantém um processo de leitura ativa dosficheiros de *log* contendo a informação transacional do sistema, identificando assim quais as transações que

interessam e que tenham sido terminadas com sucesso, guardando a sua informação numa tabela criada propositadamente criada para este efeito.

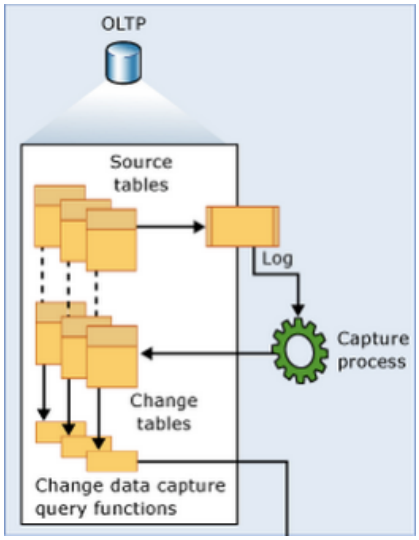


Figura 2-8: Processo de captura de dados em *SQL Server* (Microsoft n.d.).

A Figura 2-8 ilustra de uma forma simplificada o funcionamento do mecanismo de captura de dados do *SQL Server*. Como já foi referido, a informação das transações é continuamente guardada nos ficheiros de *log* do sistema. À medida que a informação é aí introduzida, os dados são identificados e guardados em tabelas com as alterações. Esta funcionalidade pode ser, também, ativada para fazer a captura de novos dados em tabelas específicas, identificadas para esse efeito. Para isso é necessário, primeiro, ativar a captura de dados na base de dados, utilizando um procedimento armazenado designado *sys.sp_cdc_enable_db*. E só depois é possível ativar a captura de dados nas tabelas alvo, recorrendo ao procedimento armazenado *sys.sp_cdc_enable_table*. Como suporte a esse processo, é criada uma tabela específica na qual vão ser guardados os dados de captura.

Na Figura 2-9, podemos ver um exemplo de uma tabela contendo dados capturados. Nela podemos verificar que nos primeiros 5 campos estão guardados os meta dados relativos à captura de dados efetuada.

	__$start_lsn	__$end_lsn	__$seqval	__$operation	__$update_mask	id	nome	dataNascimento	morada	nacionalidade
1	0x00000025000000E90003	NULL	0x00000025000000E90002	2	0x1F	1	miguel	1988-11-14	NULL	1
2	0x00000025000000F10004	NULL	0x00000025000000F10002	3	0x08	1	miguel	1988-11-14	NULL	1
3	0x00000025000000F10004	NULL	0x00000025000000F10002	4	0x08	1	miguel	1988-11-14	Tibaes	1
4	0x00000025000000F60005	NULL	0x00000025000000F60002	1	0x1F	1	miguel	1988-11-14	Tibaes	1

Figura 2-9: Conteúdo de uma tabela CDC em SQL Server 2008.

Na quarta coluna podemos ver os identificadores das operações realizadas internamente no sistema – na Tabela 2-1 podemos ver a sua descrição.

Identificador	**Descrição**
1	Registo removido
2	Registo inserido
3	Registo antes de atualizar
4	Registo depois de atualizar

Tabela 2-1: Descrição dos identificadores dos registos de CDC em SQL Server 2008.

Nos restantes atributos estão os dados a serem capturados. Os dados capturados através de operações de inserção e remoção são colocados em registos únicos para cada operação nessa tabela, enquanto que o resultado de uma operação de atualização é guardado em dois registos sequenciais, em que o primeiro guarda o conteúdo completo do registo antes da atualização, e o segundo guarda o mesmo conteúdo, desta feita após a atualização.

2.3.3 **O Sistema do *MySQL***

O sistema *MySQL* é o SGBD *open-source* mais popular do mundo (IT 2012). Recentemente este sistema foi comprado, sendo agora gerido pela *Oracle*. Neste sistema o conceito de captura de novos dados ou alterados não foi implementado nem está integrado como ferramenta do sistema. Por este motivo, não seria adequado mencioná-lo nesta pesquisa. Contudo, a *Oracle* criou uma API para ler e manipular os ficheiros com as *logs*

transacionais do sistema – *binary logs* (Kindahl & Thalmann n.d.). Com essa API é possível aceder ao conteúdo dos ficheiros *log*, quer lendo diretamente o ficheiro na qual estão registadas, quer acedendo ao seu conteúdo através da base de dados "*Master"* do sistema. Foi com recurso a esta API que foi possível permitir a replicação de informação a partir de um sistema *MySQL* para um outro qualquer, de forma simples e rápida. Utilizando esta abordagem para replicação, o mesmo pode ser feito para captura de dados novos ou alterados, uma vez que esta API permite a leitura e a identificação dessas alterações. Além disso, a API utiliza os ficheiros *log* transacionais do sistema, pelo que podemos beneficiar, consequentemente, das vantagens da captura de dados utilizando ficheiros de *log*.

Figura 2-10 : Exemplo de um ficheiro *log* do *MySQL*, em formato *ROW*.

Os *logs* do *MySQL* podem ser guardados de três formas distintas, nomeadamente: *Row*, *Statement* ou *Mixed*.

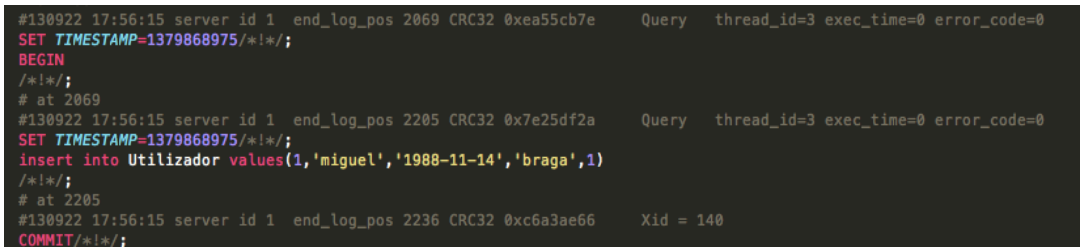


Figura 2-11: Um exemplo de um ficheiro *log* do *MySQL*, em formato *STATEMENT*.

No formato *row* os dados de cada transação efetuada com sucesso são guardados sobre a forma de um código alfanumérico, que é o resultado da codificação dessa mesma transação. Quanto ao formato *statement*, este guarda a corpo da operação executada para aquela transação, o que é bastante útil quando se pretende reexecutar alguma dessas operações. O último formato, uma combinação dos dois primeiros, permite guardar ambas as informações no *log*. O MySql apresenta ainda uma ferramenta auxiliar – designada *MySqlBinLog* – que permite traduzir o conteúdo do ficheiro *log*, quando configurado em formato *row*, para um formato legível ao ser humano. Esta ferramenta pode ser muito útil em processo de leitura dos *logs*.

2.3.4 O Sistema do *MariaDB*

O *MariaDB* surgiu após a compra do *MySQL* pela Oracle. Após essa transação, os fundadores do *MySQL* criaram um novo sistema, que funciona como uma espécie de "filho" do *MySQL,* uma vez que partilha todas as funcionalidades desse sistema, implementando, contudo, algumas novidades e melhorias sobre algumas das suas funcionalidades. O sistema *MariaDB* partilha também com o *MySQL* a falta de um mecanismo interno de CDC, mas apesar disso o *MariaDB* partilha, também, a API, referida que permite efetuar a captura de dados.

2.3.5 O *Oracle GoldenGate*

Adquirido pela *Oracle* em 2010, o *GoldenGate* é uma das "jóias da coroa" da *Oracle*. Ao contrário dos sistemas de que falámos até aqui, todos eles proprietários dos seus fornecedores, o *GoldenGate* é um sistema que consegue interagir com uma fatia importante dos sistemas de bases de dados em processos de captura de dados. Esta versátil ferramenta permite excelentes desempenhos captura em tempo real de dados transacionais, bem como atua como uma plataforma de replicação de dados, o que permite integrar essas funcionalidades com processos ETL de forma bastante simples. A tecnologia utilizada pelo *GoldenGate* permite que os seus clientes tenham uma disponibilidade de serviço praticamente contínua em sistemas críticos, bem como em projetos de integração de dados

em tempo real. A ferramenta apresenta uma plataforma adequada para processos de captura de dados, transformações, e sua entrega em repositórios de dados muito distintos,com características bastante heterogéneas (Oracle & Paper 2012b).

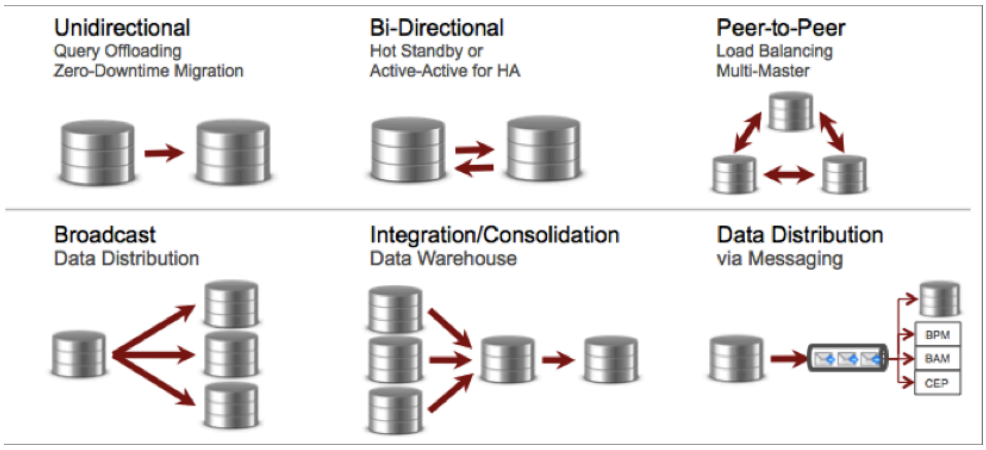


Figura 2-12: Configurações possíveis para CDC com o *Oracle GoldenGate* (Jeffries 2011).

O *GoldenGate* permite também configurar a forma de e para onde os dados são enviados, oferecendo uma série de configurações para processar a entrega dos dados entre a fonte e o destino. A Figura 2-12 apresenta as possíveis configurações para CDC do *Oracle GoldenGate*, nas quais se destacam o *broadcast* da informação, que permite que os dados extraídos de uma fonte sejam enviados para dois ou mais sistemas de destino, ou Alternativamente, a integração de dados num sistema central a partir de duas ou mais fontes heterogéneas. São, pois, várias as vantagens desta ferramenta. De um leque muito vasto, destacamos aqui as seguintes:

- Latência reduzida – o intervalo de transferência entre a captura e a entrega dos dados situa-se na ordem dos microssegundos. Ainda neste contexto, a ferramenta consegue manter essa performance em situações em quelida com grandes volumes de dados;
- Heterogeneidade – a captura e a entrega dos dados é possível entre vários sistemas relacionais, nas plataformas mais utilizadas no mercado;

- Integridade transacional – mantem a confiança e a eficácia dos dados transacionais quando estes são transferidos entre sistemas, assegurado as propriedades ACID e a integridade referencial.

Nestes casos, a captura de dados é efetuada de forma não invasiva, uma vez que se utiliza os ficheiros de *log* transacionais para identificar e capturar a informação pretendida (Khatiwada, S. 2013). Este é um dos motivos que permite ao *GoldenGate* atingir uma latência muito reduzida entre a captura dos dados e a entrega no seu destino, uma vez que os ficheiros de*log* são lidos sem que haja intrusão no sistema fonte no qualestão a ser capturados. Com esta ferramenta, para efetuar a captura de dados, é necessário configurar um processo interno – *extract* – em que definida a ligação ao conteúdo dos ficheiros de *log* sobre os quais se efetuará a captura de dados. Após a captura dos dados ter sido realizada, estes são guardados em ficheiros, designados por *trail* files. A partir daqui, tendo por base a configuração pretendida, os dados são enviados para o sistema no qual serão guardados em ficheiros *trail* e, só então, aplicados na localização pretendida.

	Oracle	***SQL Server***	***MySql***	***MariaDB***	***GoldenGate***
Colunas auditoria					
Gatilhos	Efetua uma leitura **síncrona**; Usa gatilhos para identificar alterações nas tabelas fonte, enviando-os para tabelas de destino; As tabelas de destino são guardadas na própria base de dados;				
***Logs* transacionais**	Leitura **assíncrona**, usando os ficheiros de *log* para identificar transações terminadas com sucesso; Depois de identificadas envia	Efetua uma leitura assíncrona dos ficheiros de *log*; Lê os ficheiros de *log* à procura de transações	Disponibiliza uma API dedicada à leitura e processamento dos ficheiros de*log*; A API permite	Disponibiliza uma API dedicada à leitura e processamento dos ficheiros de*log*; A API permite	Lê os ficheiros de *log* transacionais à procura de transações terminadas para. Quando as encontra

	para tabelas que guardam dados capturados; **Assíncrona** ***AutoLogs*** permite enviar dados capturados para outras bases de dados; ***HotLogs***envia os dados capturados para tabelas dentro do sistema fonte onde é feita a captura;	terminadas com sucesso, enviando-as para tabelas com os dados capturados; Tabelas de destino são guardadas na base de dados fonte;	identificar e capturar as alterações feitas numa tabela;	identificar e capturar as alterações feitas numa tabela;	envia para tabelas que guardam os dados capturados; Permite enviar os dados capturados para outros sistemas externos. Disponibilizando um vasto leque de configurações de distribuição e entrega dos dados entre vários sistemas

Tabela 2-2: Comparação de ferramentas em termos de funcionalidades de CDC.

2.4 O Método Escolhido para a Captura de Dados

A estratégia que foi escolhida para efetuar a captura de dados novos ou alterados é aquela que utiliza ficheiros de *log* transacionais, como forma de identificar as eventuais mudanças efetuadas sobre os dados fonte. Tal como já foi referido anteriormente, vários dos principais fornecedores de sistemas de bases de dados incluíram nos seus sistemas um jornal que regista todas as atividades que ocorrem nos seus sistemas, fazendo-o como uma forma de guardarem informação necessária para se evitarem alguns problemas que possam advir de falhas operacionais que possam conduzir uma base de dados a um estado inconsistente. Quando isto acontece, é possível utilizar-se o referido jornal para identificar as transações que tenham sido interrompidas nesses momentos e com base nessa informação proceder, se necessário à recuperação do estado consistente do sistema. Ora estes jornais, ao guardarem as transações e os seus resultados, tornaram-se um "alvo" natural e uma fonte de informação a explorar para um propósito que não o inicial, ou seja a captura de dados.

É sabido que estratégias que utilizam gatilhos deterioram a performance do sistema no qualestão a ser usados. O mesmo também se aplica a estratégias suportadas por atributos de etiquetas temporais. Todavia, neste último caso, o impacto é um pouco menor. Contudo, na captura de dados baseada em *logs* transacionais, a estratégia passa sempre por manter uma leitura ativa dos ficheiros *logs* transacionais, identificando em cada momento quais os novos dados (resultantes de inserções, atualizações ou remoções de registos) e, de seguida, capturá-los para posteriormente os entregar num local de destino. Desta forma, reduziremos o impacto dos processos de captura de dados na performance geral do sistema, uma vez que só os ficheiros contendo os jornais é que serão lidos, não gerando, assim,qualquer conflito com o sistema que está a escrever neles. A latência entre o momento em que são capturados e posteriormente entregues será, também, reduzida, sendo esta uma estratégia menos intrusiva (Eccles *et al.* 2010), uma vez que atua diretamente sobre o conteúdo dos *logs*, não sendo por isso necessário efetuar qualquer tipo de alteração aos esquemas das tabelas das bases de dados, nem às aplicações que utilizem o sistema operacional. Poderá haver, porém, alguma latência entre o momento em que as transações são terminadas e o momento em que a informação chega aos sistemas de destino, uma vez que, para se carregar a informação, se espera que as transações sejam dadas como terminadas. Mas nem tudo são boas notícias. Apesar das claras vantagens desta abordagem, existem alguns problemas que devem ser avaliados e resolvidos com algum cuidado. Ao longo do próximo capítulo, serão abordados os vários problemas inerentes à implementação de um processo de captura de dados, bem como quais as formas mais utilizadas para os eliminar ou atenuar.

3 Problemas na Captura de Dados

A escolha do método de captura de dados é um dos pontos mais importantes na implementação de um processo deste género. Contudo existem várias considerações que devem ser tidas em conta, pois podem limitar a escolha dos métodos a utilizar. Por isso mesmo devem ainda ser analisados alguns problemas que limitam as operações de cada método. Alguns desses problemas são específicos de cada método, enquanto que outros são bastante comuns a todos eles, necessitando de ser resolvidos (ou pelo menos atenuados) por forma a não provocarem erros de consistência ou de perda de informação relevante durante o funcionamento do processo.

3.1 Caraterização do Caso de Estudo

Uma conhecida empresa – "A ZonaFunda" – desenvolve a sua atividade na área do retalho, sendo a venda de artigos o grosso do seu volume de negócios. Os preços dos seus artigos são calculados ao final do dia, sempre com base nas vendas realizadas até a essa altura, e no tipo de clientes mais frequentes. A informação da empresa está armazenada num *data warehouse* e organizada de acordo com as necessidades de gestão dos *stocks* dos vários armazéns distribuídos em vários pontos estratégicos de entrega. Em termos gerais, com essa informação a empresa procura conhecer o tipo de produtos que cada tipo de cliente

consome, com vista a estabelecer um plano adequado de promoções para os seus clientes. Até então, o refrescamento dessa informação era feito diariamente, em modo *batch*. Isso permitia calcular os preços dos produtos e suas eventuais promoções para o dia seguinte, com base nos resultados das vendas até aquele dia.

Recentemente essa empresa apostou num aperfeiçoamento do seu sítio *Web* de venda de produtos com base na experiência adquirida com os vários processo de interação com os seus utilizadores. Entre outras funcionalidades, o sistema vai guardar todo o tipo de produtos que o utilizador pesquise ou que visualize no sítio, com o objetivo de oferecer algumas sugestões ao utilizador em tempo real, oferecendo assim uma oferta mais atrativa e personalizada ao cliente. Com isso melhora também o conhecimento que tem acerca do utilizador e dos tipo de produtos que este usualmente consome.

Esta nova experiência de utilização obrigou a empresa a repensar a forma como o seu *datawarehouse* funciona, tendo decidido reestruturar o seu sistema de *datawarehousing* para suportar processos em tempo real. Uma das características desta reestruturação passou pela alteração da forma como os dados são capturados das fontes, um processo que se faz agora por cálculo diferencial, numa janela de oportunidade noturna. Esta alteração, levou a equipa responsável a estudar várias soluções que satisfizessem esse requisito, sem causar impacto nas suas fontes de informação e que reduzisse ao mínimo a latência entre o momento da captura dos dados e a sua entrega. A estratégia que foi definida, utiliza uma abordagem de captura baseada em ficheiros de *log*, para que, a partir deles, possa capturar a informação em tempo real. Ao longo dos próximos capítulos, serão abordados os principais problemas adjacentes a este tipo de estratégia de captura de dados, quais os seus desafios principais e que soluções usualmente são adotadas para cada um deles.

3.2 A Estrutura do Ficheiro *Log*

A identificação da estrutura do ficheiro de *log* é, talvez, a maior barreira que se levanta a um processo de captura de dados, quando se pretende utilizar uma abordagem baseada em jornais de transações. O que torna isto um problema sério, é o facto de a forma como os fabricantes estruturam esses ficheiros ser usualmente desconhecida ou de não existir qualquer tipo de documentação sobre tal estrutura. Tal pode ser justificado pelo motivo de quererem preservar só para si o conhecimento desse conteúdo de forma a satisfazer, em exclusivo, as necessidades dos sistemas implementados. Por ser desconhecida essa estrutura, a implementação dum sistema deste género, quando criada por fabricantes que não sejam aqueles que são proprietários dessas mesmas estruturas, torna necessário fazer-se uma pesquisa e identificação de qual a estrutura desses ficheiro, por forma criar soluções que permitam trabalhar com essa estrutura e nela identificarem a informação considerada relevante para o processo de captura de dados em causa. Assim, é necessário saber identificar e responder a algumas questões que julgamos serem pertinentes no âmbito destes processos.

Em primeiro lugar, os ficheiros de *log* estão em linguagem máquina, situação que requer que qualquer ferramenta tenha que ser capaz de efetuar a sua conversão para uma linguagem "legível" pelo ser humano, de forma a se conseguir identificar como é que o ficheiro está estruturado, através dos seus conteúdos e padrões pertinentes. Mais ainda, é necessário um estudo mais aprofundado relativamente as aspectos relacionados com a criação de algoritmos capazes de interpretar e processar a estrutura de um qualquer ficheiro *log* e, consequentemente, capturar a informação pretendida.

Todos os fabricantes apresentados anteriormente (secção2.3) utilizam os próprios *logs* para fazer a captura de dados, uma vez que conhecem, com ninguém, a sua estrutura e conteúdo. Em particular, o *GoldenGate*, como ferramenta *multi*-sistema, é capaz de capturar dados a partir de ficheiros *log* de vários sistemas, revelando que conhece a estrutura dos sistemas mais utilizados a nível mundial. Esta circunstância seria muito

vantajosa para o presente estudo mas a informação associada com o *GoldenGate* também está muito bem guardada, o que não é uma surpresa. Contudo, existem alguns detalhes na sua documentação que ajudam a compreender a forma como se pode atuar sobre ambientes de dados heterogéneos num processo de CDC.

Em segundo, temos que assumir que a estrutura dos ficheiros *log* varia conforme os requisitos e necessidades organizacionais de cada fabricante para o sistema. Além disso, essa estrutura não é, usualmente, divulgada pelos fabricantes, sendo por isso necessário estudar e criar algoritmos capazes de ler e identificar a estrutura de qualquer um dos formatos. Porém, existem algumas ferramentas, criadas por outros fabricantes, que são capazes de ler e traduzir os ficheiros *log* de vários SGBD e outras fontes de informação mas que não são de acesso livre, o que traz as desvantagens conhecidas em termos de custos, especialmente os relacionados com a manutenção futura do sistema.

3.2.1 *MySql* e *MariaDB*

Já referimos anteriormente que, o *MariaDB* é um sistema de bases de dados que funciona sobre um sistema *MySQL,* partilhando todas as suas funcionalidades .Assim, não será de surpreender o facto de que as *logs* geridas por cada um desses sistema sejam semelhantes, ou mesmo iguais. Os ficheiros *log* transacionais, e a forma como estão estruturados, podem não vir configurados para ficarem ativos no momento em que o sistema é instalado. Como tal, devem ser validados e ativados, se necessário. Para que possam ser utilizados, precisamos de indicar a localização física na qual serão armazenados, bem como o formato no qual serão armazenados. Na prática, o MySQL permite que a forma como as operações dentro das transações são escritas nos ficheiros *log*. Possa ser realizada de várias maneiras, nomeadamente *STATEMENT*, *ROW* e *MIXED*. Estas podem ser alteradas a qualquer momento. A primeira dessas formas – *STATEMENT* – guarda o corpo da instrução (ou das instruções) da transação que foi mandada executar. O segundo – *ROW*– guarda os resultados das transações linha a linha, o que significa que quando uma operação manda

executar 5 inserções, este guarda o corpo completo de cada uma das inserções requeridas. Porém, a informação neste formato é guardada sobre a forma de um valor alfanumérico, que é resultado de uma operação de encriptação de *base64* (Anon n.d.). O terceiro e ultimo formato – MIXED– guarda as instruções utilizando os formatos das duas possibilidades anteriores, sendo a escolha do formato efetuada em tempo real, no momento da escrita.

Embora o *GoldenGate* não especifique a forma como os ficheiros de *log* dos sistemas que reconhece estão estruturados, os documentos sobre a forma de configurar um processo com o *MySQL* indicam, como pré-requisito, que os ficheiros *log* sejam previamente alterados para o formato *ROW*. Apesar de neste formato as transações estarem codificadas em *base64*, uma forma de ultrapassar este problema é com recurso às ferramentas incluídas disponíveis no pacote do *MySQL*, em especial a uma que é capaz de traduzir os *logs* para um formato legível pelo ser humano: o *MySqlBinLog*. Esta ferramenta, associada com a opção *VERBOSE* permite gerar código comentado do resultado de cada uma das transações realizadas, o que facilita imenso o processo de captura.

3.2.2 **SQL *Server***

No SQL *Server*, na sua versão se 2008, a forma de aceder ao conteúdo dos *logs* é distinta do sistema referido anteriormente. A forma mais completa de aceder ao conteúdo dos *logs* – em vez de aceder ao ficheiro físico - é através de duas funções internas do sistema: a *fn_dblog()* e a *fn_dump_dblog()* , sendo queaqui iremos abordar a primeira. Apesar de ser conhecida a existência da função *fn_dblog()*, não existe, porém, documentação oficial sobre o tipo de resultados que esta devolve, nem sobre a forma como os ficheiros estão estruturados. Contudo, existe alguma documentação, não oficial, que expõe algumas considerações sobre esta função e que se revelaram bastante valiosas para o desenvolvimento dos trabalhos desta dissertação. A função *fn_dblog()* permite obter uma resposta sobre o conteúdo de uma parcela dos *logs*, referente ao bloco de memória em que está a ser escrito naquele momento, sendo por isso muito útilem processos de captura de

dados em tempo real. De acordo com Emergis n.d., sabe-se que esta função devolve um conjunto de registos, com 101 campos, dos quais 10 deles são muito pertinentes para a captura de dados.

Na Figura 3-1 podemos ver o tipo de resultado que essa função disponibiliza. Os campos *Operation, AllocUnitName, [Row Log Content 0]* mostram-nos, respetivamente, o tipo de operação ocorrida, o nome da tabela onde ocorreu e o último mostra um campo em formato hexadecimal que mostra o conteúdo dos *logs*. Este último campo, representa a informação do corpo da instrução que foi mandada executar, mas guardando seguindo a estrutura interna do *SqlServer*, estrutura essa que segundo (Delaney *et al.* n.d.), se apresenta da seguinte forma e pode ser descodificada com recurso à Tabela 3-1.

SPID	Begin Time	End Time	Current LSN	Previous LSN	Operation	Context	AllocUnitName	RowLog Contents 0	Modify Size
54	2013/12/07 19:12...	NULL	00000036:00...	00000000:0...	LOP_BEGIN_XACT	LCX_NULL	NULL	NULL	NULL
NULL	NULL	NULL	00000036:00...	00000036:0...	LOP_INSERT_ROWS	LCX_HEAP	dbo.Utilizador	0x30000F000100000026140B0100000005000801001C006D696775656C	0
NULL	NULL	2013/1...	00000036:00...	00000036:0...	LOP_COMMIT_XACT	LCX_NULL	NULL	NULL	NULL

Figura 3-1: Resultado de uma *query* com 10 dos107 campos retornados pela função*fn_dblog()*.

Com a informação apresentada na Tabela 3-1, a descodificação e a interpretação do registo torna-se bastante mais simples, o que facilita muito o processo de captura de dados .No caso da captura de dados deste sistema, e no que diz respeito ao *OracleGoldenGate*, novamente, as configurações mostram um pré-requisito que deixa a entender, que se pode utilizar esta função para realizar um processo de captura de dados.

Item	**Espaço em memória**	**Descrição**
1	1 byte	Byte de estado que contem propriedades da linha
2	1 *byte*	Não utilizado em *SqlServer* 2008
3	2 *bytes*	*Offset* na linha até à posição onde se indicar o número total de colunas do registo
Somatório do tamanho das colunas fixas	N *bytes*	Localização das várias colunas de tamanho fixo com conteúdo no registo
5	2 *bytes*	Numero total de colunas do registo
6	1 *bit* por cada	Identificação das colunas cujo valor é *null* –

	coluna do registo	*NullBitmap*
7	2 *bytes*	Número de colunas de tamanho variável do registo
8	2 bytes por cada coluna de tamanho variável	Intervalo com posição de fim de cada coluna variável no registo
Somatório das colunas de tamanho variável	N *bytes*	Localização das várias colunas de tamanho variável com conteúdo no registo

Tabela 3-1: Tabela com a estrutura interna de um registo em Sql Server 2008.

Segundo Oracle n.d., esse pré-requisito de configuração do *GoldenGate,* para *SqlServer 2008*, é a criação de um objeto *ODBC* para que seja possível o processo *Extract* ler as transações ocorridas. Assim, a função *fn_dblog*() pode estar por trás desse processo, com o objetivo de fornecer informação para a leitura dos ficheiros de *log*.

3.3 Eliminação de Transações sem Sucesso

A importância da forma como os ficheiros de *log* estão estruturados é fundamental para a estruturação de um algoritmo eficaz para a captura dos dados neles contidos. Relacionada com essa estrutura, está a escolha do momento e a forma como se vai proceder à identificação da transação e do seu conteúdo, isto é, definir a forma como se vai identificar o inicio e o final de uma transação e como proceder no caso da transação ter sido ou não efetuada com sucesso. Este é um desafio importante, uma vez que permite delimitar as várias secções dos ficheiros *log* que se quer tratar, mas também para saber quando é que o conteúdo capturado deve ser ou não efetivado. A latência entre o momento que os dados são capturados e o momento em que são entregues no destino é também importante, pois influencia diretamente a disponibilização dos dados em tempo real ou tempo útil. A forma lógica de resolver isto, que é algo que todos os sistemas que são abordados neste documento utilizam, passa por guardar apenas as transações que foram terminadas com sucesso. Para isso, é guardada a informação de todas as operações que ocorrem na transação até ao momento em que se encontra a instrução que finaliza a transação, com sucesso ou não, através das instruções *COMMIT* ou *ROLLBACK,* respetivamente. Caso seja encontrado o estado *COMMIT*, nesse momento todas as instruções guardadas até ali

são guardadas efetivamente nas estruturas que depois serão entregues no sistema de destino. Caso se encontre o estado *ROLLBACK*, as operações guardadas até ali são eliminadas.

3.4 Algoritmos para a Filtragem de Conteúdos

Depois de encontrada e delimitada a estrutura dos ficheiros de *log*, tem-se que definir uma estratégia capaz de filtrar a informação que é escrita nesses ficheiros, que é um autêntico desafio. Anteriormente, na secção 3.3, foram já indicadas algumas formas para delimitar o conteúdo dos ficheiros *log* apenas ao conteúdo das transações terminadas com sucesso. Isto pode ser visto como um primeiro e importante passo de filtragem. O passo seguinte passará por identificar as operações de inserção, remoção e atualização de registos, bem como o seu conteúdo, de forma a que seja possível filtrar essa informação e guardá-la em estruturas para enviar para outros destinos. A forma como os sistemas referidos anteriormente filtram a informação dentro dos ficheiros de *log*, ou como delimitam esse conteúdo para captura, é desconhecida, e carece de documentação. Porém, com a realização do primeiro passo temos uma base para realizar esse processo de filtragem.

3.5 O Tamanho de um Ficheiro *log*

Os ficheiros *log* dos sistemas podem ser configurados para que seja criado um novo ficheiro, quando se atinja um determinado limite superior ou, então, um determinado tempo limite para a utilização do ficheiro *log*com que se está a trabalhar naquele momento, passando os dados transacionais a serem guardados no novo ficheiro. Este processo repete-se ao longo da execução do sistema, de acordo com uma dada configuração.Ora, o tamanho dos ficheiros *log* pode crescer de uma forma exponencial em poucos minutos, conforme o volume das transações realizadas no sistema. Esse crescimento pode ter algum impacto no desempenho do processo de registo de transações. Convencionalmente, a resolução desse problema passa por enviar os ficheiros para serem armazenados no exterior

do sistema ou então proceder à remoção do conteúdo do ficheiro de log quando se considera que passou o seu tempo de utilidade. Tudo isto,porque, por exemplo, pode já existir uma cópia de segurança que cubra a informação do sistema escrita naqueles ficheiros ou porque, simplesmente o tamanho do ficheiro torna-se um problema.

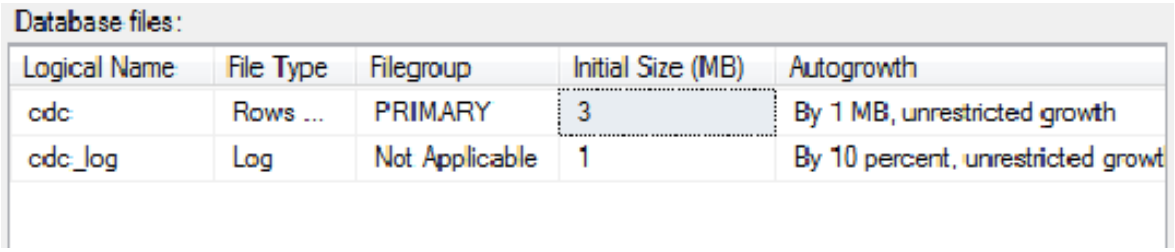
Um mecanismo de captura de dados baseado em ficheiros *log* deve ser capaz de identificar as alterações efetuadas sobre os dados antes que parte da informação que se pretende obter dos ficheiros, se torne inacessível no momento, quer por os ficheiros *log* terem sido deslocados para localizações físicas externas, quer por terem sido removidos.Para contornar este problema, o *Goldengate* pode atuar de várias formas. Quando se trata do *MySQL*, é necessário adaptar o ficheiro de configuração do servidor para que o tamanho máximo de cada ficheiro log seja de 4096 *bytes*. No caso do *SQL* Server 2008, o Goldengate obriga que as opções de truncar os conteúdos dos ficheiros *logs* sejam desativadas, de forma a que esse conteúdo não possa ser eliminado.

3.6 Ficheiros de *Log* Armazenados Exteriormente

Na secção anterior referimos que quando os ficheiros *log* atingem um determinado tamanho ou quando o seu conteúdo se torna desnecessário é criado um novo ficheiro para fazer o acolhimento da *log* e os ficheiros anteriores podem ser enviados para serem armazenados no exterior. No caso em que se pretenda efetuar um processo de captura de dados em tempo real, os ficheiros *log* que estarão a ser utilizados naquele momento para escrita das transações do sistema serão os necessários para que a captura possa ser feita satisfazendo as necessidades de tempo real. Apesar disso, pode ser opção dos utilizadores agendar um processo de captura para um dado intervalo de tempo, que pode ter ocorrido num momento passado ou então para uma altura que possa agrupar informação de um período de tempo. Um exemplo deste cenário é a execução de um processo de extração em modo *batch*, que irá no final de uma semana de trabalho de uma empresa, por exemplo, capturar toda a informação existente numa fonte de dados para posteriormente ser

trabalhada por um sistema de *ETL*. Numa situação como essa, é necessário recorrer ao conteúdo dos ficheiros de *log* que possam ter sido guardados exteriormente, em datas anteriores àquele momento de captura. Por isso, deve-se manter essa informação de forma acessível ao sistema de captura de dados.

No *GoldenGate,* as configurações do *MySQL* relacionadas com o tamanho dos *logs,* a sua localização e o formato dos ficheiros *log* devem ser mantidas de acordo com as especificadas nas secções anteriores, uma vez que, no caso de ser necessário ler o conteúdo de ficheiros *log* armazenados em endereços físicos diferentes, a estrutura desses *logs* será em tudo igual à dos ficheiros onde o sistema escrever os dados transacionais após a sua ocorrência. Já no caso, do sistema *SQL Server*, deve ser indicado o endereço físico da cópia de segurança do ficheiro *log* que é gerado na criação da base de dados, para que a partir dele, possa ser lido todos os dados transacionais ocorridos no sistema, quando se dá a necessidade de uma captura num dado período de tempo passado. Para ler esses ficheiros, deve ser utilizada a função interna do sistema – *fn_dump_dblog()*. Esta função é em quase tudo igual à que foi referida anteriormente para a leitura do bloco ativo, sendo uma das diferenças a forma como acede ao ficheiro, sendo necessário especificar o caminho para a cópia de segurança do ficheiro *log* a partir do qual se pretende capturar os dados. Essa cópia de segurança do ficheiro *log* guarda uma cópia dos dados transacionais do ficheiro *log* que foi criado no momento em que a base de dados foi criada (Figura 3-2).


Database files:

Logical Name	File Type	Filegroup	Initial Size (MB)	Autogrowth
cdc	Rows ...	PRIMARY	3	By 1 MB, unrestricted growth
cdc_log	Log	Not Applicable	1	By 10 percent, unrestricted growt

Figura 3-2: Criação de uma base de dados e de um ficheiro log em SQL Server 2008

O resultado da execução da função *fn_dump_dblog()* a partir da cópia de segurança do ficheiro *log*, é em tudo igual à função *fn_dblog()*, pelo que a forma como os dados são tratados e capturados é em tudo igual. Na configuração de uma instância do sistema GoldenGate, é previamente requerido o caminho para a cópia de segurança do ficheiro *log*, para que quando necessário o sistema possa capturar dados previamente armazenados.

3.7 Outros Problemas

Além dos problemas e desafios supracitados, existem outros que devem ser relevados, uma vez que são cada vez mais recorrentes. Além do recurso a ficheiros de *log* transacionais, existem outros tipos de formatos para suporte à captura de dados que devem ser considerados. É comum, em empresas mais pequenas, os gestores de negócio guardarem a informação das suas atividades em ficheiros *Excel*, *csv* ou texto, com estruturas que só eles conhecem e só a eles interessam. Porém, tais situações podem ser problemáticas, podendo levantar inúmeras questões para além das estruturas desses ficheiros. Se os ficheiros *Excel* já possuem funções internas para efetuar captura de dados, levando o problema para a identificação dessas alterações, os ficheiros *csv* e os ficheiros de texto requerem formas alternativas para se encontrar qual a informação que neles é nova ou foi modificada. Os sistemas devem ser capazes de interpretar ou de disponibilizar formas de dar a conhecer os formatos e os conteúdos que se pretendem capturar.

4 Processos de Captura de Dados baseados em Ficheiros *log*

Desde o início dos trabalhos desta dissertação que o objetivo era o de criar um componente universal para suporte a processo de captura de dados. Esse componente universal foi projetado e implementado. Para a sua implementação foi escolhida a linguagem JAVA, dadas as suas características de aplicação universais e o vasto leque de métodos e API que coloca à nossa disposição, e o IDE Eclipse como plataforma de desenvolvimento, dada sua grande facilidade em termos de utilização e as funcionalidades que disponibiliza para o suporte de o desenho e o desenvolvimento atrativos das aplicações.

Assim, ao longo das próximas secções apresentar-se-ão as diversas formas como cada um dos problemas enunciados anteriormente (capítulo 3) foram abordados e tratados, sendo também enunciadas algumas das características que podem ser relevantes na melhoria das soluções encontradas. Apresentar-se-ão ainda as principais características da ferramenta desenvolvida, dando particular ênfase àquelas que facilitam o processo de captura de dados e que permitem uma melhor adaptação às restrições de informação de cada empresa. Para ilustrar a forma como a captura de dados é processada e como os vários problemas abordados foram resolvidos, os exemplos, os processos e objetos de dados escolhidos para

validação e teste serão como comuns aos 3 sistemas que selecionámos: *MySQL*, *MariaDB* e *SQL Server* 2008.

4.1 Eliminação de Transações sem Sucesso

De forma a capturar apenas os dados novos ou que foram modificados numa base de dados, e consequentemente registados nos ficheiros de *log* respetivos, optou-se pela estratégia de fazer uma leitura contínua desses ficheiros. Assim, começar-se-á a guardar qualquer operação a partir do momento em que se encontra a instrução de início de uma transação (Microsoft n.d.). A partir desse momento as operações que operarem sobre as tabelas nas quais queremos capturar as modificações de dados ocorrida elas serão guardadas em memória. No momento em que o processo deteta a marca de final de transação o processo de captura termina. Se o estado final for *COMMIT*, as instruções guardadas em memória serão materializadas e enviadas para o destino correspondente. Caso o estado seja *ROLLBACK*, tudo o que foi guardado em memória será eliminado, uma vez que não interessa capturar informação que não consta, de facto, na base de dados. Tal conduziria a possíveis situações de inconsistência de dados.

Na tabela 4-1 pode-se ver quais as instruções que delimitam as transações nos *logs* dos sistemas SQL *Server* 2008, *MySQL* e *MariaDB*. De referir que, aí apenas estão presentes os delimitadores para casos de sucesso. Mas se o valor final não for um destes, então os dados da transação são descartáveis. Estas serão as condições que os algoritmos terão que ter em conta para delimitar e identificar as operações num ficheiro *log*.

	SQL Server 2008	**MySQL**	**MariaDB**
Inicio de transação	LOP_BEGIN_XACT	BEGIN	BEGIN
Final de transação	LOP_COMMIT_XACT	END	END

Tabela 4-1: As marcas de delimitação das transações.

4.2 A Estrutura do Ficheiro *log*

Anteriormente verificámos que a identificação e caracterização da estrutura de um ficheiro *log* era um dos problemas mais importantes que devem ser resolvidos para o bom funcionamento de um processo de captura de dados baseada em *logs* transacionais. Exatamente por isso, foi a esse processo que dedicámos mais atenção, pesquisando com persistência e de forma detalhada toda a informação que pudemos sobre a forma como os *logs* dos sistemas SQL *Server* e *MySQL*, em particular, estão estruturados. Após tal trabalho ter sido realizado, conseguiu-se identificar e caracterizar alguns blocos de informação bastante completos sobre os referidos ficheiros, que permitiram estabelecer de forma bastante concreta o seu modo de funcionamento.

A identificação da estrutura dos ficheiros de *log* é importante para que o processo de captura de dados seja capaz de filtrar a informação de cada registo de forma completa, acompanhando o que acontece no sistema relativamente a cada uma das três instruções – inserção, remoção e atualização – que operam sobre o conteúdo que queremos capturar. Assim, a partir do ficheiro *log,* e para cada uma dessas instruções, deve ser possível encontrar a informação completa de um registo que possa ter sido inserido ou removido e, no caso de um registo ter sido atualizado, ser possível reter a informação completa do registo, o que neste caso, pretende-se guardar o conteúdo do registo antes e após ser atualizado, para que se possam compreender as diferenças. Além da estrutura de um ficheiro *log* poder variar de acordo com os requisitos do próprio fornecedor, pode variar também com a própria versão desse sistema. Neste trabalho vamos considerar os sistemas *MySQL* 5.3 ou superior, *MariaDB* a partir da versão 5.2, uma vez que assenta numa versão *MySQL*, e o sistema SQL *Server* 2008.

4.2.1 O *MySQL* e o *MariaDB*

Como já referido, o sistema *MySQL* disponibiliza uma aplicação – a *Mysqlbinlog* – que permite ler os ficheiros de *log* em formato binário e converter os dados que aí estão contidos para um formato legível pelo ser humano (Bell et al. 2010). Sabe-se, também, que é possível guardar a informação dos seus ficheiros de *log* em três formatos diferentes: *STATEMENT*, *ROW* ou *MIXED*. Segundo a documentação sobre configurações do *GoldenGate* para *MySQL* (Oracle 2012), um dos pré-requisitos dessa configuração é que os ficheiros *log* sejam guardados no formato *ROW*. E de facto, esta alteração mostrou-se muito importante, uma vez que quando se aplica essa função com a marca "*—verbose*" sobre um ficheiro *log* nesse formato, os resultados tornam-se esclarecedores (figura 4-1).

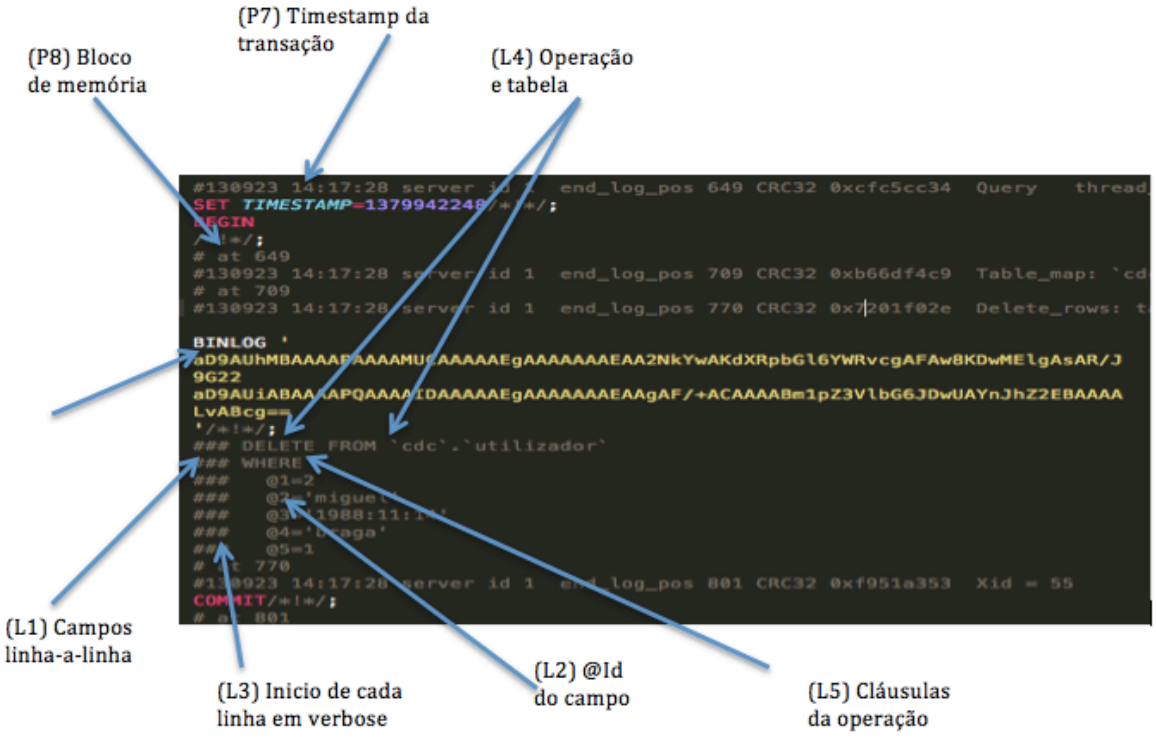


Figura 4-1: Identificação das características de um ficheiro *log* em *MySQL* formato *ROW*

Na Figura 4-1 podemos verificar que, por exemplo, o resultado da conversão de cada transação de tipo alfanumérico (formato *ROW*) para um formato "legível", coloca o corpo

de uma instrução separado por linhas (L1).Cada uma dessas linhas, com o corpo da instrução, é iniciada com a sequência de caracteres '###', que indica que essas linhas não passam de comentários (L3). Quando a instrução é escrita neste formato, o tipo de operação – inserção, remoção ou atualização –, e o nome da tabela na base de dados onde a operação foi executada é escrita na primeira linha (L4), Após ter sido escrito esse conteúdo, é inserida informação sobre o tipo de cláusula da operação – *SET* ou *WHERE* - que ocorre sobre os atributos (L5). Esses, os atributos e o seu conteúdo, são escritos um por linha, com a respetiva informação em relação à cláusula da operação. Importa salientar que a designação dos atributos não é escrita, sendo identificado pelo carácter '@' seguido de um valor inteiro, correspondente à posição do atributo na ordenação tabela aquando da sua criação (L2). Aí, também se conseguem identificar algumas linhas com código adicional sobre a data - em formato *timestamp* –, na qual ocorreu a transação (L7), e um valor do bloco de memória do ficheiro no qual foi guardado o registo da transação (P8).

Como foi referido inicialmente, é importante conseguir identificar a estrutura completa de um registo no ficheiro *log*, tenha ele sido operado por uma inserção, atualização ou remoção. De facto, é possível pode constatar que essa informação está toda presente. Numa inserção saltam à vista todos os campos de uma tabela, mesmo aqueles que o sistema sabe que possuem valores pré-definidos. No caso das remoções, o ficheiro *log* guarda na cláusula WHERE cada um dos campos do registo, garantindo assim que a informação dos registos fique completa, mesmo quando a instrução que tenha conduzido àquela operação não constasse sequer a totalidades dos atributos escritos no *log* (Figura 2-10). Isto pode acontecer quando se elimina um registo em que um atributo respeita a condição para ser eliminado, sendo que no *log*, a cláusula guarda a informação completa do registo. Por fim, a instrução de atualização coloca todos os atributos da tabela na cláusula *WHERE*, tal como acontece também com a instrução de remoção. Neste casos os dados novos do registo são colocados na cláusula *SET*.

Com esta informação acerca de como traduzir os ficheiros em formato binário para um formato que pode ser lido pelo ser humano, torna-se bem mais simples criar um algoritmo capaz de filtrar e capturar os dados que sejam considerados pertinentes para cada utilizador. Contudo, e apesar de se ter conseguido encontrar uma estrutura concreta do esquema dos ficheiros *logs*, para se obter este resultado é sempre necessária utilizar-se a função *mysqlbinlog* para fazer a interpretação do ficheiro *log*. Apesar deste processo de interpretação poder ser considerado um problema, uma vez que não se utiliza diretamente o ficheiro binário, existem algumas vantagens na utilização desta função, a qual pode ter um impacto bastante positivo num processo de filtragem de informação.

4.2.2 O *SQL Server* 2008

Ao invés do sistema anterior, no *SQL Server* 2008 a forma de aceder ao conteúdo dos ficheiros de *log* não é feita acedendo diretamente ao ficheiro físico do sistema, mas fazendo a leitura do conteúdo do bloco de memória onde estão a ser escritos os dados transacionais ou a leitura da cópia de segurança dos ficheiro *log*, para leitura de transações que não estão no disponíveis naquele momento no *log*. Para este caso em particular, e porque se pretende efetuar um processo de captura de dados em tempo real, isto é, capturar os dados novos ou modificados à medida que as operações vão ocorrendo no sistema, optámos pela opção de fazer a leitura do bloco de memória no qual os dados transacionais são escritos a cada momento – utilização da função *fn_dblog()*. Esta função já foi explicada anteriormente, bem como o tipo de resultados que retorna. É importante referir que, o resultado que é devolvido é especifico da base de dados em causa, o que impõe algumas restrições aos processos de captura da informação. Além disto, esta função poder ser parametrizada, com dois parâmetros, ambos opcionais, que indicam as posições inicial e final do bloco de memória no qual estão as transações a serem escritas. Esses dois parâmetros correspondem aos valores *[Previous LSN]* e *[Current LSN],* que indicam, respetivamente, os blocos de memória inicial e final em que cada operação é registada. Estes são apenas dois dos 101 campos que são devolvidos quando se faz a invocação da

função *fn_dblog()*. Porém, bastam apenas dez desses campos para satisfazer os requisitos de captura de dados que pretendemos para este trabalho (Tabela 4-2).

Atributo	**Descrição**
SPID	Identificador do processo da transação
Begin Time	Data de início da transação
End Time	Data de fim da transação
AllocUnitName	Nome da tabela onde ocorre a operação
Context	Contexto em que se aplica a instrução
Operation	Nome da operação ocorrida
Current LSN	Bloco de memória atual
Previous LSN	Bloco de memória inicial
Row Log Contents 0	Valor hexadecimal com o conteúdo do *log*
Modify Size	Valor que indica se um registo sofreu alterações

Tabela 4-2: Conjunto dos atributos considerados nos processos de captura e suas descrições.

Quando se sabe quais as linhas a filtrar, saber-se-á também qual a tabela em que ocorreu a operação e a informação contida na instrução que foi enviada ao sistema. A informação sobre o conteúdo da instrução, é guardada no campo *[Row Log Contents 0]* sobre a forma de um valor hexadecimal, estruturado de acordo com um registo interno do SQL Server 2008 (Delaney et al. n.d.). Importa salientar o procedimento necessário para efetuar a captura de dados a partir de registos atualizados. Para que se possa fazer essa captura de forma eficaz, é necessário adicionar informação suplementar ao conteúdo dos *logs*. Ora essa informação adicional, requer que seja ativado o mecanismo interno de CDC do *SQL Server* 2008, sobre a base de dados a partir da qual se pretende efetuar a captura. Quando ativado, o mecanismo vai criar dois processos, responsáveis por capturar e limpar os dados, no entanto, e para este caso, os processos serão criados para que a informação adicional seja registada nos *logs*, e em seguida removidos.

A partir deste momento, a informação de registos atualizados será guardada em dois registos consecutivos, sendo o primeiro, uma remoção com a informação anterior à atualização e o segundo uma inserção, já com a nova informação. Mas para que se possa compreender, quando essa sequencia se refere a uma atualização, é necessário que o campo *[Modify Size]* do registo de emoção seja diferente de zero.

Operação	**Contexto**	**Tamanho Modificado**	**Descrição**
LOP_BEGIN_XACT			Demarca o inicio de uma transação
LOP_INSERT_ROWS	*LCX_CLUSTERED*		Indica a existência uma inserção
	LCX_HEAP		
LOP_DELETE_ROWS	*LCX_MARK_AS_GHOST*		Indica a existência uma remoção
	LCX_CLUSTERED		
	LCX_HEAP		
LOP_DELETE_ROWS	*LCX_MARK_AS_GHOST*	>0	Indica a existência de uma atualização
	LCX_CLUSTERED		
	LCX_HEAP		
LOP_END_XACT			Demarca o final de uma transação

Tabela 4-3: Tabela com as combinações dos campos mais importantes para processos de captura.

Tendo em conta os campos apresentados na Tabela 4-3 podemos definir uma *query* que permita restringir o subconjunto dos dados que se quer obter como resultado. Essa *query* poderá ser:

```
SELECT SPID, [Begin Time], [End Time] [Current LSN],
       Previous LSN], Operation, Context, AllocUnitName,
       [Row Log Contents 0], [Modify Size]
       FROM fn_dblog(NULL,NULL);
```

que se for aplicada após a execução da seguinte transação:

```
BEGIN
      INSERT INTO Utilizador
            VALUES(1,'miguel','1988-11-14',NULL,1);
END
```

vamos obter como resultado os valores apresentados anteriormente na Figura 4-1. A partir deste resultado, e com a informação relativa aos elementos apresentados na Tabela 4-3, podemos ver, nessa figura, que as linhas 1 e 3 demarcam o inicio e final da transação. Na linha 1, vemos que o campo *AllocUnitName* indica qual é a tabela sobre a qual a operação vai ocorrer e, na linha 3, vemos o conteúdo do *[Row Log Contents 0]* que contém o resultado dessa operação. De acordo com a figuraFigura 3-1 podemos ver que no registo respetivo à operação de inserção, o campo *[Row Log Contents 0]* contém o valor:

```
0x30000F000100000026140B0100000005000801001C006D696775656C
```

Tendo em consideração a Tabela 3-1, podemos descodificar esse registo e saber qual valor que representa cada legenda, bem como verificar se o resultado dessa descodificação é igual ao valor inserido. Numa próxima secção será explicado todo o processo que leva a esta descodificação, uma vez que existem algumas questões que precisam de ser revistas.

4.3 Algoritmos para a Filtragem de Conteúdos

Após termos identificado devidamente a estrutura dos ficheiros de *log*, o passo seguinte foi fazer o estudo de estratégias e de algoritmos capazes de capturar a informação necessária. Da mesma forma que as estruturas de cada ficheiro *log* variam de fornecedor para fornecedor, também os algoritmos que interpretam esses mesmos ficheiros para fazer captura de também variam de estrutura em estrutura e, ainda, de versão em versão.

4.3.1 O *MySQL* e o *MariaDB*

A pesquisa pela estrutura dos ficheiros de *log* do *MySQL* permitiu-nos trabalhar diretamente sobre o ficheiro no qual os dados transacionais são escritos. Uma vez que esses ficheiros estão em formato binário, só após serem convertidos para formato ASCII é

que poderão ser tratados. Para se extrair informação deles é necessário ter um *parser*, com a capacidade de ler o ficheiro de *log* e encontrar as marcas ou os padrões de delimitação, referidos anteriormente, para que se possa extrair daí os conteúdos necessários. Além disso é preciso ter em conta algumas questões, em particular aquelas que envolvem a manipulação de caracteres especiais ou campos de tamanho variável.

Requisitos de Metadados

Em primeiro lugar, devido às características próprias do ficheiro de *log* foi necessário satisfazer alguns requisitos, para que um algoritmo pudesse funcionar corretamente – por exemplo, nas instruções nos *logs*, o nome dos campos da tabela estão na forma *'@'* número do campo na tabela. Para garantir que os valores de cada campo vão ser extraídos de forma a garantir que o tipo associado a esse atributo é o correto, é necessário guardar uma lista com os metadados de cada tabela, onde se destacam os campos de cada tabela, os seus tipos, e a posição atribuída a esse campo na ordenação interna da tabela, aquando da sua criação (para corresponder um campo ao seu identificador na instrução). Desta forma torna-se bem mais simples garantir-se que o tipo do atributo é o correto, evitando-se assim alguns erros na entrega dos dados. Além desta lista, é preciso também guardar uma segunda lista, do mesmo género, mas agora com os metadados das tabelas e dos campos em que se se pretende que seja capturada a informação modificada, para que seja guardada apenas a informação necessária nos *logs* correspondentes a esses dados.

Identificação de Valores e de Caracteres Especiais

Em segundo lugar, para se garantir que os dados a capturar - independentemente de serem aqueles que estão na lista de meta dados a capturar – respeitam o tipo de dados a si associados(e novamente vemos a importância de manter os meta dados completos e ordenados de acordo com a base de dados), foi necessário encontrar uma forma de evitar caracteres especiais que poderiam de alguma forma contrariar ou "enganar" o algoritmo de *parsing*, ou então delimitar os intervalos para garantir que são capturados nos intervalos certos. Felizmente o *MySQL*, guarda os caracteres especiais com o carácter '\' como

prefixo, e dessa forma podem usar-se conjuntos de caracteres como delimitadores. E ainda, devido ao facto de a associação de cada atributo ao seu conteúdo ser feito linha a linha, torna-se simples de identificar esses dados. Mas importa saber o tipo de dados que se estão a capturar, para ignorar caracteres especiais do *SGBD*.

Como forma de aumentar o desempenho deste algoritmo, foi considerada uma marca existente na estrutura do *log*, que identifica o bloco de memória no ficheiro onde os dados de uma transação foram guardados (Figura 4-1). Este bloco de memória – na figura indicado por P8 - veio a revelar-se bastante importante como forma de melhorar o tempo de execução e registos a capturar, uma vez que a ferramenta *Mysqlbinlog* permite adicionar uma marca com o valor do último bloco de memoria, e assim retornar apenas os dados no log a partir desse bloco. Então, vai sempre ser guardado o valor da última posição de memória e o nome do ficheiro log, onde foi adicionada informação de uma transação. Assim garante-se que o conteúdo do log que vai ser lido, é aquele que ainda não foi capturado, e não a totalidade do mesmo até ao valor onde os dados são novos.

O Algoritmo de Captura de Dados

Para se garantir que os dados que vão ser capturados serão sempre novos e os mais recentes, será necessário guardar o valor de várias variáveis: o nome do último ficheiro de *log* pesquisado, o último bloco de memória guardado para esse ficheiro, e a data da última captura de dados realizada. Assim, será possível verificar quais os ficheiros que foram alterados desde o último processo de captura de dados e proceder apenas à captura desses ficheiros. Nos casos em que o ficheiro é o mesmo, utiliza-se a posição de bloco de memória para limitar o volume de dados projetado. Com base nestes pressupostos, propôs-se um algoritmo, que, posteriormente, foi implementado para identificar as operações realizadas nas transações. O algoritmo desenvolvido pode ser visto como uma união de dois algoritmos distintos. Esses algoritmos estão apresentados de seguida, sobre a forma de pseudo-codigo (Figura 4-2 e Figura 4-3). Um deles (Figura 4-2) é responsável por

identificar as transações efetuadas com sucesso e, para esses casos, invocam uma função para capturar os dados das operações sobre as tabelas em que se pretendem capturar os dados (Figura 4-3).

```
   Input: Lista com os meta dados das tabelas da base de dados fonte
   Input: Lista com os meta dados das tabelas ou campos a capturar
   Input: Bloco de texto com a informao de captura do log
   Output: Dados capturados das transacoes bem sucedidas
 1 linha ← logConteudo.linha()
 2 while linha! = Vazio do
 3     if contem(linha, BEGIN)) then
 4         begin(Transaction);
 5         transaction ← true;
 6     if contem(linha, BEGIN) AND Transaction == true then
 7         ignore(Transaction);
 8         transaction ← true;
 9     if contem(linha, COMMIT) then
10         commit(listaTransacoes);
11         transaction ← false;
12     if contem(linha, INSERT INTO) then
13         begin(Operation);
14         decode(Operation, linha, logConteudo, INSERT, DBMetaData, CDCMetaData);
15         commit(listaTransacoes, Operation);
16     if contem(linha, DELETE FROM) then
17         begin(Operation);
18         decode(Operation, linha, logConteudo, DELETE, DBMetaData, CDCMetaData);
19         commit(listaTransacoes, Operation);
20     if contem(linha, UPDATE) then
21         begin(Operation);
22         decode(Operation, linha, logConteudo, UPDATE, DBMetaData, CDCMetaData);
23         commit(listaTransacoes, Operation);
24     linha ← logConteudo.linha()
25 return listaTransacoes;
```

Figura 4-2: O algoritmo para identificar as transações com sucesso e suas operações, em *MySQL*.

4.3.2 O *SQL Server* 2008

A processo de descoberta da estrutura dos ficheiros de *log* do *SQL Server* 2008 foi um pouco complexo. Ao se tentar encontrar as formas mais adequadas para capturar os dados no *SQL Server* 2008, concluiu-se que um algoritmo de *parsing* de texto como o do *MySQL*

não seria adequado, porque os ficheiros de *logs* do *SQLServer 2008* utilizam registos internos para armazenar a informação. Isto fez com que fosse necessário dividir o processo de captura de dados neste sistema em dois passos sequenciais, nomeadamente:

1. Identificar as transações e os registos resultantes de operações executadas em tabelas alvo do processo de captura de dados;
2. Descodificar as instruções do resultado obtido.

Assim, um algoritmo para efetuar a captura de dados no *SQL Server* 2008 deve seguir esta sequência de passos. Todavia, de forma a ser mais perceptível a cada momento as ações desenvolvidas, vamos seguir a ordem inversa e explicar com detalhe cada um dos passos envolvidos.

Descodificação das Operações

Consideremos a Tabela 3-1 na qual está apresentada a estrutura de um registo interno do sistema. Com base no seu conteúdo podemos retirar desde já algumas indicações acerca de como como é que deve funcionar um processo de descodificação. O primeiro passo, será converter o registo para um *array,* cujo tamanho será o número de valores hexadecimais do registo e, em seguida, atribuir a cada posição o seu respetivo valor. Esta conversão é essencial para que se possa fazer o cálculo posicional baseado na posição do *array*, uma vez que a descodificação vai utilizar posições de alguns intervalos para identificar os valores. A descodificação do registo é feita realizando duas pesquisas sobre o registo: uma primeira para identificar 4 intervalos e, uma segunda, para efetuar a captura a partir de intervalos derivados a partir dos primeiros.

```
   Input: linha - Linha do bloco onde se inicia a instrucao
   Input: DBMetaData - Meta dados da base de dados onde se captura
   Input: CDCMetaData - Meta dados das tabelas e campos a capturar
   Input: logConteudo - Bloco completo com informao do log
   Input: instrucao - Nome da instrucao a tratar
   Output: Conteudo dos campos capturados daquela instrucao
 1 Conteudo ← [];
 2 ConteudoDepois ← [];
 3 while !contem(linha,' # at') do
 4     flag ← (−1)
 5     if contem(linha,' # at') then
 6         commit(Conteudo); Inicia o log
 7         commit(ConteudoDepois); Inicia o log
 8     if equal(instrucao, INSERT)) OR equal(instrucao, DELETE))
       then
 9         begin(Conteudo); Inicia o log
10     if equal(instrucao, UPDATE)) then
11         begin(Conteudo); Inicia o log
12         begin(ConteudoDepois); Inicia o log
13     if equal(linha, SET)) then
14         flag ← (1);
15     if equal(linha, WHERE)) then
16         flag ← (2);
17     if contem(linha,' ###   @')) then
18         if flag == 1 then
19             adiciona(linha, ConteudoDepois); Verifica se faz parte da lista de captura;
20         if flag == 1 then
21             adiciona(linha, ConteudoAntes); Verifica se faz parte da lista de captura;
22     linha ← logConteudo.linha();
23 return (Conteudo, ConteudoDepois);
```

Figura 4-3: O algoritmo para captura dos dados das operações em *MySQL*.

Na primeira pesquisa serão identificados 4 tipos de intervalos, sendo um deles fundamental para os restantes intervalos e para toda a operação de descodificação. Esses intervalos são os seguintes:

Null Bitmap (2*bytes* ou mais)

Que quando convertido para formato binário, cada *bit* com o valor 1 indica que a coluna nessa posição possui valor *NULL*; se o valor for 0, essa coluna possui um valor regular.

Este intervalo é fundamental para interpretação de todo o registo, uma vez que guarda a informação acerca de quais os campos no registo que foram preenchidos com valor *NULL*. Um exemplo da importância deste intervalo no registo, pode ser o da Figura 4-4, no qual podemos ver um registo, onde um atributo de tamanho variável (Morada) foi deixado a *NULL*.No momento de escrever no *log*, o sistema indicou que o atributo naquela posição era vazio. Dessa forma é possível saber quais os campos que seguindo a ordenação interna da tabela, foram deixados a *NULL*.

Número de colunas de tamanho fixo (2 byte em formato *littleendian)*

Entende-se por tamanho fixo, tipo de dados como *INT*, *DATE*, *DATETIME*. O valor contido neste intervalo deve ser convertido para formato decimal, e subtraído 2 *bytes*, relativos aos primeiros dois itens do registo. Desta forma é possível determinar o numero de bytes do registo (a partir dos anteriores) correspondentes ao campos de tamanho fixo que foram guardados no registo. A ordem pela qual os campos são guardados no registo, é a mesma da ordenação interna dos campos da tabela(mesmo que os valores sejam alternados por só estarem os campos de tamanho fixo).

Número de colunas de tamanho variável (*2bytes* em formato *littleendian)*

O valor contido neste intervalo deve ser convertido para decimal. Esse valor vai corresponder ao numero de colunas de tamanho variável que foram registados no *log*. Essas colunas são "escritas" no registo, com a mesma ordenção das colunas de tamanho variável na tabela. Entende-se por tamanho variável, tipos de dados como *VARCHAR*.

Com a informação da importância destes dois intervalos, para se conhecer as localizações dos atributos que foram guardados no registo, e percebendo que todos eles, mesmo que em fragmentos diferentes, seguem a ordenação de ocorrência interna da tabela, não é dificl de

perceber a real importância do intervalo de *NullBitmap* para identificar quais os campos que estão vazios. Este campo não é mais que um mapa, dos campos e as suas posições dentro da ordenação da tabela, que estão vazios ou não. Se soubermos quais desses campos são de tamanho fixo ou variável, e as respetivas posições internas antes de efetuar a descodificação desses intervalos, facilmente podemos identificar quais os atributos que não foram registados no *log*, e assim evitar possíveis estrangulamentos na descodificação.

Intervalos com a posição final de cada valor de tamanho variável

Uma vez que os campos são de tamanho variável, é determinar dentro do registo, os fragmentos onde está guardada a informação de cada registo desse tipo que foi guardado. Cada intervalo deve ser convertido para um valor decimal, que corresponderá à posição do registo onde cada atributo de tamanho variável termina. Desta forma evitam-se problemas, caso de serem lidos mais ou menos *bytes*, e dessa forma levar a inconsistência na informação.

	Id	Nome	DataNascimento	Morada	País
Posição na tabela	1	2	3	4	5
Valor no campo	1	'miguel'	'14-11-1988'	NULL	1

Figura 4-4: Um exemplo de conversão e compreensão do valor de *NullBitmap*.

As posições de cada um dos intervalos deve ser identificada em primeiro lugar e antes de ser efetuada qualquer captura. Desta forma é possível saber a partir de que posições se pode começar a realizar a captura dos dados. Para isso, será necessário analisar o registo para encontrar esses valores, bem como as posições de cada um deles nesse registo, para

que seja possível, assim, efetuar uma segunda pesquisa a partir dessas posições, para que, por fim, se possa proceder à captura dos dados.

A partir do intervalo que determina o número de colunas fixas, vão existir tantos intervalos de 2 *bytes* quanto o número de colunas desse género que tenham sido preenchidas, indicando cada uma delas a posição final de um campo específico. A partir dessa sequência de intervalos, vão estar os respetivos valores, que terão serão acedidos através de um processo de cálculo posicional, tendo em conta os valores de cada posição. A partir daqui, e tendo em conta o valor da entrada do ficheiro *log* apresentada anteriormente na secção 4.2.2, é possível aceder à descodificação completa daquele registo (Hogg, A. 2010), identificar cada um dos intervalos, e compreender e descodificar cada um dos campos envolvidos. A Figura 4-5 mostra como todo este processo para extrair e descodificar a informação pode ser efetuado.

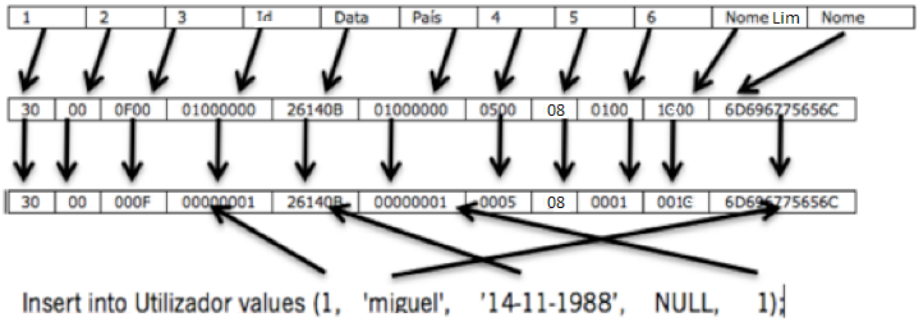


Figura 4-5: Descodificação e associação de cada valor do ficheiro de log

Após a realização deste processo ficámos em condições de criar um algoritmo capaz de descodificar cada uma destas entradas. Esse algoritmo seguirá uma estratégia de desenvolvimento em dois passos, nomeadamente:

- Identificação dos intervalos, das posições e do valor do *NullBitmap*.
- Utilização de cada um dos valores do primeiro passo, para determinar cada posição e descodificar cada valor envolvido.

De seguida apresenta-se esse algoritmo (Figura 4-6), representado em pseudo-código a forma como pode ser feita a descodificação de um registo.

Identificação e descodificação de Transações

Com o acesso às *querys* que são efetuadas sobre o bloco de memória no qual os dados transacionais estão a ser escritos, conseguimos ter acesso ao conteúdo necessário para proceder à captura de dados. Para se conseguir identificar as instruções necessárias, utilizámos os dados da Tabela 4-2. Desta forma torna-se simples encontrar os dados que terão que ser descodificados. Este algoritmo é já responsável por capturar a informação a partir de transações que foram efetuadas com sucesso (secção 3.3)

Um dos atributos que consideramos ser bastante relevante para efetuar a captura é o *[Current Lsn]*. Este campo é um identificador da posição de memória onde cada operação é armazenada. Está estruturado sob a forma de três valores hexadecimais separados por (:) - ver linha 2 da Figura 4-1. A função *fn_dblog()* permite passar dois parâmetros opcionais, da forma destes códigos, como forma de limitar o conteúdo disponibilizado. Caso se pretenda utilizá-los, é necessário fazer a conversão de cada um dos valores hexadecimais para decimal e, em seguida, aceder aos dados com a *query:*

```
SELECT *
      FROM fn_dblog(__,NULL)
```

e um valor do bloco de memória do ficheiro no qual foi guardado o registo da transação
Nessa query, coloca-se o valor do segundo parâmetro da função a *NULL,* porque queremos toda a informação contida no *log* até ao momento, sem a limitar de alguma forma.

```
   Input: Lista com os meta dados das tabelas da base de dados fonte
   Input: Lista com os meta dados das tabelas ou campos a capturar
   Input: Sequencia de valores hexadecimais A = {a_1, a_2, ..., a_n}, que a
          cada posicao corresponde um byte
   Input: Nome da tabela
   Output: Lista com dados descodificados e de acordo com a captura
 1 totalColunas ← 0; colunasFixas ← 0; colunasVariaveis ← 0;
   bitsNull ← []; posicao ← 0; listColunasFixas ← [];
   listColunasVariaveis ← []; listaIntervalos ← []; listaColunas ← [];
 2 colunasFixas ← endian(instrucao[4, 8]); posicao ← 8 + colunasFixas;
 3 totalColunas ← instrucao[posicao, (posicao + 2)]; posicao ← posicao + 2;
 4 bitmapLByes ← lengthToBytes(totalColunas);
   bitsNull ← hexToBitList(instrucao[posicao, (posicao + bitmapLByes)]);
   posicao ← posicao + lengthToBytes(totalColunas);
   colunasVariaveis ← endian(instrucao[posicao, (posicao + 2)]);
   posicao ← posicao + colunasVariaveis; i ← 0
 5 while i ≤ colunasVariaveis do
 6 |   listaIntervalos[i] = endian(instrucao[posicao, (posicao + 2)]);
 7 |   posicao ← posicao + 2;
 8 |   i ← i + 1;
 9 metaFixas ← [colunasFixasMetadados];
10 metaVariaveis ← [colunasVariaveisMetadados];
11 j ← 0;
12 posicao_ ← 8;
13 while j ≤ colunasFixas do
14 |   posicaoFixa ← IdDaColunaNosMetadadas;
15 |   if bitsNull[j] ≠ 1 then
16 |   |   tamanho = TamanhoColunaMetaFixas[j];
17 |   |   if eCapturavel then
18 |   |   |   listaColunas[posicaoFixa] =
   |   |   |   instrucao[posicao_, (posicao_ + tamanho)];
19 |   |   posicao_ ← posicao_ + tamanho;
20 |   else
21 |   |   listaColunas[posicaoFixa] = NULL;
22 |   j ← j + 1;
23 k ← 0
24 while k ≤ colunasVariaveis do
25 |   posicaoVariavel ← Id_Da_Coluna_Nos_metadadas;
26 |   if bitsNull[k] ≠ 1 then
27 |   |   tamanho = TamanhoColunaMetaVariaveis[k];
28 |   |   if eCapturavel then
29 |   |   |   listaColunas[posicaoVariavel] =
   |   |   |   instrucao[posicao, (posicao + tamanho)];
30 |   |   posicao ← posicao + tamanho;
31 |   else
32 |   |   listaColunas[posicaoVariavel] = NULL;
33 |   k ← k + 1;
34 return listaColunas;
```

Figura 4-6: O algoritmo de descodificação de um registo seguindo a estrutura interna do SQL Server 2008

Considerámos que a utilização deste campo seria importante para aumentar o desempenho do processo e reduzir o volume de dados a serem tratados a cada momento. Esta preocupação com o aumento do desempenho do processo diminuindo o volume de dados foi igualmente tida em conta no sistema *MySQL*, no qual se considerou o valor da posição do ultimo bloco de memória lido a partir do ficheiro *log* – indicado por P8 na Figura 4-1. Ao fazer a leitura do ficheiro *log*, a função *mysqlbinlog* permite indicar o valor do bloco de memória a partir do qual se pretende iniciar a leitura, minimizando desta forma o volume de dados a tratar em cada momento.

Assim, seguindo estas abordagens e utilizando estas características para aumentar o desempenho, propôs-se o algoritmo apresentado na Figura 4-7 para fazer a captura de dados no sistema Sql Server2008.

4.4 O Tamanho do Ficheiro *Log*

A escolha do tamanho dos ficheiros de *log* vai ter influência no desempenho do processo de captura de dados e no volume de dados que este vai tratar. Neste domínio temos que ver as questões de três maneiras distintas, ou seja, a dos 3 sistemas abordados. Isto porque os formatos e a forma de aceder aos *logs* são distintos em cada um deles. No caso dos sistemas MariaDB e MySQL, e como estes acedem diretamente à localização física dos ficheiros, pretendemos diminuir o tamanho limite de cada *log*, considerando para isso o tamanho de 4046 *bytes*. O sistema *OracleGoldenGate* suporta este tipo de abordagem. Desta forma, quando um ficheiro de *log* atingir esse tamanho, será criado um novo ficheiro que passará a guardar as novas transações. E assim sucessivamente. Naturalmente que existem algumas desvantagens neste tipo de abordagem. Uma delas será o aumento significativo do número de ficheiros guardados. Porém, em contrapartida, garante-se que o volume de informação a capturar em cada ficheiro é significativamente menor e, ao mesmo tempo, não sobrecarrega a função mysqlbinlog aquando da leitura de ficheiros muito extensos.

O sistema *SQL Server 2008* funciona de uma forma distinta, utilizando uma estratégia bem mais simples. É importante que antes de utilizarmos esta ferramenta, se configure o sistema para não truncar o log, de forma a obter o máximo de informação e a permitir, posteriormente, aceder à informação não disponível de imediato. Para isto deve desligar-se a opção *TRUNCACTION*. Após esta configuração, o resto será feito pelo próprio sistema. Quando a captura dos dados for feita em tempo real, o próprio sistema limitará o volume de dados disponibilizado a uma parcela do ficheiro *log* naquele momento.

4.5 Ficheiros de *Log* Armazenados Exteriormente

O armazenamento externo dos ficheiros de log poder ser uma possibilidade bastante viável, uma vez que em alguns casos, o seu armazenamento local pode ocupar espaço útil para outras tarefas. Mas para o bom funcionamento desta ferramenta, ao invés de armazenar os *logs* noutras localizações, estes devem manter-se na mesma máquina. No caso do MySQL ou do *MariaDB*, a localização destes ficheiros pode ser alterada, desde que seja indicada à ferramenta responsável pelo processo de captura. Uma vez que a estrutura do ficheiro não muda, não há problemas com mudança de diretorias, pois a utilização e leitura dos *logs* será a mesma.

```
   Input: Resultado da pesquisa sobre o fn_dblog
   Input: Lista de meta dados da bd
   Input: Lista de meta dados para captura
   Output: Lista dos conteudos capturados nas transacoes
 1 Assumindo que quando se trata uma linha,
   existe uma variavel para cada campo;
 2 linha ← lista.getLinha();
 3 Transaction ← [];
 4 Operation ← [];
 5 while linha ≠ NULL do
 6     if equal(operation, lop_begin_xact) then
 7         begin(Transaction);
 8         transaction ← true;
 9     if equal(operation, lop_begin_xact) AND Transaction == true then
10         ignore(Transaction);
11         transaction ← true;
12     if equal(operation, lop_end_xact) then
13         commit(Transaction);
14         transaction ← false;
15     if equal(operation, lop_insert_rows)
       && equal(context, lcx_clustered) then
16         begin(Operation);
17         decode(Operation, AllocUnitName, [RowLogContents0],
           Transaction, DBMetaData, CDCMetaData);
18         commit(Transaction, Operation);
19     if equal(operation, lop_insert_rows)
       AND equal(context, lcx_clustered) AND update(TRUE) then
20         Operation = AFTER_UPDATE;
21         begin(Operation);
22         update(false);
23         decode(Operation, AllocUnitName, [RowLogContents0],
           Transaction, DBMetaData, CDCMetaData);
24         commit(Transaction, AFTER_UPDATE);
25     if equal(operation, lop_delete_rows)
       AND equal(context, lcx_mark_as_ghost) AND diferent(modify_size, 0)
       then
26         Operation = BEFORE_UPDATE; begin(Operation);
27         update(true);
28         decode(Operation, AllocUnitName, [RowLogContents0],
           Transaction, DBMetaData, CDCMetaData);
29         commit(Transaction, Operation);
30     if equal(operation, lop_delete_rows)
       AND equal(context, lcx_mark_as_ghost) then
31         begin(Operation);
32         decode(Operation, AllocUnitName, [RowLogContents0],
           Transaction, DBMetaData2, CDCMetaData);
33         commit(Transaction, Operation);
34     linha ← lista.getLinha();
35 return Transaction;
```

Figura 4-7: O algoritmo para delemitar as transações efetuadas com sucesso em SQL Server 2008.

Porém, já quando falamos do sistema SQL *Server* 2008, existem algumas considerações a ter em conta, nomeadamente no que toca à especificação do caminho para o ficheiro com o log transacional, a fim deste ser utilizado quando se pretenda capturar dados não disponíveis no momento. Para isso é necessário utilizar a função *fn_dump_dblog()* – esta função não está documentada. Os resultados disponibilizados por esta função serão os mesmos que os da função *fn_dblog()*, apenas com a diferença de que o parâmetro a passar será o endereço para o ficheiro com os dados do ficheiro *log*. Este ficheiro pode ser uma cópia integral ou diferencial do ficheiro original.

4.6 A Arquitetura da Ferramenta

Uma característica muito importante da ferramenta é a sua "universalidade", ou seja, a capacidade de interagir com vários sistemas, sem acusar as diferenças entre cada um. É neste contexto que a arquitetura definida para a ferramenta se insere, pois é, digamos assim, o motor que sustenta tal universalidade. A arquitetura definida para a ferramenta partilha algumas das soluções que a arquitetura do *GoldenGate* apresenta, em especial aquelas que permitem efetuar a captura de dados e a sincronização desses dados com outros sistemas, desde que, obviamente, sejam suportados pela ferramenta. Todas estas estratégias seguem uma abordagem de envio de dados unidirecional. Atualmente, a ferramenta apresenta as seguintes alternativas de funcionamento (Figura 4-8):

- 'Um-Para-Muitos' - Captura de dados numa fonte e seu envio para vários sistemas destino.
- 'Muitos-Para-Um' - Captura de dados em várias fontes e seu envio para um único sistema.

- 'Um-Para-Um' - Captura de dados numa fonte e seu envio para um único sistema destino.

Nenhuma destas alternativas é melhor que a outra. A sua escolha será determinada pelas necessidades da empresa e dos seus próprios processos de captura de dados.

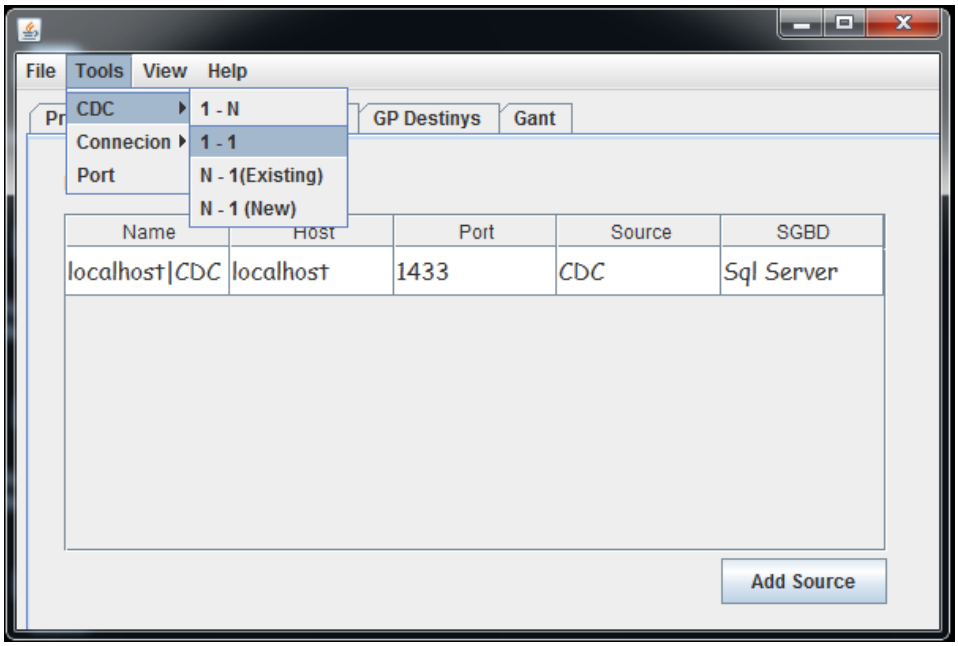


Figura 4-8: Interface principal da ferramenta de captura de dados.

Na Figura 4-8, podemos observar um exemplo de funcionamento a partir da interface principal da aplicação universal de captura de dados, desenvolvida como propósito desta dissertação. Embora a figura não seja explicita, é possível ver que foi adicionada uma fonte com o SGBD *SQL Server*, designada 'localhost|CDC', a partir da qual se poderá se poderá efetuar uma configuração para captura de dados. Ainda na figura, é possível verificar os passos para a escolha da configuração de captura de dados que se pretenda efetuar. A partir daí, será efetuada uma sequencia de passos para configurar o processo, passos esses como

a escolha da fonte e do destino, mapeamentos entre colunas de ambas as partes, escolha do tipo de captura ou o tipo de localização.

Um-Para-Um

Esta é de longe a solução mais fácil de configurar, uma vez que segue uma estratégia de uma fonte, um destino". Aqui os dados vão ser capturados a partir de um único sistema e enviados de seguida para um outro sistema, tudo em tempo real ou em *batch*, conforme as necessidades do negócio. Um exemplo prático desta solução, poder ser um caso de envio de dados de um sistema operacional para a área de estágio de um processo de ETL (Figura 4-9).

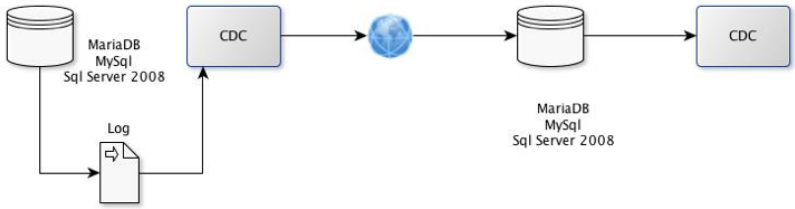


Figura 4-9: Um exemplo de um processo de 'um-para-um'.

Um-Para-Muitos

Esta solução é ligeiramente mais complexa do que a anterior, mas permite a propagação de informação para diferentes locais, com estruturas diferentes desde que estas sejam mapeáveis[1]. Esta solução é muito prática quando se quer que a informação capturada numa fonte seja enviada para destinos distintos, com propósitos distintos. Um exemplo de como integrar esta solução em processos de negócio de uma empresa (Figura 4-10), seria o caso de se ter que enviar, em paralelo, dados de um sistema operacional para uma área de estágio que suporte um processo *ETL* e, ao mesmo tempo, enviar os dados para um sistema OLAP que disponibiliza um conjunto de *dashboards* apresentando a evolução do negócio de uma filial, por exemplo.

[1] Referencia à próxima bem sobre mapeamento entre os campos na fonte e no destino.

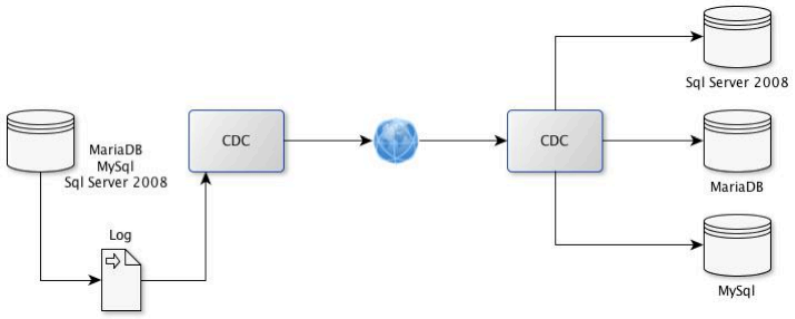


Figura 4-10: Um exemplo de um processo de 'um-para-muitos'.

Muitos-Para-Um

Esta é uma solução um pouco mais arrojada do que as anteriores, uma vez que possui um nível de complexidade superior aos anteriores, pois precisa de garantir que a informação é integrada sem falhas. Mas, ao mesmo tempo é muito natural em ambientes de empresas que acolham SDW. Nesta solução é possível imantegrar num único repositório a informação que foi capturada a partir de várias fontes, independentemente do sistema na qual esta vai ser inserida. Este tipo de solução é muito comum nas grandes empresas que utilizam este tipo de ferramentas – vejam-se os casos que acontecem quando numa empresa de retalho se tem que enviar os dados das vendas efetuadas em todas as suas filiais para um repositório central para posterior tratamento (Figura 4-11).

4.7 Mapeamento de Tabelas e de Colunas

Quando se configura uma das soluções que apresentámos, é bastante comum que o utilizador tenha a intenção de enviar os dados capturados para uma tabela já existente no sistema de destino e que pode ter uma estrutura distinta da tabela na qual os dados foram capturados. É por isso que é necessário que exista uma forma de fazer os mapeamentos requeridos entre os campos da tabela fonte e os campos da tabela destino. Esta é uma maneira de resolver este problema de uma forma simples e amigável.

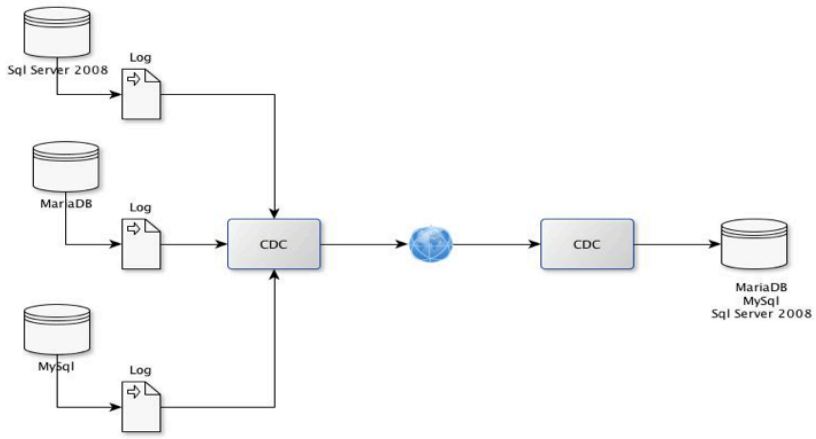


Figura 4-11: Um exemplo de um processo de ‘muitos-para-um’.

Mas, além do mapeamento entre colunas, é possível fazer a combinação dos valores de forma a facilitar alguns outros processos. Um caso desses acontece quando, por exemplo, numa tabela fonte o nome de um utilizador numa tabela “Utilizador”, está dividido em dois campos “PrimeiroNome” e “UltimoNome”. Para que se possa faze a junção dos dois campos, no momento de fazer o mapeamento tem-se que indicar que no campo destino esses dois campos serão combinados num único, “PrimeiroNome”+” ”+”UltimoNome”, e ao mesmo tempo enviar os valores desses campos para os respetivos no destino (Figura 4-12). Obviamente que, para que a combinação de campos seja possível deve existir compatibilidade entre os campos nas tabelas fonte e destino. Com esta funcionalidade, torna-se muito simples fazer a "ponte” entre os dados na fonte e no destino.

Na Figura 4-12, podemos ver a janela de configuração de um processo de captura de dados da nossa aplicação. Aqui podemos ver o momento em que é configurado o mapeamento dos atributos entre a tabela fonte e a tabela destino. Podemos ver que a tabela fonte, é designada por ‘dbo.Utilizador’ e que a tabela de destino é designada por ‘testecdc’, e por baixo de cada uma delas vemos o respetivo conjunto de atributos.

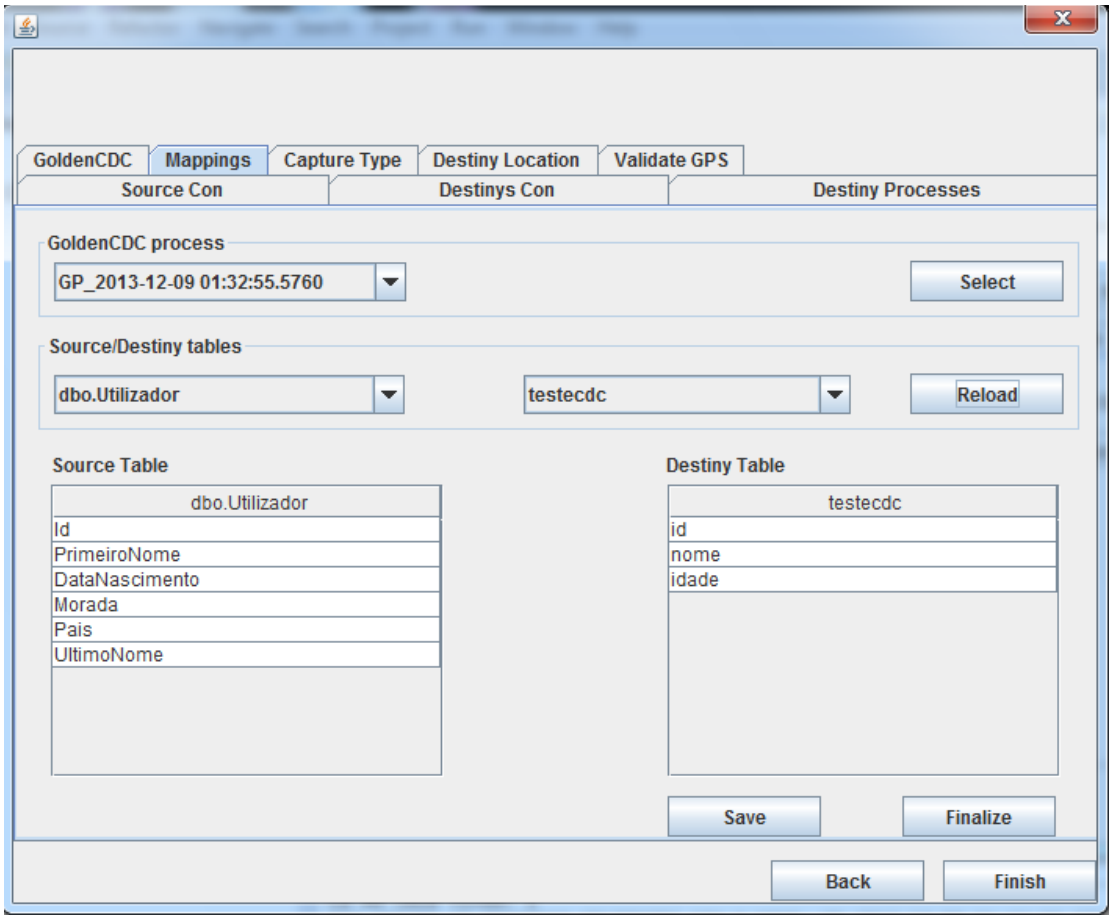


Figura 4-12: Interface com a configuração dos mapeamentos entre as colunas fonte e destino.

O exemplo dado acima para explicar o mapeamento de tabelas, pode ser visto nesta janela, em que se pretende guardar no campo nome, da tabela de destino, a concatenação dos campos 'PrimeiroNome' e 'UltimoNome' da fonte. Ainda nesta janela, é possível fazer a configuração dos mapeamentos entre todas as tabelas fonte e destino, bastando para isso, escolher da lista de tabelas, e atualizar o conteúdo das colunas. Antes de mudar as tabelas a mapear, deve 'Guardar-se' aquele estado, para que nao haja perda de informação sobre os mapeamentos.

5 Conclusões e Trabalho Futuro

5.1 Um Comentário Final

As necessidades dos analistas em conseguir coletar e armazenar informação proveniente de varias fontes num único repositório com grandes volumes de dados, levaram a uma adoção cada vez maior de sistemas analíticos, em particular os SDW. Estes sistemas permitem aos analistas, fazer consultas com um elevado grau de desempenho, orientados por tema, facilitando o processo de tomada de decisão. A ideia por trás disto tudo é a quererem tratar toda essa informação de uma forma rápida e orientada, de forma a conseguir criar relatórios detalhados com informação pertinente para os seus processos de negócio, como relatórios históricos sobre a evolução da empresa, resumos e análises de variação de índices de gestão, etc..

Usualmente para carregar informação proveniente de várias fontes para um *datawarehouse* é utilizado um processo de ETL. Este prcesso é responsável pela realização de todas as tarefas envolvidas com o povoamento de um *data warehouse*. Essas tarefas poderão englobar um conjunto muito diverso de operações, que podem ir desde a extração de dados num conjunto de fontes de informação, a sua transformação para outros formatos e valores para satisfazer as condições e restrições do negócio em causa e, por fim, carregar toda essa

informação para o *data warehouse*. A frequência com que este processo é efetuado varia com as necessidades de angariação de informação da empresa - é comum essa frequência ser semanal ou mesmo diária. Contudo, essas frequências por vezes torman-se desajustadas às necessidades de informação de alguns analistas, uma vez que a informação do dia anterior, por exemplo, pode tornar-se obsoleta no dia seguinte. Veja-se o caso de uma bolsa de valores, em que os dados podem variar drasticamente numa questão de minutos. Para casos como estes, foi preciso encontrar alternativas viáveis, como o aumento da frequência dos carregamentos dos dados no sistema. Porém, a solução de aumentar a frequência dos carregamentos acarreta algumas preocupações, nas quais se destacam a forma como os dados são carregados para o sistema destino por forma a que este possua a informação mais recente, ou a forma de extração dos dados nas fontes, de forma a extrair a informação logo após esta entrar no sistema, sem que para isso este sofra alguma diminuição no desempenho das suas tarefas diárias. O problema da captura de dados em tempo real passa por determinar quais os dados que devem ser carregados para a área de estágio de um processo ETL- o caso que considerámos. Em alguns cenários aplicacionais, a extração é feita efetuando um cálculo diferencial entre os dados que estão no sistema fonte e aqueles que estão no sistema destino. Todavia, e como sabemos, isso causa um impacto sério no processamento global do sistema, conduzindo, não raras vezes, à utilização de janelas de oportunidade noturnas, com os sistemas em offline, para realizar oprocesso de ETL.

A criação de um processo CDC específico foi a solução encontrada para contrariar tais situações. Baseado num principio de "ação-reação", este tipo de processo capturaos dados que se pretendem (novos o modificados) imediatamente após os ecventos que os produzirem ocorrerem no sistema fonte, enviando-os de seguida para um outro local. É um mecanismos bastante útil para soluções CDC que requeiram execução em tempo real. Contudo existem algumas preocupações que devem ser tomadas em consideração. De referir, o impacto que os processos de CDC possam ter no desempenho do sistema fonte ou a latência entre o momento em que os dados são capturados e entregues. Este problema

criou-nos a necessidade de realizar uma pesquisa detalhada sobre a melhor forma de efetuar a captura de dados e, ao mesmo tempo, resolver os problemas de latência e do impacto na captura dos dados. Em grande parte dos documentos sobre esta temática, a melhor solução apontada era aquela que se baseava em ficheirosde *log* transacionais. Comummente, os ficheiros de *log* guardam os dados transacionais que os sistemas de bases de dados utilizam para salvaguardar a informação dos seus sistemas em casos de desastre, por exemplo. Assim, esta abordagem consistiu em fazer uma leitura ativa desses *logs*, identificando as transações ocorridas com sucesso que satisfizessem as necessidades de captura e que enviasse a sua informação para um dado sistema de destino. Desta forma, consegue-se evitar o problema do impacto causado pelos mecanismos de CDC, uma vez que apenas se faz leitura dos ficheiros de *log*, não afetando de qualquer maneira o funcionamento do sistema. Ao mesmo tempo, permite-se também uma leitura e um envio rápido dos dados.

Desde o inicio deste trabalho que foi intenção criar um componente universal de captura de dados, ou seja, um componente que fosse capaz de se "encaixar" em qualquer SGBD, sem que as diferenças entre cada um afectassem o seu funcionamento ou mesmo o seu desempenho. Isso também justifica a opção por uma abordagem não intrusiva para efetuar essa captura de dados. Apesar da utilização de ficheiros *log* ser a mais natural para garantir uma captura não intrusiva, surgem alguns problemas sérios relacionados com o facto das estruturas desses ficheiros de *log* variarem conforme o fabricante e das dificuldades que daí advêm. Neste trabalho foram demonstradas várias das formas de aceder aos ficheiros de *log* de alguns SGBD concretos, bem como a forma como alguns deles estão estruturados.

5.2 A Abordagem Desenvolvida

Neste trabalho foi estudada a problemática relacionada com um processo de captura de dados, propondo e implementando um componente universal para captura de dados em

tempo real, baseado em *logs* transacionais. Para que a implementação deste componente "universal" de CDC fosse possível, começou-se por fazer a angariação da informação acerca do que é, sua arquitetura e como funciona um mecanismo de captura de dados. Após terminar esse processo, analisando as principais abordagens conhecidas para a sua implementação, identificou-se as vantagens e as desvantagens de que cada uma dessas abordagens, selecionando a abordagem baseada em *logs* transacionais como o nosso elemento básico de trabalho. Tal deveu-se, simplesmente, por este tipo de abordagem ser muito pouco intrusiva e por ter uma latência reduzida.

De seguida, identifcou-se alguns dos principais fornecedores de sistemas de base de dados ou de outras ferramentas que integrassem na sua estrutura componentes de CDC, como mecanismos internos ou como ferramentas auxiliares, no sentido de estudar com é que estes componentes estão organizados e são utilizados na prática. Adicionalmente, fez-se um levantamento exaustivo dos principais problemas e desafios que estas ferramentas enfrentam e que necessitam, de alguma forma, serem resolvidos.

Após estes trabalhos iniciais de estudo e de investigação aplicada, foi identificada a existência de alguns problemas considerados fundamentais, que podem tornar o tipo de abordagem adotada incompatível com os problemas que podem ser encontrados, de facto, na prática. Identificou-se também como sendo um claro obstáculo, o facto de os fabricantes estruturarem os ficheiros *log* dos seus sistemas de maneira muito própria, nem que seja somente para garantirem algumas vantagens concorrenciais.Por outro lado, a situação foi agravada quando se verificou que não há praticamente semelhanças entre os diferentes sistemas, e que a informação oficial sobre a forma como estão organizados internamente e como se pode aceder ao seu conteúdo praticamente não existe. Todavia, através da documentação do *MySQL* foi possível saber como guardar a informação em 3 formatos, aceder aos seus ficheiros físicos e ainda que existe uma ferramenta disponibilizada com o pacote *MySQL*, que permite converter o conteúdo dos *logs*, para um formato "legível" pelo ser humano. Mas se para o caso do *MySQL* foi possível encontrar

informação detalhada sobre o conteúdo dos seus ficheiros *log*, no caso do SQL*Server* 2008 isso não aconteceu. Através de Jeffries 2011 foi possível conhecer a existência de uma função não documentada– *fn_dblog()* –, interna ao próprio sistema, que devolve como resultado registos com 101 campos, contendo informação acerca dos dados das transações realizadas no sistema. Assim, foi possível saber como aceder aos *logs* e como identificar as instruções de inserção, remoção ou atualização de registos que aí são guardadas (Delaney et al. n.d.). Com essa informação tornou-se mais fácil compreender a forma de descodificar os registos de um ficheiro *log* e daí capturar a informação necessária. Porém, este trabalho não se ficou apenas pela identificação das estruturas dos ficheiros de *log*. Foram também propostos e implementados vários algoritmos capazes de interagir com os ficheiros *log* de cada sistema, que foram distribuídos em dois tipo: os que são capazes de identificar e delimitar as transações efetuadas com sucesso e os que são capazes de efetuar a descodificação e a extração dos dados que se pretendem capturar. Nesta altura, é possível afirmar que alguns dos desafios mais importantes levantados por um processo de captura de dados baseada em *logs* foram resolvidos.

Com o conhecimento das estruturas dos ficheiros de *log* do *MySQL*, do *MariaDB* e do *SQLServer* 2008 a implementação de um componente capaz de fazer a captura de dados a partir desses sistemas tornou-se bastante mais simples. Isso permitiu projetar e implementar a ferramenta de captura de dados "universal" que foi definido como o principal objetivo do trabalho apresentado nesta dissertação. Começou-se por abordar a sua construção com uma das estratégias de captura mais simples, uma fonte, um destino, para depois partir para a implementação das funcionalidades requeridas para componentes CDC que utilizassem mais do que uma fonte e mais do que um destino.

A ferramenta desenvolvida integra um interface bastante amigável para a configuração dos processos de CDC, o que torna muito intuitiva a escolha das tabelas e dos campos que se pretendem capturar. É frequente quando se captura dados de uma fonte, enviar os dados para localizações com estruturas já definidas. Para facilitar este processo, ao invés de criar

tabelas com estrutura igual às tabelas alvo do processo de captura desenvolveram-se uma série de mecanismos de mapeamento de campos, facilitando assim, em muito, a criação das várias correspondências dos atributos e dos valores necessárias para se garantir de forma correta a migração de dados (e algumas operações preliminares de transformação) entre os repositórios origem e os repositórios destino.

5.3 Algumas Linhas de Orientação para Trabalho Futuro

Este trabalho pode ser visto como um ponto de partida para a avaliação de processos de captura de dados, para compreender o funcionamento das várias abordagens realizadas no domínio deste tipo de processo. Pode ainda ser visto como um ponto de partida para uma implementação*open-source* de uma ferramentas de captura de dados baseada em *logs*. Várias são as melhorias e as possíveis evoluções que este trabalho poderá sofrer, nomeadamente, em termos da incorporação de métodos alternativos para efetuar a captura de dados a partir de ficheiros *Excel*, ficheiros de texto ou ficheiros *.csv*, com estruturas variáveis,. Além disso, poderão ser adicionados alguns módulos de captura para outros SGBD*,* como por exemplo o *Oracle*, o *DB2* ou o *Teradata*. Pela forma como a ferramenta foi implementada, este processo poderá ser simples, sendo apenas necessário respeitar a assinatura de alguns dos métodos de CDC implementados no caso da criação de novos módulos para um outro qualquer SGBD. Os algoritmos que foram criados para capturar os dados podem ser melhorados de forma a serem mais eficazes, particularmenteo algoritmo de captura de dados para o *MySQL* cujo *parser* poderá ser aprimorado com a utilização de ferramentas especificamente direcionadas ao processamento de linguagens.

Bibliografia

Ankorion, I., 2005. Change Data Capture–Efficient ETL for Real-Time BI. *Article published in DM Review Magazine*, (Cdc), pp.1–5. Disponivel em: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Change+Data+Capture+–+Efficient+ETL+for+Real++Time+BI#0 [Acedido a Setembro 25, 2013].

Khatiwada, S. 2012 . Architectural Issues in Real-time Business Intelligence. Disponivel em: http://brage.bibsys.no/uis/retrieve/4612/khatiwada,sanjeev.pdf [Acedido a Setembro 28, 2013a].

Hogg, A. 2010. How Do You Decode A Simple Entry in the Transaction Log? (Part 1) | SQL Fascination on WordPress.com. Disponivel em: http://sqlfascination.com/2010/02/03/how-do-you-decode-a-simple-entry-in-the-transaction-log-part-1/ [Acedido a Setembro 29, 2013b].

Anon, MySQL Replication - mysql-replication-excerpt-5.1-en.a4.pdf. Disponivel em: http://downloads.mysql.com/docs/mysql-replication-excerpt-5.1-en.a4.pdf [Acedido a Setembro 28, 2013c].

Barkaway, D., 2009. Change Data Capture and the Benefits to the Modern Enterprise Data Warehouse. , pp.1–12. Disponivel em: http://support.sas.com/resources/papers/sgf09/303-2009.pdf [Acedido a Setembro 24, 2013].

Bell, C., Kindahl, M. & Thalmann, L., 2010. *MySQL High Availability: Tools for Building Robust Data Centers*, O'Reilly Media. Disponivel em: http://www.amazon.com/MySQL-High-Availability-Building-Centers/dp/0596807309 [Acedido a Setembro 29, 2013].

Chen, L., Rahayu, W. & Taniar, D., 2010. Towards Near Real-Time Data Warehousing. *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pp.1150–1157. Disponivel em: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5474842 [Acedido a Setembro 4, 2013].

Connolly, T.M. & Begg, C.E., 2004. *Database Systems: A Practical Approach to Design, Implementation and Management (4th Edition)*, Addison Wesley. Disponivel em: http://www.amazon.com/Database-Systems-Practical-Implementation-Management/dp/0321210255 [Acedido a Setembro 28, 2013].

Delaney, K. et al., *Microsoft® SQL Server® 2008 Internals (Pro - Developer)*, Microsoft Press. Disponivel em: http://www.amazon.com/Microsoft®-SQL-Server®-2008-Internals/dp/0735626243 [Acedido a Setembro 28, 2013].

Eccles, M.J., Evans, D.J. & Beaumont, A.J., 2010. True Real-Time Change Data Capture with Web Service Database Encapsulation. *2010 6th World Congress on Services*, pp.128–131. Disponivel em: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5575585 [Acedido a Setembro 4, 2013].

Emergis, SQL Server Forensics. Disponivel em: https://www.blackhat.com/presentations/bh-usa-07/Fowler/Presentation/bh-usa-07-fowler.pdf [Acedido a Setembro 28, 2013].

Golfarelli, M. & Rizzi, S., 2009. *Data Warehouse Design: Modern Principles and Methodologies* 1st ed. McGraw-Hill Osborne Media, ed., Disponivel em: http://www.amazon.com/Data-Warehouse-Design-Principles-Methodologies/dp/0071610391.

Inmon, W.H., 2005. *Building the Data Warehouse* W. Publishing, ed., Wiley. Disponivel em: http://www.amazon.com/Building-Data-Warehouse-W-Inmon/dp/0764599445.

IT, S., 2012. DB-Engines Ranking - popularity ranking of database management systems. Disponivel em: http://db-engines.com/en/ranking [Acedido a Setembro 17, 2013].

JÄorg, T. & Dessloch, S., Near Real Time Data Warehousing Using State of the Art ETL Tools. Disponivel em: http://www.cs.toronto.edu/db/birte09/joergBIRTE09.pdf. [Acedido a Setembro 20, 2013]

JBSAC, Change Data Capture using Oracle Database. Disponivel em: http://www.jbsac.com/white/cdc_whitepaper.pdf [Acedido a Setembro 8, 2013].

Jeffries, J.P., 2011. *Oracle GoldenGate 11g Implementer's guide*, Packt Publishing. Disponivel em: http://www.amazon.com/Oracle-GoldenGate-11g-Implementers-guide/dp/1849682003 [Acedido a Setembro 28, 2013].

Jörg, T. & Deßloch, S., 2008. Towards generating ETL processes for incremental loading. *Proceedings of the 2008 international symposium on Database engineering & applications - IDEAS '08*, p.101. Disponivel em: http://portal.acm.org/citation.cfm?doid=1451940.1451956. [Acedido a Setembro 24, 2013]

Kimball, R. et al., 2008. *The Data Warehouse Lifecycle Toolkit Table of Contents*, Wiley. Disponivel em: http://www.amazon.com/Data-Warehouse-Lifecycle-Toolkit/dp/0470149779.

Kimball, R. & Caserta, J., 2004. *The Data Warehouse ETL Toolkit* Wiley, ed., Wiley. Disponivel em: http://www.theeuropeanlibrary.org/tel4/record/2000002542696?subject=Bases+de+donn�es+-+Conception [Acedido a Setembro 10, 2013].

Kindahl, M. & Thalmann, L., An API for Reading the MySQL Binary Log Mats Kindahl. Disponivel em: http://cdn.oreillystatic.com/en/assets/1/event/61/Binary log API_ A Library for Change Data Capture using MySQL Presentation.pdf. [Acedido a Setembro 25, 2013]

Lane, P. et al., 2009. Oracle ® Database. , 2(August). [Acedido a Setembro 25, 2013]

Lane, P. et al., 2011. Oracle ® Database. , 2(September). Disponivel em: http://docs.oracle.com/cd/E14072_01/server.112/e10810.pdf. [Acedido a Setembro 25, 2013]

Langseth, J., 2004. Real-Time Data Warehousing: Challenges and Solutions. Disponivel em: http://dssresources.com/papers/features/langseth/langseth02082004.html [Acedido a October 26, 2012].

Microsoft, Basics of Change Data Capture. Disponivel em: http://technet.microsoft.com/en-us/library/cc645937(v=sql.105).aspx [Acedido a Setembro 9, 2013].

Oracle, Oracle GoldenGate for MS SQL Server 2008. Disponivel em: http://www.oracle.com/webfolder/technetwork/tutorials/obe/fmw/goldengate/11g/GGS_Sect_Config_WIN_MSS_2008_to_WIN_MSS_2008.pdf [Acedido a Setembro 28, 2013].

Oracle, 2012. Oracle GoldenGate for MySQL Installation and Setup Guide - e27289.pdf. Disponivel em: http://docs.oracle.com/cd/E35209_01/doc.1121/e27289.pdf [Acedido a Setembro 23, 2013].

Oracle, A. & Paper, W., 2012a. Best Practices for Real-time Data Warehousing. , (Agosto). Disponivel em: http://www.oracle.com/us/products/middleware/data! integration/goldengate11g!realtime!wp!168153.pdf. [Acedido a Setembro 25, 2013]

Oracle, A. & Paper, W., 2012b. Oracle GoldenGate 11g : Real-Time Access to Real-Time Information. , (Augosto). Disponivel em: http://www.oracle.com/us/products/middleware/data-integration/goldengate11g-realtime-wp-168153.pdf. [Acedido a Setembro 25, 2013]

Corp, S. 2004. W., Data Integrator : Change Data Capture with Database Triggers. ,(Outubro). Disponivel em: http://www.sdn.sap.com/irj/boc/go/portal/prtroot/docs/library/uuid/10d27a38-7664-2b10-9696-b7d0c0cf20a9?QuickLink=index&overridelayout=true [Acedido a Setembro 25, 2013]

Sack, J., 2008. *SQL Server 2008 Transact-SQL Recipes*, Disponivel em: http://pacoge.net/Physics/Apress.SQL.Server.2008.Transact.SQL.Recipes.Jul.2008.pdf. [Acedido a Setembro 23, 2013]

Shi, J. et al., 2008. Study on Log-Based Change Data Capture and Handling Mechanism in Real-Time Data Warehouse. *2008 International Conference on Computer Science and Software Engineering*, pp.478–481. Disponivel em: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4722662 [Acedido a Setembro 4, 2013].

Skoutas, D. & Simitsis, A., 2006. Designing ETL processes using semantic web technologies. *Proceedings of the 9th ACM international workshop on Data warehousing and OLAP - DOLAP '06*, p.67. Disponivel em: http://portal.acm.org/citation.cfm?doid=1183512.1183526 [Acedido a February 6, 2013].

Tank, D.M. et al., 2010. Speeding ETL Processing in Data Warehouses Using High-Performance Joins for Changed Data Capture (CDC). *2010 International Conference on Advances in Recent Technologies in Communication and Computing*, (Cdc), pp.365–368. Disponivel em: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5656810 [Acedido a Setembro 22, 2013].

Thomas, J. & Dessloch, S., 2009. Formalizing ETL Jobs for Incremental Loading of Data Warehouses. *Notes*, pp.327–346. Available at: http://dblp.uni-trier.de/db/conf/btw/btw2009.html#JorgD09 [Acedido a Setembro 25, 2013].

Vassiliadis, P. & Simitsis, A., 2009. Near real time etl. *New Trends in Data Warehousing and Data ...*, 3. Disponivel em: http://www.springerlink.com/index/kq6pg05h84988110.pdf [Acedido a Setembro 25, 2013].