Universidade Federal de Pernambuco
Centro de Informática

Data Migration Wizard: Um Assistente de Migração de Dados para Bancos de Dados no SQL Server

TRABALHO DE GRADUAÇÃO

Recife, janeiro de 2008

Universidade Federal de Pernambuco
Centro de Informática

Data Migration Wizard: Um Assistente de Migração de Dados para Bancos de Dados no SQL Server

TRABALHO DE GRADUAÇÃO

Rodrigo Lumack do Monte Barretto

Monografia apresentada ao Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do Grau de Bacharel em Ciências da Computação.

Orientador: Prof. Ph.D. Fernando da Fonseca de Souza

Recife, janeiro de 2008

“Far and away the best prize that life has to offer is the chance to work hard at work worth doing.”

(Theodore Roosevelt)

A minha família e amigos

Agradecimentos

Primeiramente agradeço aos meus pais por terem me dado a oportunidade de ter uma educação de excelente qualidade, o que me garantiu uma vaga em um centro de excelência, e terem me mostrado o quão importante é realizar um trabalho que o faz se sentir bem. Sou eternamente grato a meus pais e irmão por seus ensinamentos e lições. São longas conversas que me ajudam a superar momentos difíceis e a planejar meu futuro. Juntamente com minha irmã, eles são o meu porto seguro.

Agradeço também aos meus amigos do Centro de Informática, que me acompanharam durante esses quatro anos estudando muito e desenvolvendo projetos interessantes (e trabalhando até tarde para concluí-los), contudo sem perder o humor e o espírito de equipe. Definitivamente fiz amizades que não cessarão após a formatura.

Por fim, contudo não menos importante, agradeço aos grandes professores que lecionaram importantes disciplinas durante o curso e que me forneceram uma base sólida para continuar acompanhando a evolução da tecnologia. Agradeço aos meus colegas de trabalho da SUATI e ao meu orientador pela idéia deste trabalho e pela ajuda que me deram.

Resumo

Um sistema é chamado de sistema legado quando resiste a evoluções significativas [4]. Normalmente, são sistemas que sofreram muitas modificações ao longo do tempo e possuem um nível de entropia tão alto, que o esforço e custo necessários para desenvolver novas funcionalidades não são compensados pelos benefícios trazidos por estas novas funcionalidades [18].

Quando refazer o sistema é uma opção inviável, uma possível solução é migrá-lo [15]. Os benefícios da migração são enormes a longo prazo: ela oferece mais flexibilidade, um melhor entendimento do sistema, fácil manutenção e custos reduzidos [19], porém a migração de dados é uma tarefa tipicamente complexa e trabalhosa nos ambientes computacionais atuais [9].

O principal objetivo deste trabalho é detalhar o desenvolvimento de um assistente de migração de dados para bancos de dados no *SQL Server*, o *Data Migration Wizard*.

Palavras-chave: migração de dados, sistemas legados, assistente de migração.

Abstract

A system is called legacy system when it resists to significative evolutions [4]. Usually, they are systems that have been modified many times throughout the years and which levels of entropy are so high, that the effort and cost needed to develop new features are not compensated by the benefits brought by those new features [18].

When redoing a system is an unfeasible option, one possible solution is to migrate it [15]. The long-term benefits of a migration are huge: it offers more flexibility, a better understanding of the system, easy maintenance and reduced costs [19], however a data migration is a complex task in today's computational environments [9].

The main purpose of the report is to detail the development of a data migration wizard for *SQL Server* databases, the *Data Migration Wizard*.

Keywords: data migration, legacy systems, migration wizard.

Sumário

Lista de Figuras

Lista de Tabelas

1. INTRODUÇÃO

Neste capítulo introdutório, serão mostrados o contexto no qual o presente trabalho se insere, a ferramenta considerada estado da arte em projetos de migração de dados, os principais pontos fracos do competidor do *Data Migration Wizard* e os objetivos do trabalho.

1.1. CONTEXTO

Um sistema é chamado de sistema legado quando resiste a evoluções significativas [4]. Normalmente, são sistemas que sofreram muitas modificações ao longo do tempo e possuem um nível de entropia tão alto, que o esforço e custo necessários para desenvolver novas funcionalidades não são compensados pelos benefícios trazidos por estas novas funcionalidades [18]. Esses mesmos sistemas de informação, apesar de extremamente importantes, constituem uma barreira para o progresso [21]. Segundo Bing Wu et al. e Lei Wu et al. [18, 22], os principais problemas que giram em torno dos sistemas legados são:

- Eles passaram por diversas revisões de código durante um longo período de tempo. Conseqüentemente, um alto nível de entropia associado a uma documentação imprecisa sobre o projeto e a arquitetura do sistema torna sua manutenção mais difícil, longa e cara. Isto impede a sua evolução a fim de prover novas funcionalidades e permanecerem competitivos no mercado; e
- Eles rodam em *hardware* obsoletos, de cara manutenção e que reduzem a produtividade devido à baixa velocidade.

Quando refazer o sistema é uma opção inviável, uma solução é migrá-lo [15]. Empresas tendem a querer migrar seus sistemas legados para novos ambientes, que permitem ao sistema de informação uma manutenção mais simples. Os sistemas são adaptados para satisfazer os novos requisitos de negócio, mas retêm as funcionalidades existentes, não sendo necessário desenvolvê-las novamente [18].

Os benefícios da migração são enormes a longo prazo: ela oferece mais flexibilidade, um melhor entendimento do sistema, fácil manutenção e custos reduzidos [19]. Porém, a migração de dados é uma tarefa tipicamente complexa e trabalhosa nos ambientes computacionais atuais, dada a miríade de servidores de aplicação, sistemas operacionais, sistemas de arquivos, equipamentos físicos e redes [9]. Vários são os desafios impostos pela migração, como o tempo no qual o sistema original ficará fora do ar, a necessidade de adicionar *software* de migração de dados em servidores, a possibilidade de perda e corrupção de dados, o surgimento de erros advindos da complexidade de ambientes heterogêneos, a possibilidade de atraso no cronograma ou estouro do orçamento definido para o projeto [9].

A solução ideal para a migração de sistemas legados é uma ferramenta que dê suporte a todas as etapas do ciclo de vida da migração de dados, desde a análise do conteúdo dos dados através de *profiling* e auditorias até a realização de transformações nos dados e mapeamento das bases. Ela deve ser flexível, escalável, requerer pouco conhecimento técnico e ser intuitiva [5].

Empresas gastam milhões de dólares migrando dados entre aplicações que lidam com informação em massa e, ainda com todo esse investimento, 75% dos novos sistemas falham em satisfazer as expectativas [5]. Muita pesquisa vem sendo conduzida na tentativa de desenvolver uma maneira segura e barata de guiar a migração de um sistema legado como um todo [2, 3, 4, 6, 7, 8, 12, 16, 17, 18, 19, 20, 21, 22] .

1.2. ANÁLISE DE COMPETIDORES

Ao realizar um projeto de migração de dados, é necessário escolher um *software* de migração. Segundo Softek [13], os atributos que devem ser levados em consideração durante a escolha desse *software* são:

- Performance - quão rápido os dados são copiados da fonte para o destino. Entretanto, é necessário balancear a performance com o consumo da largura de banda exigida ou overhead do sistema. Se o dado é copiado rapidamente, mas consume muita banda ou entrada e saída, isso pode provocar um sério impacto em aplicações ou sistemas de produção. Por outro lado, se os dados são

copiados vagarosamente, a migração deve levar mais tempo que o antecipado, prolongando o tempo que o sistema ficará fora do ar;

- Habilidade de desfazer a migração em caso de erro; e
- Tempo que o sistema de origem ficará fora do ar.

Umas das ferramentas que representa o estado da arte em termos de migração de dados é o *TDMF* da *Softek* [13]. Com relação à performance, esta ferramenta possui um sistema que permite que o usuário modifique a velocidade de transferência dos dados, de modo que torna possível balancear entre o consumo de largura de banda e o tempo de transferência. A ferramenta também possui um sistema de *rollback* que desfaz as operações realizadas durante a migração dos dados. Quanto ao tempo que o sistema fica fora do ar, o *TDMF* permite que a aplicação de origem pemaneça em funcionamento durante a transferência dos dados. Além destes atributos, a ferramenta é totalmente independente de *hardware* e permite a migração de dados entre diferentes tipos de mídia de armazenamento em disco (*e.g. DASD*, SATA ou *ATA*).

Um dos principais concorrentes do *Data Migration Wizard* é o *Data Importer/Exporter* do *SQL Server Management Studio* da *Microsoft*. O objetivo principal do *Data Migration Wizard* é superar os pontos fracos identificados no *Data Importer/Exporter.* São eles:

- A ferramenta não realiza uma análise profunda das base de dados, logo o usuário deve descobrir manualmente quais tabelas devem ser migradas em cada iteração e deve executar a ferramenta uma vez para cada iteração, tornando o processo de migração longo e cansativo;
- A ferramenta não dá suporte a um mecanismo de *rollback*, deixando a cargo do usuário desfazer as operações realizadas até então;
- A ferramenta não indica quais tabelas fazem parte de dependências cíclicas, portanto o usuário deve descobri-las por si só;
- A ferramenta não realiza qualquer tipo de mapeamento automático, o que torna o processo de migração ainda mais trabalhoso, pois o usuário deve ser

preocupar em mapear tanto o que mudou quanto o que não sofreu modificações entre as bases;

- A ferramenta não possui um motor de limpeza da base destino, deixando a cargo do usuário limpar a base antes da migração dos dados; e
- A ferramenta não dá suporte à inserção de *scripts* personalizados.

1.3. OBJETIVOS

Este trabalho tem como objetivos mostrar as metodologias de migração de sistemas legados, as possíveis estratégias a serem adotadas, os passos envolvidos, os desafios e os fatores de sucesso de um projeto de migração de dados, além de desenvolver um assistente de migração de dados para bancos de dados no *SQL Server* - o *Data Migration Wizard* - que supere os pontos fracos identificados no *Data Importer/Exporter* vistos na seção 1.2 (principal objetivo), mostrar detalhes da implementação do assistente e exibir os resultados obtidos em um estudo de caso realizado na empresa de desenvolvimento de *software* Suporte Avançado em Tecnologia da Informação – SUATI [16].

1.4. ESTRUTURA DO TRABALHO

Este trabalho é composto de cinco capítulos. No capítulo 2, serão abordados o processo de migração de dados, metodologias de migração de sistemas legados, estratégias de migração de dados, os passos envolvidos no processo, principais desafios e fatores que levam um projeto de migração de dados ao sucesso. No capítulo 3, serão vistos detalhes do desenvolvimento do assistente *Data Migration Wizard*, como seus requisitos e casos de uso, a modelagem dos dados, a arquitetura utilizada e como se dá o funcionamento do assistente. No capítulo 4 serão mostrados os resultados obtidos com a realização de um estudo de caso utilizando a ferramenta. No capítulo 5, estão as considerações finais deste trabalho e sugestões de trabalhos futuros.

2. MIGRAÇÃO DE DADOS

Freqüentemente, os dados que uma nova aplicação[1] necessita vêm de outras aplicações existentes [23]. Caso estes dados estejam disponíveis em sistemas existentes e seu volume seja muito grande, eles devem ser migrados das aplicações existentes para a nova aplicação, ao invés de recriados. O processo de transferir dados de um sistema para outro é chamado de migração de dados.

Projetos de migração de dados são complexos e o foco deve ser a migração da menor quantidade de dados possível para que a aplicação destino possa entrar em funcionamento [5]. Raramente, todos os dados de origem serão necessários, logo é importante filtrar os dados relevantes. Esta análise de relevância dos dados deve ser realizada em conjunto com os usuários que serão impactados diretamente pela migração.

A migração de dados tipicamente envolve o planejamento do projeto, a definição do escopo, a extração, a transformação (para o formato adequado), a transferência e a verificação dos dados [10]. É geralmente implementada em duas etapas: a extração e a carga dos dados.

A extração de dados é o processo de extrair dados de um sistema existente e armazená-lo em um arquivo. Se o volume dos dados for relativamente pequeno, estes podem ser extraídos para um arquivo que será referenciado diretamente pela aplicação destino. No entanto, se o volume dos dados for grande e os sistemas utilizam diferentes ambientes computacionais, estes devem ser armazenados em alguma mídia, e então, carregados na aplicação destino [23].

A carga de dados é o processo de transferir o conteúdo do arquivo gerado na etapa de extração para a base de dados da aplicação destino. Se os domínios dos dois sistemas possuem uma interpretação comum entre valores e os relacionamentos entre os dados são bem definidos, então o processo de mapeamento é relativamente direto. Caso os domínios sejam diferentes ou os relacionamentos não estejam bem definidos, é necessário

[1] Neste caso, nova aplicação entende-se como todo sistema computacional em sua versão inicial ou resultante da atualização de um sistema pré-existente.

passar por um processo de transformação dos dados, que pode ocorrer na etapa de extração ou de carga [23].

2.1. METODOLOGIAS DE MIGRAÇÃO DE SISTEMAS LEGADOS

O sucesso de um projeto de migração de sistemas legados depende da implementação de uma metodologia que seja detalhada e bem definida, já que este tipo de projeto é de enorme escala, complexidade e muito suscetível a falhas [18]. Atualmente, duas metodologias são mais aceitas pela comunidade: *Chicken Little* e *Butterfly*.

2.1.1. *Chicken Little*

Na metodologia *Chicken Little*, Brodie e Stonebraker [3, 4] propõem uma estratégia de migração composta de onze etapas genéricas que utilizam complexos *gateways.* Um *Gateway* pode ser definido como qualquer módulo de *software* introduzido entre componentes de *software* com o objetivo de mediá-los [4]. Esta metodologia deixa a cargo da equipe escolher qual método de migração utilizar: o método *Forward* ou *Database First* ou o método *Reverse* ou *Database Last*.

O método de migração *Forward* ou *Database First* envolve a migração inicial dos dados legados para uma nova base e, em seguida, a aplicação e as interfaces são migradas incrementalmente [4]. Um f*orward gateway* é utilizado para que a aplicação original acesse os dados na plataforma destino.

O método de migração *Reverse* ou *Database Last* faz com que a aplicação seja migrada gradualmente para a plataforma destino, enquanto que a base de dados permanece na plataforma original [4]. A migração dos dados é a última etapa deste processo. Um *reverse gateway* é utilizado para que a aplicação destino acesse os dados da plataforma original.

Nesses dois métodos, os sistemas de origem e destino operam paralelamente durante o processo da migração, o que adiciona complexidade [18]. Os próprios autores

reconhecem que manter a consistência dos dados entre sistemas de informação heterogêneos representa um problema técnico complexo e que ainda não foi proposta uma solução geral [4]. Os onze passos que compõem a metodologia *Chicken Little* são:

1- Analisar o sistema legado;

2- Decompor a estrutura do sistema legado;

3- Projetar a interface destino;

4- Projetar a aplicação destino;

5- Projetar a base de dados destino;

6- Instalar o ambiente destino;

7- Criar e instalar os gateways necessários;

8- Migrar as bases legadas;

9- Migrar as aplicações legadas;

10- Migrar as interfaces legadas; e

11- Mudar para o sistema destino.

2.1.2. *Butterfly*

Ao contrário da metodologia *Chicken Little*, a metodologia *Butterfly* questiona a necessidade de interoperabilidade entre as aplicações [19]. Ela leva em consideração o fato de que o sistema legado não estará em produção enquanto o processo de migração ocorre [19]. A metodologia propõe o desenvolvimento de um motor de migração de dados, a fim de que o sistema legado fique fora do ar o mínimo possível. Ela difere dos métodos *Forward e Reverse Migration* na medida em que o sistema legado deve ficar fora do ar por um tempo considerável para facilitar a migração dos dados antes do sistema destino entrar em funcionamento. Este é uma abordagem mais simples, porém mais arriscada, pois todo o fluxo de informação da aplicação passará a ser executado em um sistema não confiável [19]. Segundo Bing Wu et al. [20], os passos que compõem a metodologia são:

1- Planejar a migração;

2- Entender a semântica do sistema legado e desenvolver o esquema de dados destino;

3- Migrar todos os componentes, exceto os dados, do sistema legado para a arquitetura destino;

4- Gradualmente migrar os dados legados para o sistema destino e treinar os usuários a usar a aplicação destino; e

5- Passar a utilizar a aplicação destino.

2.2. ESTRATÉGIAS DE MIGRAÇÃO DE DADOS

Para realizar um projeto de migração de dados é necessário que uma estratégia de migração seja adotada no início do projeto, já que seu planejamento e a implementação de uma ferramenta de migração dependem da estratégia adotada. Há 2 tipos principais de estratégias de migração: *Big Bang* e *Trickle*.

2.2.1. *Big Bang*

As migrações que adotam a estratégia *Big Bang* são realizas de uma só vez. Neste caso, o sistema original deve ficar fora do ar enquanto os dados são extraídos, transformados e carregados na aplicação destino [5]. Segundo Datanomic Ltd. [5], esta estratégia tem a vantagem da migração ser finalizada no menor tempo possível. Porém, ela apresenta riscos: algumas organizações não podem ter o seu sistema fora do ar por muito tempo, o que aumenta a grande carga de pressão sobre a execução da migração e a realização de seus testes. Datanomic Ltd. [5] ainda afirma que as empresas que adotam esta estratégia deveriam realizar uma migração de testes antes da migração real, mas poucas o fazem, o que compromete a qualidade dos dados. Normalmente quando esta estratégia é adotada, o processo de execução da migração ocorre durante um fim de semana ou feriado.

2.2.2. *Trickle*

As migrações que adotam a estratégia *Trickle* são realizadas de forma incremental. Os dois sistemas executam em paralelo e os dados são migrados em fases. Segundo Datanomic Ltd. [5], isto pode ser implementado utilizando processos em tempo real para transferir os dados da base de origem para a base destino e para manter os dados da base destino atualizados.

Adotar esta estratégia adiciona complexidade ao projeto e há duas possíveis maneiras de desenvolver o projeto: na primeira, os dois sistemas executam paralelamente e os usuários devem chavear entre as aplicações dependendo de onde está a informação que ele precisa no momento. Na segunda, os usuários utilizam o sistema antigo até que a migração termine, porém quaisquer mudanças nos dados da aplicação fonte exigem que os novos registros sejam migrados, de modo que a base destino permaneça atualizada.

2.3. PASSOS

Migrações de dados são realizadas freqüentemente, porém nem todas obtêm êxito. Através de migrações de sucesso, boas práticas foram compiladas, de modo que é possível enumerar uma seqüência de passos que ajudam a aumentar a probabilidade de sucesso do projeto [5].

2.3.1. Passo 1 – Planejamento

O processo de planejamento envolve descrever em detalhes o escopo do projeto, determinar os requisitos da migração, identificar os ambientes atual e futuro, criar e documentar o plano de migração e definir um cronograma a ser seguido. Os requisitos de projeto incluem a arquitetura de migração, os requisitos de *hardware* e *software*, o procedimento de migração e os planos de teste e implantação [5, 10, 13]. O plano de migração deve determinar se a migração será realizada em fases, incluindo quantas fases

serão necessárias e que tipos de dados serão migrados em casa fase. Deve também determinar o responsável por cada tarefa e seus prazos, além de descrever o impacto da migração no negócio em termos de necessidade de treinamento de pessoal, custos envolvidos e o tempo de parada do sistema e estabelecer um plano de contingência, especificando como lidar com processos de negócio críticos em termos de tempo, como pagamentos, em caso de atraso no cronograma [10]. Quanto maior a importância dos dados para as operações da empresa e maior a complexidade do ambiente, mais crítico é o planejamento da migração [13].

O cronograma e o orçamento devem levar em consideração todo o material e tempo necessários para auditar os dados, desenvolver as especificações de mapeamento, escrever o código de migração, construir as regras de transformações e limpeza dos dados, carregar e testar a migração. O cronograma também deve incluir quando o dado estará pronto para análise e teste, quando o sistema de origem ficará fora do ar (dependendo da metodologia adotada) e quando o novo sistema estará pronto para os usuários [10]. Um projeto de migração típico tem uma duração média de seis meses a dois anos com uma equipe de cinco desenvolvedores em tempo integral [5].

2.3.2. Passo 2 - *Profiling* e Auditorias

Quando dados são migrados para um novo sistema, pode ficar aparente que eles contêm redundâncias e imprecisões. Enquanto que estes dados são perfeitamente adequados para o funcionamento do sistema original, eles podem não ser adequados (em termos de estrutura e conteúdo) para a nova aplicação [5]. Sem o entendimento suficiente de ambos os sistemas, a transferência de dados de um para o outro pode ampliar o impacto negativo de qualquer dado incorreto ou irrelevante [5].

Para desenvolver um programa de migração de dados eficaz é importante entender por completo as fontes de dados antes de especificar o código de migração. Isto é melhor alcançado através da realização de uma auditoria completa dos dados que fazem parte do escopo da migração nos estágios iniciais do projeto. Através do uso de ferramentas de *profiling* e auditoria, é possível analisar o conteúdo dos dados e identificar quais valem a pena serem migrados, já que uma migração requer tempo, dinheiro e esforço [5, 10]. O

principal resultado alcançado com a análise dos dados é o refinamento do escopo e a definição do que será migrado automaticamente, manualmente e o que não será migrado [10]. Segundo Datanomic Ltd. [5], os benefícios trazidos por esta prática são:

- Com uma visibilidade completa de todos os dados de origem, a equipe pode identificar potenciais problemas que permaneceriam escondidos até um estágio mais avançado do projeto;
- As regras para planejamento, mapeamento, execução e teste da migração são baseadas na análise de todos os dados de origem ao invés de uma pequena amostra;
- As decisões podem ser realizadas baseadas em fatos ao invés de suposições; e
- Uma auditoria completa dos dados pode reduzir o custo de reparos no código na etapa de testes em até 80%.

2.3.3. Passo 3 - Construção e *Design*

Nesta etapa são desenvolvidas as especificações de mapeamento. Projetos de migração são mais eficientes quando segmentados, pois os dados podem ser auditados, mapeados, testados e transferidos em fases, facilitando o seguimento do cronograma, orçamento e possibilitando a realização de revisões de progresso [5]. Segundo Datanomic Ltd [5], uma vez que as especificações de mapeamento forem convertidas em código de migração, elas devem ser verificadas individualmente, a fim de identificar possíveis erros.

2.3.4. Passo 4 – Execução

Neste passo, os dados são extraídos da fonte, transformados e carregados na aplicação destino utilizando regras de migração carregadas em uma ferramenta de migração escolhida durante a etapa de planejamento [5, 13].

2.3.5. Passo 5 - Testes e Validação dos Dados

Apesar dos dados terem sido validados ao longo do processo de migração, testes unitários, de sistema e de carga devem ser feitos para garantir que todos os dados foram migrados e que o novo sistema se comporta como esperado [10]. Caso os passos 2 e 3 não tenham sido realizados de maneira eficaz, a chance da ocorrência de erros aumenta, acarretando em repetição de trabalho, o que leva a um aumento no custo do projeto e um possível atraso no cronograma [5].

2.4. DESAFIOS

De acordo com IBM, Manek e Youn [9, 10, 23], os principais desafios impostos por um projeto de migração de dados são:

- A necessidade de minimização do tempo em que o sistema ficará fora do ar, pois normalmente a organização congela a entrada de dados enquanto realiza a migração dos dados;
- A necessidade de aquisição e instalação de *software* de migração de dados em servidores;
- A ocorrência de inconsistência de valores. Isto acontece quando múltiplos sistemas de origem possuem o mesmo conceito (entidade), mas as representações (valores) variam (*e.g.* em um sistema o campo que armazena números de telefone utiliza o formato *XXXXXXXX*, já em outro sistema, o mesmo campo utiliza o formato *(XX) XXXX-XXXX*);
- A necessidade de preservação da integridade referencial entre as entidades e relacionamentos da aplicação destino durante o processo de carga;
- A sincronização dos dados. Apesar destes serem migrados do sistema original para o sistema destino, o sistema original pode continuar em operação. Para preservar a integridade dos dados, quando estes forem adicionados e/ou

modificados no sistema original, também devem ser adicionados e/ou alterados no sistema destino;

- Necessidade de escrita das especificações e códigos de mapeamento orientada a conteúdo, ao invés de orientada a metadados. Para o propósito de migração de dados, a informação que descreve a origem (*e.g.* o nome da base de dados, o nome da tabela, o nome da coluna) e as características de cada coluna (*e.g.* o tipo, o tamanho, a escala, a precisão) são considerados metadados. Conteúdo é o que está contido em cada registro da coluna. Uma migração de dados orientada a metadados assume que o conteúdo reflete a sua descrição, o que nem sempre acontece. Somente a realização de *profiling* e auditorias podem confirmar de fato o conteúdo do registro;
- Possibilidade de perda e corrupção de dados durante o processo;
- Alocação insuficiente de tempo para a realização de testes; e
- Cronograma imprevisível. Mesmo com um planejamento meticuloso, situações inesperadas podem surgir durante a migração, ocasionando problemas que impactam no cronograma.

2.5. FATORES DE SUCESSO

Apesar dos desafios presentes em um projeto de migração de dados, existem algumas boas práticas que diminuem a probabilidade de falha do projeto. Segundo Datanomic Ltd. e Manek [5, 10], estas boas práticas são as que se seguem:

- A migração de dados deve ser vista como um projeto completo. Portanto, se faz necessário alocar recursos, definir claramente o escopo do projeto, definir um plano de projeto com um cronograma que leva em consideração a possibilidade de ocorrência de problemas inesperados e angariar fundos necessários para a execução do projeto;
- Levar em consideração o tempo necessário para testar o resultado da migração e resolver eventuais problemas ao definir o cronograma do projeto;

- Utilizar ferramentas de *profiling* e auditoria para analisar completamente os dados, refinar o escopo do projeto e escrever as especificações de mapeamento com mais segurança. Entender que informação é importante e como deve ser usada é crítico para o sucesso da empreitada;
- Minimizar a quantidade de dados a serem migrados;
- Testar o resultado da migração o quanto antes; e
- Enquanto que muitas tarefas podem ser automatizadas, outras devem ser realizadas manualmente. Realizar estas tarefas com atenção ajuda a manter a consistência dos dados.

3. DATA MIGRATION WIZARD

Neste capítulo será detalhado o processo de desenvolvimento de um assistente de migração de dados para Sistemas de Gerenciamento de Banco de Dados *Microsoft SQL Server*, o *Data Migration Wizard*. Na primeira seção, será vista a definição de alguns termos e conceitos que serão utilizados no capítulo. Na segunda seção, serão mostradas as características gerais da ferramenta. Na terceira seção, serão abordadas suas especificações técnicas, tais quais seus requisitos, casos de uso, modelagem de dados e arquitetura. Na seção seguinte, o uso da ferramenta poderá ser observado.

3.1. CONCEITOS

Esta seção tem a finalidade de explicar alguns conceitos e termos que serão utilizados no decorrer do trabalho e que seu entendimento é de fundamental importância.

Neste trabalho são utilizadas as seguintes abreviaturas:

- *UCXX* para referenciar um caso de uso cujo número é representado pelo *XX*;
- *RFXX* para referenciar um requisito funcional cujo número é representando pelo *XX*; e
- *RNFXX* para referenciar um requisito não-funcional cujo número é representado pelo *XX*.

Quanto às prioridades dos casos de uso, caso ela seja **essencial**, o caso de uso deve ser implementado, já que a sua falta impede o funcionamento do sistema. Já um caso de uso de prioridade **importante** não necessariamente precisa ser implementado, pois sua falta não impede o funcionamento do sistema, porém o sistema não funcionará de maneira satisfatória. Quanto a um caso de uso de prioridade **desejável**, caso este não seja implementado, a operacionalização da aplicação não será comprometida.

Um conceito que deve ser entendido é a **dependência de tabelas**. Quando uma tabela B **depende** de uma tabela A significa dizer que a tabela B possui uma chave estrangeira, que é a chave primária (ou faz parte da chave primária, caso ela seja

composta) da tabela A. Este conceito é levado em consideração durante a definição de iterações, pois se uma tabela B depende de uma tabela A, os dados devem ser inseridos na tabela A **antes** que os dados sejam inseridos na tabela B. Portanto, a tabela A pertence a uma iteração que será executada antes da iteração a qual a tabela B pertence. Caso contrário, a restrição de integridade referencial será violada. Analogamente, os registros da tabela B devem ser apagados **antes** da limpeza dos registros da tabela A. Na Figura 3.1 a seguir pode-se observar que a tabela Empresas depende da tabela TipoEmpresa:

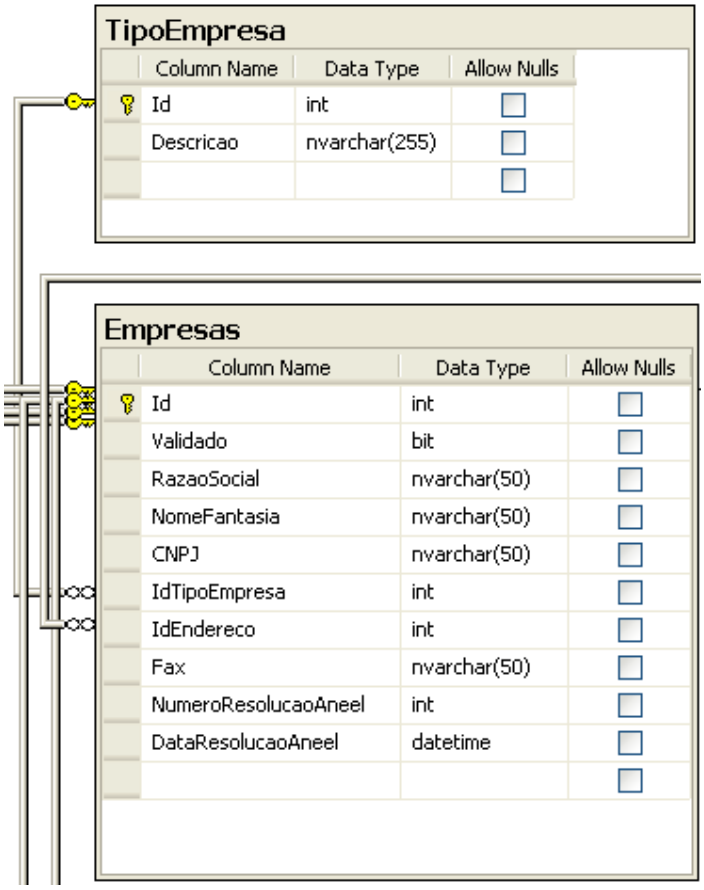

Figura 3.1: Dependência entre tabelas.

Um outro conceito relevante é o de **nível** de uma tabela. O nível de uma tabela representa o quão distante da raiz (nível 0) a tabela se encontra. Dado que uma base de dados pode ser modelada como uma floresta (um conjunto de árvores onde cada nó do grafo é uma tabela), as raízes (tabelas de nível 0) são as tabelas que não dependem de

outras tabelas. Já as tabelas de nível 1 são as tabelas que dependem de tabelas cujo nível máximo é 0, e assim por diante.

O conceito de **iteração** foi criado com o intuito de agrupar os mapeamentos e *scripts* que possuem o mesmo nível de modo a tornar possível a migração dos dados de uma única vez, satisfazendo o *RF01*, já que executar as iterações em cadeia não violará a restrição de integridade referencial da base.

O conceito de **análise** de uma base de dados é utilizado para indicar a execução de uma série de operações que têm por finalidade criar um objeto do tipo *Database* (detalhes do seu conteúdo serão vistos na subseção 3.3.3) na memória.

3.2. CARACTERÍSTICAS GERAIS

O *Data Migration Wizard* foi concebido de modo a superar os pontos fracos identificados na ferramenta *Data Importer/Exporter* da *Microsoft* (vistos na seção 1.2). A seguir são listadas suas características gerais:

- Foi desenvolvido em C# sobre a plataforma *.NET* 2.0;
- Foi criado para a realização de migrações de dados entre bases de dados *SQL* Server de aplicações que sofrem atualizações em suas versões;
- Possui o formato de um assistente para tornar o processo de migração intuitivo;
- A ferramenta indica quais tabelas fazem parte de dependências cíclicas (seu tratamento automático não faz parte do escopo do trabalho);
- A ferramenta divide o processo de migração em iterações (identificadas automaticamente), de modo que todos os dados podem ser migrados de uma única vez, já que não violam a restrição de integridade referencial da base;
- A ferramenta provê o suporte necessário para que uma tabela da base de origem forneça dados para mais de uma tabela da base destino, porém o inverso não faz parte do escopo do trabalho, já que, na prática, esta é uma situação que não ocorre com freqüência;

- Caso ocorra algum problema durante a execução da migração, o mecanismo de *rollback* é acionado;
- O assistente realiza mapeamentos automáticos entre tabelas e colunas, poupando o usuário de um esforço desnecessário e agilizando o processo de migração;
- Possui uma arquitetura expansível, o que permite o desenvolvimento de componentes que realizem migrações entre diversos tipos de bancos de dados; e
- A ferramenta é flexível o suficiente para permitir que o usuário crie e execute seus próprios *scripts SQL.*

3.3. ESPECIFICAÇÕES TÉCNICAS

3.3.1. Requisitos

Durante o processo de elicitação de requisitos foi utilizada a técnica da entrevista. Esta foi realizada com funcionários da empresa Suporte Avançado em Tecnologia da Informação – SUATI pelo fato deles estarem acostumados a utilizar a ferramenta de migração de dados *Data Importer/Exporter* pertencente ao *Microsoft SQL Server Management Studio*. Portanto, eles têm conhecimento acerca das possíveis melhorias que o software necessita.

Como resultado da entrevista, os seguintes requisitos funcionais (RF) foram elicitados:

- [RF01] - O sistema deverá ser capaz de realizar a migração de dados entre bases de dados de aplicações que sofrem atualizações em suas versões. Esta migração (excluindo as tabelas que fazem parte de dependências cíclicas) deve ser realizada de uma vez;
- [RF02] - O sistema deverá ser capaz de identificar as tabelas que fazem parte de dependências cíclicas;

- [RF03] - O sistema deverá indicar ao usuário todas as operações que são realizadas no momento em que estiverem sendo efetuadas;
- [RF04] - O sistema deverá ser capaz de permitir a inserção de scripts SQL e que a escolha do momento de sua execução fique a cargo do usuário;
- [RF05] - O sistema deverá realizar mapeamentos automáticos sempre que possível; e
- [RF06] - O sistema deverá limpar a base de dados destino automaticamente antes da realização da migração dos dados.

Além destes requisitos funcionais, foi possível identificar os seguintes requisitos não-funcionais (*RNF*):

- [*RNF01*] - Caso ocorra algum erro durante a migração dos dados, uma operação de *rollback* deverá ser executada;
- [*RNF02*] - O sistema deverá ser flexível o suficiente para permitir a inserção de *scripts SQL* customizáveis (ver RF04); e
- [*RNF03*] - O sistema deve ser intuitivo, de modo que o usuário saiba exatamente o que deve ser feito e como fazê-lo.

3.3.2. Casos de Uso

Para a realização da migração dos dados, é necessário que o usuário realize certas atividades, tais quais configurar os dados das conexões com os bancos de dados, realizar mapeamentos entre tabelas, entre outras. Estas atividades foram agrupadas em casos de uso (*UC*). O diagrama de casos de uso do sistema pode ser visualizado na Figura 3.2 e o detalhamento de cada caso de uso em particular será abordado logo em seguida.

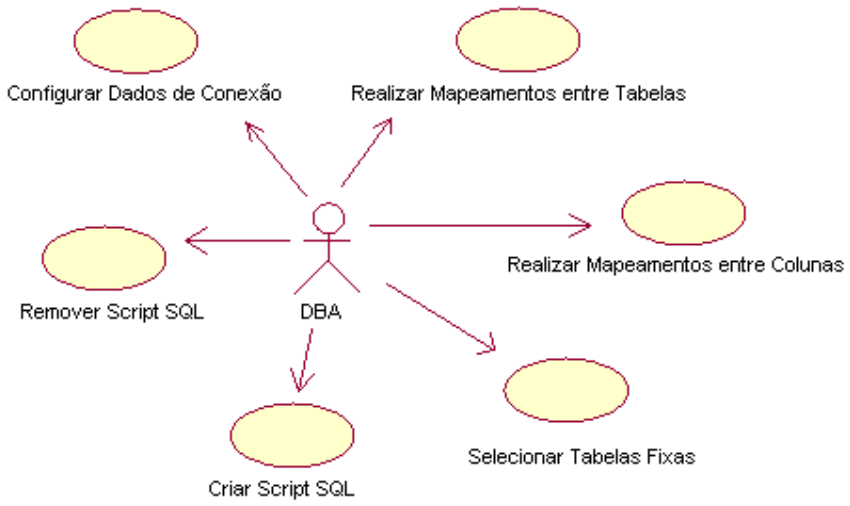

Figura 3.2: Diagrama de casos de uso do *Data Migration Wizard*.

3.3.2.1. *UC01* – Configurar Dados de Conexão

Prioridade:	☒ Essencial	☐ Importante	☐ Desejável
Ator:	DBA		

Descrição:

Este caso de uso tem por finalidade permitir com que o ator configure os dados das conexões com as das bases de dados utilizadas pela ferramenta: a base de origem e a base destino.

Pré-condições:

- O ator deve estar no passo referente ao gerenciamento das conexões com as bases de dados;
- O processo do SQL Server deve ter sido inicializado; e
- As bases de dados devem ter sido adicionadas ao SQL Server Management Studio.

Pós-condições:

- As *strings* de conexão com as bases de dados são geradas.

Fluxo Principal:

1. O ator preenche os dados relativos às conexões com as bases de dados a serem utilizadas pelo *Data Migration Wizard*:

  - os nomes dos servidores;
  - os nomes das bases de dados;
  - login; e
  - senha.

Caso o ator tenha escolhido a opção *Windows Authentication*, não será necessário informar seu login e senha;

2. O ator deve clicar no botão “Avançar”.

Fluxo Alternativo – Campos Obrigatórios:

- **Pré-condição:**

  Algum dos dados do passo 1 não foram preenchidos.

- **Fluxo:**

Caso o dado não preenchido faça parte da *string* de conexão com a base de dados de origem, então a mensagem de erro “*Preencha todos os campos relativos à conexão com a base de dados de origem.*” é exibida ao ator. Caso contrário, a seguinte mensagem é exibida: “*Preencha todos os campos relativos à conexão com a base de dados destino.*”.

Fluxo Alternativo – Erro de Conexão:

- **Pré-condição:**

Algum dos dados do passo 1 foram informados incorretamente.

- **Fluxo:**

Caso o dado preenchido erroneamente faça parte da *string* de conexão com a base de dados de origem, então a mensagem de erro "*Verifique os dados informados a respeito da conexão com a base de dados de origem.*" é exibida ao ator. Caso contrário, a seguinte mensagem é exibida: "*Verifique os dados informados a respeito da conexão com a base de dados destino.*".

3.3.2.2. *UC02* – Realizar Mapeamentos entre Tabelas

Prioridade:	☒ Essencial	☐ Importante	☐ Desejável
Ator:	DBA		

Descrição:

Este caso de uso tem por finalidade permitir com que o ator realize os mapeamentos entre as tabelas das bases de dados, de modo que o sistema seja capaz de, dado o mapeamento, realizar a migração dos dados da tabela de origem para a tabela destino.

Pré-condições:

- O ator deve estar no passo referente à realização dos mapeamentos entre tabelas.

Pós-condições:

- O mapeamento realizado pelo ator é salvo em memória, a fim de ser utilizado durante a migração dos dados.

Fluxo Principal:

1. A tabela da base de dados de origem é fixada pelo sistema, logo, o ator deve selecionar uma tabela da base de dados destino cujos dados serão provenientes da tabela fixada pelo sistema. Caso o mapeamento seja perfeito, ou seja, as duas tabelas possuem a mesma estrutura interna (quantidade de colunas, nomes e tipos das colunas), então um ícone indicará tal característica e o mapeamento foi finalizado com sucesso. Caso contrário, um ícone indicará a necessidade de realização do mapeamento entre as colunas das duas tabelas em questão.

3.3.2.3. *UC03* – Realizar Mapeamentos entre Colunas

Prioridade:	☒ Essencial	☐ Importante	☐ Desejável
Ator:	DBA		

Descrição:

Este caso de uso tem por finalidade permitir com que o ator realize o mapeamento entre as colunas de duas tabelas: a tabela de origem e a tabela destino.

Pré-condições:

- O ator deve estar no passo referente à realização do mapeamento entre colunas.

Pós-condições:

- O mapeamento realizado pelo ator é salvo em memória, a fim de ser utilizado durante a migração dos dados.

Fluxo Principal:

1. A coluna da tabela destino é fixada pelo sistema, logo, o ator deve selecionar uma coluna da tabela de origem, cujos dados serão migrados para a coluna fixada pelo sistema.

3.3.2.4. *UC04* – Selecionar Tabelas Fixas

Prioridade:	☒ Essencial	☐ Importante	☐ Desejável
Ator:	DBA		

Descrição:

Este caso de uso tem por finalidade permitir com que o ator possa escolher quais tabelas da base de dados destino não deve ter seu conteúdo apagado antes da migração (*e.g.* uma tabela de permissões recém-criada).

Pré-condições:

- O ator deve estar no passo referente à seleção das tabelas fixas.

Pós-condições:

- As tabelas escolhidas pelo ator são salvas em memória e serão consultadas durante a execução da limpeza da base de dados destino.

Fluxo Principal:

1. O ator escolhe as tabelas fixas clicando com o *mouse* em *checkboxes*. Caso alguma tabela escolhida dependa de outras tabelas, estas tabelas serão marcadas automaticamente, pois caso seus dados sejam apagados, a tabela

escolhida ferirá a integridade referencial da base, já que ela fará referência a *ids* que não existem mais.

3.3.2.5. *UC05* –Criar *Script SQL*

Prioridade:	☒ Essencial	☐ Importante	☐ Desejável
Ator:	DBA		

Descrição:

Este caso de uso tem por finalidade permitir com que o ator possa criar seus próprios *scripts SQL* e escolher quando eles devem ser executados.

Pré-condições:

- O ator deve estar no passo referente à visualização das iterações.

Pós-condições:

- Um novo *script SQL* é salvo em memória e será executado posteriormente.

Fluxo Principal:

1. O ator deve selecionar a localização do novo *script*;

2. O ator deve clicar no botão "*Criar Script*";

3. O ator deve preencher os dados relativos ao novo *script;* e

4. O ator deve finalizar a criação do novo *script* clicando no botão "S*alvar*".

Fluxo Alternativo – Campos Obrigatórios:

- **Pré-condição:**

Algum dos dados do passo 3 não foram preenchidos.

- **Fluxo:**

  A mensagem de erro "*Preencha todos os campos*" é exibida na barra de *status*.

3.3.2.6. *UC06* – Remover *Script SQL*

Prioridade:	☒ Essencial	☐ Importante	☐ Desejável
Ator:	DBA		

Descrição:

Este caso de uso tem por finalidade permitir com que o ator possa remover *scripts SQL* criados previamente.

Pré-condições:

- O ator deve estar no passo referente à visualização das iterações.

Pós-condições:

- O *script* selecionado é apagado da memória.

Fluxo Principal:

1. O ator deve selecionar o *script* a ser apagado; e
2. O ator deve clicar no botão "*Remover Script*" a fim de apagar o *script* da memória.

3.3.3. Modelagem dos Dados

Esta subseção tem por objetivo detalhar a modelagem dos dados utilizada no desenvolvimento do *Data Migration Wizard*. A Figura 3.3 a seguir ilustra as entidades criadas:

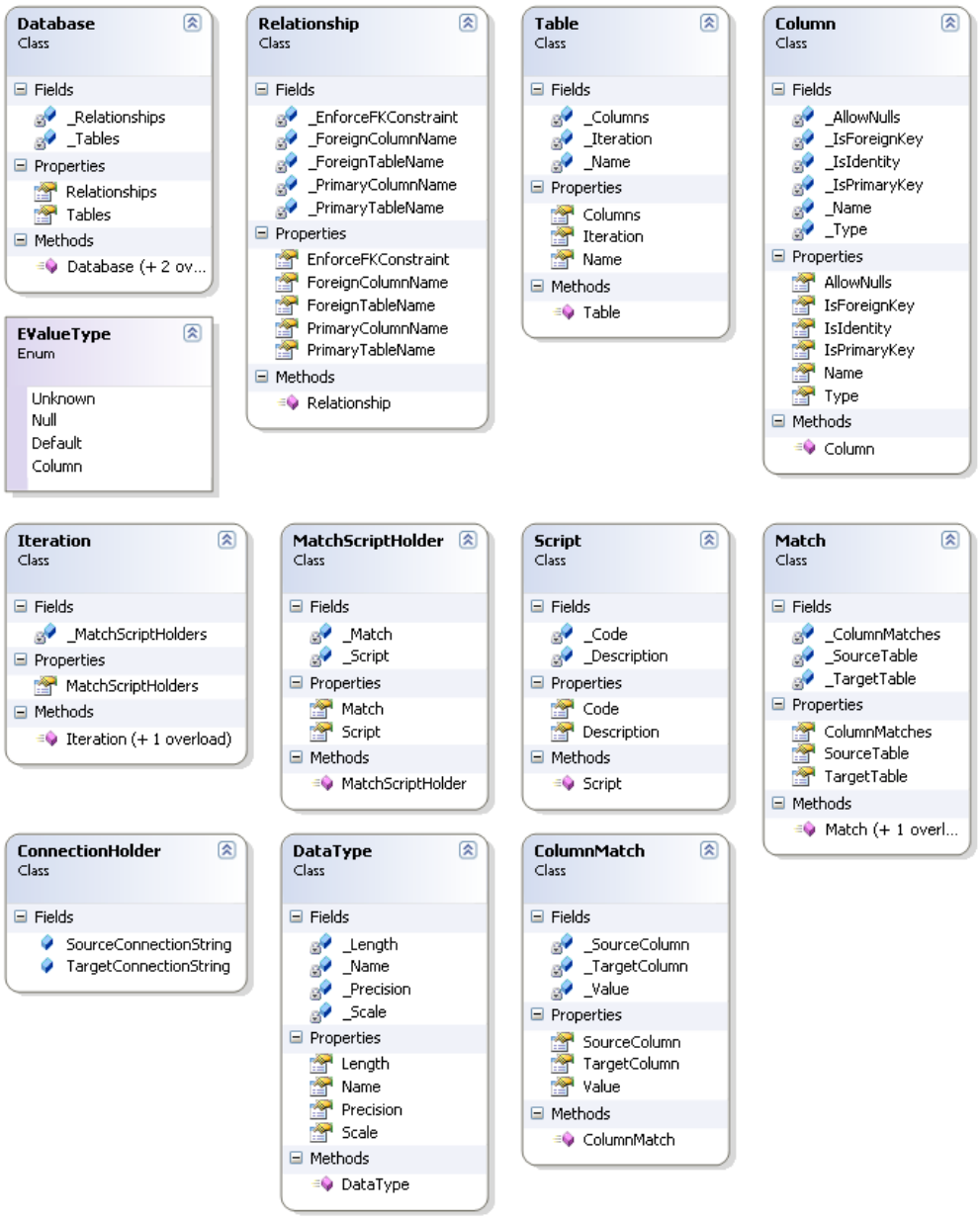

Figura 3.3: Diagrama das entidades do *Data Migration Wizard*.

O detalhamento de cada classe é o que se segue:

1. A classe *Database* representa uma base de dados e possui os seguintes atributos: uma lista de tabelas e uma lista de relacionamentos. A lista de tabelas foi implementada utilizando uma estrutura de dados proveniente da plataforma *.NET* que permite com que a tabela possa ser indexada pelo seu nome, ou seja, existe um mapeamento direto entre o nome da tabela e o objeto que a representa. Esta estrutura é o *Dictionary*, que faz parte do *namespace System.Collections.Generic*;
2. A classe *Relationship* representa um relacionamento e possui os seguintes atributos: um *booleano* que indica se este relacionamento segue a restrição de integridade referencial, os nomes da coluna e da tabela onde se encontra a chave primária do relacionamento e os nomes da coluna e da tabela onde se encontra a chave estrageira do relacionamento;
3. A classe *Table* representa uma tabela e possui os seguintes atributos: o nome da tabela, um inteiro que indica a qual iteração a tabela pertence e uma lista de colunas, que foi implementada utilizando a mesma estrutura (*Dictionary*) descrita no detalhamento 1;
4. A classe *Column* representa uma coluna e possui os seguintes atributos: um *booleano* que indica se a coluna pode receber um valor nulo, um *booleano* que indica se a coluna é uma chave primária, um *booleano* que indica se a coluna é uma chave estrangeira, um *booleano* que indica se a coluna é auto-incremento, o nome da coluna e seu tipo, que é representado pela classe *DataType*;
5. A classe *DataType* representa o tipo de uma coluna e possui os seguintes atributos: o nome do tipo (como *varchar, decimal, bit, int,* entre outros), seu tamanho, escala e precisão;
6. A classe *Iteration* representa uma iteração e possui uma lista de *MatchScriptHolder* como atributo;
7. A classe *MatchScriptHolder* serve de *wrapper* das classes *Script* e *Match*, pois a ferramenta trabalha tanto com mapeamentos quanto com *scripts* criados pelo administrador da base de dados;

8. A classe *Script* representa um *script SQL* criado pelo administrador da base de dados e possui os seguintes atributos: uma descrição e o código *SQL*;
9. A classe *Match* representa um mapeamento entre tabelas e possui os seguintes atributos: a tabela de origem, a tabela destino e uma lista de mapeamentos das colunas das tabelas, representada pela classe *ColumnMatch*;
10. A classe *ColumnMatch* representa o mapeamento entre uma coluna de uma tabela da base de origem e outra de uma tabela da base destino. Vale ressaltar que nem sempre existe um mapeamento direto entre colunas. Neste caso, a coluna da tabela da base destino será nula e a coluna da tabela da base destino receberá algum valor *default* ou nulo. Esta classe possui como atributos as colunas e um enumerador que indica se a coluna da tabela da base destino receberá um valor nulo, *default* ou se este valor virá diretamente de outra coluna;
11. A classe *EValueType* representa o tipo do valor que uma coluna receberá, podendo este ser nulo, *default* ou proveniente de outra coluna; e
12. A classe *ConnectionHolder* é a classe responsável por armazenar as *strings* de conexão com as duas bases em questão: de origem e destino.

3.3.4. Arquitetura

Durante a etapa de projeto do *Data Migration Wizard*, foi decidido utilizar uma arquitetura em camadas, criando-se então uma camada provedora de serviços chamada *Services Provider*, uma camada responsável pelo tratamento das regras de negócio chamada *Business Provider* e uma camada responsável por lidar diretamente com o código relativo à base de dados trabalhada (*SQL Server*) chamada *Data Provider.*

O motivo pelo qual foi feita esta escolha está na expansibilidade que este tipo de arquitetura traz. Apesar de não ser um requisito não-funcional da ferramenta, foi considerado importante torná-la expansível, de modo que, caso haja a necessidade de utilizar algum sistema de gerenciamento de banco de dados diferente do *SQL Server*, será necessário apenas implementar a camada de dados da ferramenta, já que as camadas

são independentes. Esta arquitetura também torna possível fazer com que a *interface* gráfica possa executar em uma máquina cliente e o resto da aplicação em uma máquina servidora através do uso de *web services* que se conectam diretamente à camada provedora de serviços. A Figura 3.4 ilustra a arquitetura utilizada na ferramenta.

Figura 3.4: Arquitetura do *Data Migration Wizard*.

3.4. FUNCIONAMENTO

Nesta seção será abordado o funcionamento do *Data Migration Wizard* através da utilização de uma base de dados real. Também serão discutidos alguns detalhes mais relevantes acerca da implementação da ferramenta.

Como foi visto na seção 3.1, o assistente foi concebido para realizar a migração de dados entre bases de dados de aplicações que sofrem atualizações em suas versões. A

base de origem utilizada neste exemplo é composta de 30 tabelas e a base destino é composta de 34 tabelas. Além da adição de novas tabelas, outras sofrem modificações:

- Foram adicionadas duas colunas à tabela EmailsLog;
- Foi removida uma coluna da tabela Log; e
- Uma coluna da tabela Empresas teve seu tipo modificado.

Ao executar o assistente, uma tela de apresentação é exibida ao administrador da base de dados, como mostra a Figura 3.5:

Figura 3.5: Tela de apresentação.

3.4.1. Passo 1 – Configurar as Conexões com as Bases de Dados

Neste passo, que corresponde ao *UC01*, o usuário deve preencher os campos com os dados que formarão as *strings* de conexão com as bases de dados utilizadas durante o processo de migração. É válido destacar que, sempre que alguma operação é executada, esta informação é exibida ao usuário através da barra de *status* localizada na parte inferior da tela, satisfazendo o *RF03*. O passo 1 é ilustrado pela Figura 3.6 a seguir:

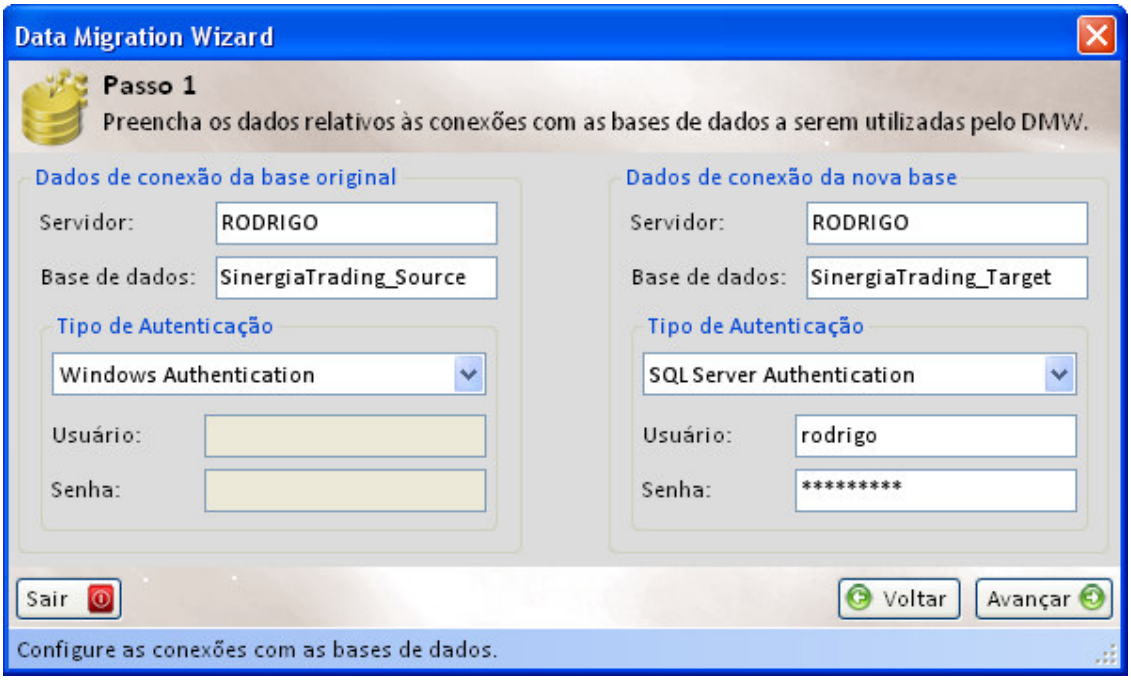

Figura 3.6: Tela referente ao passo 1.

É interessante atentar para o fato de que é possível fazer uso das credenciais utilizadas pelo sistema operacional, não havendo a necessidade de informar o nome do usuário e sua senha. Tal característica é obtida ao escolher a opção "*Windows Authentication*" como tipo de autenticação.

Ao clicar no botão "A*vançar*", o sistema realiza a análise das bases de origem e destino, define as iterações das tabelas destino, exibindo as dependências cíclicas identificadas, e avança para o passo 2 do processo de migração.

Para implementar as operações que analisam uma base de dados, foram utilizadas classes que facilitam a manipulação de informações de bases de dados *SQL Server*. Estas classes pertencem aos *namespaces Microsoft.SqlServer.Management.Smo* e *Microsoft.SqlServer.Management.Common*, providos pela plataforma *.NET*.

Para desenvolver a operação que define as iterações das tabelas destino, foi utilizado o pseudo-código a seguir:

- O algoritmo recebe como entrada a base de dados destino, onde todas as tabelas possuem nível -1;
- Todas as tabelas que não dependem de outras tabelas possuirão nível 0;

- Enquanto há tabelas com nível -1 e alguma tabela teve seu nível definido neste laço, faça:
    - Para cada tabela com nível -1, faça:
        - Se as tabelas das quais ela depende tiverem seu nível definido, o nível desta tabela será o maior dentre os níveis das tabelas das quais ela depende, acrescido de 1.

Como resultado do algoritmo utilizado, caso haja tabelas que não tiveram o seu nível identificado, a migração de seus dados deve ser tratada manualmente, já que elas fazem parte de dependências cíclicas. O tratamento automático de dependências cíclicas não faz parte do escopo deste trabalho (como afirmado na seção 3.2), mas sua identificação satisfaz o *RF02*. A Figura 3.7 a seguir ilustra como as tabelas identificadas são exibidas ao usuário:

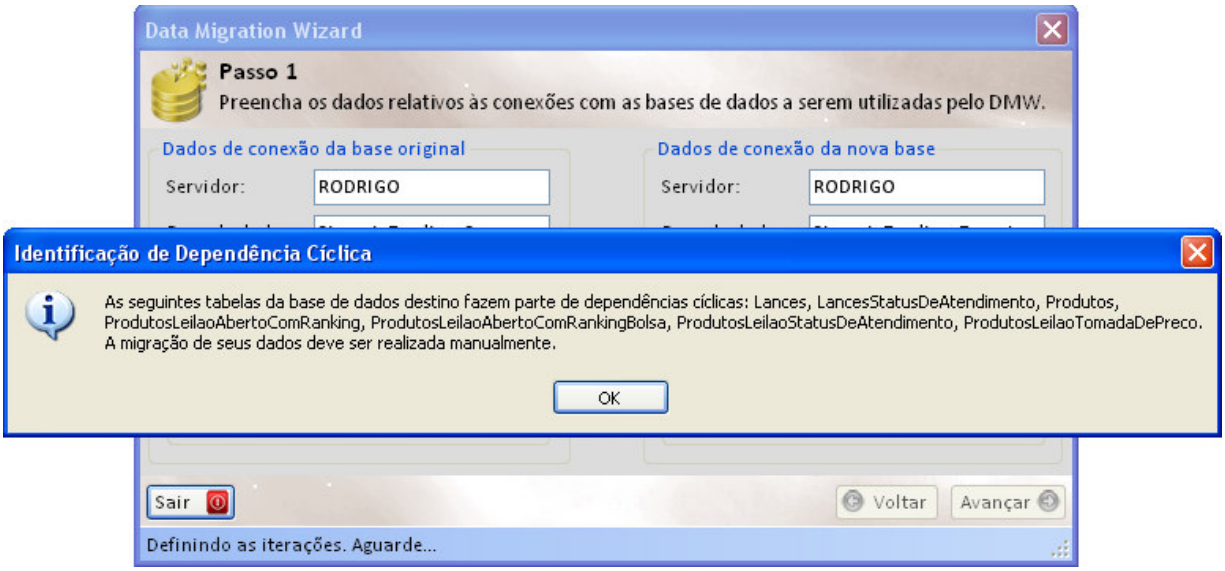

Figura 3.7: Identificação de dependências cíclicas.

Caso ocorra algum erro durante o uso da ferramenta, este é exibido ao usuário na barra de *status*, como mostra a Figura 3.8:

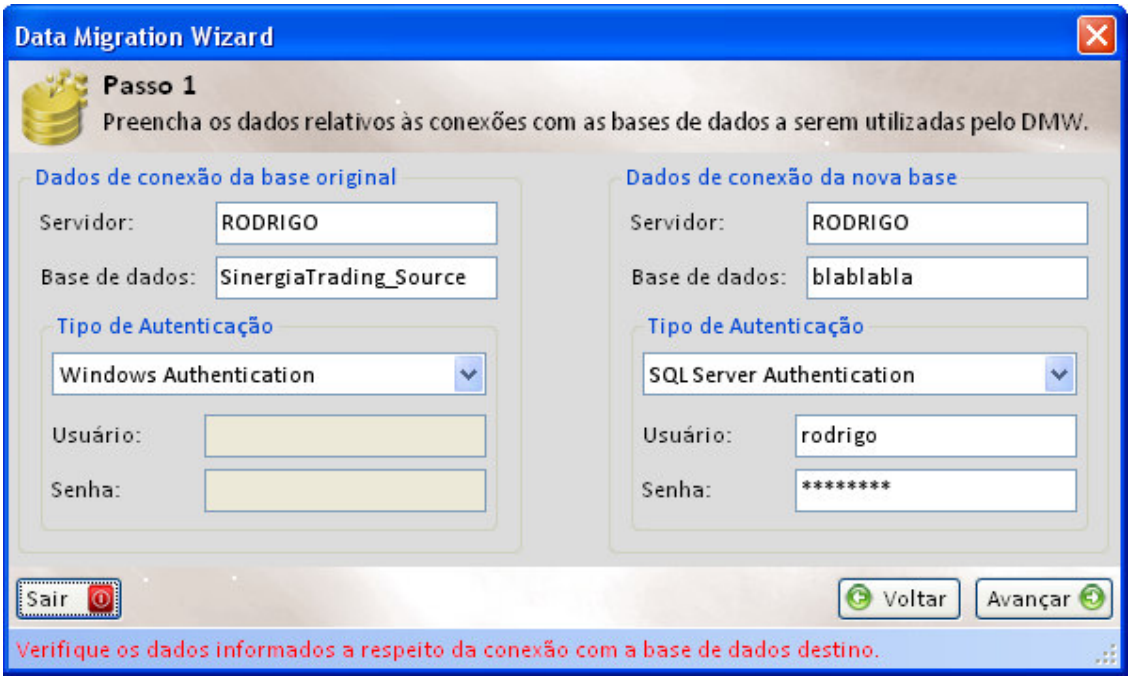

Figura 3.8: Exibição de erro ao usuário.

3.4.2. Passo 2 – Realizar Mapeamentos entre Tabelas

Neste passo, o usuário deve realizar os mapeamentos entre as tabelas e entre as colunas das bases de origem e destino. O mapeamento entre as tabelas é considerado "orientado à origem", pois são exibidas todas as tabelas da base de origem e o usuário deve escolher uma ou mais tabelas (a possibilidade de uma tabela de origem prover dados para mais de uma tabela destino será abordada mais adiante) da base destino para realizar o mapeamento. Já o mapeamento entre as colunas é considerado "orientado ao destino", pois dadas as duas tabelas pertencentes ao mapeamento, são exibidas todas as colunas da tabela destino e o usuário deve escolher dentre as colunas da tabela de origem para realizar o mapeamento.

No momento em que a tela referente ao passo 2 é exibida ao usuário, as tabelas da base de origem são carregadas e os mapeamentos automáticos são realizados. Neste caso, um mapeamento é realizado automaticamente quando uma tabela da base de origem possui uma tabela correspondente (de nome igual) na base destino, onde todas as colunas da tabela de origem possuem colunas correspondentes (iguais em nome e tipo) na tabela destino. A realização de mapeamentos automáticos satisfaz o *RF05* e garante que o

usuário não perca tempo realizando mapeamentos óbvios e se preocupe apenas em mapear o que de fato difere entre as bases. A Figura 3.9 a seguir ilustra a tela referente ao passo 2 da ferramenta:

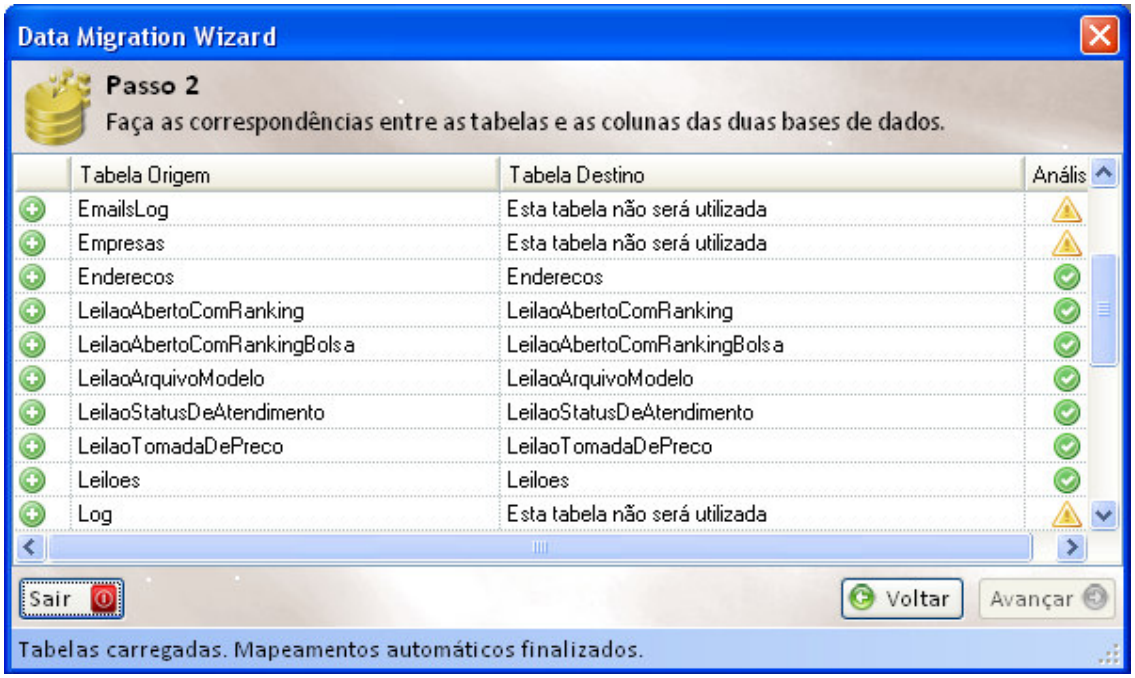

Figura 3.9: Tela referente ao passo 2.

Caso uma tabela de origem não exista na base destino, o usuário simplesmente deve escolher a opção "*esta tabela não será utilizada*", de modo que o sistema ignore esta tabela a partir de então. Uma outra opção de uso para esta tela consiste da possibilidade do usuário utilizar uma mesma tabela para prover dados para mais de uma tabela da base destino (o contrário não faz parte do escopo da aplicação, como foi apresentado na seção 3.2). Neste caso, o usuário deve clicar duas vezes sobre o ícone indicativo de adição e realizar o mapeamento da tabela replicada com outra tabela da base destino. A Figura 3.10 mostra este tipo de ação:

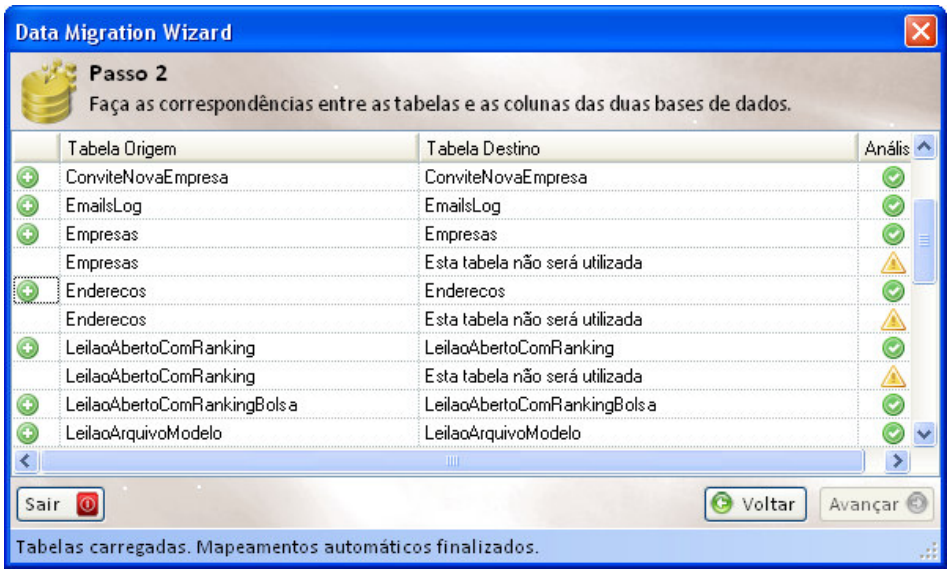

Figura 3.10: Replicação de tabela de origem.

Quando o sistema não consegue realizar o mapeamento automático, um ícone indicativo de atenção é exibido na linha pertencente à tabela em questão. Caso o usuário clique sobre o ícone duas vezes, o sistema abre uma nova tela (que corresponde ao *UC03*), na qual o usuário terá a possibilidade de realizar o mapeamento entre as colunas das duas tabelas exibidas pela linha. A Figura 3.11 a seguir ilustra o momento quando o usuário clica duas vezes sobre o ícone de atenção pertencente à linha da tabela *EmailsLog*:

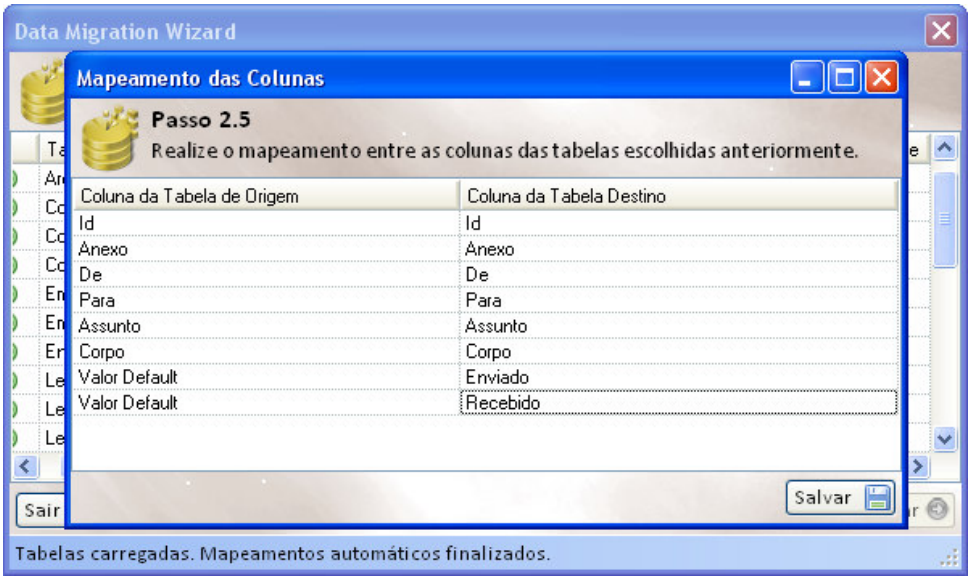

Figura 3.11: Tela de mapeamento de colunas.

Vale ressaltar que no momento em que a tela é exibida, o sistema realiza o mapeamento automático entre as colunas de mesmo nome e tipo, fazendo com que o usuário se preocupe apenas em mapear as colunas que impediram a realização do mapeamento automático na tela anterior. Após a realização dos mapeamentos, o usuário pode continuar o processo de migração clicando no botão "A*vançar*", o qual o levará para a tela referente ao passo 3.

3.4.3. Passo 3 – Selecionar tabelas fixas

Neste passo, que corresponde ao *UC04*, o usuário terá a possibilidade de escolher quais tabelas da base destino não deverão ter seus registros apagados antes da migração dos dados. Este passo tem a sua importância devido ao fato de que é possível que tabelas cujos registros apenas sejam adicionados manualmente tenham sido criadas na base de dados destino (*e.g.* uma tabela de permissões) e que o administrador da base deseje manter seus registros, já que adicioná-los novamente é um trabalho desnecessário. A Figura 3.12 a seguir mostra a tela referente ao passo 3 do assistente:

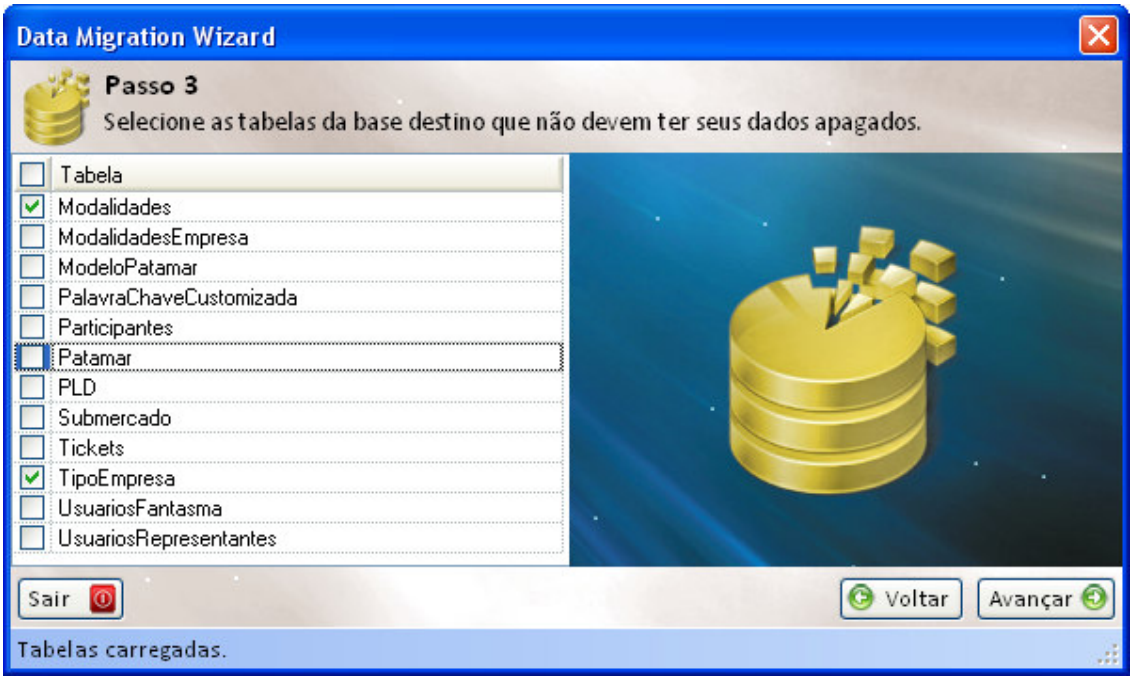

Figura 3.12: Tela referente ao passo 3.

Ao selecionar as tabelas fixas, vale atentar para o seguinte fato: caso a tabela escolhida dependa de outra tabela, esta será selecionada automaticamente de modo que a restrição de integridade referencial não será violada durante a limpeza da base de dados. Utilizando o exemplo visto na seção 3.1, caso o usuário selecione a tabela Empresas, automaticamente a tabela TiposEmpresa será selecionada. Após selecionar as tabelas fixas desejadas (caso existam), o usuário deve clicar no botão "A*vançar*" para ser levado à tela referente ao passo 4 do assistente.

3.4.4. Passo 4 – Visualizar a limpeza da base de dados destino

Neste passo, que satisfaz o *RF06*, o usuário simplesmente visualiza a limpeza da base de dados destino. São informados ao usuário o momento em que a limpeza de cada tabela é iniciada e o estado da operação (em andamento ou concluído). É importante salientar que a limpeza da base ocorre em ordem decrescente de iteração, ou seja, as tabelas que possuem o maior nível de dependência são apagadas primeiro, de modo que não ocorra uma violação da integridade referencial no banco de dados. A Figura 3.13 a seguir ilustra o momento em que os registros de uma tabela são apagados:

Figura 3.13: Tela referente ao passo 4.

O botão “A*vançar*” somente é desabilitado quando a limpeza da base é finalizada, como pode ser observado na Figura 3.14:

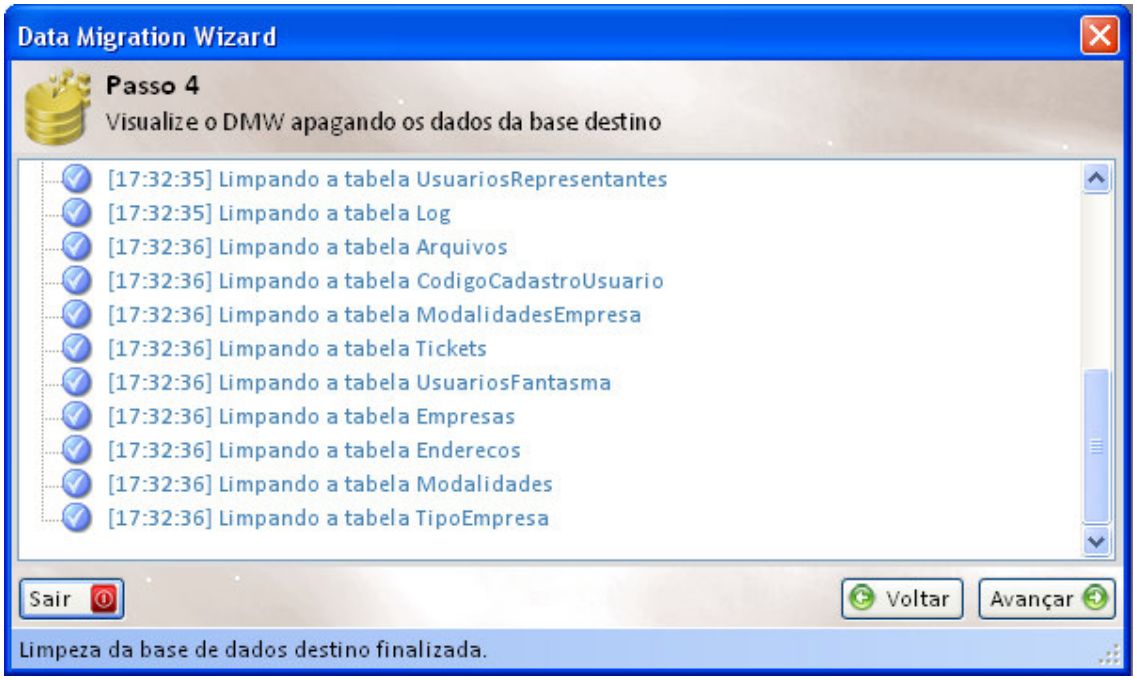

Figura 3.14: Conclusão da limpeza da base de dados destino.

3.4.5. Passo 5 – Visualizar as iterações e criar *scripts SQL*

Neste passo, o usuário pode visualizar as iterações criadas pelo *Data Migration Wizard*, criar e remover *scripts SQL*. As ações descritas em cada iteração são resultado dos mapeamentos elaborados pelo usuário no passo 2 e da criação de *scripts SQL*. A Figura 3.15 a seguir ilustra a tela referente ao passo 5 da ferramenta:

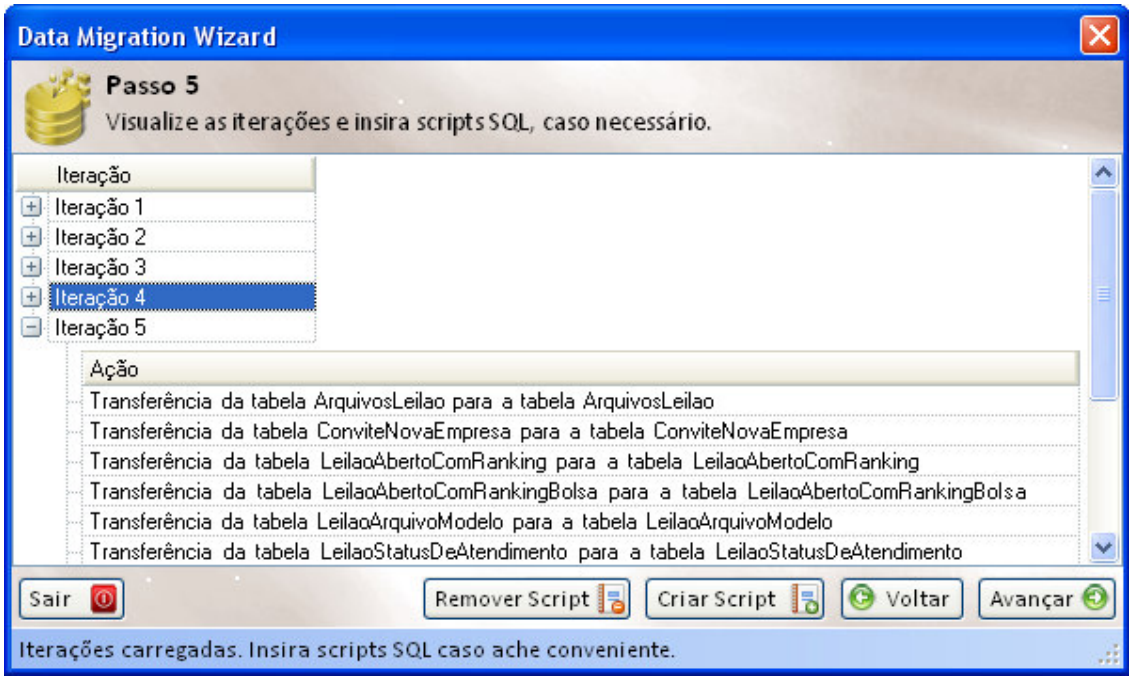

Figura 3.15: Tela referente ao passo 5.

Para criar um novo *script* (*UC05, RF04*), basta que o usuário escolha o momento no qual ele será executado e clique no botão "*Criar Script*", o qual abrirá uma tela de configuração dos dados do *script*, como mostra a Figura 3.16:

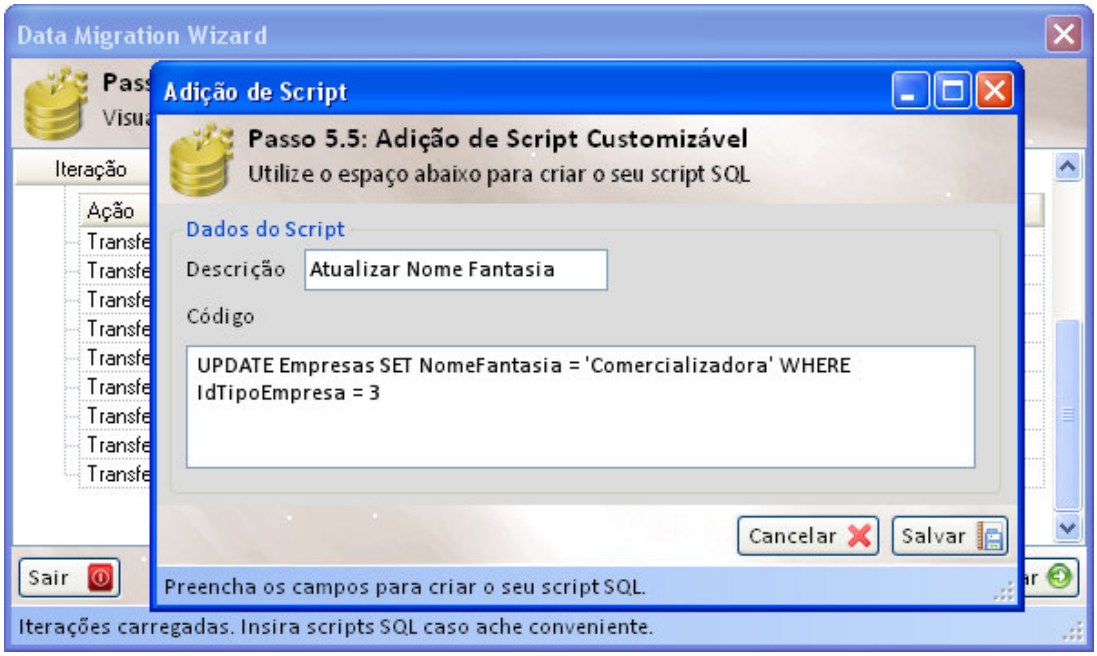

Figura 3.16: Tela de configuração de um novo *script SQL*.

Ao preencher os campos e clicar no botão "*Salvar*", o novo *script* é adicionado à lista de ações da iteração correspondente, como mostra a Figura 3.17 a seguir:

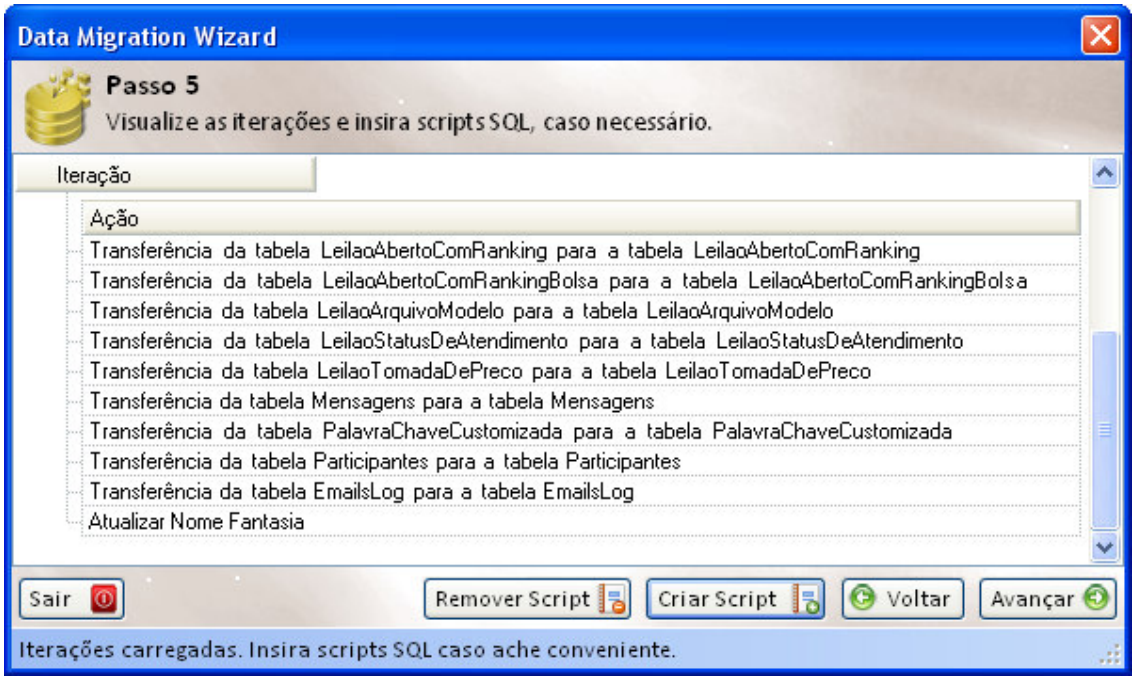

Figura 3.17: Visualização de novo *script*.

Caso o usuário deseje remover algum *script* criado por ele (*UC06*), basta selecioná-lo e clicar no botão "*Remover Script*". Clicando no botão "*Avançar*", o usuário é levado para a tela referente ao último passo da ferramenta.

3.4.6. Passo 6 – Visualizar a migração dos dados

Neste passo, o usuário simplesmente visualiza a execução da migração dos dados. São informados ao usuário o momento em que cada ação (seja resultado do mapeamento ou um *script* criado pelo próprio administrador da base) é iniciada e o estado da mesma (em andamento ou concluído). A Figura 3.18 a seguir ilustra o momento em que uma ação ocorre:

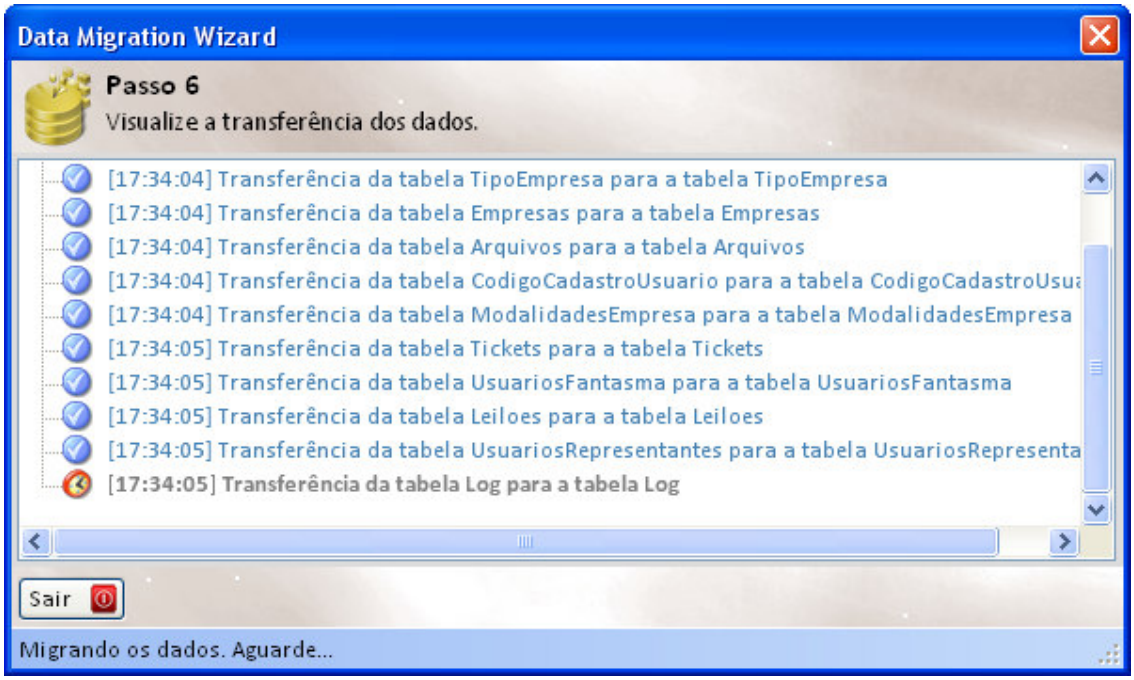

Figura 3.18: Tela referente ao passo 6.

Um problema acerca da operação de transferência que vale a pena destacar é o seguinte: quando uma tabela depende de outra, a primeira tem armazenado em seus registros um valor correspondente ao identificador da segunda. Quando os dados da segunda tabela são migrados (lembrando que seus dados serão migrados antes dos dados da primeira tabela, já que seu nível de dependência é menor), um novo valor para o identificador na tabela destino é criado automaticamente. Quando os dados da tabela dependente forem migrados, o valor armazenado na coluna que é a chave estrangeira deste relacionamento está ultrapassado, logo não pode ser utilizado. As figuras 3.19 e 3.20 ilustram este problema:

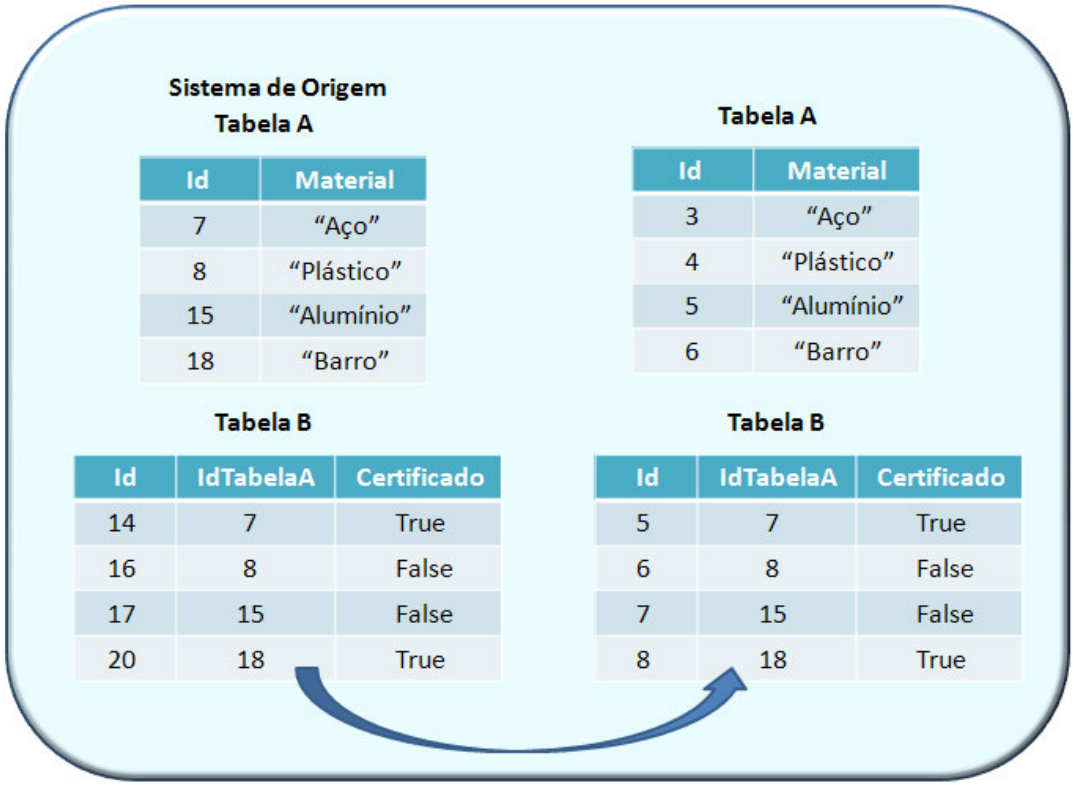

Figura 3.19: Problema relacionado à dependência entre tabelas.

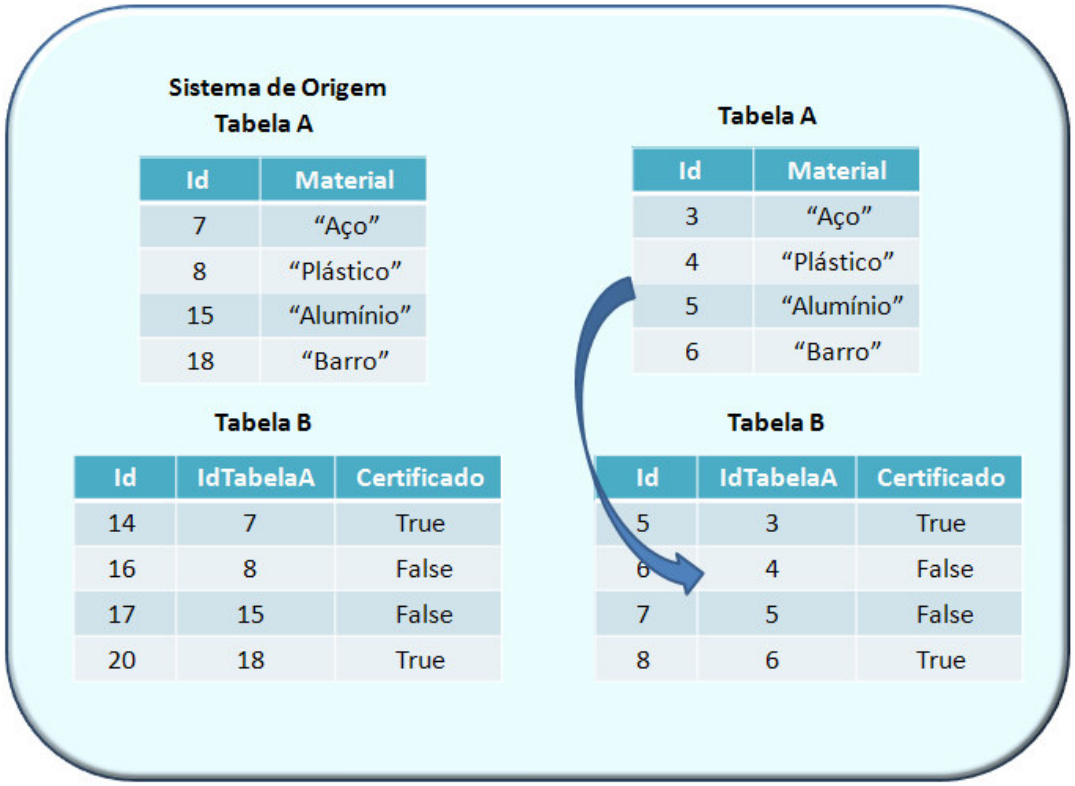

Figura 3.20: Comportamento ideal da migração diante de relacionamentos entre tabelas.

Portanto, havia a necessidade de realizar um mapeamento entre o valor do identificador original e o novo valor gerado pela base de dados. Para realizar tal mapeamento foi utilizado um *Dictionary<string, int>*, que é uma estrutura fornecida pela plataforma *.NET* que mapeia uma chave do tipo *string* em um valor do tipo inteiro. A chave armazenada é composta pela concatenação do nome da tabela com o valor antigo do identificador e o valor resultante é o novo identificador do registro em questão. Pode-se perceber que, desta maneira, cada registro de uma tabela irá compor um par chave/valor único. No momento em que os dados de uma tabela dependente são migrados, o valor do campo que é chave estrangeira é obtido na tabela de mapeamentos. A Tabela 3.1 a seguir ilustra este mapeamento:

Chave	Valor
A7	3
A8	4
A15	5
A18	6

Tabela 3.1: Mapeamento entre valores de identificadores.

Quando o processo de migração dos dados é concluído, uma mensagem de conclusão com sucesso é exibida e a única ação que o usuário pode realizar é fechar o assistente, como mostra a Figura 3.21:

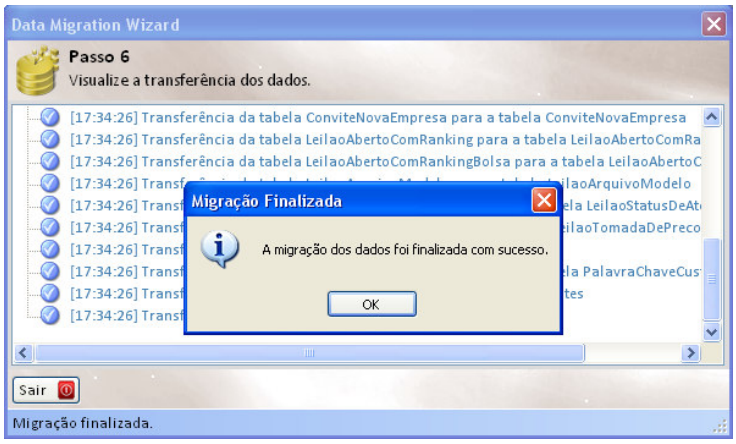

Figura 3.21: Conclusão da migração com sucesso.

4. ESTUDO DE CASO

Com o intuito de validar os requisitos implementados no assistente e verificar se, de fato, a ferramenta supera os pontos fracos identificados no *Data Importer/Exporter* do *SQL Server Management Studio* da *Microsoft* (mostrados na seção 1.2), foi realizado um estudo de caso com duas bases de dados geradas a partir da evolução de um projeto da empresa de desenvolvimento de software Suporte Avançado em Tecnologia da Informação – SUATI. A base de dados de origem contém 148 tabelas, cuja quantidade de registros varia de 2 (tabela que armazena os tipos de unidades de energia - UnidadesEnergia) a 1.751.107 (tabela que armazena as medições feitas pelo cliente da empresa - Medicao). A única modificação ocorrida que deu origem à base destino foi a adição de 5 tabelas, resultantes da agregação de novas funcionalidades requisitadas pelo cliente da SUATI. Na próxima seção, serão mostrados os resultados obtidos com a execução do estudo de caso.

4.1. RESULTADOS OBTIDOS

Após a realização do estudo de caso pelo líder técnico da empresa, David Cardoso, este foi convidado a responder um questionário, de modo que fosse possível verificar seu nível de contentamento com a ferramenta e se esta supera a atual ferramenta de migração de dados utilizada na empresa, o *Data Importer/Exporter*. As perguntas e respostas foram transcritas a seguir:

1. O assistente é intuitivo o suficiente para tornar o processo de migração uma atividade simples?

    R. Sim. O assistente automatiza todas as etapas do processo de transferência de dados entre bancos de dados com estruturas similares de maneira simples e intuitiva. O assistente solicita ao usuário o preenchimento das informações estritamente necessárias para a conclusão do processo e que não podem ser inferidas a partir da análise da estrutura dos bancos de dados.

2. Você acha que o assistente é completo?

   R. Sim. O assistente contempla todas as possibilidades de migração de dados que necessitamos, analisa as bases para identificar as tabelas que fazem parte de dependências, descobre a ordem correta de transferência automaticamente, permite a criação de consultas personalizadas para atualização dos dados e ainda realiza mapeamentos automáticos.

3. Quais são os pontos fortes e fracos do assistente de migração de dados?

   R. **Pontos fortes:** a facilidade com que bancos de dados grandes e complexos podem ser migrados sem que o usuário precise analisar minuciosamente cada diferença entre os bancos de dados, possibilitando uma maior eficiência devido à redução do esforço/tempo gasto no processo e diminuindo substancialmente o risco de falha humana e perda de dados no processo de migração.

   Ponto fracos: o assistente exige o uso de um computador com disponibilidade de memória e processamento elevados.

4. No que o assistente supera a ferramenta da Microsoft?

   R. **Automação** - A ferramenta da *Microsoft* faz uma análise superficial das estruturas dos bancos de dados envolvidos, sem considerar relacionamentos e chaves primárias e, portanto, é incapaz de inferir automaticamente a ordem com que as tabelas devem ser importadas nem quais tabelas fazem parte de dependências cíclicas. Cabe ao usuário identificar a ordem correta e executar a ferramenta diversas vezes, manualmente, a fim de concluir o processo de migração. O assistente realiza o processo de migração completo de uma só vez. E o assistente também faz muitos mapeamentos automáticos, fazendo com que o usuário se preocupe apenas com o que está diferente entre as bases. Além disso ele ainda limpa a base destino, deixando a cargo do usuário quais tabelas não devem ser apagadas;

   Flexibilidade - A ferramenta permite a introdução de consultas *SQL* personalizadas durante o andamento do processo de migração. Este

recurso aumenta consideravelmente as possibilidades que o usuário possui de manipular e personalizar cada etapa do processo; e

Segurança de consistência dos dados - Como o assistente realiza toda a migração de uma vez, caso ocorra algum erro, ele desfaz todas as operações executadas antes da ocorrência do erro. Já na ferramenta da *Microsoft*, caso ocorra algum erro, todas as operações realizadas previamente não são desfeitas, pois o programa tem que ser executado várias vezes para concluir o processo de migração.

5. Você substituiria o *Data Importer/*Exporter pelo *Data Migration* Wizard?

   R. Sim.

5. CONSIDERAÇÕES FINAIS

Empresas gastam milhões de dólares migrando dados entre aplicações que lidam com informação em massa e, ainda com todo esse investimento, 75% dos novos sistemas falham em satisfazer as expectativas [5]. Muita pesquisa vem sendo conduzida na tentativa de desenvolver uma maneira segura e barata de guiar a migração de um sistema legado como um todo [2, 3, 4, 6, 7, 8, 12, 16, 17, 18, 19, 20, 21, 22] .

Ao longo deste trabalho, foram vistas duas metodologias de migração de sistemas legados, duas estratégias de migração de dados, os passos envolvidos neste processo, seus desafios e fatores de sucesso. Foi mostrada a ferramenta que representa o estado da arte acerca de projetos de migração de dados e a principal concorrente do assistente desenvolvido neste trabalho. Também foram mostradas com detalhes as especificações técnicas do *Data Migration Wizard*, suas características gerais e seu funcionamento. O trabalho foi finalizado com a realização de um estudo de caso, a fim de verificar se o assistente supera os pontos fracos identificados no seu principal concorrente.

Na próxima seção são sugeridas melhorias no *Data Migration Wizard* a fim de tornar a ferramenta mais sofisticada.

5.1. TRABALHOS FUTUROS

Como foi mostrado na seção 4.1 deste trabalho, o *Data Migration Wizard* pode ser considerado um exemplo de projeto de sucesso, pois ele satisfez todos os seus requisitos. Porém, o trabalho não está completo. Existem funcionalidades e características que podem ser agregadas ao assistente de modo que ele se torne uma ferramenta mais sofisticada e capaz de resolver outras necessidades de negócio. Estas sugestões são enumeradas a seguir:

1. Tornar possível que o usuário escolha qual tipo de SGBD utilizar, não se restringindo apenas ao *SQL Server* (e.g. a base de origem pode ser em um banco *Oracle* e a base destino pode ser um banco *MySQL*);

2. Dar suporte à possibilidade dos dados que compõem uma tabela destino serem provenientes de mais de uma tabela da base de origem;
3. Migrar automaticamente dados de tabelas que fazem parte de dependências cíclicas; e
4. Desenvolver o algoritmo de transferência de dados utilizando *stored procedures*, a fim de otimizar a performance e o uso de memória do servidor.

Referências Bibliográficas

[1] Aebi, D.; **Data Reengineering: A Case Study**, Advances in Databases and Information Systems, Springer-Verlag, Berlim, 1997.

[2] Bennett, K. H.; **Legacy Systems: Coping with Success**, IEEE Software, Volume 12, Número 11, Páginas 19 - 23, Janeiro 1995.

[3] Brodie, M.; Stonebraker, M.; **DARWIN: On the Incremental Migration of Legacy Information Systems**, Relatório Técnico TR-022-10-92-165 GTE Labs Inc., Março 1993.

[4] Brodie, M.; Stonebraker, M.; **Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach**, Morgan Kaufmann Publishers Inc. 1995.

[5] Datanomic Ltd.; **Data Migration: Reengineering Data for Optimized Value.**

http://whitepaper.informationweek.com/shared/write/collateral/WTP/52344_87837_94801_Successful_Business_Guide_-_Data_Migration_September_2007.pdf?ksi=1578523&ksc=1291504496

Último acesso em 6 de outubro de 2007.

[6] Dayal, U.; Hwang H.; **View Definition and Generalization for Database Integration in a Multidatabase System**, IEEE Transactions on Software Engineering, Volume 10, Número 6, Novembro 1984.

[7] Elmasri, R.; Navathe S. B.; **Object Integration in Database Design**, IEEE Conference on Data Engineering, Los Angeles, Califórnia, Abril 1984.

[8] Ganti, N.; Brayman W.; **Transition of Legacy Systems to a Distributed Architecture**, John Wiley & Sons Inc. 1995.

[9] IBM; **IBM Hardware-Assisted Data Migration Services**.

http://www-03.ibm.com/systems/storage/solutions/services/whitepapers/pdf/g522-2583.pdf

Último acesso em 6 de outubro de 2007.

[10] Manek, P.; **Microsoft CRM Data Migration Framework**, Abril 2003.

[11] March, S. T.; **Special Issue on Heterogeneous Databases**, Relatórios de Computação da ACM, Volume 22, Número 3, 1990.

[12] Navathe, S. B.; Gadgil S. G.; **A Methodology for View Integration in Logical Database Design**, 8th International Conference on Very Large Databases, Cidade do México, México, Setembro 1992.

[13] Softek; **Best Practices for Data Migration**.

http://www.cio.co.uk/whitepapers/index.cfm?whitepaperid=1744

Último acesso em 6 de outubro de 2007.

[14] Softek Survey; **Softek's Worldwide Data Migration Survey**, 2005.

[15] Sommerville, I.; **Software Engineering Environments**, 5ª Edição, Addison-Wesley.

[16] SUATI; http://www.suati.com.br Último acesso em 20 de janeiro de 2008.

[17] Tilley, S. R.; Smith D. B.; **Perspectives on Legacy System Reengineering**, Novembro 1996.

[18] Wu, B.; Lawless, D.; Bisbal, J.; Richardson, R.; Grimson, J.; Wade, V.; O'Sullivan, D.; **An Overview of Legacy Information System Migration**, Asia Pacific Software Engineering Conference e International Computer Science Conference, Páginas: 529 – 530, Dezembro 1997.

[19] Wu, B.; Lawless, D.; Bisbal, J.; Grimson, J.; **Legacy Information Systems: Issues and Directions**, IEEE Software, Volume 16, Número 5, Páginas: 103 - 111, Setembro/Outubro 1999.

[20] Wu, B.; Lawless, D.; Bisbal, J.; Grimson, J.; Wade, V.; O'Sullivan, D.; Richardson, R.; **Legacy Systems Migration - A Method and its Tool-Kit Framework**, Asia Pacific Software Engineering Conference e International Computer Science Conference. Páginas: 312 – 320, Dezembro 1997.

[21] Wu, B.; Lawless, D.; Bisbal, J.; Richardson, R.; Grimson, J.; Wade, V.; O'Sullivan, D.; **The Butterfly Methodology: A Gateway-free Approach for Migration Legacy Information Systems**, 3rd IEEE International Conference on Engineering of Complex Computer Systems, Páginas 200 - 205, Setembro 1997.

[22] Wu, L.; Sahraoui, H.; Valtchev, P.; **Coping with Legacy System Migration Complexity**, 10th IEEE International Conference on Engineering of Complex Computer Systems, Páginas: 600 - 609, Junho 2005.

[23] Youn, C.; Ku, C. S.; **Data Migration**, IEEE International Conference on Systems, Man and Cybernetics, Volume 2, Páginas: 1255 - 1258, Outubro 1992.

Orientador

_____

Prof. Ph.D. Fernando da Fonseca de Souza

Aluno

_____

Rodrigo Lumack do Monte Barretto