Evolução de Bancos de Dados Utilizando Planejamento Automático

M.B.P. Domingues e J. R. de Almeida Jr

***Abstract*—O projeto e a implementação de bancos de dados evolutivos são importantes desafios para o desenvolvimento de sistemas de computação, tendo em vista as frequentes mudanças de requisitos das aplicações e a necessidade de fazer as atualizações em bancos de dados que atendem, simultaneamente, várias aplicações. Atualmente na literatura, existem soluções síncronas e assíncronas que utilizam refatorações para evoluir o banco de dados. Essas refatorações representam pequenas alterações estruturais, arquiteturais, de integridade e qualidade de dados que não incluem funcionalidades novas ao sistema. É muito comum a necessidade de implementação de uma grande alteração no banco de dados, a qual envolva o uso de um conjunto de refatorações programadas. Esse conjunto deve ser organizado de modo que, os requisitos de mudança dos usuários sejam atendidos e que o conjunto de restrições presentes no modelo de dados sejam satisfeitos. Atualmente técnicas de planejamento automatizado têm sido usadas para modelar problemas de planejamento de tarefas em diferentes domínios de problemas. Tais modelos possuem, de forma geral, suas ações com precondições e efeitos de suas transições e estados, restrições, recursos disponíveis e objetivos a serem atingidos O objetivo deste trabalho é utilizar técnicas de planejamento automatizado para representar o conjunto de refatorações e propor um modelo que inclua as fases para a conclusão de uma grande alteração no banco de dados.**

***Keywords*— Evolução de bancos de dados, Refatorações de banco de dados, Planejamento automático.**

I. INTRODUÇÃO

A EVOLUÇÃO de esquemas de bancos de dados tem sido reconhecida como um dos principais obstáculos que dificultam as atualizações em sistemas de informação[1]. Estudos mostram que essas dificuldades aumentaram com o uso de sistemas *Web*, nos quais a frequência de mudanças é muito grande e praticamente não há tolerância para erros ou indisponibilidade dos serviços.

Com a chegada das Metodologias Ágeis para Desenvolvimento de Software alguns conceitos das metodologias tradicionais foram substituídos por modelos iterativos e incrementais, que não precisam ter todo o modelo lógico e físico do banco de dados para o início do desenvolvimento do software[2]. Isso significa organizar o desenvolvimento do sistema em várias iterações de desenvolvimento e entregas ao cliente. Cada iteração contempla todas as fases já conhecidas nos modelos tradicionais como planejamento, especificação de requisitos, projeto, codificação e testes[3]. Essas iterações que normalmente duram em torno de duas semanas, têm como principais vantagens a detecção antecipada de erros de requisitos e de implementação e também a flexibilidade de mudar os requisitos no decorrer do projeto. Para atender a essas mudanças de requisitos é necessário que o banco de dados também tenha um desenvolvimento iterativo, ou seja, à medida que o software é desenvolvido, o banco de dados precisa ser alterado para acompanhar os novos requisitos. As alterações no esquema de banco de dados são chamadas de refatorações de bancos de dados.

Refatorar um banco de dados pode ser mais trabalhoso do que refatorar um código-fonte de uma aplicação, embora, segundo Martin Fowler[2]possa seguir as mesmas premissas. Tal afirmação baseia-se no fato de que é preciso se preocupar com todas as instâncias das aplicações que utilizam o banco de dados; realizar as alterações respeitando os requisitos dos usuários; preservar os dados existentes; não modificar a semântica do esquema atual e manter a integridade do banco de dados.

As técnicas de refatoração de código de Martin Fowler[2] foram adaptadas para bancos de dados por Ambler e Sadalage [1]e inseridas na literatura de Métodos Ágeis para Desenvolvimento de *Software*. Um refatoração de banco de dados foi definida como uma pequena mudança no esquema que melhora o projeto, não adiciona funcionalidades e não altera a semântica informacional ou comportamental do esquema.

Entretanto, nem todas as alterações se encaixam nessa definição de refatoração. É muito comum a necessidade de implementação de grandes alterações em bancos de dados. Isso implica escolher um conjunto de refatorações que represente o estado final desejado do esquema do banco de dados. Para a escolha desse conjunto de refatorações é necessário o conhecimento prévio do esquema atual do banco de dados, bem como suas limitações e objetivos. Também é necessária a escolha de um conjunto de tarefas que modifique o esquema sem negligenciar a integridade do banco de dados[4].

Neste trabalho foram utilizadas técnicas de planejamento automatizado, pararepresentar um conjunto de refatorações utilizado em uma grande alteração de banco de dados.Esta representação, foi adaptada dos trabalhos de Reiter [5],McCarth[6]e Nau et al. [7]. Ainda neste trabalho é proposto um modelo para o planejamento e análise de uma grandealteração de banco de dados.

II. REFATORAÇÃO DE BANCO DE DADOS

Segundo Ambler e Sadalage [1]uma refatoração em um banco de dados pode ser definida como uma alteração simples em seu esquema, com o objetivo de melhorar seu projeto, preservando sua semântica informacional e sua semântica

comportamental. Assim, para se realizar uma refatoração não é possível adicionar novas funcionalidades ou modificar alguma existente, não se pode alterar um dado e nem alterar o significado de um dado existente. Quando há necessidade de modificar ou adicionar algo existente é necessário aplicar o conceito de transformação, também definido por [1].

Em outras palavras, após a execução de uma refatoração ou um conjunto de refatorações no esquema de dados, o banco de dados deve responder às mesmas consultas que eram realizadas antes deste processo de refatoração.

Esta estratégia de refatoração é aplicada em ambientes complexos, nos quais existem várias aplicações acessando simultaneamente o mesmo banco de dados. Quando esse banco precisa ser alterado, praticamente todas as aplicações precisam sofrer algum tipo de adaptação, sendo que não é possível supor que todas essas aplicações serão alteradas ao mesmo momento [8]. Por esse motivo, foi proposto por [1]que se tenha um período de transição entre o início e o fim de uma refatoração, no qual o esquema antigo e o esquema novo coexistam, como mostrado na Fig. 1.

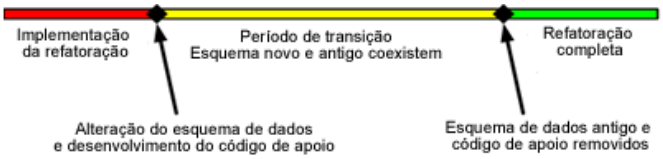


Figura 1. Ciclo de vida de uma refatoração.

Ambler e Sadalage [1]destacam alguns problemas no banco de dados que podem exigir refatorações como: tabelas ou colunas com múltiplas funções; tabelas com muitas colunas; e tabelas com colunas ou relacionamentos obsoletos.

Essas refatorações estão apresentadas na Tabela I, divididas em 4 grupos: estrutural, qualidade de dados, integridade referencial e arquitetural.

TABELA I. CATEGORIAS DE REFATORAÇÕES DE BANCO DE DADOS

Tipo	Aplicação
Estrutural	▪ Remover tabelas, colunas ou visões; ▪ Dividir ou unir tabelas ou coluna; ▪ Renomear tabelas, colunas ou visões.
Qualidade de Dados	▪ Introduzir restrição de coluna, valor padrão ou formato comum; ▪ Aplicar código ou tipo padrão; ▪ Remover restrição de coluna, valor padrão ou restrição de não nulo.
Integridade referencial	▪ Adicionar ou remover restrição de chave estrangeira; e ▪ Adicionar ou remover a exclusão em cascata.
Arquitetural	▪ Introduzir índice, tabela espelho, tabela somente para leitura; e ▪ Adicionar métodos (*Stored Procedures*).

III. PLANEJAMENTO AUTOMÁTICO

Um plano é uma sequência de ações para se atingir um objetivo. Planejamento é um processo de deliberação que escolhe e organiza ações para antecipar seus resultados esperados. Esta deliberação tem por objetivo atingir, da melhor forma possível, alguns objetivos conhecidos. Planejamento automatizado é uma área de Inteligência Artificial (IA) que estuda, computacionalmente, este processo de deliberação[7].

Este plano que segue ações apropriadas e o objetivo pode ser especificado de várias maneiras diferentes. A mais simples consiste na especificação de um estado objetivo (*goal state*) S_g, ou um conjunto de estados objetivo. Neste caso, o objetivo é alcançado por qualquer sequência de transições de estado que termina em um dos estados objetivo. Em outras palavras, o objetivo é satisfazer alguma condição sobre uma sequência de estados seguida pelo sistema. Outra alternativa consiste em indicar os objetivos como tarefas que o sistema deve executar, conforme está apresentado na Fig. 2. Estas tarefas podem ser definidas recursivamente como um conjunto de ações e outras tarefas.

Nau et al. [7] descreve esse modelo por meio da interação entre três principais componentes como mostrado na Fig. 2 e descritos a seguir.

Figura 2. Modelo Conceitual de um planejamento automático.

Planejador (*Planner*) recebe como entrada a descrição de um sistema Σ, uma situação inicial e objetivos que sintetizam o plano para o controlador executar e atingir o objetivo;

Controlador (*Controller*) recebe o estado atual (s) do sistema e providencia como saída uma ação, de acordo com o plano escolhido pelo planejador;

Sistema estado-transição Σ (*System* Σ) evolui de acordo com as ações e eventos que este recebe (um sistema estado-transição pode ser representado por um grafo direcionado onde os nós são estados e os arcos são ações ou eventos).

No modelo clássico de planejamento o sistema estado-transição pode ser definido formalmente como uma 4-tupla Σ=(S,A,E,γ), onde:

- $S = \{s_1, s_2, \ldots\}$ é o conjunto finito ou recursivo dos estados;
- $A = \{a_1, a_2, \ldots\}$ é o conjunto finito ou recursivo de ações;
- $E = \{e_1, e_2, \ldots\}$ é o conjunto finito ou recursivo dos eventos exógenos (não controlados pelo agente);
- $\gamma: S \times (A \cup E) \rightarrow 2^s$ é a função de estado-transição.

O sistema Σ da Fig. 2 pode ser representado por um grafo dirigido cujos nós são estados em *S*. Se $s' \in \gamma(s,u)$, onde *u* é o par (*a*, *e*), para todo $a \in A$ e $e \in E$, então o grafo contem um arco (chamado de estado transição) de *s* para s', rotulado com *u* como mostrado na Fig. 3.

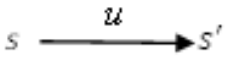


Figura 3. Estado transição de s para s'.

A ação *a* aplicada ao estado *s* leva a um outro estado $\gamma(s,a)$, assim o sistema evolui através de eventos e ações. A função de observação $\eta: S \rightarrow O$ faz o mapeamento de *S* em um conjunto discreto $O = \{o_{1,} o_{2,\dots}\}$ de possíveis observações. A entrada do controle é então a observação $o = \eta(s)$ que corresponde ao estado atual *s*, a entrada do planejador é a descrição de Σ, estado inicial $s_{0,} \in S$ e um estado objetivo e a saída do planejador produz um plano para guiar o controle.

Segundo Nau et al. [7]este modelo conceitual não pretende ser diretamente operacional. Pelo contrário, é usado como uma referência para as representações. Considera-se este modelo como um ponto inicial para avaliação de propriedades restritivas, particularmente as que seguem:

1. Σ **Finito:** O *sistema estado-transição* Σ possui um conjunto finito de estados $S = \{s_{1,} s_{2,\dots,} s_{k,}\}$para algum *k;*
2. Σ **Totalmente Observável:** O sistema Σ é totalmente observável, ou seja, conhece-se por completo o estado de Σ. Nesse caso a função η é a função identidade.
3. Σ **Determinístico:** O sistema Σ é determinístico, ou seja, cada ação ou evento tem apenas uma saída possível *u* em *A*∪*E*, |γ(*s*,*u*)| = 1;
4. Σ **Estático:** O sistema Σ é estático, isto é, o conjunto de *Eventos* exógenos E é vazio. Nenhuma mudança ocorre exceto aquelas efetuadas pelo controle
5. **Objetivos restritos:** Os planejadores lidam apenas com *objetivos restritos* que são especificados explicitamente no estado objetivo (*goal state*) s_g ou um conjunto de estados objetivo S_g. O objetivo é qualquersequência de estado-transição que termina em um dos estados objetivo. Os *objetivos estendidos*, tais como, estados a serem evitados, restrições na trajetória da solução ou funções de otimização, não são permitidos;
6. **Planos Sequenciais:** Um plano-solução de um problema de planejamento é uma sequência finita de ações ordenadas (a_1, a_2, *... a_k*);
7. **Tempo Implícito:** Ações e eventos não possuem duração. Elas são transições de estado instantâneas;
8. **Planejamento *Offline*:** O planejador não percebe mudanças do sistema Σ enquanto está planejando. Ele planeja apenas com as condições iniciais e os objetivos, independentemente da dinâmica que ocorre em Σ. O planejador não sabe o status da execução.

O planejamento clássico tem como suposições o conhecimento completo sobre um sistema determinístico, estático, estado-finito com satisfação de metas e tempo implícito. Segundo Nau et al. [7]o planejamento pode ser reduzido ao seguinte problema:

Dado$\Sigma = (S, A, \gamma)$, um estado inicial S_0, um conjunto de estados objetivo S_g encontre uma sequência de ações (a_1, a_2, *... a_k*) que corresponda a uma sequência de estados-transições ($s_{1,} s_{2,\dots,} s_{k,}$) tal que, $s_1 \in \gamma(s0, a1)$, $s_2 \in \gamma(s_1, a_2)$,..., $s_k \in \gamma(s_k–1, a_k)$ onde s_n pertença à S_g.

As restrições e suposições descritas acima pretendem tornar o planejamento apenas um processo de busca por um caminho no grafo de estados, o que é um problema bem conhecido e bem resolvido pela Ciência da Computação. De fato, se possuíssemos o grafo que representa Σ explicitamente não haveria muito planejamento a se fazer. Todavia, mesmo para um domínio de planejamento bem simples, o grafo de Σ pode ser tão grande que, representá-lo, seria inviável. Por esse motivo, é necessário representar o domínio com uma representação reduzida do sistema Σ facilitando a busca por uma solução.

Vários modelos interessantes são obtidos através do relaxamento das suposições restritivas do modelo clássico. Esse relaxamento fez com que os planejadores fossem capazes de lidar com problemas mais complexos, e esse novo modelo foi chamado de Planejamento Neoclássico [7].

O avanço das técnicas de Planejamento em Inteligência Artificial fez com que a Engenharia de Requisitos e a Engenharia do Conhecimento pudessem ser usadas como importantes ferramentas para projeto de engenharia (*Engineering Design*). Esta crescente área da IA está presente em cenários como: planejamento de trajetória e movimentação de sistemas automáticos móveis; planejamento de percepção envolvendo ações de sensoriamento para captação de informação do ambiente; planejamento de navegação que combina sensoriamento e definições de trajetórias; planejamento de manipulação relacionado com movimentação de objetos como, por exemplo, montagem de peças, organização de containers, gestão de processos de negócios e aplicações de *Business Intelligence* [7] e [9].

Uma entrada necessária para qualquer algoritmo de planejamento é uma descrição do problema para ser resolvido. Na prática, seria impossível para esta descrição do problema incluir uma enumeração explícita de todos os possíveis estados e transições de estado. A descrição do problema seria excessivamente grande e mais trabalhosa que resolver o próprio problema de planejamento, assim é mais fácilcomputá-los e descreve-los quando forem necessários[7]. Para o Planejamento Automático, o mais importante é a representação e modelagem dos domínios, ou seja, a representação do sistema Σ. Essa representação deve ser feita em alguma linguagem que possa ser compreendida por um planejador.

IV. PLANEJAMENTO AUTOMÁTICO PARA REFATORAÇÃO DE BANCOS DE DADOS

Os trabalhos de Ambler e Sadalage [1] e Domingues [10], apresentaram como alternativas o uso de pequenas mudanças no esquema do banco de dados chamadas de refatorações. Ambos os trabalhos não consideram a execução de um conjunto de refatorações em um mesmo esquema de dados. Essa situação é bastante comum em ambientes reais, já que os requisitos dos usuários, muitas vezes, envolvem diversas funcionalidades do sistema. Ambler e Sadalage [1]sugeririamo uso das refatorações bem como as tarefas de cada uma delas, mas não estudaram como seria a execução de várias refatorações para se atingir um estado final do esquema do banco de dados.

A proposta deste trabalho é representar uma grande alteração no esquema do banco de dados, por meio de um conjunto de refatorações que expressem os desejos de mudança dos usuários. As expectativas dos usuários são diferentes, dependendo do envolvimento deles com o desenvolvimento do projeto. Um usuário desenvolvedor tem um ponto de vista de requisito diferente do usuário que administra o banco de dados e do usuário que gerencia o projeto.

Esses diferentes pontos de vista podem dificultar a fase de levantamento de requisitos e implicar em mudanças equivocadas no modelo do banco de dados. Para esses casos, os diagramas de UML representam uma boa solução para resolver dúvidas sobre os requisitos. Os estados iniciais, do esquema de banco de dados, e final, após as mudanças são conhecidos por todos os usuários envolvidos. Deste modo, entre o estado inicial e final do esquema de dados deve ser implementado um conjunto de refatorações que atinjam o estado final após a grande alteração no esquema de dados, como apresentado na Fig. 4.

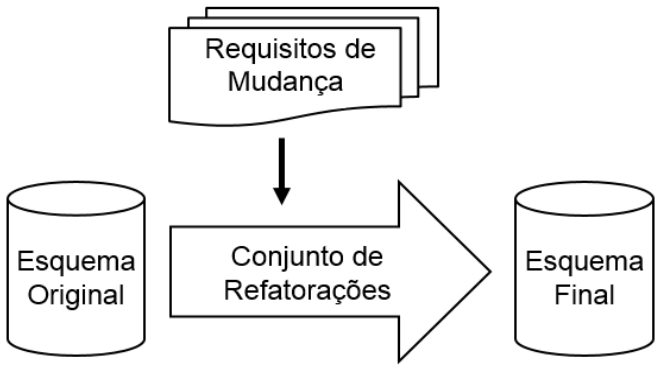


Figura 4. Modelo para evolução do esquema original para o esquema final do banco de dados.

O conjunto de refatorações deve ser escolhido utilizando o catálogo de Ambler e Sadalage [1]. Embora todos os usuários conheçam o estado final do esquema de dados, somente o administrador do banco de dados (DBA) irá se preocupar com o planejamento dessa mudança, ou seja, quais scripts serão criados e qual a ordem que eles serão executados para que se atinja o esquema final.

As refatorações de um conjunto podem ser arranjadas e combinadas de formas diferentes para que atinjam o esquema final, mas que tenham tempos diferentes para o término da grande alteração.

O conjunto de refatorações deve ser executado considerando todas as restrições impostas pelo modelo do banco de dados, ou seja, nem todo arranjo ou combinação das refatorações pode levar ao estado final esperado.

Considerando que um conjunto de refatorações seja um conjunto de ações a serem executadas no esquema inicial do banco de dados, e que a cada refatoração executada, aqui chamada de ação, o esquema de dados atinge um estado diferente, propõe-se neste trabalho representar uma grande alteração no banco de dados como sistema estado-transição definido por pelo sistema $\Sigma = (S, A, \gamma)$, onde:

- $S = \{s_{1,} s_{2,\ldots}\}$ é o conjunto finito de estados do esquema do banco de dados;
- $A = \{a_{1,} a_{2,\ldots}\}$ é o conjunto finito de ações (refatorações);

Dado $\Sigma = (S, A, \gamma)$, um estado inicial S_0, um conjunto de estados objetivo S_g deve-se encontrar uma sequência de ações (a_1, a_2, ... a_k) que corresponda a uma sequência de estados-transições ($s_{1,} s_{2,\ldots,} s_{k,}$) tal que, $s_1 \in \gamma(s0, a1)$, $s_2 \in \gamma(s_1, a_2)$,..., $s_k \in \gamma(s_k{-}1, a_k)$ onde s_n pertença à S_g. Tal como proposto por Nau et al. [7]. No sistema Σ de uma grande alteração ainda é necessário estudar o conjunto de restrições que envolvem cada ação do planejamento.

Para o Sistema estado-transição Σ de uma grande alteração pode-se relacionar as propriedades restritivas do modelo clássico de Nau et al. [7]. Assim o sistema Σ é:

- Finito, pois o conjunto de refatorações é conhecido;
- Totalmente observável;
- Determinístico, pois a cada ação (refatoração) executada existe apenas uma saída possível, ou seja, existe um único estado possível para o banco de dados após a execução da refatoração.
- Estático: Nenhuma mudança ocorre exceto aquelas executadas pelo controle;
- Objetivos restritos: O objetivo é qualquer sequência de estado-transição que leve a um estado final.
- Planos sequenciais;
- O tempo pode ser um parâmetro para execução das ações;
- Planejador off-line;

É necessário criar um plano de conhecimento do domínio, considerando o sistema estado-transição proposto, em linguagem PDDL(*Planning Domain Definition Language)*descrita porMcDermott [11].Nos estudos preliminares deste trabalho, identificou-se a necessidade de uma notação gráfica e uma notação formal, para intermediar os requisitos de uma grande alteração e a execução do seu plano de execução.

Na literatura são encontradas algumas técnicas de levantamento de requisitos que envolvem desde a criação,

representação e análise por diagramas esquemáticos de UML (*Unified Modeling Language*) e Redes de Petri.É possível combinar um método semiformal como diagramas de UML e métodos formais como representação em Redes de Petri. O diagrama de caso de uso de UML define uma visão que constitui um modelo funcional do sistema a partir da perspectiva do usuário [12]. Ambiguidades, contradições, omissões e imprecisões são frequentemente encontradas utilizando esse diagrama. Uma estrutura bem definida de casos de uso serve como base para futuras evoluções do banco de dados.

Em Domingues [13]as tarefas das refatorações foram descritas em Redes de Petri[14]. Nesse trabalho, também foi mostrado como é possível combinar as tarefas de cada uma delas, utilizando uma notação formal. Neste artigo, a pesquisa avançou com o objetivo de formalizar as refatorações utilizando o planejamento automático e propor um modelo para automatizar o plano de uma grande alteração em um banco de dados. Na Fig. 5 éapresentado um diagrama para o estudo do planejamento de uma grande alteração.

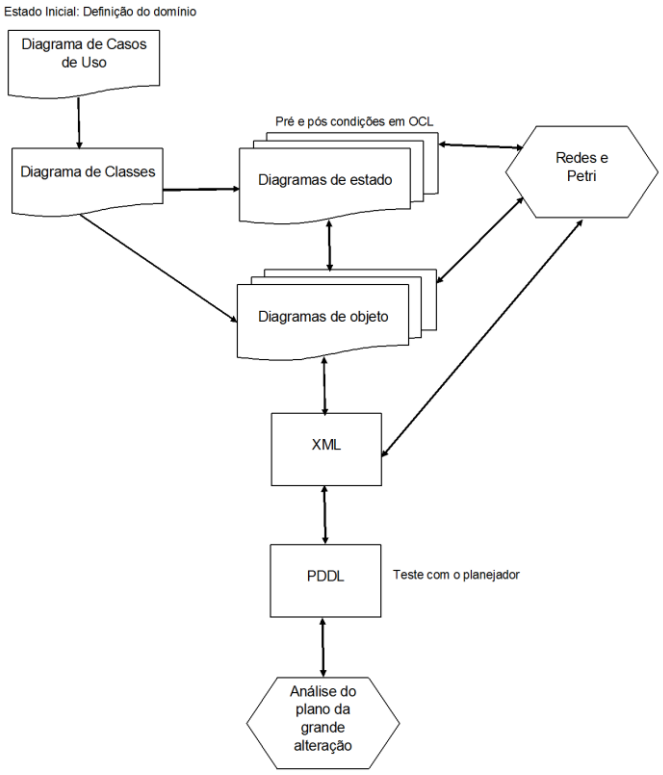


Figura 5. Modelo para o planejamento de uma grande alteração em um banco de dados

Nesse planejamento são esperados como entradas os diagramas casos de uso e de classes para representar o estado inicial do esquema de dados e os requisitos do usuário. Para traçar o plano de ações é necessária uma tradução dos diagramas UML, tais como diagramas de classes, objetos e sequencia, para linguagem PDDL que é interpretada por softwares planejadores. As pré e pós-condições são traduzidas para ações OCL (*Object Constraint Language*), a partir dos diagramas e estado de cada classe. Com o plano escrito em linguagem PDDL é possível a conversão para XML e posteriormente para Redes de Petri, na qual é possível realizar uma análise dinâmica do modelo e também das dependências do sistema.

Com a análise do plano das refatorações foi possível validar o plano de ações proposto pelos planejadores com os requisitos iniciais do usuário. Concluída essa etapa, foi possível traduzir as ações de saída do planejador para SQL (*Structured Query Language*). Desde modo, a solução proposta para planejamento de uma grande alteração pode ser resumida como segue na Fig. 6.

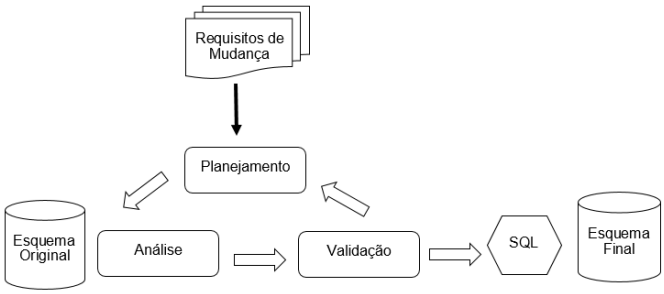


Figura 6. Modelo de transformação para SQL

Para Ambler e Sadalage [1]as alterações do modelo que não se encaixam no conceito de refatoração, por não modificarem ou adicionarem funcionalidades são chamadas de transformações. Assim a Fig. 4 é completada na Fig. 7 com o conceito de transformações para abranger qualquer alteração que esteja nos requisitos dos usuários.

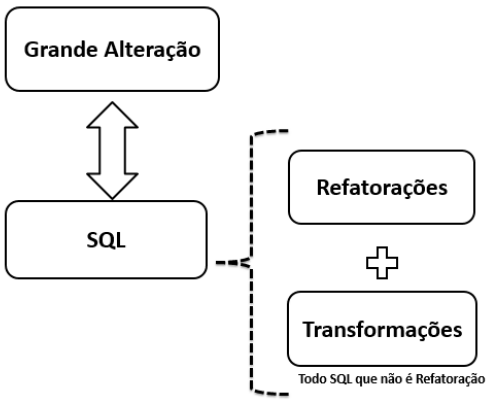


Figura 7. Modelo abrangendo as transformações

V. APLICAÇÃO DA PROPOSTA

A Fig. 8 mostra um exemplo de uma refatoração na qual será renomeada a coluna Pnome da Tabela Cliente para PrimeiroNome. Esse procedimento exige que sejam seguidos os passosda Fig. 9.

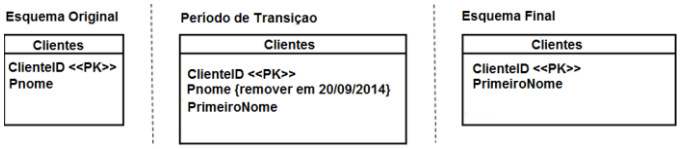


Figura 8. Exemplo de refatoração renomear coluna

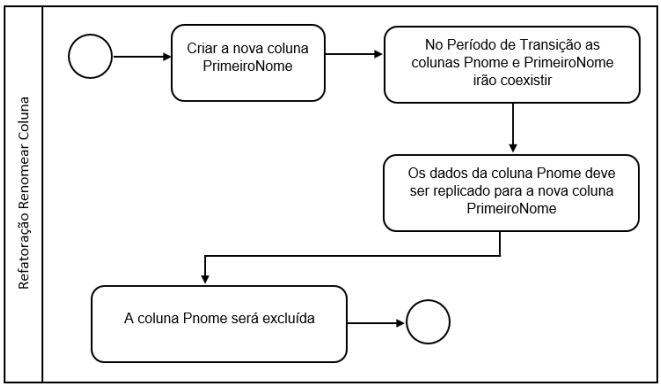


Figura 9. Passos para a refatoração renomear coluna

A refatoração Renomear Coluna está dentro do contexto de uma grande alteração (Fig. 4). O processo de uma grande alteração foi representado em um diagrama de classes conforme a Fig. 11. Neste diagrama conceitual todas as tabelas do modelo do banco de dados são representadas pela entidade "Tabela" e estão em relacionamentos com cardinalidade 1:N com colunas. Cada coluna, além dos atributos comuns como tipo de dados, possuem ainda atributos booleanos apenas para controle: excluída, quando a coluna não pertence mais à tabela; backup, quando os dados da coluna já foram replicados para a nova coluna e PeriodoTransicao, quando a coluna nova e a coluna antiga coexistem para manter a consistência do banco de dados. No diagrama de classe da Fig. 10 é representada a classe Refatoração Remover Coluna. Essa Classe possui métodos para realizar os passos descritos na Tabela I e herdam atributos e métodos da interface "Grande Alteração" como mostrado na Fig. 11.

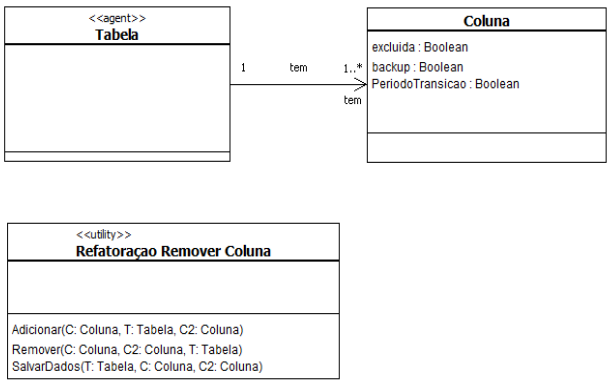


Figura 10. Exemplo de diagrama de classes conceitual para a Refatoração Remover Colunas

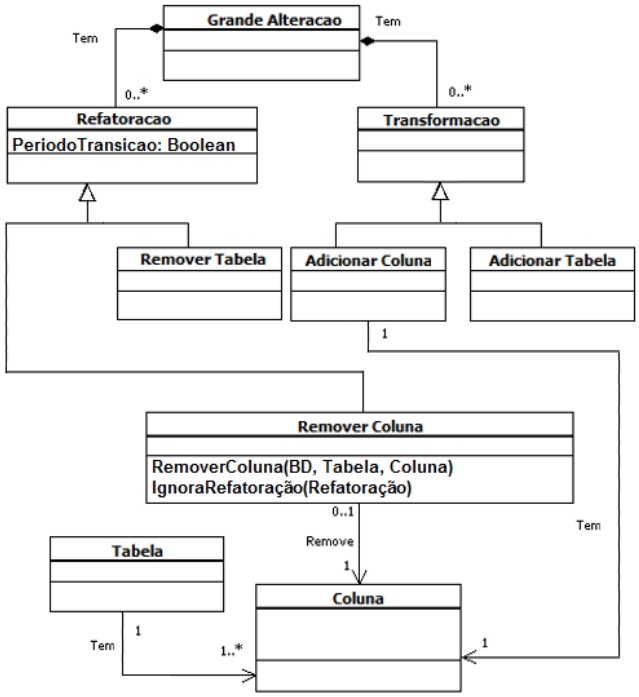


Figura 11. Diagrama de Classes: Conceitual para uma grande alteração

Na próxima fase é necesário instanciar os objetos reais do modelo do banco de dados, ou seja, instanciar a tabela "Cliente" e dois objetos da classe "Coluna": Coluna1 e Coluna2. Coluna1 representa a coluna existente Pnome que deverá ser substituída. Coluna2 representa a nova coluna PrimeiroNome. Ambas possuem os mesmos atributos, por se tratarem de instâncias da mesma classe. A Fig. 12 mostra estas instâncias e também o relacionamento já existente entre o objeto "Cliente" (tabela) e o objeto "Pnome" (coluna).

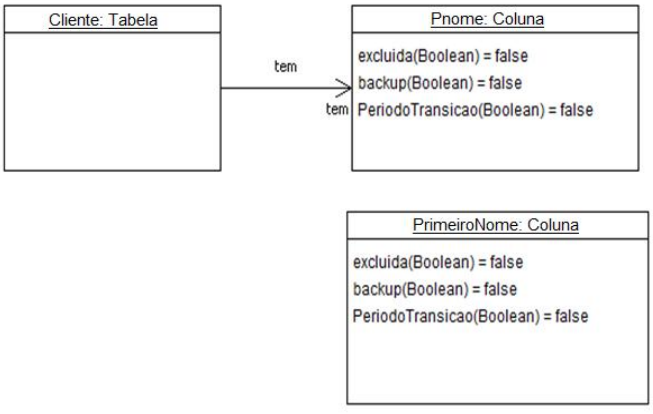


Figura 12. Instância dos objetos do para representar as tabelas do modelo do banco de dados.

Outra forma de representar os passos da Fig. 9 é utilizar o um diagrama de estados. Este diagrama além de poder ser utilizado para simular os passos, pode ser traduzido para linguagem de planejadores automáticos como a PDDL. A Fig. 13 apresenta os estados para conferência da Coluna 1 já

existente (Pnome), a entrada no período de transição onde a Coluna 1 e Coluna 2 (PrimeiroNome) coexistem no modelo e o último estado no qual a Coluna 1 é excluída do modelo do banco de dados. Neste caso os atributos do objeto Pnome do diagrama da Fig. 12 são atualizados quando o processo entra no período de transição, no qual os dados precisam ser replicados para a nova coluna (backup = true) e ao término o valor do atributo Excluída é alterado para true para representar a exclusão física da coluna Pnome do modelo do banco de dados, como é mostrado na Fig. 13.

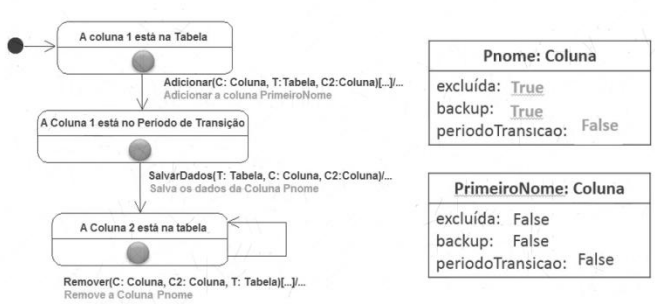


Figura 13. Diagrama de estados para remoção física da coluna Pnome do modelo do banco de dados

Após excluir a coluna Pnome é necessário refazer o relacionamento do diagrama conceitual da Fig. 10, para que o objeto "Cliente" (Tabela) se relacione agora como o objeto "PrimeiroNome" (Coluna) como é mostrado na Fig. 14.

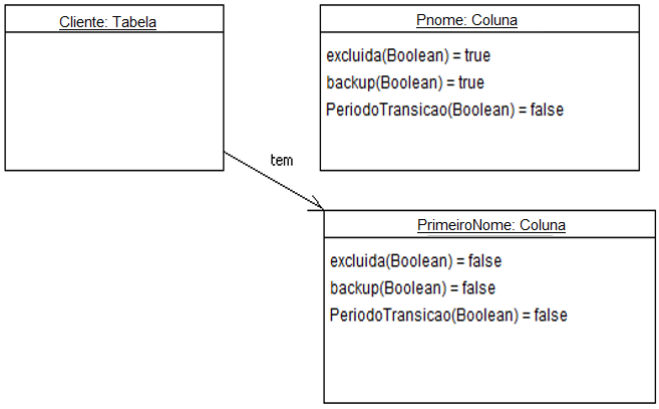
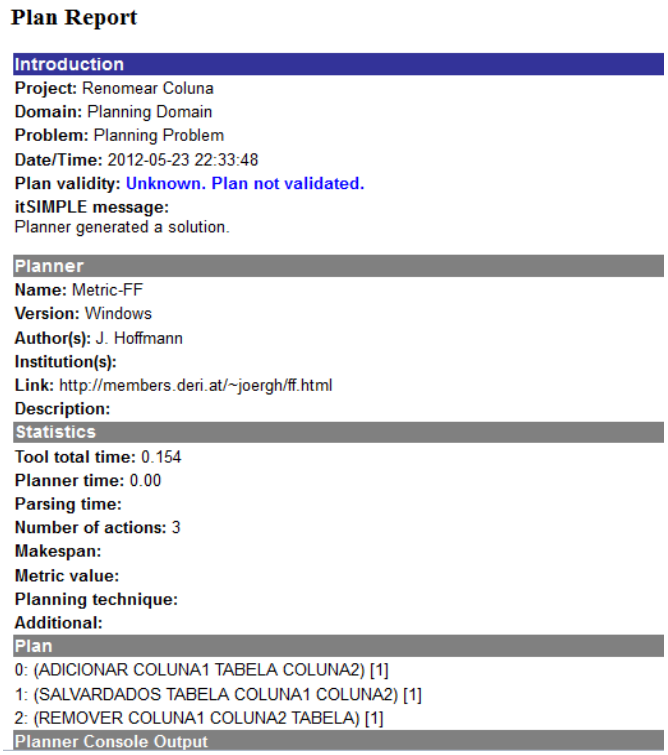


Figura 14.Instância dos objetos do para representar o relacionamento com a nova coluna criada.

O diagrama de estados da Fig. 14 foi traduzido para linguagem PDDL para ser executado por um aplicativo planejador. Como sugestão fizemos os testes com o planejador Metric-FF [16]. A seguir é apresentada a implementação em PDDL da refatoração Remover Coluna, como parte do processo de uma grande alteração, descrito na Fig. 11. A Fig. 15 mostra a execução do plano em PDDL para a refatoração Remover Coluna utilizando o planejador Metric-FF.

```
(define (domain Renomear_Coluna)
    (:requirements :typing :negative-
    preconditions)
(:types
      Tabela - object
      Coluna - object
 )
    (:predicates
      (tem ?tab - Tabela ?col - Coluna)
      (excluida ?col - Coluna)
      (backup ?col - Coluna)
      (PeriodoTransicao ?col - Coluna)
    )
    (:action Adicionar
     :parameters (?C - Coluna ?T - Tabela
     ?C2 - Coluna)
     :precondition
       (and
         (not (excluida ?C))
         (not (backup ?C))
(tem ?T ?C)
       )
     :effect
       (and
         (tem ?T ?C2)
         (not (excluida ?C2))
         (not (backup ?C2))
         (not (tem ?T ?C))
       )
 )

(:action SalvarDados
     :parameters (?T - Tabela ?C - Coluna
     ?C2 - Coluna)
:effect
       (and
         (backup ?C)
         (PeriodoTransicao ?C)
       )
 )
(:action Remover
     :parameters (?C - Coluna ?C2 -
     Coluna ?T - Tabela)
:precondition
       (and
(tem ?T ?C2)
         (backup ?C)
)
     :effect
       (and
         (excluida ?C)
         (not (tem ?T ?C))
         (not (PeriodoTransicao ?C))
)
 )
```



Plan Report

Introduction
Project: Renomear Coluna
Domain: Planning Domain
Problem: Planning Problem
Date/Time: 2012-05-23 22:33:48
Plan validity: Unknown. Plan not validated.
itSIMPLE message:
Planner generated a solution.

Planner
Name: Metric-FF
Version: Windows
Author(s): J. Hoffmann
Institution(s):
Link: http://members.deri.at/~joergh/ff.html
Description:

Statistics
Tool total time: 0.154
Planner time: 0.00
Parsing time:
Number of actions: 3
Makespan:
Metric value:
Planning technique:
Additional:

Plan
0: (ADICIONAR COLUNA1 TABELA COLUNA2) [1]
1: (SALVARDADOS TABELA COLUNA1 COLUNA2) [1]
2: (REMOVER COLUNA1 COLUNA2 TABELA) [1]

Planner Console Output

Figura 15. Execução do plano para a Refatoração Remover Coluna utilizando o planejador Metric-FF.

CONCLUSÕES

O trabalho inicial do administrador de banco de dados é coletar os requisitos de alterações dos usuários, identificar os problemas de modelagem ou que podem influenciar o desempenho de consultas. Esse conjunto de alterações no esquema de banco de dados é a entrada do processo refatoração que leva o esquema do banco de dados do modelo original até o final.Este trabalho teve como objetivo apresentar uma nova forma de realizar refatorações de banco de dados utilizando técnicas de planejamento automático. Estas técnicas incluem atividades multidiciplinares de diagramação das tabelas físicas do modelo do banco de dados em UML, com o objetivo de conhecer os passos e restrições existentes para se realizar uma refatoração no modelo do banco de dados. O presente trabalho apresentou testes em linguagem PDDL que no futuro podem ser utilizadas para o desenvolviimento de uma ferramenta de interface amigável para o usuários de banco de dados planejarem e documentarem as etapas de uma grande alteração de banco de dados.

REFERÊNCIAS

[1] S. W. Ambler and P. J. Sadalage, *Refactoring databases: evolutionary database design*. New York: Addison-Wesley Professional, 2006.

[2] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas, “Manifesto for Agile Software Development,” 2014. [Online]. Available: http://www.agilemanifesto.org/. [Accessed: 07-Jan-2014].

[3] R. S. Pressman, *Software engineering: a practitioner’s approach*, 7th ed. New York: McGraw-Hill, 2010.

[4] M. B. P. Domingues, J. R. de Almeida Júnior, W. F. Costa, and A. M. Saraiva, “Refactoring database to improve queries performance,” in *In: 21th Italian Symposium on Advanced Database Systems (SEBD 2013)*, 2013.

[5] R. Reiter, “Formalizing Database Evolution in the Situation Calculus,” in *FGCS*, 1992, pp. 600–609.

[6] J. McCarthy, “Programs with Common Sense,” in *Semantic Information Processing*, 1968, pp. 403–418.

[7] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: theory & Practice (The Morgan Kaufmann Series in Artificial Intelligence)*. San Francisco: Morgan Kaufmann, 2004, p. 635.

[8] H. H. Domingues, F. Kon, and J. E. Ferreira, “Replicação assíncrona em modelagem evolutiva de banco de dados,” in *In: SBBD*, 2009, pp. 121–135.

[9] S. Biundo, R. Aylett, M. Beetz, D. Borrajo, A. Cesta, T. Grant, L. McCluskey, A. Milani, and G. Verfaille, *Technological Roadmap on AI Planning and Scheduling*. PLANET, 2003.

[10] H. H. Domingues, F. Kon, and J. E. Ferreira, “Asynchronous replication for evolutionary database development: a design for the experimental assessment of a novel approach,” in *In: Proceedings of the 2011th Confederated international conference on On the move to meaningful internet systems - Volume Part II*, 2011, pp. 818–825.

[11] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, “{PDDL} -- {T}he {P}lanning {D}omain {D}efinition {L}anguage -- {V}ersion 1.2,” 1998.

[12] D. Xu and X. He, “Generation of test requirements from aspectual use cases,” in *Proceedings of the 3rd workshop on Testing aspect-oriented programs*, 2007, pp. 17–22.

[13] M. B. P. Domingues, J. R. de Almeida Júnior, and J. R. Silva, “Evolution of Databases Using Petri Nets,” in *In: CBA - Congresso Brasileiro de Automática*, 2012.

[14] T. Murata, “Petri nets: Properties, analysis and applications,” *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[15] S. Ambler, *Agile modeling: effective practices for eXtreme programming and the unified process*. New York, NY: John Wiley & Sons, 2002.

[16] B. Nebel, “The FF Planning System: Fast Plan Generation Through Heuristic Search,” *J. Artif. Intell. Researc*, vol. 14, pp. 253–302, 2001.

Márcia Beatriz Pereira Domingues: é graduada em Bacharelado em Ciência da Computação pelasFaculdades Adamantinenses Integradas (FAI), Adamantina, São Paulo, Brasil, em 2002. Obteve o título de mestre em Engenharia Elétrica pela Universidade Estadual Paulista (Unesp), Ilha Solteira, São Paulo, Brasil, em 2008 e de Doutora, em Engenharia da Computação pela Universidade de São Paulo (USP), São Paulo, São Paulo, Brasil, em 2014. Atualmente é pos-doutoranda na Universidade de São Paulo (USP) e suas pesquisas se concentram na área de Engenharia de Software, com ênfase em Bancos de Dados e desenvolvimento de sistemas, sistemas de informação e sistemas de apoio a decisão.

Jorge Rady de Almeida Jr.: Possui mestrado em Engenharia Elétrica pela Escola Politécnica da Universidade de São Paulo (1990) e doutorado também em Engenharia Elétrica pela Escola Politécnica da Universidade de São Paulo (1995). É Professor Associado do Departamento de Engenharia de Computação e Sistemas Digitais (PCS) da Escola Politécnica da Universidade de São Paulo desde 2003. Suas áreas de pesquisa são o Desenvolvimento e Avaliação de Sistemas Críticos, com ênfase nos aspectos de tempo real e a área de Engenharia de Software, com ênfase em Bancos de Dados, Data Warehouse e Data Mining, além da segurança em Sistemas de Informação.