Pós-Graduação em Ciência da Computação

Gênesis Jeferson Ferreira Pereira de Lima

Uma Linguagem de Modelagem para Refatoração Estrutural de Banco de Dados Relacionais

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
http://cin.ufpe.br/~posgraduacao

Recife
2018

Gênesis Jeferson Ferreira Pereira de Lima

Uma Linguagem de Modelagem para Refatoração Estrutural de Banco de Dados Relacionais

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Bancos de Dados
Orientador: Robson do Nascimento Fidalgo

Recife
2018

Catalogação na fonte
Bibliotecária Monick Raquel Silvestre da S. Portes, CRB4-1217

L732l Lima, Gênesis Jeferson Ferreira Pereira de
Uma linguagem de modelagem para refatoração estrutural de banco de dados relacionais / Gênesis Jeferson Ferreira Pereira de Lima. – 2018.
174 f.: il., fig., tab.

Orientador: Robson do Nascimento Fidalgo.
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação. Recife, 2018.
Inclui referências e apêndices

1. Banco de dados. 2. Banco de dados relacional. I. Fidalgo, Robson do Nascimento. (orientador) II. Título.

025.04 CDD (23. ed.) UFPE-MEI 2018-126

Gênesis Jeferson Ferreira Pereira de Lima

Uma Linguagem de Modelagem para Refatoração Estrutural de Banco de Dados Relacionais

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 09/03/2018

BANCA EXAMINADORA

_____

Prof. Dr. Vinicius Cardoso Garcia
Centro de Informática/UFPE

_____

Prof. Dr. João Henrique Correia Pimentel
Unidade Acadêmica de Cabo de Santo Agostinho/UFRPE

_____

Prof. Dr. Robson do Nascimento Fidalgo
Centro de Informática/UFPE
(**Orientador**)

A Deus, minha família e meus amigos.

AGRADECIMENTOS

Este trabalho não seria possível sem o suporte daqueles que me ajudaram nesta jornada. Gostaria de agradecer e dedicar este trabalho aos que seguem:

Meu orientador, Robson Fidalgo, que possui um nível de conhecimento motivacional e muita paciência, além de ser um professor e orientador excepcional.

Aos colegas que fiz ao longo do percurso por meio das discussões entusiasmadas e compartilhamento de conhecimento. Sobretudo, meu colega de pesquisa Edson Alves pelo compartilhamento da sua experiência em pesquisas cientificas.

A todos os professores do Centro de Informática da Universidade Federal de Pernambuco, que ministraram aulas fantásticas, repassando uma boa carga de conhecimento e sabedoria.

À minha família e amigos. Em especial ao meu filho, minha esposa e minha mãe por todo apoio.

RESUMO

Refatoração é uma técnica utilizada para melhorar o código de um software ou de um banco de dados. No contexto de banco de dados relacionais, a literatura aponta 64 padrões de refatoração categorizados em: Estrutural (17), Qualidade de Dados (13), Integridade Referencial (7), Arquitetural (12) e de Métodos (15). Por cobrirem as operações de refatoração mais frequentes, os padrões estruturais são os mais utilizados. Contudo, considerando que o esquema relacional é normalmente modelado como um diagrama, o levantamento do estado da arte mostra que os trabalhos relacionados não oferecem um suporte diagramático adequado para cobrir a maioria desses padrões. Por exemplo, nenhum trabalho permite remover visão, adicionar campo derivado ou chave substituta ou dividir coluna. O objetivo desse trabalho é propor uma Linguagem de Modelagem Específica de Domínio (do inglês, Domain Specific Modeling Language - DSML) que ofereça um suporte diagramático para especificar refatorações estruturais em banco de dados relacionais.O desenvolvimento desta proposta inicia com o levantamento do estado da arte (i.e., fundamentação teórica e trabalhos relacionados), o que permite entender os principais conceitos relacionados, bem como as necessidades existentes no contexto de refatoração de banco de dados. Na sequencia, apresenta-se o método utilizado para especificar a DSML proposta e a sua Ferramenta CASE (Computer Aided Software Engineering). Por fim, de modo a avaliar o trabalho proposto, a ferramenta CASE desenvolvida é utilizada para realizar uma avaliação sistematizada da expressividade das refatorações estruturais em um banco de dados relacional. Como resultados tem-se a definição da sintaxe concreta (notação), sintaxe abstrata (metamodelo) e semântica estática (regras de boa formação) da DSML proposta, bem como a implementação da sua ferramenta CASE. Além disso, mostra-se que o trabalho proposto oferece uma melhor cobertura diagramática em relação aos trabalhos relacionados.

Palavras-chaves: Evolução e Refatoração de Banco de Dados. Banco de dados relacional. Linguagem de Modelagem de Domínio Específico. Computer-Aided Software Engineering.

ABSTRACT

Refactoring is a technique applied to improve both software code and the objects of a database. In the relational database context, the literature points out 64 refactoring patterns categorized in: Structural (17), Data Quality (13), Referential Integrity (7), Architectural (12) and Methods (15). Since the structural patterns covers the most frequent refactoring operations, they are the most used. However, the state of the art survey shows that the related works does not provide adequate support to cover most of these patterns. Since the relational schema is usually modeled by diagram, this work aims to provide a Domain Specific Modeling Language (DSML) that delivers better support for specifying structural refactorings in relational databases. The development of this proposal starts with a survey of the state of the art (i.e., theoretical foundation and related works), which bring to light the key concepts as well as the existing needs in the field of database refactoring. Next, the method used to specify the proposed DSML and its Computer Aided Software Engineering (CASE) tool are shown. Finally, in order to evaluate the proposed work, the developed CASE tool is used to perform a systematic expressiveness evaluation of the structural refactorings in a relational database. The results of the proposed DSML concrete syntax (notation), abstract syntax (metamodel) and static semantics (well-formedness rules) are introduced, as well as the implementation of its CASE tool. In addition, it is shown that the proposed work offers a better coverage than findings.

Key-words: Database Evolution and Refactoring. Relational Database. Domain-Specific Modeling Language. Computer-Aided Software Engineering.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CASE	*Computer-Aided Software Engineering*
DBA	*Database Admnistrator*
DMP	*Database Modification Primitive*
DS	*Design Science*
DSML	*Domain-Specific Modeling Language*
EGL	*Eclipse Generation Language*
EMF	*Eclipse Modeling Framework*
EOL	*Eclipse Object Language*
EVL	*Eclipse Validation Language*
GMF	*Graphical Modeling Framework*
LOB	*Large Object*
M2M	*Model To Model*
M2T	*Model To Text*
MDD	*Model Driven Development*
MDE	*Model Driven Engineering*
MOF	*Meta-Object Facility*
OCL	*Object Constraint Language*
PRISM	*Panta Rhei Information Schema Manager*
RMM	*Relational Metamodel*
SEB	*Schema Evolution Benchmark*

SUMÁRIO

1 INTRODUÇÃO

Neste capítulo apresenta-se o contexto desta pesquisa, junto com o problema e a motivação da realização deste trabalho. Além disso, detalham-se os objetivos, a delimitação do escopo e a metodologia da pesquisa. Por fim, destaca-se a estrutura da dissertação.

1.1 CONTEXTUALIZAÇÃO

O desenvolvimento evolucionário de dados é uma abordagem que promove a evolução do modelo de dados de acordo com as mudanças ocorridas na aplicação ((AMBLER SCOTT W. E SADALAGE, 2006)). Dentre as técnicas evolucionárias de dados destaca-se a refatoração de banco de dados, que consiste em aplicar mudanças simples no esquema de um banco de dados visando melhorar sua estrutura. Além disso, a refatoração também visa manter a semântica informacional do esquema. Com isto, pode-se garantir a consistência dos resultados das consultas, sobretudo após as modificações do esquema. A semântica informacional se sustenta nos seguintes mecanismos: 1) Migração de Dados, que define como os dados (e.g., coluna ou tuplas) devem ser migrados ao aplicar a refatoração; e 2) Período Transicional, que especifica um meio de sincronização (e.g., gatilho ou visão) entre os componentes (e.g., colunas, restrições e/ou tabelas) que fazem parte da refatoração e buscam garantir a integridade e disponibilidade dos dados durante um período preestabelecido. Ambler Scott W. e Sadalage (2006) definem 64 padrões de refatoração categorizados em estruturais, qualidade de dados, integridade referencial, arquiteturais e comportamentais (ou de métodos). Dentre estes destacam-se as refatorações estruturais, pois estas são realizadas com maior frequência (QIU; LI; SU, 2013) (cf. Seção 2.1).

Refatorar um bancos de dados é uma tarefa não trivial (AMBLER SCOTT W. E SADALAGE, 2006). No entanto, por meio do desenvolvimento orientado a modelos, do inglês *Model Driven Development* (MDD) (BRAMBILLA; CABOT; WIMMER, 2012), pode-se definir representações diagramáticas para as refatorações e abstrair a complexidade da implementação. O paradigma MDD utiliza o modelo como seu principal artefato de construção e torna possível abstrair os conceitos do domínio no intuito de criar facilidades para geração automática de código interpretável e/ou executável (BRAMBILLA; CABOT; WIMMER, 2012). MDD torna possível aumentar a produtividade, representar e validar o modelo, reduzir a probabilidade de erros e aumentar a qualidade dos artefatos gerados. Além disso, esta abordagem também promove o baixo custo de produção, pois o produto da execução de um modelo (i.e., código ou modelo gerado) tende a ser mais barato e trazer mais benefícios que o custo de construção por meios tradicionais (i.e., codificação escrita) (SELIC, 2003) (PICEK; STRAHONJA, 2007).

Uma das formas de beneficiar-se do paradigma MDD é o desenvolvimento de uma

linguagem de modelagem de domínio específico (do inglês, *Domain-Specific Modeling Language* – DSML). Uma DSML é especificada a partir de uma sintaxe abstrata (i.e., metamodelo), uma sintaxe concreta (i.e., notação) e uma semântica estática (regras de boa formação) (KELLY; TOLVANEN, 2008). Além disso, as DSML são implementadas por ferramentas do tipo *Computer-Aided Software Engineering* (CASE), as quais fornecem um ambiente para representar diagramaticamente os conceitos de um determinado domínio. Assim, torna-se possível que profissionais e estudantes possam gerar soluções de forma automática a partir dos modelos, principalmente nos casos onde não se tem domínio sobre os detalhes técnicos de implementação (e.g., codificação).

1.2 PROBLEMA E MOTIVAÇÃO

Bancos de dados evoluem com frequência, expondo uma série de mudanças que podem ocorrer durante todo o ciclo de vida de uma aplicação. Efetuar modificações no esquema demanda a execução de atividades complexas que podem ser realizadas de forma equivocada tanto pelos especialistas em bancos de dados (e.g., *Database Admnistrator* (DBA)) quanto por outros profissionais (e.g., Desenvolvedor). Como resultado, isto pode gerar inconsistências ou até mesmo perda de dados ((QIU; LI; SU, 2013)).

No sentido de prover uma categorização para as transformações sofridas pelo esquema durante seu ciclo de vida, Ambler Scott W. e Sadalage (2006) sumarizam as evoluções em padrões de refatoração. Diversos trabalhos acadêmicos e ferramentas CASE permitem a modelagem, engenharia reversa e engenharia direta de representações visuais de um banco de dados (e.g., DBVis[1], Visual Paradigm[2] e MySQL Workbench[3]). No entanto, quando se trata das refatorações, especialmente as estruturais (as mais frequentemente aplicadas), a revisão da literatura realizada nesta proposta não encontrou nenhum trabalho acadêmico ou ferramenta que apresentasse uma linguagem de modelagem de domínio específico para impedir erros sintáticos ou semânticos para refatoração estrutural de banco de dados. Em suma, não foram encontradas propostas científicas ou tecnológicas que, de forma diagramática, auxiliassem na geração do código *SQL*, migração de dados, período transicional e na engenharia direta destas refatorações.

1.3 OBJETIVO

O objetivo deste trabalho é especificar e desenvolver uma DSML e sua ferramenta CASE que auxiliem na tarefa de refatoração estrutural de banco de dados. Com este objetivo em mente, têm-se os seguintes objetivos específicos:

[1] dbvis.com
[2] visual-paradigm.com
[3] mysql.com/products/workbench

1. Efetuar a análise do domínio para identificar as características para cada refatoração estrutural elencada neste trabalho;
2. Definir a notação para a linguagem visual que irá compor a DSML;
3. Especificar a gramática da DSML de modo a evitar a ocorrência de erros sintáticos;
4. Definir as principais regras de boa formação da DSML, visando especificar um conjunto básico de regras semânticas para os construtores da linguagem;
5. Implementar uma ferramenta CASE que permita a geração automática do código SQL referente a cada refatoração estrutural.

1.4 ESCOPO

Esta proposta busca abstrair a complexidade inerente às refatorações de bancos de dados não só para o público industrial, mas também para o acadêmico, pois ambos podem se beneficiar da modelagem e geração de código promovida pela abordagem. Com isto, tanto a aplicação das refatorações quanto o aprendizado tornam-se mais produtivos. O domínio desta proposta limita-se às refatorações estruturais definidas por Ambler Scott W. e Sadalage (2006). Ressalta-se que as refatorações *Replace LOB With Table* e *Split Column* ainda não possuem cobertura. O primeiro, recebe como entrada uma estrutura de dados semiestruturados (e.g., XML, JSON), portanto demanda um tratamento complexo no sentido de identificar e processar estas estruturas. Já o segundo, requer um tratamento diferenciado para cada coluna, que pode ter várias composições e ser de diferentes tipos de dados.

1.5 METODOLOGIA

A metodologia utilizada neste trabalho baseia-se na aplicação do método *Design Science* (DS) (PEFFERS et al., 2007), o qual segue as seguintes seis etapas: 1) identificação e motivação do problema. Nesta etapa busca-se apontar os problemas relacionados à evolução de bancos de dados e como o processo de refatoração é aplicado para endereçar este problema; 2) definição dos objetivos. Para executar esta etapa faz-se uma análise comparativa não só dos trabalhos identificados na literatura, como também das ferramentas existentes relacionadas à evolução de banco de dados e refatoração. Uma vez definidos estes objetivos, especifica-se uma proposta; 3) projeto e desenvolvimento. Uma vez fechado o escopo na etapa 1, inicia-se o desenvolvimento da proposta, a qual é baseada no conhecimento obtido na etapa 2; 4) demonstração. Nesta etapa são construídos exemplos, os quais buscam validar a proposta; 5) avaliação. Executa-se a avaliação da proposta por meio de exemplos de uso definidos no planejamento efetuado na etapa anterior. Assim, torna-se possível fazer uma comparação entre o resultado e seu objetivo. 6) comunicação.

Nesta etapa comunica-se todo o processo de desenvolvimento da solução para o problema identificado.

1.6 ESTRUTURA DA DISSERTAÇÃO

No sentido de demonstrar como os objetivos foram alcançados, bem como descrever a aplicação dos conceitos e tecnologias utilizadas, estruturou-se este trabalho de acordo com a seguinte descrição: No Capítulo 2 mostram-se os conceitos que formam o bloco de construção da proposta. No Capítulo 3 faz-se uma análise crítica das forças e fraquezas dos trabalhos relacionados ao estado da arte do tema em questão. No Capítulo 4 apresentam-se as características da DSML proposta, o qual consiste no metamodelo, na sintaxe concreta e na semântica estática. No Capítulo 5 mostra-se a ferramenta desenvolvida para validar o metamodelo por meio da avaliação da expressividade dos construtores. Também faz-se um comparativo entre a abordagem apresentada neste trabalho e as propostas analisadas no Capítulo 3. Por fim, no Capítulo 6 apresentam-se as conclusões deste trabalho, junto com as principais contribuições, limitações e trabalhos futuros. Sugestões de uso das citações:

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo apresentam-se os conceitos fundamentais para a compreensão desta proposta. Inicialmente, na Seção 2.1, introduz-se a definição de refatoração e sua aplicação no domínio de bancos de dados relacionais. Após isto, na Seção 2.2, discorre-se sobre a análise de domínio para identificar os conceitos presentes na DSML proposta. Na Seção 2.3 descrevem-se as definições do Desenvolvimento Orientado a Modelos e seu suporte no desenvolvimento de uma DSML. Na Seção 2.4 apresentam-se as tecnologias MDD utilizadas na proposta deste trabalho. Na Seção 2.5 exibe-se o metamodelo para o modelo relacional usado neste trabalho (i.e., RMM). Por fim, encerra-se o capítulo com as considerações finais.

2.1 REFATORAÇÃO DE BANCO DE DADOS

A refatoração é uma prática de engenharia de software que reestrutura o código existente alterando a sua estrutura interna sem afetar a semântica comportamental do software, ou seja, seu comportamento externo (FOWLER, 1999). Com refatoração é possível melhorar não só o design do código, mas também do sistema de forma geral. Além disso, também proporciona um código mais legível e reutilizável, ao passo que torna mais fácil estender e manter o código fonte (MENS; DEMEYER, 2008; MARIANI; VERGILIO, 2017). De acordo com Mens e Demeyer (2008), a refatoração é umas das práticas chaves para alcançar a evolução de software. Por isso, deve ser aplicada de forma disciplinada, demandando a definição de padrões de refatoração.

Assim como *Code Smells* (EVA; MOONEN, 2002) sugerem categorias de problemas que necessitam de refatoração, *Database Smells* também indicam que este tipo de intervenção seja aplicada em objetos de bancos de dados (AMBLER SCOTT W. E SADALAGE, 2006). O conceito de *Bad Smells* foi introduzido por Fowler (1999) para descrever códigos que apresentam deficiências de design e necessitam de restruturação (RATZINGER; FISCHER; GALL, 2005). Do mesmo modo, *Database Smells* podem afetar a interpretação dos dados de forma negativa, como por exemplo: 1) colunas e tabelas com vários propósitos, que podem ser produtos de falha de design; 2) dados redundantes fora de controle, que podem dar oportunidade para inconsistência dos dados; e 3) tabelas com várias colunas ou com um número excessivo de linhas, que podem apresentar falha de coesão e problemas de performance (AMBLER SCOTT W. E SADALAGE, 2006). Estes e outros *Database Smells* que podem trazer sérios riscos técnicos devem ser evitados e quando identificados, corrigidos.

Uma vez que a refatoração serve de alicerce para a evolução de software, outros componentes podem se beneficiar desta prática (MENS; DEMEYER, 2008). Na perspectiva do bancos de dados, a refatoração também melhora o design sem afetar sua semântica. No

entanto, além de manter a semântica comportamental, como faz a refatoração de código, também se preocupa com a semântica informacional. Este conceito está relacionado ao significado da informação contida no banco de dados do ponto de vista do usuário, implicando que o seu sentido deve ser preservado para que não haja impacto negativo na interpretação (AMBLER SCOTT W. E SADALAGE, 2006). O processo de refatoração de bancos de dados também torna-se mais complexo porque várias aplicações podem estar acopladas a um único esquema. Esta complexidade expõe a necessidade de uma forma sistemática de aplicar refatoração, além de boas ferramentas e técnicas que ofereçam suporte à tarefa.

Ambler Scott W. e Sadalage (2006) descrevem 64 padrões de refatoração de banco de dados categorizados em estruturais, qualidade de dados, integridade referencial, arquiteturais e comportamentais ou de método. Junto com estes padrões são definidas a motivação, os *tradeoffs*, mecânicas de atualização e migração dos dados. Além disso, estas refatorações podem ser aplicadas em ambientes onde uma única aplicação acessa os dados ou em ambientes onde várias aplicações acessam a mesma base, ou seja, ambientes multi-aplicações (AMBLER SCOTT W. E SADALAGE, 2006).

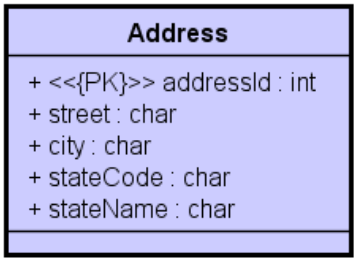
Dentre as refatorações de bancos de dados presentes na literatura está o padrão *Split Table*. Por meio desta refatoração torna-se possível dividir uma tabela existente por colunas. Com isto, removem-se tanto as anomalias das tabelas quanto a redundância dos dados, melhorando a qualidade do esquema como um todo. Um exemplo desta abordagem pode ser visto na Figura 1a, onde os atributos *stateCode* e *stateName* estão presentes em *Address*. Já na Figura 1b, nota-se que estes atributos foram movidos para *State*, gerando um modelo normalizado.

Split Table encontra-se na categoria de refatorações estruturais. No entanto, a literatura define outras refatorações distribuídas em outras categorias (i.e., qualidade de dados, integridade referencial, arquitetural, métodos e estruturais). Todas as refatorações podem ser vistas nos *bullets* i ao v.

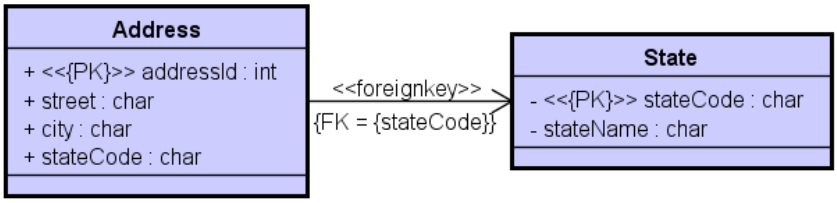
i **Refatorações Estruturais.** Refatorações estruturais têm como objetivo mudar a estrutura da tabela provendo melhorias para o banco de dados como um todo. As refatorações desta categoria são: *Drop Column*, *Drop Table*, *Drop View*, *Introduce Calculated Column*, , *Introduce Surrogate Key*, *Merge Columns*, *Merge Tables*, *Move Column*, *Rename Column*, *Rename Table*, *Rename View*, *Replace Large Object (LOB) With Table*, *Replace Column*, *Replace One-to-Many With Associative Table*, *Replace Surrogate Key With Natural Key*, *Split Column* e *Split Table*.

ii **Refatorações de Qualidade de dados.** As refatorações contidas nessa categoria são responsáveis por melhorar a qualidade da informação, garantindo a consistência e o uso dos valores armazenados na base de dados. As principais refatorações são: *Add Lookup Table*, *Apply Standard Codes*, *Apply Standard Type*, *Consolidate Key*

Figura 1 – Representação da solução para um *database smell* aplicando *Split Table*.



(a) *Address* contendo *database smell*



(b) Resultado da Refatoração *Split Table*

Fonte: (AMBLER SCOTT W. E SADALAGE, 2006)

Strategy, *Drop Column Constraint*, *Drop Default Value*, *Drop Non-Nullable Constraint*, *Introduce Column Constraint*, *Introduce Common Format*, *Introduce Default Value*, *Make Column Non-Nullable*, *Move Data* e *Replace Type Code With Property Flags*.

iii **Refatorações de Integridade Referencial.** Visam garantir que as referencias entre tabelas se mantenham válidas, além de assegurar a remoção dos dados de forma apropriada. As principais refatorações são: *Add Foreign Key Constraint*, *Add Trigger For Calculated Column*, *Drop Foreign Key Constraint*, *Introduce Cascading Delete*, *Introduce Hard Delete*, *Introduce Soft Delete* e *Introduce Trigger For History*.

iv **Refatorações Arquiteturais.** Preocupam-se com a qualidade da comunicação entre o banco de dados e suas interfaces externas. As principais refatorações são: *Add CRUD Methods*, *Add Mirror Table*, *Add Read Method*, *Encapsulate Table With View*, *Introduce Calculation Method*, *Introduce Index*, *Introduce Read-Only Table*, *Migrate Method From Database*, *Migrate Method To Database*, *Replace Method(s) With View*, *Replace View With Method(s)* e *Use Official Data Source*.

v **Refatorações de Métodos.** São usadas para evoluir as *Stored Procedures*, *Functions* e *Triggers*. Estas refatorações dividem-se em duas subcategorias, sendo os padrões da subcategoria *Interface Changing Refactoring* (I) responsável por melhorar o contrato dos métodos e os da subcategoria *Internal Refactorings* (II) por melhorias relacionadas ao corpo das rotinas. As principais refatorações do tipo *Inter-*

face Changing Refactoring (I) são: *Add Parameter*, *Parameterize Method*, *Remove Parameter*, *Rename Method*, *Reorder Parameters* e *Replace Parameter with Explicit Methods*. As principais refatorações do tipo *Internal Refactorings* (II) são: *Consolidate Conditional Expression*, *Decompose Conditional*, *Extract Method*, *Introduce Variable*, *Remove Control Flag*, *Remove Middle Man*, *Rename Parameter*, *Replace Literal with Table Lookup*, *Replace Nested Conditional with Guard Clauses*, *Split Temporary Variable* e *Substitute Algorithm*.

Qiu, Li e Su (2013) apresenta um estudo detalhado no qual 10 aplicações *open-source* de vários domínios são avaliadas. Este estudo utiliza as categorias de refatoração supracitadas e identifica quais evoluções são mais realizadas, constatando que as mais aplicadas são refatorações estruturais. Baseando-se nesse estudo, este trabalho utiliza os padrões de refatoração estruturais como escopo da pesquisa. Vale ressaltar que as refatorações desta categoria são detalhadas no apêndice A.

Na próxima seção, apresenta-se a abordagem adotada para levantamento dos conceitos do domínio relevantes para a solução proposta, junto com o processo de análise adotado neste trabalho.

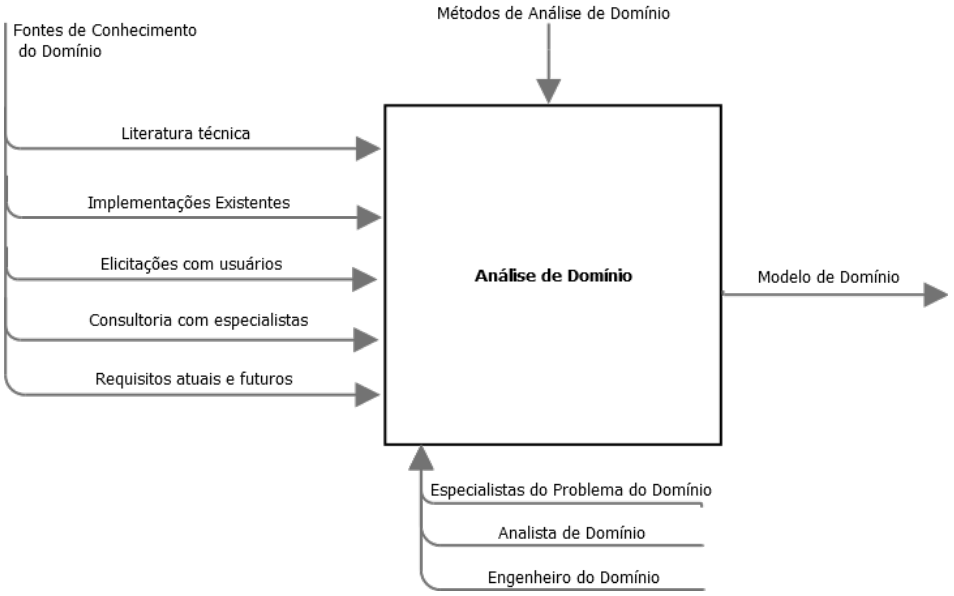
2.2 ANÁLISE DE DOMÍNIO

A análise de domínio tem como pontos-chave a identificação, captura, organização e estruturação da informação utilizada no desenvolvimento de sistemas com o propósito de torná-la reutilizável (PRIETO-DIAZ, 1990). Além disso, Neighbors (1980) afirma que a análise de domínio visa trazer a perspectiva dos especialistas para o que é de fato importante. Com isto, o analista de domínio pode definir as fronteiras que delimitam a capacidade operacional, as operações e relacionamentos dos elementos do domínio. Na Figura 2 ilustra-se como esta perspectiva pode ser observada.

Como descrito na Figura, uma das entradas do processo de Análise de Domínio é o Método de Análise, que serve como um guia para identificação das características que o usuário espera encontrar em uma solução. Outras entradas do processo estão relacionadas às Fontes de Conhecimento do Domínio (e.g., Literatura Técnica, Requisitos Atuais e Futuros) e aos Especialistas do Domínio (e.g., Analista do Domínio, Engenheiro do Domínio). A primeira, serve como base de informação para captura, identificação e estruturação dos conceitos do domínio. Já o segundo, busca extrair, analisar e abstrair os conceitos que são organizados e encapsulados para produzir o Modelos do Domínio (e.g., Componentes, Linguagens Específicas do Domínio).

Uma vez identificados os conceitos do domínio, na Seção 2.3 são introduzidos os conceitos relacionados ao desenvolvimento orientado a modelos e seu suporte no processo de definição de uma linguagem de domínio específico.

Figura 2 – Processo de Análise de Domínio definido por Prieto-Diaz (1990)



Fonte: Adaptado de Prieto-Diaz (1990)

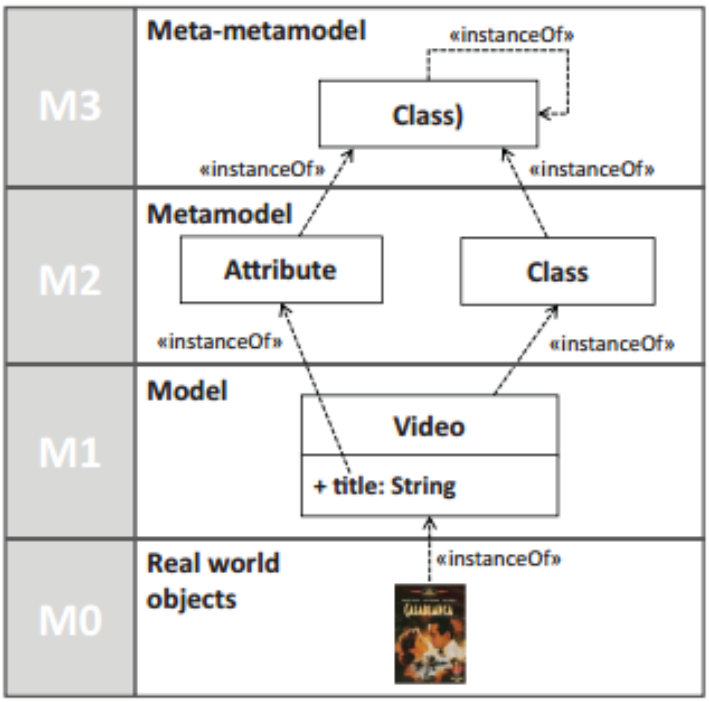
2.3 DESENVOLVIMENTO ORIENTADO A MODELOS

A Engenharia Orientada a Modelos, do inglês *Model Driven Engineering* (MDE), é uma abordagem de engenharia de software que tem como principais objetivos construir software de qualidade, trazer mais produtividade e melhorar o processo de desenvolvimento, facilitando a manutenção e diminuindo o tempo de desenvolvimento (HUTCHINSON; WHITTLE; ROUNCEFIELD, 2014). Além disso, também possui papel fundamental no processo de interoperabilidade entre sistemas e intercâmbio de dados, trazendo padronização para o processo. A MDE traz consigo várias técnicas para definição e exploração de modelos, pois considera estes artefatos como entidades de primeira classe (WHITTLE; HUTCHINSON; ROUNCEFIELD, 2014; TOPCU et al., 2016).

Uma forma de aplicar MDE é por meio do Desenvolvimento Dirigido por Modelos, do inglês *Model Driven Development* (MDD). O MDD é um paradigma que utiliza modelos como principal artefato do processo de definição e construção de software. Segundo Siegel (2014), modelos são representações de alto nível que podem ser abstraídos por uma linguagem de modelagem. Do mesmo modo que os modelos abstraem elementos da realidade modelada, esta ideia também pode ser aplicada sobre os próprios modelos. Partindo desta perspectiva, tem-se que os metamodelos são abstrações dos modelos. Os metamodelos também podem descrever outros metamodelos definindo o conceito de meta-metamodelo, como pode ser observado na Figura 3, na qual apresenta-se o modelo de 4 camadas definido pela OMG[1], conhecido como *Meta-Object Facility* (MOF)[2] (ATKINSON; KUEHNE, 2003).

[1] https://www.omg.org/
[2] http://www.omg.org/spec/MOF/2.5.1/

Figura 3 – Camadas do MOF definido pela OMG

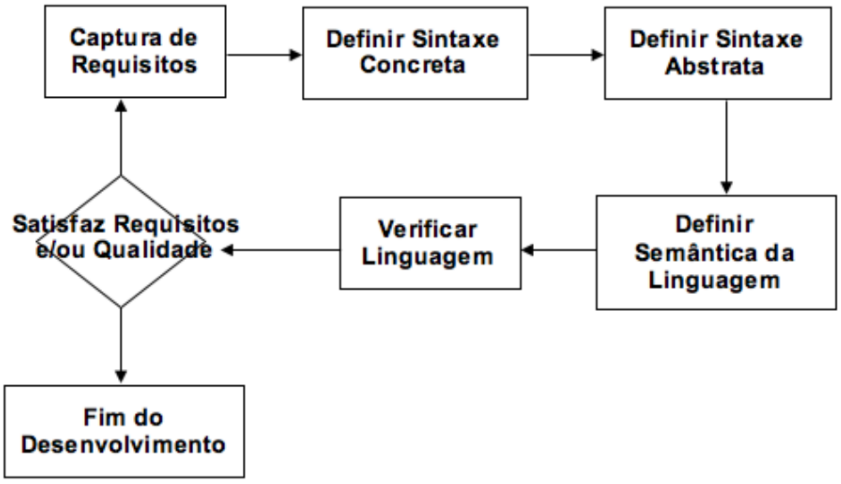


Fonte: Adaptado de Brambilla, Cabot e Wimmer (2012)

Com MDD é possível definir metamodelos para representar diferentes aspectos de um domínio e também aplicar vários tipos de processamentos (e.g., transformações, comparações e geração automática de artefatos e código), auxiliando na redução da complexidade em expressar conceitos do domínio (SCHMIDT, 2006).

Uma das atividades básicas da abordagem MDD é a transformação entre modelos. Esta transformação deve estar em conformidade com o metamodelo e pode ser do tipo M2M e/ou M2T, as quais visam prover transformações de modelo para modelo e de modelo para texto, respectivamente (VOELTER et al., 2013). Todo este processo pode ser aplicado na construção de uma Linguagem de Modelagem de Domínio Específico (do inglês, *Domain-Specific Modeling Language* (DSML)). As DSML estão diretamente relacionadas a um domínio mais restrito, focando na solução de problemas inerentes ao um contexto específico (e.g., EER) (MERNIK; HEERING; SLOANE, 2005). Na Figura 4 apresenta-se o processo de desenvolvimento de uma DSML (CHO; GRAY; SYRIANI, 2012). Uma vez que este processo demanda conhecimento sobre o domínio, o desenvolvimento se inicia pela fase de Captura de Requisitos, que visa identificar o domínio do problema e elicitar os requisitos. É de suma importância que nesta fase os especialistas do domínio identifiquem o que se espera da linguagem de modelagem, bem como as representações deste domínio. O processo de desenvolvimento de uma DSML ainda contempla as fases Definir Sintaxe Concreta, Definir Sintaxe Abstrata e Definir Semântica da Linguagem. A definição da Sintaxe Concreta resulta na notação que será definida para detalhar visualmente os construtores da linguagem. Já a definição de Sintaxe Abstrata, busca construir uma especificação dos

Figura 4 – Processo de desenvolvimento de DSML.



Fonte: Adaptado de Cho, Gray e Syriani (2012)

elementos da linguagem e seus relacionamentos por meio de um metamodelo. Na etapa Definir Semântica Estática, o desenvolvedor da linguagem preocupa-se em especificar as regras de boa formação que não são capturadas pelo metamodelo (e.g., tabelas com mesmo nome ou chaves estrangeiras que não referenciam uma chave primária). Por fim, na fase Verificar Linguagem, os especialistas avaliam a composição da linguagem, fornecendo *feedbacks* que podem marcar mais iterações de aprimoramentos. Por ser incremental, este processo se repete até que a DSML atenda as necessidades dos especialistas e se encerra caso seja satisfatória.

Na Seção seguinte são introduzidas as tecnologias que servem como bloco de construção para o processo de desenvolvimento de uma DSML por meio do paradigma de desenvolvimento orientado a modelos.

2.4 FERRAMENTAS PARA MDD

A construção da DSML proposta se dá por meio do arcabouço de tecnologias entregues pelos *frameworks Eclipse Modeling Framework* (EMF)[3], *Graphical Modeling Framework* (GMF)[4] e *Epsilon Framework*[5].

O EMF permite a modelagem e geração de código para representação de modelos de forma automática. Já o GMF adiciona uma camada de abstração aos conceitos do domínio representados pelos elementos do metamodelo, gerando como resultado uma ferramenta de modelagem visual. O GMF busca entregar um conjunto de insumos para construção de editores gráficos que se baseiam nos metamodelos definidos por meio da tecnologia EMF (MOORE, 2004) (SHATALIN; TIKHOMIROV, 2006).

[3] https://www.eclipse.org/modeling/emf/
[4] https://www.eclipse.org/gmf-tooling/
[5] https://www.eclipse.org/epsilon/doc/

O Epsilon é uma família de linguagens e ferramentas que suportam a geração de código, transformação, validação, comparação, migração e refatoração de modelos (KOLOVOS et al., 2012). Dentre as linguagens do *Epsilon* utilizadas neste projeto estão *Epsilon Validation Language* (EVL), *Epsilon Generation Language* (EGL) e *Epsilon Object Language* (EOL).

A linguagem EVL possui sintaxe similar a *Object Constraint Language* (OCL), sendo construída para especificar e avaliar as restrições sobre os elementos de um metamodelo. Com isto, pode-se definir as regras de boa formação impostas sobre os modelos e consequentemente validar a conformidade com semântica estática especificada. Na Listagem 2.1, mostra-se um trecho de código escrito na linguagem EVL. Da linha 2 à 4, define-se uma restrição chamada *"HasName"* indicando que a tabela deve ter um nome. Em caso negativo, uma mensagem informando a violação desta restrição deve ser retornada. Já da linha 7 à 13, ajusta-se a crítica *"NameShouldBeUpperCase"* para alertar o usuário que o nome da tabela deveria ser escrito com letras maiúsculas. Além disso, como disposto da linha 11 à 13, é possível auxiliar o usuário na correção do modelo. Vale ressaltar que, diferente da restrição, uma crítica é utilizada para capturar erros que não invalidam o modelo.

Listagem 2.1 – Exemplo de código EVL

```
1  context Table {
2      constraint HasName {
3       check : self.name.isDefined()
4       message : self.eClass().name + ' must have a name.
5       Set the name for '+ self.eClass().name + '.'
6                         }
7      critique NameShouldBeUpperCase {
8       guard : self.satisfies('HasName')
9       check : self.name.toUpperCase() <> self.name
10      message : self.eClass().name + ' ' + self.name +' should be uppercase.'
11     fix {
12      title : 'Rename to ' + self.name.toUpperCase()
13      do { self.name := self.name.toUpperCase();}
14        }
15                               }
16                }
```

Fonte:(Autor, 2018)

A linguagem EGL é utilizada para geração de código guiada por *templates* que viabiliza a tradução M2T. Sendo assim, pode-se gerar instruções baseadas nos modelos definidos por meio da DSML. Um trecho de código contendo instruções EGL pode ser observado na Listagem 2.2. Neste exemplo, comandos SQL são gerados e disponibilizados na saída padrão. Na linha 1, concatenam-se os comandos para criação de uma nova tabela com o nome da tabela contendo letras maiúsculas. Da linha 2 à 13, são definidas as colunas da

tabela e seus respectivos tipos, verificando se cada coluna possui um valor padrão e se sua definição é obrigatória. Na linha 14, encerra-se com o fechamento do bloco de instruções iniciado na linha 1.

Listagem 2.2 – Exemplo de código EGL

```
1  out.print("CREATE TABLE IF NOT EXISTS "+table.name.toUpperCase()+"(");
2  for(column in table.columns){
3    out.print(column.name.toUpperCase()+" "+column.getDataType(attribute));
4    if(column.defaultValue.isDefined()){
5    out.print(" DEFAULT "+attribute.defaultValue);
6    }
7    if(column.isNotNull = true){
8    out.print(" NOT NULL ");
9    }
10   if(not table.columns.last().equals(column)){
11   out.print(",");
12   }
13 }
14 out.println(");");
```

Fonte:(Autor, 2018)

A linguagem EOL entrega um conjunto de operações reutilizáveis para gerenciamento de modelos como instruções para tratamento de coleções, condicionais, de repetição, definição de tipos de dados e etc (KOLOVOS et al., 2012). Na Listagem, ilustra-se um fragmento de código com a definição de uma operação que retorna o nome da chave primária de uma determinada tabela.

Listagem 2.3 – Exemplo de Código EOL

```
1
2  operation getPrimaryKeyName(table:Table):EString{
3      for(constraint in table.constraints){
4      if(constraint.isKindOf(PrimaryKey)){
5      return constraint.name;
6      }
7      }
8  }
9  }
```

Fonte:(Autor, 2018)

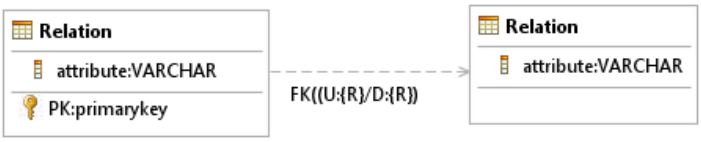
Na Seção 2.5, detalha-se o metamodelo *Relational Metamodel* (RMM) e suas metaclasses (FERNANDES, 2017). Este metamodelo permite definir construtores para representar o modelo relacional.

2.5 METAMODELO RMM

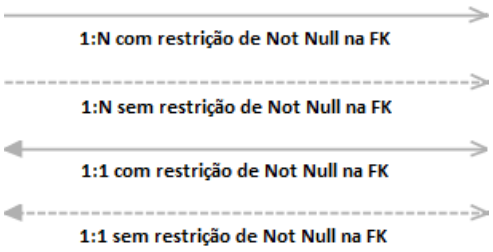
Uma vez que as Refatorações Estruturais de bancos de dados são aplicadas sobre os bancos de dados relacionais, necessita-se de um metamodelo para abstrair os conceitos deste modelo de dados. Em seu trabalho, Fernandes (2017) faz uma análise qualitativa dos modelos de dados existentes e suas notações, concluindo que os modelos analisados possuem deficiências quanto à cobertura dos conceitos do modelo relacional. O autor propõe um modelo que possui aderência total ao modelo relacional, entregando uma notação e um metamodelo nomeado *Relational Metamodel* (RMM) para representar estes conceitos. Ressalta-se que só serão descritos nesta seção as metaclasses e construtores utilizados neste trabalho.

2.5.1 Sintaxe Concreta

Figura 5 – Construtores Básicos do metamodelo RMM



(a) Construtores *Relation*, *Attribute*, e restrições (i.e., *Primary Key* e *Foreign Key*)



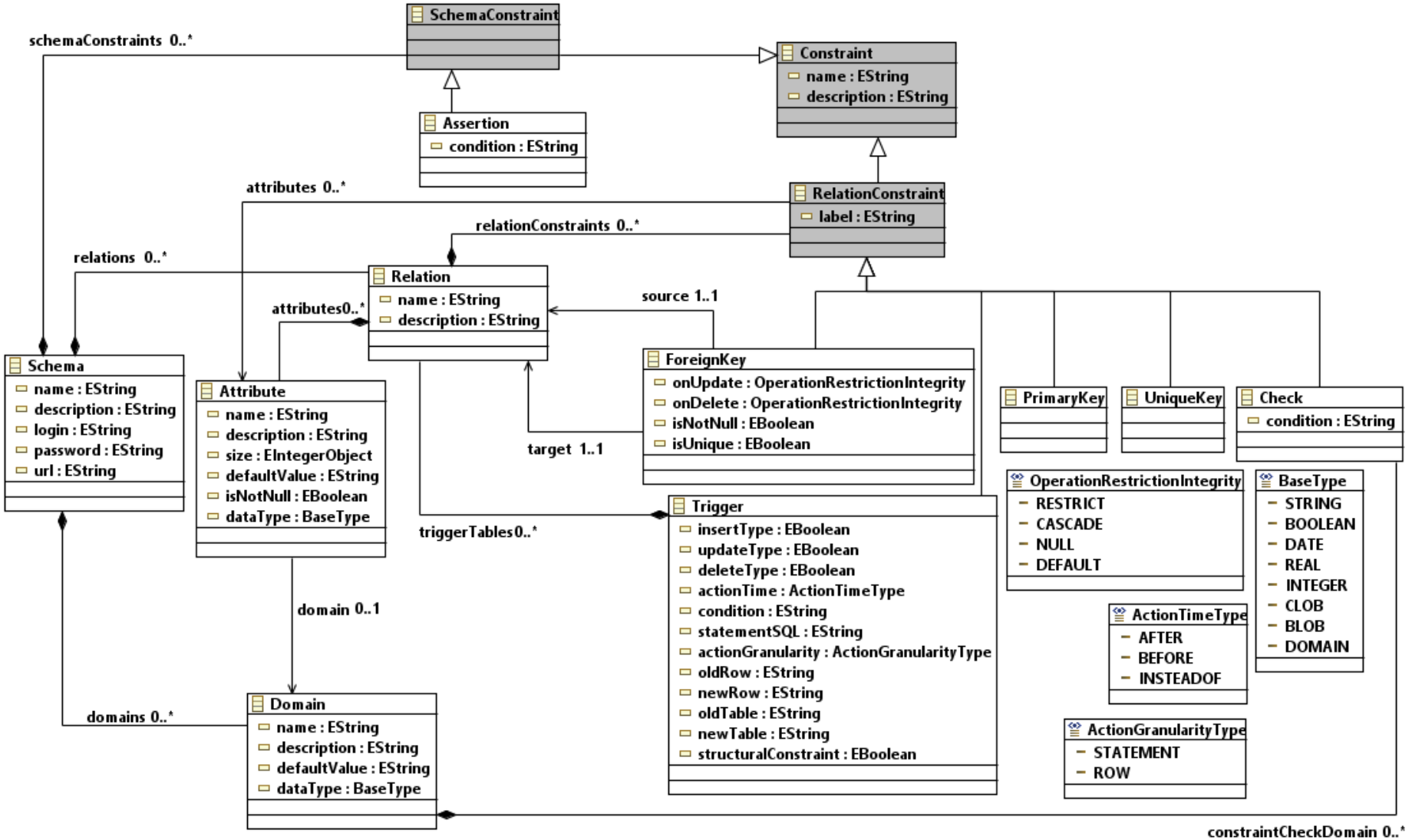
(b) Construtores de Ligação entre Relações

Fonte: (Autor, 2018)

Na Figura 5 apresenta-se a notação dos construtores básicos do modelo Relacional utilizados neste trabalho. Como disposto na Figura 5a, a representação de uma tabela é composta por um rótulo e dois compartimentos. O rótulo é formado por um pictograma que representa uma tabela e pelo nome da relação. Já os compartimentos, de cima para baixo, são: 1) compartimento de atributos, os quais são identificados por um pictograma semelhante a uma coluna, um rótulo contendo seu nome e seu tipo de dado; e 2) compartimento de restrições (e.g., chave primária e/ou restrição de unicidade), que tem seus construtores representados por um pictograma e um rótulo contendo o nome da restrição. Além disso, identifica-se a chave estrangeira (*Foreign Key*) com uma linha e seu

rótulo, que contém o nome da chave estrangeira e as primeiras letras das operações de restrição e/ou cascateamento (i.e., *RESTRICT* e/ou *CASCADE*) realizadas ao atualizar (*UPDATE*) e ao remover (*DELETE*) uma tupla. Já na Figura 5b, especificam-se as semânticas relacionadas às ligações entre relações. Nota-se que tanto os relacionamentos do tipo 1:N (i.e., um para muitos) quanto os do tipo 1:1 (i.e., um para um) possuem uma seta aberta para indicar a relação referenciada. No entanto, no segundo caso, especifica-se uma seta fechada para indicar que os atributos que compõem a chave estrangeira na tabela fonte possuem uma restrição de unicidade. Além disso, identifica-se a obrigatoriedade da definição destes atributos (i.e., restrição *Not Null*) por meio de uma linha sólida. Caso a especificação dos atributos seja opcional, representa-se com a linha tracejada.

Figura 6 – Metamodelo RMM



Fonte:(FERNANDES, 2017)

2.5.2 Sintaxe Abstrata

Na Figura 6, apresenta-se o metamodelo RMM, onde são especificadas as metaclasses e enumerações referentes aos construtores do modelo relacional. Vale ressaltar que as metaclasses abstratas são identificadas pelo cor cinza, enquanto as concretas são identificadas pela cor branca. Além disso, somente as metaclasses utilizadas neste trabalhos são descritas nesta seção, que são:

- *Schema* esta metaclasse representa o esquema da base de dados e possui os meta-atributos *name*, *description*, *login*, *password* e *url*. Estes meta-atributos são todos do tipo *EString*, sendo os três últimos definidos para armazenar o nome do usuário, senha e o caminho de acesso à base de dados. Ressalta-se que a composição *relations* refere-se aos construtores do tipo relação dispostos na área de desenho.

- A metaclasse *Relation* representa a relação (i.e., tabela da base de dados) na DSML proposta e possui os meta-atributos *name* e *description*, ambos do tipo *EString*. Além disso também mantém as seguintes relações de composição: *relationConstraints* - que refere-se às metaclasses que representam as restrições que podem ser impostas sobre a tabela (e.g., *PrimaryKey* e *ForeignKey*); e *attributes* - que especifica os atributos que podem compor a tabela.

- *Attribute* é a metaclasse que representa o atributo (i.e., coluna da tabela) na DSML proposta e além das propriedades *name* e *description* possui os atributos: *size* para definir a largura da coluna; *defaultValue*, para que se possa ajustar um valor padrão e o booleano; e *isNotNull* para definir a obrigatoriedade ou não de um valor para a coluna na Relação; Além disso, esta metaclasse também define um atributo nomeado *dataType* do tipo Enumeração *BaseType*. Esta Enumeração define os tipos de dados suportados pelo construtor *Attribute*.

- As metaclasses *PrimaryKey* e *ForeignKey* representam, respectivamente, as chaves primárias e chaves estrangeiras das Relações no ambiente gráfico. Estas metaclasses são especializações da metaclasse abstrata *RelationConstraint*, que possui um atributo *label*. A metaclasse *RelationConstraint* também herda os atributos *name* e *description* da metaclasse abstrata *Constraint*.

As demais metaclasses também são utilizadas na ferramenta CASE para dar suporte completo à modelagem de diagramas do modelo Relacional, porém não são descritas aqui por estar fora do escopo deste trabalho.

2.6 CONSIDERAÇÕES FINAIS

Neste capítulo, apresentam-se os conceitos que formam a base para a compreensão dos demais capítulos deste trabalho. Sendo assim, inicia-se pela introdução do conceito de

Refatoração e seu uso no contexto de bancos de dados. Estas classes de evoluções se preocupam com a modificação das características estruturais e comportamentais de um esquema sem afetar sua propriedade informacional.Também são introduzidas a análise de domínio e a abordagem MDD. A primeira, tem papel fundamental no sentido de auxiliar o entendimento dos conceitos do domínio. Com isto, pode-se identificar, capturar, organizar e estruturar as informações coletadas e torná-las reutilizáveis. A segunda, busca auxiliar no processo de abstração dos conceitos do domínio em modelos. Esta abordagem utiliza modelos como principal bloco de construção de artefatos, tornando possível definir uma sintaxe abstrata, uma sintaxe concreta e uma semântica estática. Assim, estes modelos podem ser traduzidos em outros modelos ou em texto, suportando a construção de uma DSML. Uma DSML e desenvolvida por um processo bem definido e se propõe a reduzir a complexidade na resolução de problemas absorvendo o conjunto de características estruturais ou comportamentais, fornecendo uma representação simplificada para realizar tarefas complexas. Por meio da família de linguagens e ferramentas MDD dispostas no *framework Epsilon*, EMF e GMF pode-se conceber uma ferramenta CASE que abstraia os construtores do modelo Relacional. Do mesmo modo, estas tecnologias fornecem insumos para a especificação e desenvolvimento de uma DSML para as refatorações estruturais de bancos de dados.

No próximo capítulo são introduzidos os trabalhos relacionados, bem como suas forças e fraquezas. Ao final, exibe-se o nível de aderência de cada trabalho em relação aos critérios de seleção e de avaliação definidos no desenho da pesquisa.

3 REVISÃO DA LITERATURA

Neste capítulo analisam-se os trabalhos relacionados e ferramentas que propõem métodos para evolução de bancos de dados relacionais. Sobretudo, busca-se identificar trabalhos que utilizem padrões de refatoração estruturais de banco de dados como técnica para evolução do esquema e MDD como abordagem para solução. Este capítulo estrutura-se da seguinte forma: na Seção 3.1 apresentam-se os critérios de avaliação dos trabalhos, os quais são usados para analisar a cobertura aos padrões de refatoração estrutural de bancos de dados e a adoção do paradigma MDD. Também avalia-se o suporte à engenharia reversa, engenharia direta, migração de dados e período transicional, uma vez que são características chaves para a definição de uma DSML que automatize o processo. Na Seção 3.2, analisam-se os principais trabalhos e ferramentas relacionados a esta proposta, baseando-se nos critérios definidos na seção anterior. Por fim, na Seção 3.3 apresenta-se a análise e a avaliação consolidada dos trabalhos e ferramentas.

3.1 CRITÉRIOS PARA AVALIAÇÃO

Visando fazer uma análise comparativa entre os trabalhos relacionados, considera-se importante a definição de critérios com o objetivo de identificar se há algum mecanismo na ferramenta/abordagem ou elemento no metamodelo que viabilize a implementação dos Padrões de Refatoração Estrutural, que são: *Drop Column*, *Drop Table*, *Drop View*, *Introduce Calculated Column*, *Introduce Surrogate Key*, *Merge Columns*, *Merge Tables*, *Move Column*, *Rename Column*, *Rename Table*, *Rename View*, *Replace LOB with Table*, *Replace Colum*, *Replace One-to-Many With Associative Table*, *Replace Surrogate Key With Natural Key*, *Split Column* e *Split Table*, bem como seus mecanismos de atualização do esquema (i.e., Período Transicional e/ou a Migração de Dados). Estes critérios são baseados nas refatorações estruturais descritas por Ambler Scott W. e Sadalage (2006), pois estas são as refatorações mais usadas (QIU; LI; SU, 2013). Além disso, para avaliar se o trabalho faz uso de MDD, analisa-se se este define uma Sintaxe Concreta, Sintaxe Abstrata e Semântica Estática. Por fim, investiga-se a existência de uma abordagem para Engenharia Reversa e Engenharia Direta, uma vez que são componentes chaves do processo de evolução de bancos de dados.

Na próxima seção, são apresentados os trabalhos relacionados, bem como os critérios de seleção e avaliação.

3.2 TRABALHOS SELECIONADOS

Nesta Seção, busca-se analisar os trabalhos selecionados. Para isso, realizou-se um levantamento bibliográfico. Neste levantamento foram utilizadas as bibliotecas digitais ACM

Digital Library e IEEEXplore. Além disso, foram consultados tanto os indexadores Scopus e Science Direct quanto a ferramenta de busca Google Scholar. A *string* de busca executada nos repositórios foi a seguinte: "('relational schema' OR 'relational database' OR 'relational model') AND ('metamodel' OR 'meta model') AND ('schema evolution' OR 'schema refactoring' OR 'database evolution' OR 'database refactoring')". Foram excluídos desta pesquisa trabalhos que: 1) não estão escritos em inglês; e 2) não foi possível ter acesso ao texto completo. Por outro lado, foram incluídas propostas que possuem os seguintes critérios: 1) propõem uma abordagem para evolução de bancos de dados relacionais; e 2) apresentam uma análise ou proposta de ferramentas e técnicas para evolução de banco de dados relacional. Nesta busca foram retornados 909 artigos, os quais foram analisados o título e resumo. Após esta leitura, foram descartados 904 artigos (permanecendo cinco artigos), pois estes não estavam relacionados com evolução ou refatoração de bancos de dados relacionais. Ressalta-se que apesar de ter como objetivo identificar trabalhos que utilizam MDD como abordagem para construção de uma DSML, entende-se que é prudente comparar ferramentas que adotem propostas distintas, porém alcancem o mesmo resultado. Sendo assim, ferramentas como *IBM Optim Database Administrator*[1], *Oracle Enterprise Manager Management Packs*[2], *Acoda*[3] e *MSSQL Refactor Studio*[4] também foram analisadas. Vale ressaltar que, dentre estas ferramentas, nenhuma utiliza MDD. Porém, a ferramenta *MSSQL Refactor Studio* destaca-se por oferecer suporte parcial às refatorações, portanto encontra-se dentre as abordagens e ferramentas analisadas neste capítulo. As demais não possuem suporte às refatorações ou operações que entreguem um resultado semelhante.

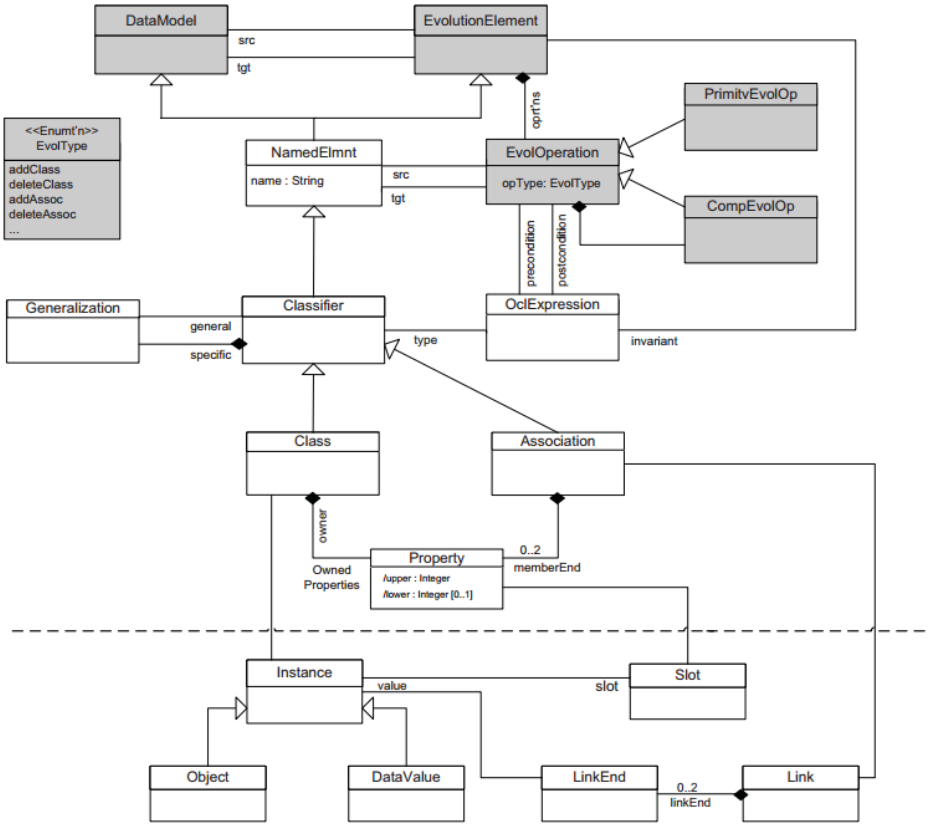
3.2.1 Aboulsamh & Davies

Aboulsamh Mohammed A. e Davies (2010) utilizam uma abordagem orientada a modelos para coevolução de sistemas de informação. Para isto, cria-se um metamodelo que define uma série de operações de evolução que são traduzidas para um modelo de classes UML. Esta representação do modelo de dados do sistema é transformado em uma série de instruções SQL para evoluir o esquema do banco de dados. Além disso, uma extensão do metamodelo UML foi especificada para representar as classes da aplicação e, por meio desta extensão, são efetuados mapeamentos de mudanças do modelo de classes para o modelo de banco de dados. Na Figura 7 apresenta-se o metamodelo definido neste trabalho.

Como pode ser visto na Figura 7, o metamodelo estende não só elementos do metamodelo UML, mas também da OCL. Um dos principais elementos é a enumeração *EvolType*, que representa os tipos de evolução que podem ser aplicadas sobre o modelo e, consequen-

[1] http://www.ibm.com/developerworks/data/library/ techarticle/dm-0704henry/index.html
[2] http://www.oracle.com/technetwork/articles/oem-packs-overview-195704.html
[3] http://webdsl.org/selectpage/Manual/acoda
[4] https://sqlrefactorstudio.com/

Figura 7 – Extensão do Metamodelo UML proposta por Aboulsamh Mohammed A. e Davies (2010)



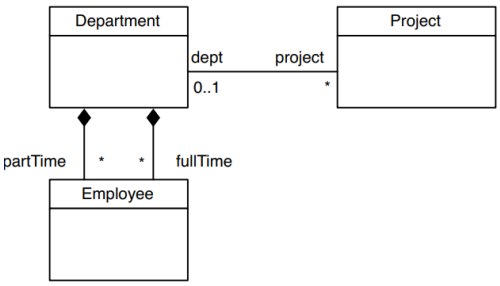
Fonte: (ABOULSAMH MOHAMMED A. E DAVIES, 2010)

temente, o esquema. Estas evoluções estão presentes na classe *EvolOperation*, que possui duas especializações: *PrimitvEvolOp* e *CompEvolOp*. O autor afirma que *PrimitvEvolOp* aplica uma operação simples sobre o modelo como, por exemplo, adicionar um atributo ou remover uma classe. Já *CompEvolOp*, comporta uma série de operações no intuito de redefinir o esquema. Porém, ao descrever que *EvolOperation* possui uma composição de *CompEvolOp* e *PrimitvEvolOp* é sua especialização, acaba quebrando a semântica do modelo, haja vista que *PrimitvEvolOp* deixa de representar somente operações simples e passa a compor operações complexas por meio de *CompEvolOp*.

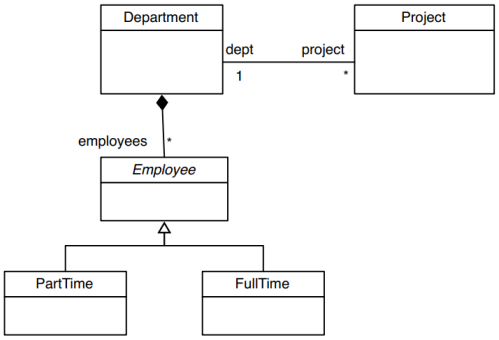
Para exemplificar, o autor apresenta um protótipo onde uma lista de mudanças são efetuadas em um diagrama de classes da UML e propagadas para o esquema do banco de dados. Segundo o autor, todo processo é suportado pelo EMF. Na Figura 8 apresenta-se como este processo de coevolução se desdobra.

Como pode ser visto na Figura 8c, as alterações podem ser efetuadas compondo operações que são aplicadas sobre o modelo de classes, gerando instruções de banco de dados como criação de tabelas e consultas. Ressalta-se que o autor não define uma representação para o elemento *View*, o que inviabiliza a definição das refatorações *Drop View* e *Rename View*. Também, dado o conjunto de operações presentes na proposta, não foram

Figura 8 – Exemplo de coevolução do Aboulsamh Mohammed A. e Davies (2010)



(a) Modelo de Dados Existente



(b) Modelo de Dados Após Evolução Proposta

```
1 addClass(FullTime);
2 addGeneralization(FullTime, Employee);
3 deleteAssociation(Department, fullTime);
4 addClass(PartTime);
5 addGeneralization(PartTime, Employee);
6 deleteAssociation(Department, partTime);
7 addAssociation(Department, employees,
   Employee,*);
8 modifyAssociation(Project, department,
   Department, 1)
```

(c) Conjunto de Operações de Evolução

```
CREATE TABLE fulltime AS
  (SELECT empid FROM department_fulltime);
CREATE TABLE department_employees AS
  (SELECT empid, depid
     FROM employee, department,
       department_fulltime
 WHERE employee.empid =
          department_fulltime.empid AND
          department.depid =
     department_fulltime. depid);
...
DROP TABLE employee;
DROP TABLE department_fulltime;
DROP TABLE department_parttime;
```

(d) Declarações SQL para Evolução Proposta

Fonte:Adaptado de Aboulsamh Mohammed A. e Davies (2010)

encontradas operações que permitam fundir elementos. Consequentemente, as refatorações *Merge Column* e *Merge Tables* não possuem cobertura. Além disso, também não é possível evidenciar que as refatorações *Replace LOB with Table*, *Replace One-to-Many with Associative Table*, *Introduce Surrogate Key*, *Replace Surrogate Key with Natural Key* e *Split Column* possam ser aplicadas. Por fim, o autor também não define um Período Transicional e não cita Engenharia Reversa em seu trabalho.

Ao analisar o exemplo, bem como as operações apresentadas na proposta, identifica-se que as refatorações *Drop Column*, *Drop Table*, *Move Column*, *Rename Column*, *Rename Table*, *Replace Column* e *Split Table* podem ser efetuadas. Nota-se, também, a presença da Sintaxe Abstrata, Sintaxe Concreta e da Semântica Estática, por meio de regras OCL, além da Migração de Dados. Contudo, nenhuma ferramenta é disponibilizada para que uma análise mais detalhada seja efetuada. No Quadro 3.2, mostra-se uma análise deste trabalho com relação aos critérios definidos na Seção 3.2.

Quadro 2 – Avaliação do trabalho proposto por Aboulsamh Mohammed A. e Davies (2010)

Característica	**Aboulsamh Mohammed A. e Davies (2010)**
Drop Column	SIM
Drop Table	SIM
Drop View	NÃO
Introduce Calculated Column	NÃO
Introduce Surrogate Key	NÃO
Merge Columns	NÃO
Merge Tables	NÃO
Move Column	SIM
Rename Column	SIM
Rename Table	SIM
Rename View	NÃO
Replace LOB with Table	NÃO
Replace Column	SIM
Replace One-to-Many with Associative Table	NÃO
Replace Surrogate Key with Natural Key	NÃO
Split Column	NÃO
Split Table	SIM
Período Transicional	NÃO
Sintaxe Abstrata(Metamodelo)	SIM
Sintaxe Concreta(Notação)	SIM
Semântica Estática	SIM
Engenharia Reversa	NÃO
Engenharia Direta	SIM
Migração de Dados	SIM

Fonte: (Autor, 2018)

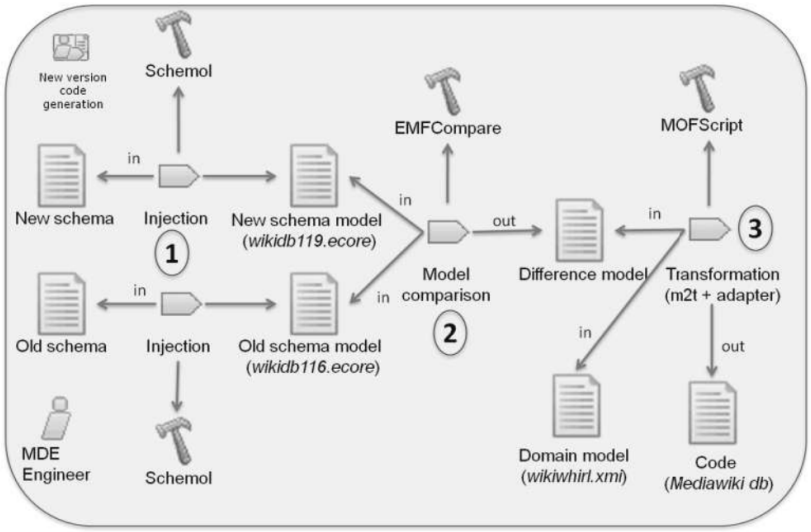
3.2.2 Garcia et al.

García e Díaz Oscar e Cabot (2014) traz como elemento central da coevolução um componente que recebe como entrada dois esquemas diferentes e aplica heurísticas para resolver a distinção e atualizar o esquema da aplicação. Um panorama geral deste processo de evolução pode ser visto na Figura 9. O processo inicia com a injeção de duas versões de

um mesmo esquema. Após isto, são aplicadas instruções *Schemol*[5] com o intuito de extrair um metamodelo para cada esquema. *Schemol* é uma DSL projetada pra extrair metamodelos de um esquema de banco de dados. Os modelos resultantes do processo anterior servem como entrada para o *EMFCompare*[6], que é responsável por efetuar comparações entre dois artefatos Ecore[7]. A saída desta comparação é um novo metamodelo chamado *difference*, o qual contem a diferença entre o esquema velho e o novo. Por fim, *difference* torna-se alvo de transformações M2T por meio das instruções *MOFScript*[8] e o *Adapter* retorna o código SQL de acordo com a modificação. O *MOFScript* é um gerador de código baseado em *templates* que utiliza declarações *print* para gerar código e retornar elementos do modelo.

O papel do *Adapter* consiste em sobrepor a instrução *print* com uma chamada customizada *printSQL*. Ressalta-se que este algoritmo também é responsável por identificar mudanças nas tabelas presentes na declaração SQL. Caso alguma mudança seja detectada, o *Adapter* retorna o código SQL de acordo com o novo esquema (GARCíA; DíAZ OSCAR E CABOT, 2014).

Figura 9 – Processo genérico de coevolução proposto por García e Díaz Oscar e Cabot (2014)



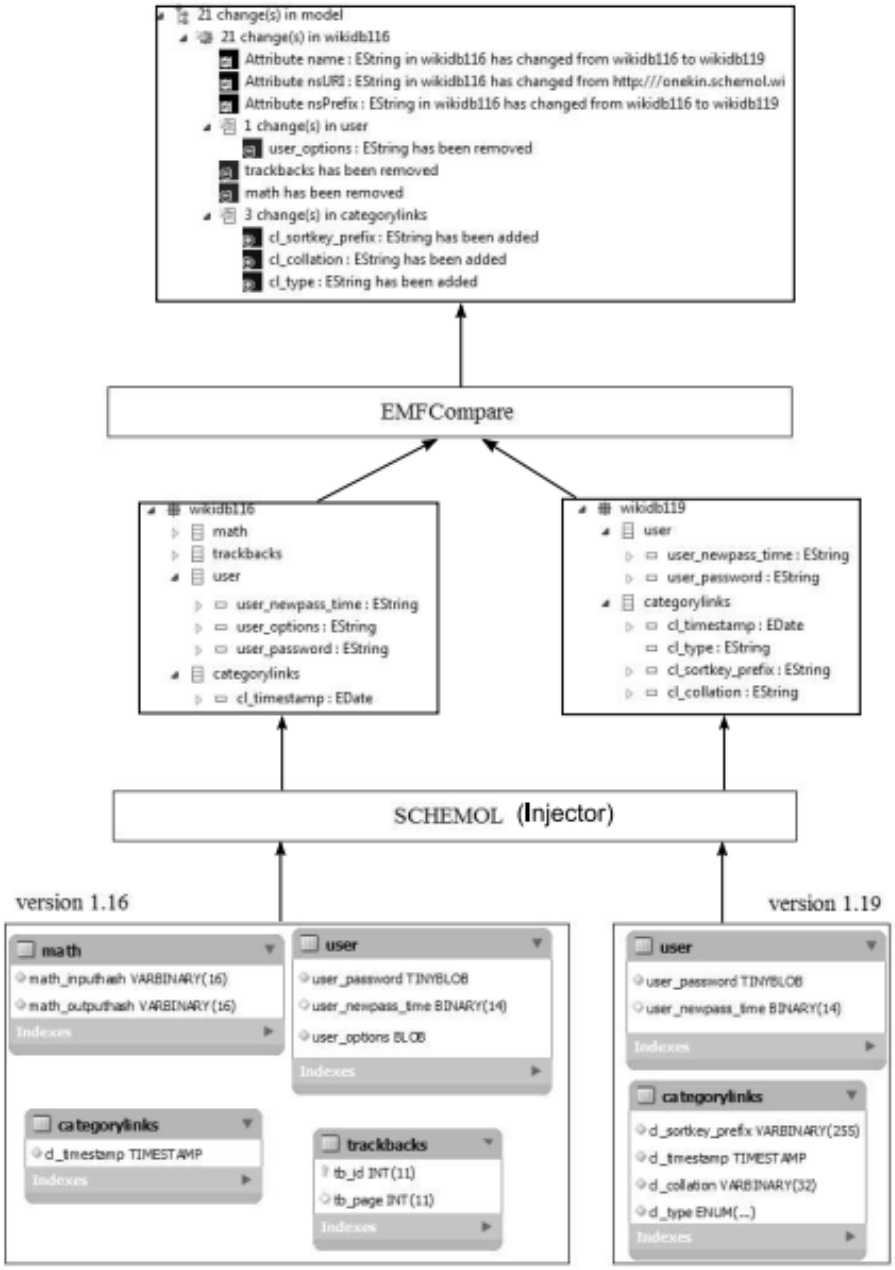
Fonte: (GARCíA; DíAZ OSCAR E CABOT, 2014)

Na Figura 10 observa-se o processo de refatoração em ação, no qual executa-se a remoção das tabelas *MATH* e *TRACKBACKS*, além da coluna *USER_OPTIONS* da tabela *USER*. Também é possível perceber a adição das colunas *CL_SORTKEY_PREFIX*, *CL_COLLATION* e *CL_TYPE* na tabela *CATEGORYLINKS*. Após identificar as alterações nas instâncias das classes, que são representadas pelas instâncias dos elementos do

[5] http://www.onekin.org/portal/schemol
[6] https://www.eclipse.org/emf/compare/
[7] http://www.eclipse.org/ecoretools/doc/
[8] https://marketplace.eclipse.org/content/mofscript-model-transformation-tool

metamodelo, o componente *EMFCompare* identifica as distinções e o algoritmo de transformação *MOFScript* junto com o *Adapter* submete as mudanças para o banco de dados. Observando este processo, fica claro que os metamodelos são gerados dinamicamente e são utilizados somente como suporte ao processo de tradução feito pelo *Adapter*.

Figura 10 – Comparação entre metamodelos que representam as relações em evolução.



Fonte: (GARCíA; DíAZ OSCAR E CABOT, 2014)

Ao analisar este trabalho, entende-se que é possível obter resultados semelhantes às seguintes refatorações: *Drop Colmn*, *Drop Table*, *Move Column*, *Merge Columns*, *Rename Column*, *Rename Table*, *Replace Column* e *Split Table*. Identifica-se, também, a utilização de MDD por meio da definição de um metamodelo *Ecore*. No entanto, nenhuma sintaxe concreta e semântica estática foi apresentada. Esta última característica é impor-

tante, uma vez que o *EMFCompare* compara instâncias de metaclasses e a ausência de validações estruturais pode gerar inconsistências ao evoluir esquemas. Ainda que utilize um metamodelo *Ecore*, o trabalho não define uma DSML. Por outro lado, a Engenharia Reversa se mostra presente na abordagem, como pode ser visto na Figura 9 quando o *Schemol* traduz o esquema do banco de dados em modelo *Ecore*. O autor também apresenta em seu trabalho a Engenharia Direta e a Migração de Dados feita pelo *Adapter*. Por fim, a proposta não esta disponível para acesso, o que inviabiliza a análise das características totais da ferramenta. Sendo assim, torna-se impossível saber se o trabalho leva em consideração um Período Transicional. No Quadro 3.4 apresenta-se uma análise detalhada.

Quadro 3 – Avaliação consolidada do trabalho proposto por García e Díaz Oscar e Cabot (2014)

Característica	**García e Díaz Oscar e Cabot (2014)**
Drop Column	SIM
Drop Table	SIM
Drop View	NÃO
Introduce Calculated Column	NÃO
Introduce Surrogate Key	NÃO
Merge Columns	NÃO
Merge Tables	NÃO
Move Column	SIM
Rename Column	SIM
Rename Table	SIM
Rename View	NÃO
Replace LOB with Table	NÃO
Replace Column	SIM
Replace One-to-Many with Associative Table	NÃO
Replace Surrogate Key with Natural Key	NÃO
Split Column	NÃO
Split Table	SIM
Período Transicional	NÃO
Sintaxe Abstrata(Metamodelo)	SIM
Sintaxe Concreta(Notação)	NÃO
Semântica Estática	NÃO
Engenharia Reversa	SIM
Engenharia Direta	SIM
Migração de Dados	SIM

Fonte:(Autor, 2018)

3.2.3 Milovanovic & Milicev

Milovanovic Vukasin e Milicev (2015) aplica uma abordagem semelhante as apresentadas em Aboulsamh Mohammed A. e Davies (2010) e García e Díaz Oscar e Cabot (2014) para promover a coevolução das classes de entidade da aplicação para as tabelas do banco de dados.

Esta proposta é integrada ao *Framework SOLolist*[9], que utiliza-se da ferramenta de modelagem visual StarUML[10] para criar as classes modeladas pelo usuário e transformá-las em artefatos da aplicação. O destaque deste trabalho está no uso de *Database Modification Primitive* (DMP), que são operações embutidas no algoritmo responsável por identificar as mudanças nas classes (MILOVANOVIC VUKASIN E MILICEV, 2015). As DMPs permitem as seguintes operações: *createTable(tableDesc)*, *renameTable(oldTableName,newTableName)*, *deleteTable(tableName)*, *addColumn(columnDesc)*, *deleteColumn(tableName, columnName)*, *modifyColumn(tableName, oldColumnDesc, newColumnDesc)*, *addForeignKey(columnDesc, FKColumnDesc)*, *dropForeignKey(olumnDesc)*, *insertData(tableName, values)*, *deleteData(tableName, condition)* e *updateData(tableName, columnName, value, condition)* (MILOVANOVIC VUKASIN E MILICEV, 2015).

A principal contribuição deste trabalho é o algoritmo responsável por comparar os pacotes que contém as classes. Ao detectar alterações nestes pacotes, o algoritmo compara a representação antiga do modelo com nova, iterando sobre os pacotes e buscando mudanças efetuadas nas classes. Caso sejam detectadas modificações que o algoritmo de comparação não consiga resolver, o usuário é requisitado pela ferramenta para marcar quais tipos de transformações serão aplicadas nas classes (MILOVANOVIC VUKASIN E MILICEV, 2015). Desse modo, ele por intervir nos casos onde um atributo movido para uma classe que já possui uma propriedade com mesmo nome, solicitando que o usuário escolha se continua a modificação.

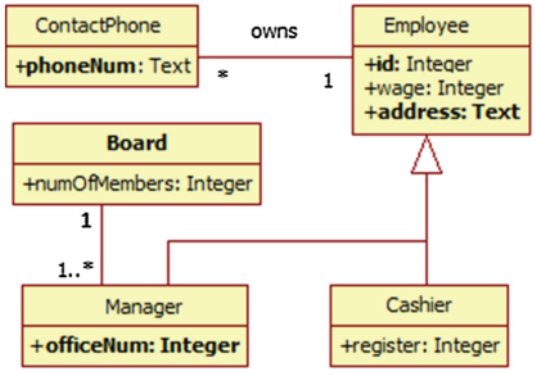
Na Figura 11 apresenta-se um modelo de classes modelado na ferramenta StarUML. Neste exemplo, observa-se que as seguintes refatorações podem ser efetuadas: *Rename Column*, *Rename Table*, *Move Column* e *Drop Column*. A primeira, ocorre na classe *ContactPhone* com a mudança no nome da propriedade *phoneNum* (a) para *number* (b). Já na classe *Employee* a mudança ocorreu na propriedade *id* (a) que mudou para *identification* (b). A segunda evolução ocorre quando o nome da classe *Board* (a) muda para *Committee* (b). A terceira, move a coluna *officeNum* (a) da classe *Manager* para *Employee* (b). A ultima materializa-se ao remover a propriedade *address* da classe *Employee*.

Dada a lista de operações DMPs apresentadas no trabalho, destacam-se algumas deficiências, como: 1) a ausência de mecanismos para efetuar fusão de atributos (*Merge Columns*) ou a introdução de uma coluna calculada (*Introduce Calculated Column*); 2) a falta de suporte para particionar (*Split Table*) ou fundir(*Merge Table*) classes; 3) o suporte às refatorações *Split Column*, *Replace LOB with Table*, *Introduce Surrogate Key*, *Replace One-to-Many with Associative Table*, *Replace Surrogate Key with Natural Key*, *Split Column* e *Split Table*; 4) a falta de evidências que corroborem a definição de um Período Transicional; e 5) a falta de regras de boa formação para as evoluções, o que pode gerar construções inválidas no modelo de classes, as quais podem ser propagadas para as tabe-

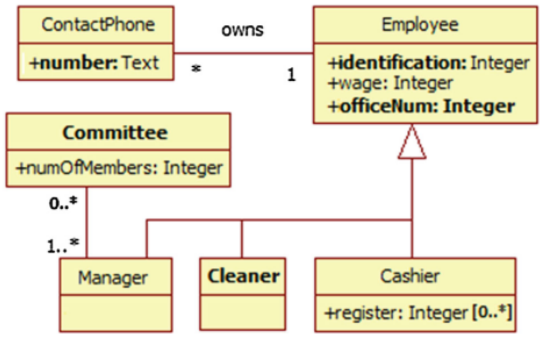
[9] https://www.soloist4uml.com/
[10] http://staruml.io/

Figura 11 – Exemplo dado por Milovanovic Vukasin e Milicev (2015) do modelo de classes antes e após evolução.

(a) Versão antiga do modelo de classe



(b) Nova versão do modelo de classe



Fonte: (MILOVANOVIC VUKASIN E MILICEV, 2015)

las. Além disso, observa-se que o autor utiliza MDD, auxiliada pela ferramenta *SOList*, a qual é responsável pelo algoritmo que faz a distinção entre modelos de classe. O diagrama de classes UML serve como notação (sintaxe concreta) para fazer o mapeamento das classes para as relações presentes no banco de dados. Por outro lado, nenhuma Engenharia Reversa é apresentada. Contudo, a Engenharia Direta e a Migração de dados mostram-se presentes. No Quadro 3.5, apresentam-se quais refatorações poderiam ser efetuadas ao aplicar as evoluções presentes na proposta.

Quadro 4 – Avaliação do trabalho proposto por Milovanovic Vukasin e Milicev (2015)

Característica	**Milovanovic Vukasin e Milicev (2015)**
Drop Column	SIM
Drop Table	SIM
Drop View	NÃO
Introduce Calculated Column	NÃO
Introduce Surrogate Key	NÃO
Merge Columns	NÃO
Merge Tables	NÃO
Move Column	SIM
Rename Column	SIM
Rename Table	SIM
Rename View	NÃO
Replace LOB with Table	NÃO
Replace Column	SIM
Replace One-to-Many with Associative Table	NÃO
Replace Surrogate Key with Natural Key	NÃO
Split Column	NÃO
Split Table	NÃO
Período Transicional	NÃO
Sintaxe Abstrata(Metamodelo)	SIM
Sintaxe Concreta(Notação)	SIM
Semântica Estática	NÃO
Engenharia Reversa	NÃO
Engenharia Direta	SIM
Migração de Dados	SIM

Fonte: (Autor, 2018)

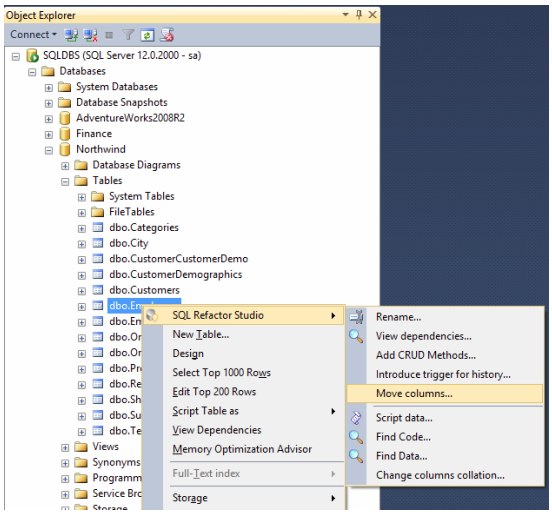
3.2.4 MSSQL Refactor Studio

As ferramentas comerciais de gerenciamento de bancos de dados relacionais possuem um suporte limitado ao processo de evolução de bancos de dados (MILOVANOVIC VUKASIN E MILICEV, 2015). Porém, *MSSQL Refactor Studio*[11] torna-se relevante nesta análise por viabilizar a aplicação dos padrões de refatoração descritas por Ambler Scott W. e Sadalage

[11] https://sqlrefactorstudio.com/

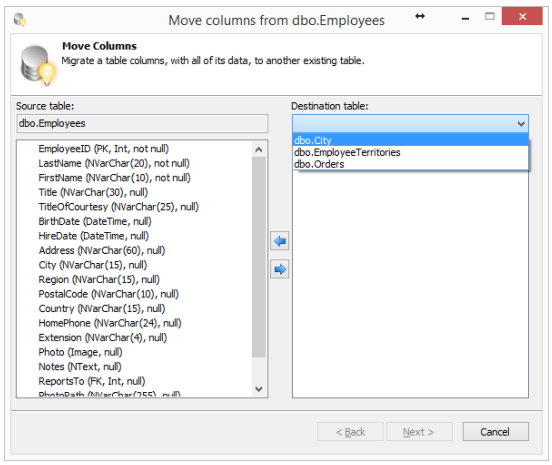
(2006) utilizando uma ferramenta visual. Esta ferramenta é um plugin para o *SQL Server Management Studio*[12], que entrega várias facilidades para o usuário do *Microsoft SQL Server*[13]. Dentre elas está a possibilidade de promover refatorações. Na Figura 12, observa-se como acionar as refatorações dispostas na ferramenta. Já nas Figuras 13, 14 e 15 ilustra-se como o usuário aplica a refatoração *Move Column*.

Figura 12 – Refatoração *Move Column* na tabela DBO.EMPLOYEE com *MSSQL Refactor*



Fonte: sqlrefactorstudio.com (2018)

Figura 13 – Definição da tabela que irá receber a nova coluna

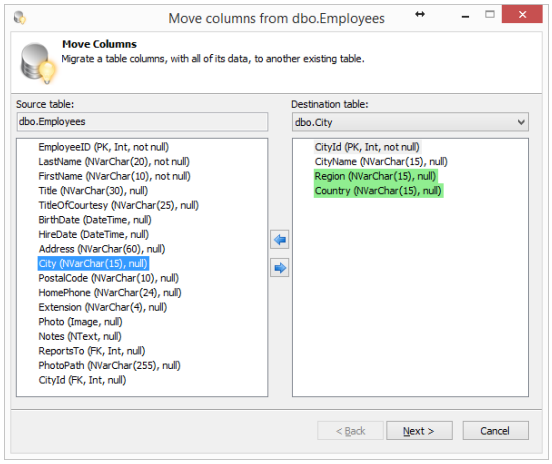


Fonte: sqlrefactorstudio.com (2018)

Além de disponibilizar algumas refatorações estruturais como *Rename Table*, *Rename Column*, *Rename View* e *Move Column*, outras refatorações de outras categorias também podem ser efetuadas, como *Add Lookup Table*, *Introduce Trigger For History* e *Add Crud Methods*. Estes padrões são da categoria de Qualidade de Dados, Integridade Referencial

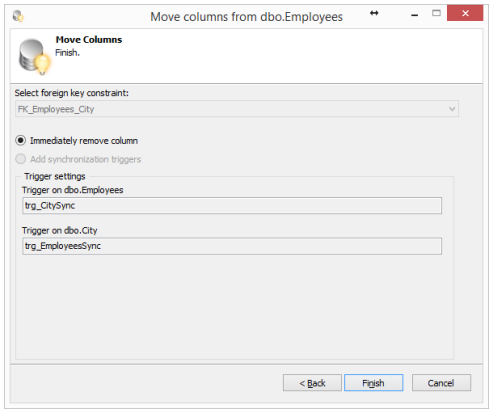
[12] https://docs.microsoft.com/pt-br/sql/ssms/sql-server-management-studio-ssms
[13] https://www.microsoft.com/pt-br/sql-server

Figura 14 – Definição das colunas que serão movidas



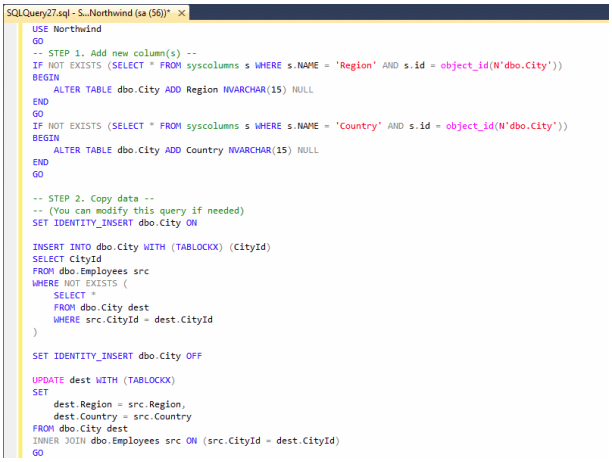
Fonte: sqlrefactorstudio.com (2018)

Figura 15 – Definição do Período Transicional.



Fonte: sqlrefactorstudio.com (2018)

Figura 16 – Código gerado pela MSSQL Refactor Studio



```
SQLQuery27.sql - S...Northwind (sa (56))*
USE Northwind
GO
-- STEP 1. Add new column(s) --
IF NOT EXISTS (SELECT * FROM syscolumns s WHERE s.NAME = 'Region' AND s.id = object_id(N'dbo.City'))
BEGIN
    ALTER TABLE dbo.City ADD Region NVARCHAR(15) NULL
END
GO
IF NOT EXISTS (SELECT * FROM syscolumns s WHERE s.NAME = 'Country' AND s.id = object_id(N'dbo.City'))
BEGIN
    ALTER TABLE dbo.City ADD Country NVARCHAR(15) NULL
END
GO

-- STEP 2. Copy data --
-- (You can modify this query if needed)
SET IDENTITY_INSERT dbo.City ON

INSERT INTO dbo.City WITH (TABLOCKX) (CityId)
SELECT CityId
FROM dbo.Employees src
WHERE NOT EXISTS (
    SELECT *
    FROM dbo.City dest
    WHERE src.CityId = dest.CityId
)

SET IDENTITY_INSERT dbo.City OFF

UPDATE dest WITH (TABLOCKX)
SET
    dest.Region = src.Region,
    dest.Country = src.Country
FROM dbo.City dest
INNER JOIN dbo.Employees src ON (src.CityId = dest.CityId)
GO
```

Fonte: sqlrefactorstudio.com (2018)

e Arquitetural, respectivamente. A ferramenta é aderente à proposta de Ambler Scott W. e Sadalage (2006), pois suporta a definição de um Período Transicional. Além disso, *MSSQL Refactor* também fornece suporte à Engenharia Reversa, Engenharia Direta e a

Migração de Dados. Vale ressaltar que a ferramenta não utiliza MDD como paradigma de construção e também não são apresentados evidências de pontos de extensão para que novos padrões possam ser adicionados. No Quadro 3.7, mostram-se os detalhes da avaliação do *MSSQL Refactor Studio*.

Quadro 5 – Avaliação da ferramenta MSSQL Refactor Studio

Característica	**MSSQL Refactor Studio**
Drop Column	NÃO
Drop Table	NÃO
Drop View	NÃO
Introduce Calculated Column	NÃO
Introduce Surrogate Key	NÃO
Merge Columns	NÃO
Merge Tables	NÃO
Move Column	SIM
Rename Column	SIM
Rename Table	SIM
Rename View	SIM
Replace LOB with Table	NÃO
Replace Column	NÃO
Replace One-to-Many with Associative Table	NÃO
Replace Surrogate Key with Natural Key	NÃO
Split Column	NÃO
Split Table	NÃO
Período Transicional	SIM
Sintaxe Abstrata(Metamodelo)	NÃO
Sintaxe Concreta(Notação)	NÃO
Semântica Estática	NÃO
Engenharia Reversa	SIM
Engenharia Direta	SIM
Migração de Dados	SIM

Fonte: (Autor, 2018)

3.2.5 Curino et. al.

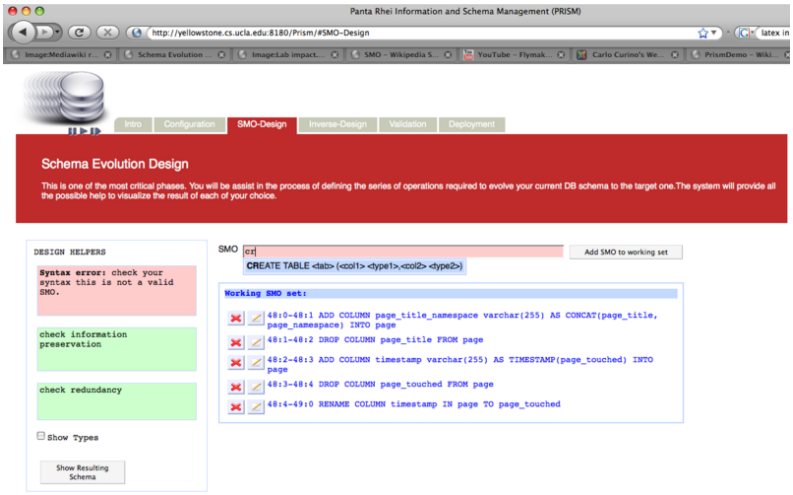
O *Panta Rhei Information Schema Manager* (PRISM) é um projeto que busca desenvolver métodos e ferramentas para evolução de bancos de dados. Como resultado, uma ferramenta de mesmo nome foi desenvolvida no intuito de tornar o processo mais controlado e previsível, buscando reduzir o período em que o banco de dados necessita ter o seu serviço parado e restabelecido após reparo (CURINO et al., 2013). Para cumprir com este propósito, dados foram coletados e analisados utilizando um sistema chamado *Schema Evolution Benchmark* (SEB) resultando em uma coleção de dados históricos relacionados às evoluções aplicadas em bancos de dados de projetos *open source* e científicos (CURINO et al., 2013). Outro produto do SEB são os *Schema Modification Operations* (SMOs).

SMOs são operações que podem expressar quase todas as transformações detectadas na aplicação do SEB e possui sintaxe semelhante às declarações SQL padrão (CURINO et al., 2013). As seguintes operações estão contidas no conjunto de SMOs: *Add Column*, *Drop Column*, *Rename Column*, *Copy Column*, *Create Table*, *Drop Table*, *Rename Table*, *Copy Table*, *Join Table*, *Decompose Table(Vertical)*, *Merge Table(Union)*, *Partition Table(Horizontal)*, *Add Primary Key*, *Add Foreign Key*, *Drop Foreign Key*. Vale ressaltar que algumas destas operações não são de caráter estrutural, mas sim de Integridade Referencial como *Add Foreign Key* e *Drop Foreign Key*. Já *Copy Table* possui comportamento semelhante à refatoração *Add Mirror Table* e está situado na categoria das Refatorações Arquiteturais. *Decompose table* possui semelhança com a refatoração *Move Data* presente na categoria dos padrões de Qualidade de Dados. Uma vez que estas refatorações não pertencem a categoria de refatorações estruturais, estão fora do escopo deste trabalho. O restante das SMOs possuem comportamento semelhante aos padrões estruturais, havendo somente algumas distinções na nomenclatura, como no caso do SMO *Partition Table* que equivale ao *Split Table*.

Como apresentado na Figura 17, o usuário do PRISM utiliza uma interface gráfica para enviar as SMOs para o *PRISM Server*. Ao receber os comandos recebidos pela interface web da ferramenta, o componente se encarrega de submeter as alterações para o esquema presente no banco de dados. A ferramenta também permite efetuar operações inversas (e.g., *Join Table* e *Decompose Table*) e reescrita de consultas. Na Figura 17 ilustra-se como o usuário pode submeter as instruções para evoluir o esquema.

Esta abordagem não define uma DSML, mas sim uma DSL para representar as SMOs. Além disso, não utiliza MDD como alicerce para construção da ferramenta. Apesar de ser uma ferramenta que pode servir no suporte à refatoração, algumas modificações estruturais não são suportadas pelo PRISM, como pode ser visto no Quadro 5. Outras características ausentes são a possibilidade de Engenharia Reversa e o suporte ao Período Transicional, porém a Engenharia Direta e a Migração de Dados estão presentes na proposta. Além disso, a versão demo encontra-se indisponível, impossibilitando que uma análise mais profunda de suas capacidades seja efetuada. Uma análise completa em

Figura 17 – Interface Gráfica do PRISM



Fonte: (CURINO et al., 2013)

relação à aderência aos critérios de avaliação encontra-se no no Quadro 5.

Quadro 6 – Avaliação do trabalho proposto por Curino et al. (2013)

Característica	**Curino et al. (2013)**
Drop Column	SIM
Drop Table	SIM
Drop View	NÃO
Introduce Calculated Column	NÃO
Introduce Surrogate Key	NÃO
Merge Columns	NÃO
Merge Tables	SIM
Move Column	SIM
Rename Column	SIM
Rename Table	SIM
Rename View	NÃO
Replace LOB with Table	NÃO
Replace Column	SIM
Replace One-to-Many with Associative Table	NÃO
Replace Surrogate Key with Natural Key	NÃO
Split Column	NÃO
Split Table	SIM
Período Transicional	NÃO
Sintaxe Abstrata(Metamodelo)	NÃO
Sintaxe Concreta(Notação)	NÃO
Semântica Estática	NÃO
Engenharia Reversa	NÃO
Engenharia Direta	SIM
Migração de Dados	SIM

Fonte: (Autor, 2018)

Na próxima seção, os dados são apresentados de forma consolidada, fazendo-se um comparativo no nível de cobertura de cada trabalho selecionado e da ferramenta *MSSQL Refactor Studio* em relação as critérios de avaliação.

3.3 ANÁLISE DOS TRABALHOS

Nesta seção, sumariza-se a análise dos trabalhos selecionados. No Quadro 6 apresenta-se uma matriz consolidada que apontando os pontos fortes e fracos de cada trabalho de acordo com os critérios dispostos na Seção 3.1. Algumas destas características são consideradas essenciais, como a Engenharia Reversa, Engenharia Direta, o Período Transicional e a Migração de Dados, pois estão diretamente relacionados com evolução de bancos de dados.

Nos dados contidos no Quadro 6 destacam-se que: 1) 48% dos critérios analisados estão presentes no trabalho de Aboulsamh Mohammed A. e Davies (2010); 2) o trabalho de García e Díaz Oscar e Cabot (2014) cobre 44% dos critérios; 3) a abordagem apresentada por Curino Carlo e Moon (2009) (PRISM) não utiliza MDD, porém fornece cobertura a 40% dos critérios analisados; 4) Milovanovic Vukasin e Milicev (2015) contemplam 40% dos critérios; 5) a ferramenta *MSSQL Refactor Studio* possui aderência a 32% dos critérios; 6) as refatorações *Rename Table*, *Move Column* e *Rename Column* estão presente em 100% dos trabalhos. Desse modo, pode-se considerar estas alterações no esquema como fundamentais para os trabalhos relacionados à evolução de bancos de dados; 7) as refatorações *Drop Table*, *Drop Column*, *Replace Column* aparecem em 80% dos trabalhos, sendo consideradas como critérios bastante relevantes para os trabalhos que promovem evoluções no esquema; 8) a refatoração *Split Table* pode ser realizada por 60% dos trabalhos, o que ainda pode ser considerado relevante para quem aplica alterações nos bancos de dados; 9) apenas 20% das abordagens dão cobertura *Merge Tables* e *Rename View*; 10) o suporte ao Período Transicional foi encontrado em apenas 20% dos trabalhos; 11) nenhuma trabalho viabiliza a implementação do *Drop View*, *Replace LOB With Table*, *Introduce Calculated Column*, *Merge Columns*, *Introduce Surrogate Key*, *Replace Surrogate Key with Natural Key*, *Split Column* e *Replace One-to-Many with Associative Table*; 12) 60% dos trabalhos definem uma Sintaxe Abstrata e 20% utilizam Semântica Estática para o metamodelo; 13) 40% dos trabalhos consideram a definição de uma Sintaxe Concreta. O que destaca a escassez de ferramentas que utilizam MDD para resolver problemas relacionados à evolução de bancos de dados; 14) 40% dos trabalhos efetuam Engenharia Reversa e 100% dos trabalhos viabilizam tanto realização da Engenharia Direta quanto a Migração de Dados, reafirmando a importância destas características no processo de evolução de banco de dados; e, por fim, 15) com a exceção das refatorações *Replace LOB With Table* e *Split Column*, todas as características presentes nesta análise encontram-se neste trabalho, que tem como diferenciais as refatorações *Drop View*, *Introduce Calculated Column*, *Introduce Surrogate key*, *Merge Columns*, *Replace One to Many with Associative Table* e *Replace Surrogate Key with Natural Key*, além do Período Transicional.

Quadro 6 – Analise consolidada dos trabalhos acadêmicos e da ferramenta *MSSQL Refactor Studio*.

Característica	Aboulsamh & Davies	Garcia et al.	Milovanovic & Milicev	*MSSQL Refactor*	Curino et. Al.	
Drop Column	SIM	SIM	SIM	NÃO	SIM	80%
Drop Table	SIM	SIM	SIM	NÃO	SIM	80%
Drop View	NÃO	NÃO	NÃO	NÃO	NÃO	0%
Introduce Calculated Column	NÃO	NÃO	NÃO	NÃO	NÃO	0%
Introduce Surrogate Key	NÃO	NÃO	NÃO	NÃO	NÃO	0%
Merge Columns	NÃO	NÃO	NÃO	NÃO	NÃO	0%
Merge Tables	NÃO	NÃO	NÃO	NÃO	SIM	20%
Move Column	SIM	SIM	SIM	SIM	SIM	100%
Rename Column	SIM	SIM	SIM	SIM	SIM	100%
Rename Table	SIM	SIM	SIM	SIM	SIM	100%
Rename View	NÃO	NÃO	NÃO	SIM	NÃO	20%
Replace LOB with Table	NÃO	NÃO	NÃO	NÃO	NÃO	0%
Replace Column	SIM	SIM	SIM	NÃO	SIM	80%
Replace One-to-Many with Associative Table	NÃO	NÃO	NÃO	NÃO	NÃO	0%
Replace Surrogate Key with Natural Key	NÃO	NÃO	NÃO	NÃO	NÃO	0%
Split Column	NÃO	NÃO	NÃO	NÃO	NÃO	0%
Split Table	SIM	SIM	NÃO	NÃO	SIM	60%
Periodo Transicional	NÃO	NÃO	NÃO	SIM	NÃO	20%
Sintaxe Abstrata(Metamodelo)	SIM	SIM	SIM	NÃO	NÃO	60%
Sintaxe Concreta(Notação)	SIM	NÃO	SIM	NÃO	NÃO	40%
Semântica Estática	SIM	NÃO	NÃO	NÃO	NÃO	20%
Engenharia Reversa	NÃO	SIM	NÃO	SIM	NÃO	40%
Engenharia Direta	SIM	SIM	SIM	SIM	SIM	100%
Migração de Dados	SIM	SIM	SIM	SIM	SIM	100%
Cobertura	48%	44%	40%	32%	40%	

Fonte: (Autor ,2018)

3.4 CONSIDERAÇÕES FINAIS

Neste capítulo definem-se os critérios de avaliação que servem para delimitar o escopo da revisão bibliográfica. Além disso, tanto a seleção dos repositórios quanto a definição da *string* de busca que contém as palavras chaves relacionadas compõem o processo. Como parte do levantamento, também avalia-se cada trabalho relevante e ferramenta encontrada, efetuando um comparativo com os critérios de seleção e avaliação definidos. No sentido de apresentar um comparativo entre as propostas, sumarizam-se os trabalhos selecionados, que tem suas características detalhadas para demonstrar o nível de cobertura de cada trabalho e da ferramenta *MSSQL Refactor Studio* em relação aos critérios de avaliação.

No capítulo seguinte são introduzidos os componentes da DSML proposta e detalhadas sua sintaxe concreta, sintaxe abstrata e semântica estática.

4 UMA DSML PARA REFATORAÇÃO DE BANCOS DE DADOS RELACIONAIS

Neste capítulo apresenta-se a DSML proposta neste trabalho. Para isto, detalha-se o metamodelo *EvoDB*, que visa especificar os conceitos e relacionamentos entre estes para implementar uma DSML que ofereça suporte aos padrões de refatoração estruturais propostos por Ambler Scott W. e Sadalage (2006). Esta DSML surge como solução às deficiências encontradas nos trabalhos analisados no Capítulo 3. Para isso, na Seção 4.1, inicia-se apresentando uma visão geral dos objetivos e principais características da abordagem. Após isso, nas Seções 4.2, 4.3 e 4.4, descreve-se a sintaxe concreta (i.e., notação), sintaxe abstrata (i.e., metamodelo) e semântica estática (i.e., regras de boa formação) obtidas por meio do processo de desenvolvimento de DSML definido por Cho, Gray e Syriani (2012). Por fim, encerra-se com a apresentação das considerações finais do capítulo.

4.1 VISÃO GERAL

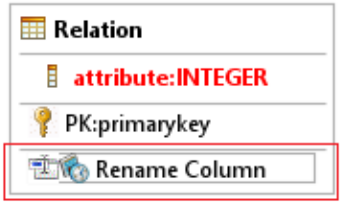
A DSML proposta tem como objetivo viabilizar, por meio de diagramas, a modelagem de refatorações estruturais de bancos de dados. Suas principais características são o suporte às Refatorações Estruturais descritas por Ambler Scott W. e Sadalage (2006), a Engenharia Reversa, a Engenharia Direta, o Período Transicional e a Migração de Dados. Esta linguagem busca reduzir a complexidade da aplicação de Refatorações no esquema do banco de dados, fornecendo subsídios para que profissionais e estudantes possam entender e promover evoluções no esquema de dados por meio de Refatorações.

Estruturalmente, esta DSML foi desenvolvida baseando-se no arcabouço definido por Brambilla, Cabot e Wimmer (2012), o qual consiste na especificação da: 1) sintaxe concreta (i.e., representação visual); 2) sintaxe abstrata (i.e., metamodelo); e 3) semântica estática (i.e., regras de boa formação). Além disso, orienta-se pelo processo de desenvolvimento estabelecido por Cho, Gray e Syriani (2012). Nas próximas seções pode-se observar os aspectos da linguagem resultantes da execução deste processo.

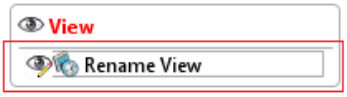
4.2 SINTAXE CONCRETA

A especificação da sintaxe concreta serve como ponto de partida para o desenvolvimento da linguagem proposta e ocorre por meio da extensão à representação visual do modelo relacional (cf. Seção 2.5). Para isto, adiciona-se um compartimento à representação da relação ou visão, permitindo a inserção dos construtores que especificam as refatorações. Na Figura 7 mostra-se o compartimento refatoração tanto na tabela quanto na visão. Ressalta-se que todos os construtores possuem um ícone e um rótulo, como apresentado no Quadro 7. Além disso, destaca-se o rótulo, atributos e/ou restrições da tabela, ou da visão, na cor vermelha para indicar quais elementos servem de entrada na modelagem da

Figura 7 – Compartimento Refatoração



(a) Compartimento refatoração no construtor Relação



(b) Compartimento refatoração no construtor Visão

Fonte:(Autor, 2018)

Quadro 7 – Pictogramas que representam as refatorações

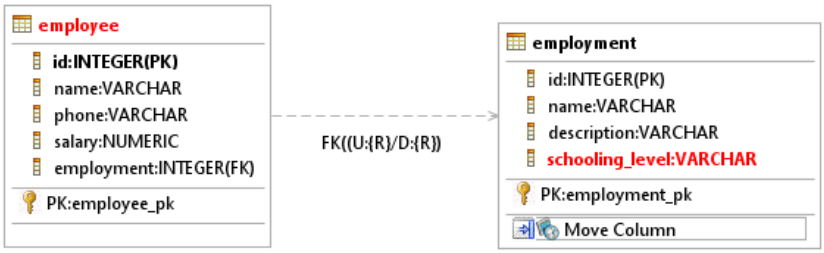
Notação	Refatoração
	*Move Column*
	*Introduce Surrogate Key*
	*Introduce Natural Key*
	*Introduce Calculated Column*
	*Merge Tables*
	*Merge Columns*
	*Extract Associative Table* (i.e., *Replace 1:N With Associative Table*)
	*Rename Column*
	*Replace Column*
	*Rename Table*
	*Rename View*
	*Drop Table*
	Drop View
	*Drop Columns*
	*Split Table*

refatoração. Vale ressaltar que: 1) o ícone de uma tabela contendo um relógio ao lado do rótulo do construtor da refatoração indica o ajuste do período transicional; e 2) nos casos das refatorações *Introduce Surrogate Key*, *Introduce Calculated Column*, *Merge Columns*, *Extract Associative Table*, *Rename Column*, *Replace Column*, *Rename View* e *Split Table*

o rótulo do construtor é substituído pelos valores definidos do usuário.

Na Figura 8 exibe-se o construtor *Move Column* por meio de um exemplo. Nota-se que tanto o atributo *SCHOOLING_LEVEL* da relação *EMPLOYMENT* quanto o rótulo da relação *EMPLOYEE* estão destacados para especificar quais elementos servem como entrada para a refatoração, ou seja, qual coluna deve ser movida e sua tabela destino. Além disso, o modelo deixa claro o ajuste do período transicional, pois pode-se observar o ícone que identifica a opção ao lado do rótulo que contém o nome da refatoração.

Figura 8 – Construtor *Move Column*



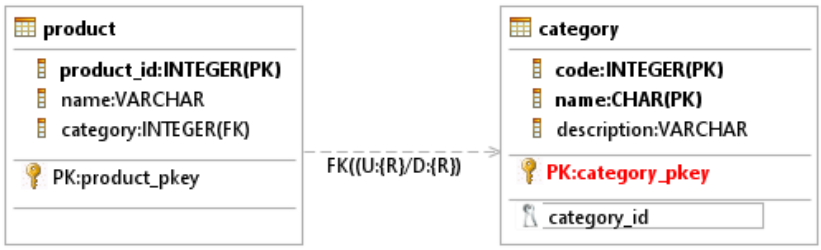
Fonte:(Autor, 2018)

Na Figura 9 apresentam-se os construtores *Introduce Surrogate Key* e *Introduce Natural Key*. O construtor *Introduce Surrogate Key* permite especificar uma chave substituta na tabela. Nota-se que, diferente do exemplo anterior (i.e., Figura 8), o rótulo do construtor é substituído pela coluna que representa a chave substituta. Vale ressaltar que isto ocorre no momento em que a coluna é definida. Além disso, destaca-se o elemento que representa a chave primária (i.e., *CATEGORY_PKEY*), o qual serve como entrada para a refatoração. A especificação da refatoração *Introduce Natural Key* ocorre de modo semelhante. No entanto, rótulo do construtor permanece inalterado e tanto a chave primária (i.e., *DRIVE_PKEY*) quanto a chave natural (i.e., *LICENSE*) formam a entrada da refatoração.

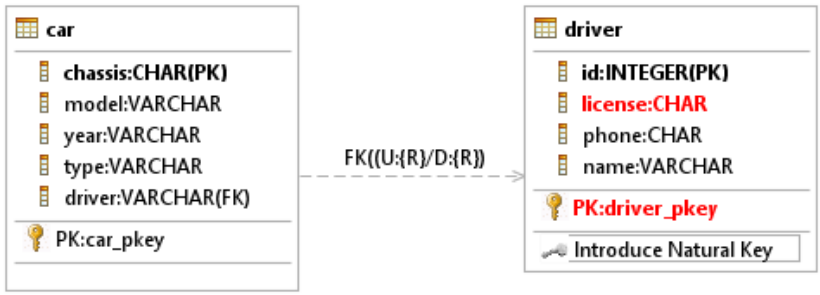
Na Figura 10 mostra-se o construtor *Introduce Calculated Column* o qual permite especificar uma nova coluna para armazenar o resultado de uma função definida pelo usuário. Vale frisar que este construtor tem seu rótulo substituído para manter a especificação da nova coluna junto com a função definida pelo usuário. No exemplo descrito na figura, note que a coluna que serve como entrada para a refatoração(i.e., *QUANTITY*) encontra-se destacada e a nova coluna (i.e., *AMOUNT*) deve armazenar o resultado de uma função (i.e., *SUM(QUANTITY)*).

Na Figura 11 ilustram-se os construtores *Merge Tables* e *Merge Columns*. Observa-se que o primeiro permite especificar os atributos que servem como entrada para a refatoração, bem como a tabela destino por meio da escolha da chave estrangeira. O segundo, especifica o nome da nova coluna e os atributos que compõem sua formação. Note que, em ambos os casos, os construtores que participam da refatoração encontram-se destacados

Figura 9 – Construtores *Introduce Surrogate Key* e *Introduce Natural Key*



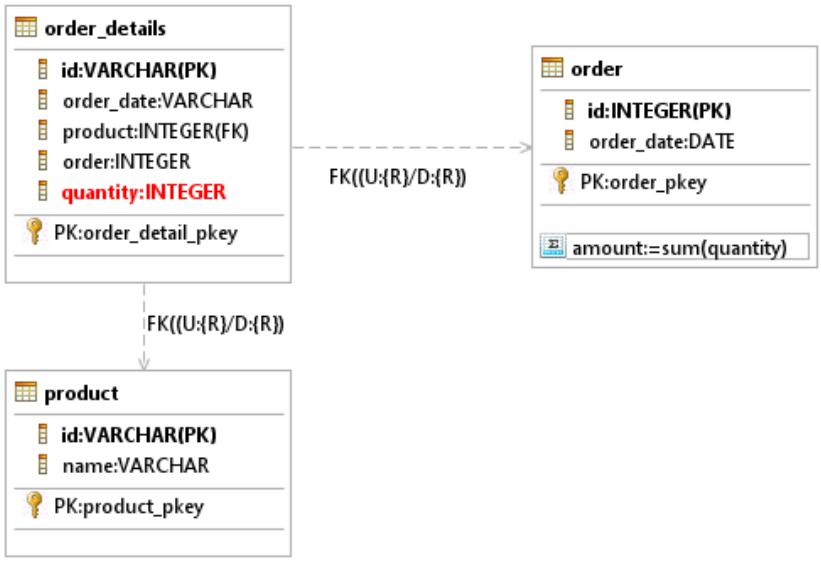
Construtor *Introduce Surrogate Key*



Construtor *Introduce Natural Key*

Fonte:(Autor, 2018)

Figura 10 – Construtor *Introduce Calculated Column*

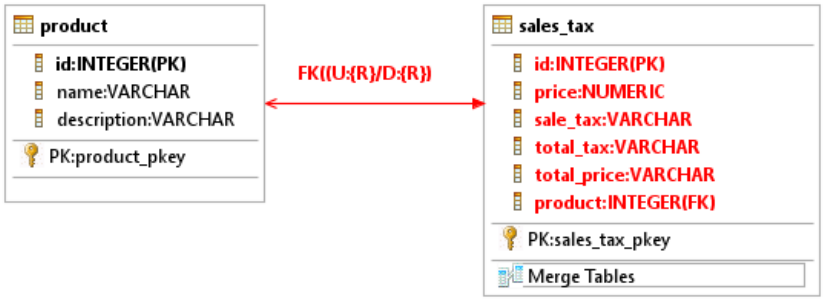


Fonte:(Autor, 2018)

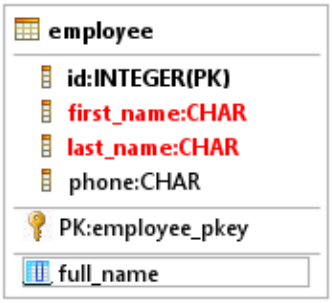
em vermelho. No caso do construtor *Merge Columns*, seu rótulo é substituído pelo nome da nova coluna (i.e., *FULL_NAME*).

Na Figura 12, exibe-se a especificação do construtor *Extract Associative Table*. Neste

Figura 11 – Construtores *Merge Tables* e *Merge Columns*



Construtor *Merge Tables*

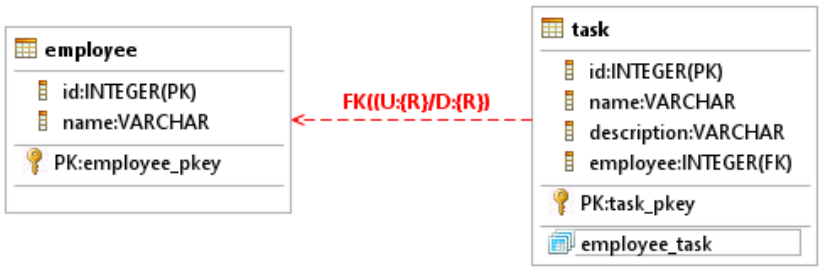


Construtor *Merge Columns*

Fonte:(Autor, 2018)

caso, especificam-se: 1) a chave estrangeira a qual deseja-se extrair a tabela associativa; e 2) o nome desta nova tabela. Ainda sobre a Figura 12, nota-se que o construtor que representa a chave estrangeira (i.e., *link*) encontra-se em destaque para indicar quais relações participam do relacionamento e o rótulo do construtor apresenta o nome da nova tabela associativa (i.e., *EMPLOYEE_TASK*).

Figura 12 – Construtor *Extract Associative Table*

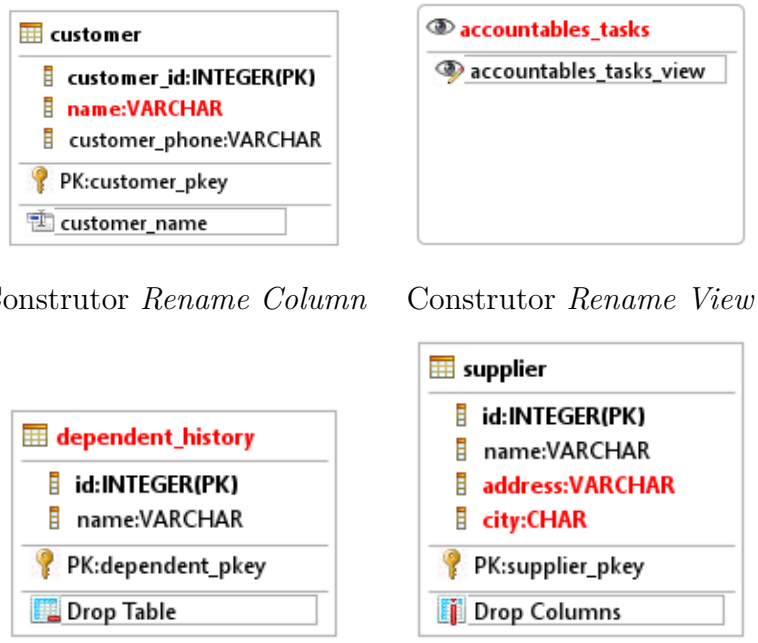


Fonte:(Autor, 2018)

As especificações das refatorações *Rename Column*, *Rename View*, *Drop Table* e *Drop Columns* são efetuadas a partir dos construtores apresentados na Figura 13. Por meio do construtor *Rename Column* especifica-se a coluna que será renomeada (i.e., *NAME*)

e seu novo nome (i.e., *CUSTOMER_NAME*). Observa-se que o atributo tem seu nome destacado e o rótulo do construtor é substituído pelo novo nome especificado. Por sua vez, o construtor *Rename View* permite definir um novo nome para uma visão existente. Nota-se que o rótulo da visão é destacado para indicar o nome atual deste objeto. Já o rótulo do construtor que representa a refatoração, exibe o novo nome da visão (i.e., *ACCOUNTABLES_TASK_VIEW*). Utilizando-se o construtor *Drop Table* destaca-se o rótulo da tabela para indicar sua remoção do banco de dados. Por fim, o construtor *Drop Columns* permite especificar quais colunas devem ser removidas ao aplicar esta refatoração (i.e., *ADDRESS* e *CITY*). Vale ressaltar que a especificação das seguintes refatorações ocorrem de forma idêntica: 1) *Rename Column* e *Replace Column*; 2) *Rename View* e *Rename Table* ; e 3) *Drop Table* e *Drop View*. Por este motivo, *Replace Column*, *Rename Table* e *Drop View* não estão presentes na Figura.

Figura 13 – Construtores *Rename Column* e *Rename View*



Construtor *Rename Column* Construtor *Rename View*

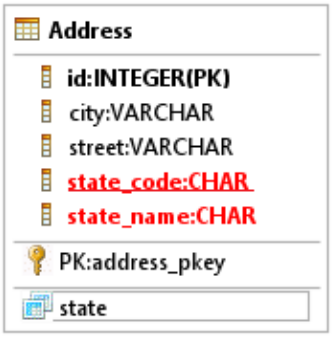
Construtor *Drop Table* Construtor *Drop Columns*

Fonte:(Autor, 2018)

Para especificar a refatoração *Split Table*, utiliza-se o construtor presente na Figura 14. Para isto, definem-se tanto as colunas que compõem a nova tabela quanto as colunas que formam sua chave primária. Especifica-se, também, o nome da tabela gerada como resultado da refatoração. No exemplo presente na figura, nota-se que os atributos que fazem parte da refatoração encontram-se destacados (i.e., *STATE_CODE* e *STATE_NAME*). Além disso, o atributo que compõe a chave primária da nova relação apresenta-se sublinhado (i.e., *STATE_CODE*) e o rótulo do construtor da refatoração é substituído

pelo nome da nova tabela (i.e., *STATE*).

Figura 14 – Construtor *Split Table*



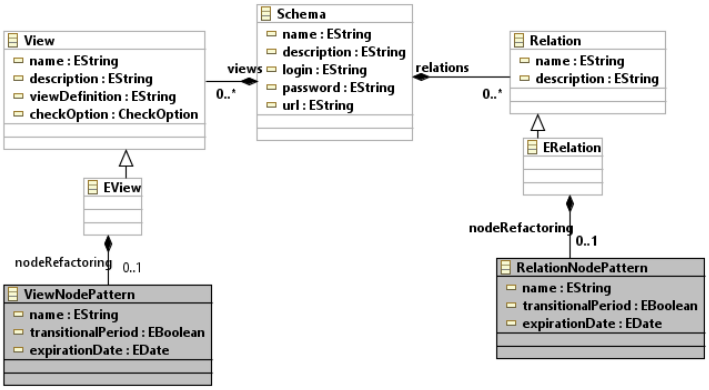
Fonte:(Autor, 2018)

Os construtores supramencionados possibilitam a especificação, de forma diagramática, dos padrões de refatoração estrutural descritas por (AMBLER SCOTT W. E SADALAGE, 2006). Com isto, facilita-se a implementação e o entendimento da aplicação destes padrões por meio da abstração entregue pelos construtores da linguagem.

4.3 SINTAXE ABSTRATA

Nesta seção descreve-se a sintaxe abstrata da linguagem proposta. Para isto, apresentam-se tanto os conceitos quantos as relações válidas entre eles na forma de um metamodelo, o qual tem como objetivo especificar os elementos que compõem as refatorações estruturais de bancos de dados relacionais. Isto viabiliza que uma representação concreta (cf. Seção 4.2) seja fornecida para cada conceito. Este metamodelo é denominado *EvoDB* e, além das classes estendidas do metamodelo *RMM* (cf. Seção 2.5), é composto por 19 metaclasses das quais 2 são abstratas. O metamodelo *EvoDB* é apresentado em 2 partes, que são: 1) metaclasses de integração com o metamodelo *RMM*; e 2) metaclasses relacionadas com a especificação das refatorações aplicadas às tabelas e visões.

Figura 15 – Detalhamento da metaclasse principal da DSML proposta.



Fonte:(Autor, 2018)

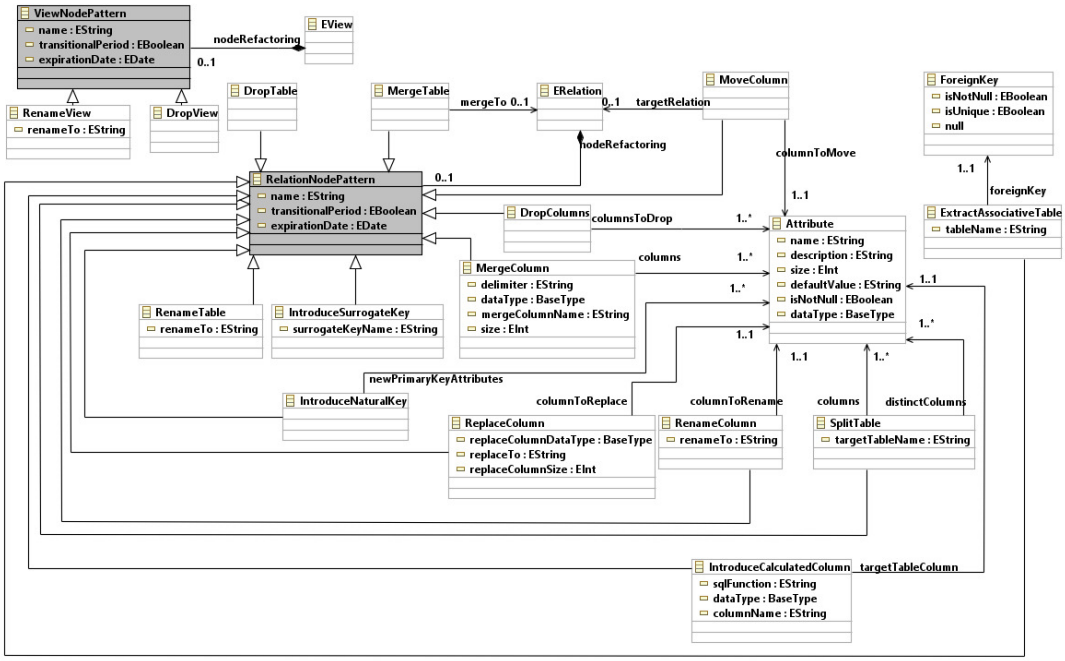
Na Figura 15 apresenta-se a primeira parte do metamodelo onde são descritas as metaclasses relacionadas com os construtores que integram os metamodelos *EvoDB* e *RMM*. Esta parte possui 7 metaclasses das quais: i) 3 metaclasses fazem parte do metamodelo *RMM*(i.e., *Schema*, *Relation* e *View*); ii) 2 metaclasses são especializações que se referem aos construtores fundamentais sobre os quais especificam-se as refatorações (i.e., *ERelation* e *EView*); e iii) 2 metaclasses abstratas que agrupam as características comuns às metaclasses referentes às refatorações estruturais (i.e., *RelationNodePattern* e *ViewNodePattern*). Ressalta-se que: 1) como supramencionado (cf. Seção 2.5), a metaclasse *Schema* é a metaclasse raiz, portanto corresponde a área de desenho onde são especificados os modelos da linguagem; 2) as metaclasses abstratas são identificadas na cor cinza; e 3) as metaclasses estendidas do metamodelo *RMM* são identificadas por seu nome precedido pela letra "E"de *Extends* para reafirmar a relação de extensão. A seguir são apresentadas as 4 metaclasses principais que são: *ERelation*, *EView*, *RelationNodePattern* e *ViewNodePattern*. Note que as relações de composição presentes nas metaclasses *EView* e *ERelation* denotam os compartimentos de refatoração apresentados na sintaxe concreta (cf. Figura 7).

- *ERelation* é a metaclasse que representa construtor *ERelation* na linguagem proposta. Esta metaclasse estende a metaclasse *Relation* do metamodelo *RMM*, portanto herda seus atributos e associações, além de ser composta pela metaclasse *Schema* por meio do compartimento *relations*. Além disso, esta metaclasse possui o compartimento *nodeRefactoring* do tipo *RelationNodePattern*. Isto permite que seu construtor seja composto por uma das representações especializadas de *RelationNodePattern* (cf. Figura 16). Vale ressaltar que, estruturalmente, a metaclasse *EView* possui as mesmas características da metaclasse *ERelation*, porém esta metaclasse representa o construtor *EView* e sua composição, *nodeRefactoring*, é do tipo *ViewNodePattern*.

- *RelationNodePattern* é a metaclasse abstrata que mantém as propriedades referentes a todas as refatorações que se aplicam às relações. As especializações desta metaclasse são instanciadas no compartimento de refatoração do construtor *ERelation* e especializam os seguintes meta-atributos: *name* - que define o nome da refatoração; *transitionalPeriod* - que permite indicar se haverá período transicional; e *expirationDate* - que determina uma data de expiração para o mecanismo que mantém o período transicional (i.e., *Trigger* ou *View*). Note que a metaclasse *ViewNodePattern* possui a mesma definição presente em *RelationNodePattern*, no entanto é instanciada no compartimento de refatoração do construtor *EView* e mantém as propriedades referentes às refatorações que aplicam-se às visões.

Na Figura 16 exibe-se a segunda parte do metamodelo, onde definem-se as metaclasses relacionadas com a especificação das refatorações que se aplicam às relações e visões

Figura 16 – Metaclasses relacionadas com a especificação dos construtores que representam as refatorações aplicadas sobre as Tabelas e Visões.



Fonte:(Autor, 2018)

(i.e.,são instanciadas no compartimento de refatoração de uma *ERelation* ou *EView*). Esta parte possui 15 metaclasses que representam refatorações estruturais com exceção das metaclasses estendidas do metamodelo *RMM* (i.e., *Attribute* e *ForeignKey*) e das metaclasses descritas na primeira parte (i.e., *ERelation*, *EView*, *ViewNodePattern* e *RelationNodePattern*). A seguir são apresentadas as metaclasses contidas neste fragmento:

- *MoveColumn* é a metaclasse que representa o construtor *Move Column*. Esta metaclasse deve manter as seguintes referências: *columnToMove* - que mantém o atributo (i.e., coluna) que será movido; e *targetRelation* - que referencia a relação (i.e., tabela) destino do atributo.

- As metaclasses *IntroduceSurrogateKey* e *IntroduceNaturalKey* referem-se, respectivamente, ao construtor que permite inserir uma chave substituta e inserir uma chave natural. A metaclasse *IntroduceSurrogateKey* possui o meta-atributo *surrogateKeyName* que viabiliza a definição do nome da chave primária substituta. Já a metaclasse *IntroduceNaturalKey*, mantém uma referencia nomeada *newPrimaryKeyAttributes* que permite a adição de um ou mais atributos para compor a nova chave natural.

- A metaclasse *IntroduceCalculatedColumn* representa o construtor *Introduce Calculated Column* e possui a referência *targetTableColumn* do tipo *Attribute*. Esta referência serve como parâmetro para a função definida pelo usuário do construtor

e é armazenada no meta-atributo *sqlFunction*. Além deste, definem-se os seguintes meta-atributos: *columnName* - que recebe o nome da nova coluna calculada; e *dataType* - que especifica o tipo de dado da nova coluna.

- As metaclasses *MergeTables* e *MergeColumns* viabilizam, por meio dos respectivos construtores, fazer fusão entre tabelas e entre colunas, nesta ordem. A metaclasse *MergeTables* mantém uma referência do tipo *ERelation* nomeada *mergeTo*. Esta referência deve indicar qual a tabela alvo da fusão, ou seja, destino das colunas e restrições da tabela fonte. A metaclasse *MergeColumns* possui uma referência de nome *columns* do tipo *Attribute* para indicar quais colunas serão mescladas. Esta metaclasse também possui os seguintes meta-atributos: *mergeColumnName* - que mantém o nome da nova coluna; *dataType* - que especifica o tipo de dado da nova coluna; *delimiter* - que permite a especificação de um separador para a nova coluna com valores mesclados (e.g., 999-9999); e *size* - que define um tamanho para a nova coluna.

- A metaclasse *ExtractAssociativeTable* representa a refatoração *Replace One-to-Many With Associative Table* por meio do construtor *Extract Associative Table*, resultando em uma tabela associativa. Esta metaclasse possui uma referência nomeada *foreignKey* do tipo *ForeignKey*. Esta referência deve ser definida no construtor para possibilitar a extração do relacionamento entre as tabelas fonte e destino da chave estrangeira. Além disso, esta metaclasse também mantém o meta-atributo *tableName* para especificar o nome da nova tabela que representa o relacionamento.

- As metaclasses *RenameTable*, *RenameColumn* e *ReplaceColumn* permitem, por meio de seus construtores, renomear uma tabela, uma coluna e alterar o nome e tipo de dado da coluna, nesta ordem. A metaclasse *RenameTable*, assim como *RenameColumn*, possui um meta-atributo nomeado *renameTo*. No contexto da primeira, permite renomear a tabela. Na segunda, possibilita renomear uma coluna. A metaclasse *RenameColumn* também mantém uma referência, cujo nome é *columnToRename* e tipo *Attribute*, para especificar qual coluna deve ser renomeada. A metaclasse *ReplaceColumn* também possui uma referência do tipo *Attribute*. Esta referência, nomeada *columnToReplace*, especifica qual coluna deve ser substituída. Além disso, esta metaclasse possui os seguintes meta-atributos: *replaceTo* - que define o nome da nova coluna; *replaceColumnDataType* - que especifica o tipo de dado da nova coluna; e *replaceColumnSize* - para especificar o tamanho da nova coluna.

- As metaclasses *RenameView* e *DropView* viabilizam, por intermédio dos respectivos construtores, renomear e remover uma visão. A metaclasse *RenameView* possui somente o meta-atributo *renameTo* para especificar o novo nome da visão. Já a

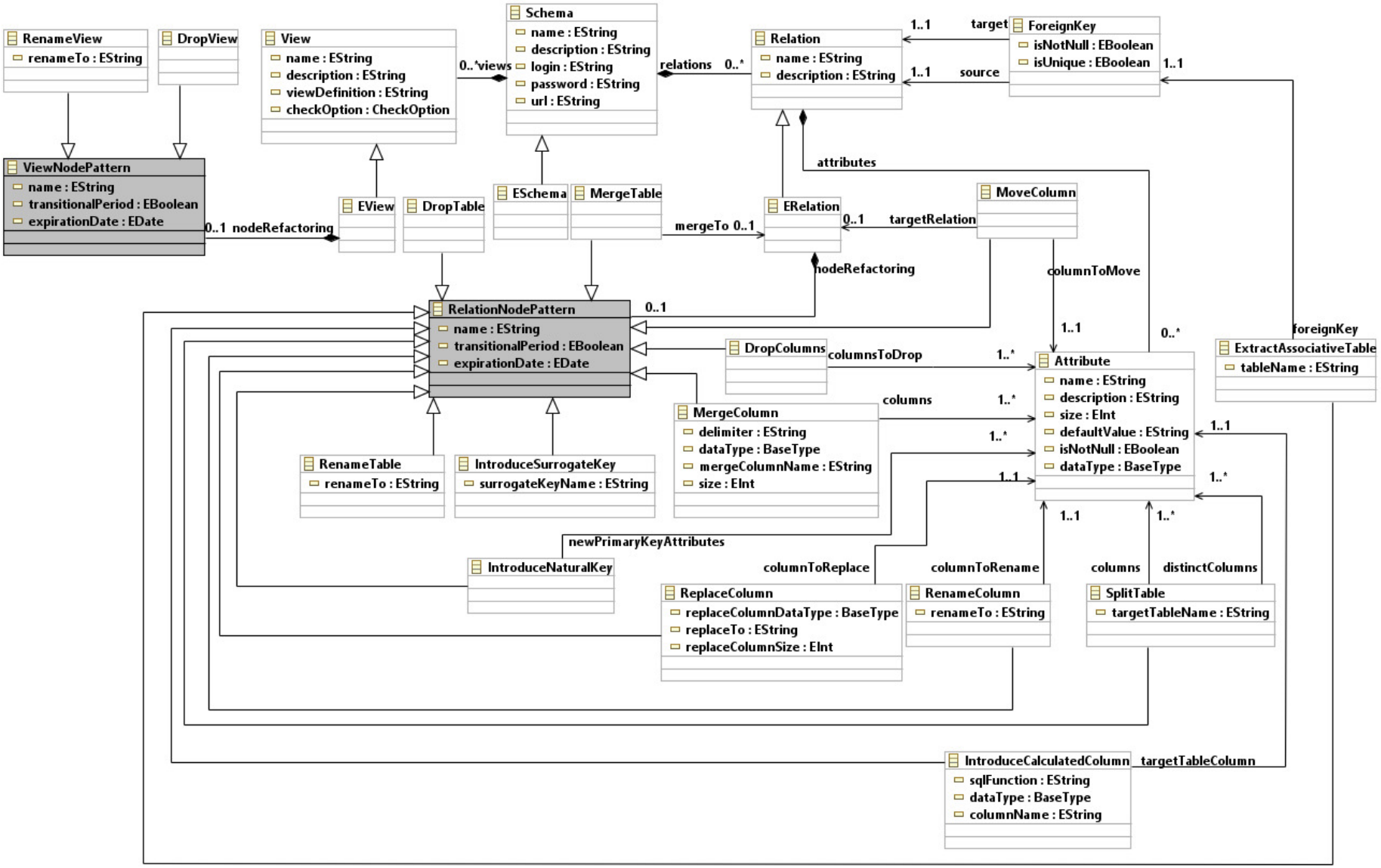
metaclasse *DropView* não possui propriedades e toma como referência a visão (i.e., construtor *EView*) que mantém a sua instância no compartimento.

- As metaclasses *DropTable* e *DropColumns* referem-se aos construtores que permitem remover uma tabela e remover colunas, respectivamente. A metaclasse *DropTable* não possui atributo próprio e, assim como as demais especializações de *RelationNodePattern*, toma como referência a relação que o mantém no compartimento. A metaclasse *Drop Columns* também não possui atributo próprio, porém especifica, por meio de uma referência, quais colunas devem ser removidas. Esta referência tem como especificação o nome *columnsToDrop* e tipo *Attribute*.

- A metaclasse *SplitTable* permite especificar a divisão de uma tabela por meio do construtor *Split Table*. Esta metaclasse possui o meta-atributo *targetTableName* para definir o nome da nova tabela. Além disso, mantém duas referências que são: *columns* - que especifica quais colunas serão remanejadas pra a nova tabela; e *distinctColumns* - que define quais colunas devem formar a chave primária da nova tabela.

Na Figura 17, monstra-se o metamodelo *EvoDB* em sua completude.

Figura 17 – Metaclasses relacionadas com a especificação dos construtores fundamentais da linguagem proposta.



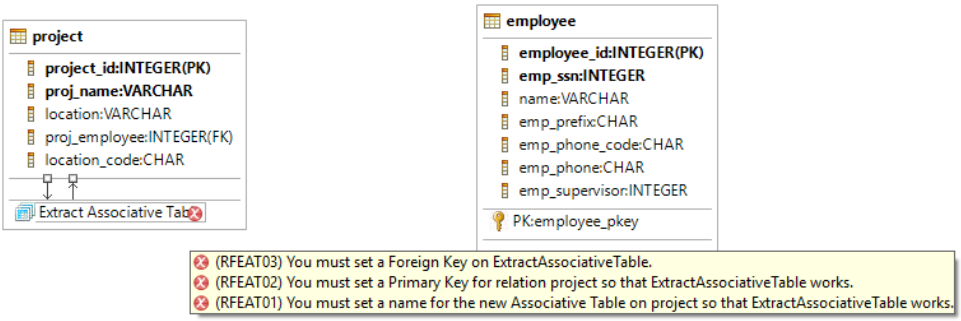
Fonte: (Autor, 2018)

Na próxima seção, são descritas as regras de semântica estática relacionadas a cada construtor da DSML. Com isto, torna-se possível alertar os usuários sobre erros de modelagem que possam afetar a boa formação das refatorações.

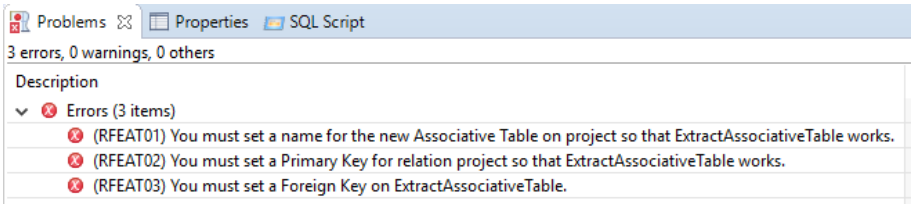
4.4 SEMÂNTICA ESTÁTICA

Apesar da definição de restrições estruturais do metamodelo, como associações obrigatórias ou relações de composição, algumas restrições não são possíveis de serem capturas, pois estas não são de caráter estrutural, mas sim de semântica estática. Construções inválidas como aplicar o *Merge Tables* apontando para a própria tabela ou definir um *Introduce Calculated Column* com tipo de dados que não seja numérico não devem ser permitidas. Exemplos de erros de semântica estática podem ser vistos na Figura 18. Ressalta-se que na Listagem 2.4 ilustra-se um exemplo de código EVL, o qual é responsável por uma regra de boa formação.

Figura 18 – Representação de um erro sintático na DSML proposta



(a) Visualização dos erros no objeto disposto no compartimento de refatoração



(b) Visualização de um erro na aba de problemas da ferramenta CASE

Fonte: (Autor, 2018)

Na figura 18a, apresenta-se um erro reportado pela ferramenta, o qual indica que uma construção inválida foi efetuada, junto com uma mensagem informando que tipo de restrição foi violada. Na Figura 18b, este mesmo erro pode ser visto na aba *Problems* da ferramenta CASE.

Apesar de haver a possibilidade da existência de regras ainda não implementadas, definem-se 26 regras que visam a boa formação na modelagem das refatorações de bancos de dados. Ressalta-se que além destas regras também especifica-se uma crítica comum a

todos os construtores de refatoração. Esta crítica alerta sobre a definição de uma data de expiração por meio da propriedade *Expiration Date*, caso o usuário ajuste a propriedade *Transitional Period* com o valor *"true"* para indicar o período transicional. As regras de semântica estática são detalhadas a seguir:

Move Column

1. Verifica se a coluna que será realocada no construto *MoveColumn* foi definida. Tendo em vista que o objetivo deste padrão é mover uma coluna entre tabelas, a definição da propriedade *Column To Move* é mandatória.

2. Valida se a relação fonte e destino são distintas . Já que a coluna está presente na relação fonte, não faz sentido tê-la como alvo.

Introduce Surrogate Key

3. Verifica se a propriedade *Surrogate Key Name* foi definido para geração da nova chave substituta. Esta propriedade especifica um novo nome para a chave substituta.

Replace Surrogatey Key With Natural Key

4. Verifica se a propriedade *New Primary Key Attributes* foi especificada para geração da nova Chave Natural. Esta propriedade deve ser definida, pois os atributos mantidos pela propriedade compõem a nova chave primária natural.

Introduce Calculated Column.

8. Verifica se um nome foi definido na propriedade *Column Name* do construtor. Um nome é uma propriedade obrigatória, pois a nova coluna calculada precisa de um nome no banco de dados.

Introduce Calculated Column

5. Verifica se o tipo de dado é do tipo numérico. Já que o produto desta refatoração é um cálculo, a coluna que armazena o resultado deve ser do tipo numérico.

6. Valida a presença das colunas participantes da equação definida na propriedade *Sql Function*. Na propriedade em questão armazena-se a expressão que irá definir o resultado persistido na coluna derivada. Portanto, a definição desta propriedade torna-se obrigatória.

7. Checa se uma coluna foi definida para o construtor *Introduce Calculated Column.*

8. Verifica se um nome foi definido na propriedade *Column Name* do construtor. Um nome é uma propriedade obrigatória, pois a nova coluna calculada precisa de um nome no banco de dados.

Merge Table

9. Verifica se a Relação de destino e de origem são distintas.

12. Verifica se a Relação fonte é filha da relação destino. A Relação fonte deve ser possuir uma restrição de integridade referencial para a relação alvo.

Merge Columns

10. Verifica se a propriedade *Columns* possui pelo menos dois atributos.

11. Checa se os atributos mantidos pela propriedades *Columns* são do mesmo tipo. Uma vez que a coluna resultante da fusão é do tipo alfanumérico, deve-se verificar a compatibilidade dos tipos.

12. Verifica a definição do nome da nova coluna na propriedade *Merge Column Name*. A nova coluna produzida pela refatoração deve ter um nome.

Extract Associative Table

13. Verifica a definição de uma chave primária. Deve existir uma chave primária na tabela fonte. Esta chave será inserida na Tabela Associativa.

14. Avalia o ajuste de um nome para a Tabela Associativa. Para garantir uma geração válida, uma tabela deve possuir um nome definido na propriedade *Table Name*.

15. Avalia a definição de uma chave estrangeira para a Tabela Associativa. Deve-se apontar uma chave estrangeira na propriedade *Foreign Key*. Esta chave será inserida na tabela associativa.

16. Checa se a tabela fonte é filha da tabela destino. Para implementação adequada do padrão, a tabela fonte deve manter Integridade Referencial para a tabela alvo.

Rename Column

17. Verifica se uma coluna foi selecionada para mudança de nome. Tendo em vista que o objetivo deste padrão é renomear uma coluna, a escolha desta coluna é mandatória para o atribute *Column To Rename*.

18. Verifica se um novo nome foi definido para a coluna. Já que esta refatoração visa renomear a coluna selecionada, um novo nome deve ser ajustado na propriedade *Rename To*.

Replace Column

19. Verifica se uma coluna foi definida na propriedade *Replace To*. Uma nova coluna deve ser apontada para substituir a que está contida na propriedade *Column To Replace*

20. Alarma sobre o risco de perda de dados para substituições com tipos distintos. Para substituir uma coluna é indicado que seus tipos sejam equivalentes. Para implementar esta

refatoração, uma crítica foi adicionada, alertando o usuário do construtor sobre o risco de perda de dados em casos onde os tipos sejam distintos.

Rename Table

21. Verifica se um novo nome foi definido para a tabela em questão. Uma vez que o objetivo deste padrão é renomear uma tabela, definição de um novo nome na propriedade *Rename To* é obrigatória.

Rename View

22. Checa se um novo nome foi inserido para a *View*. O objetivo deste padrão é substituir o nome da Visão. Para isto, deve-se definir um novo para permutação através da propriedade *Rename To*.

Drop Column

23. Verifica se a coluna que será removida na refatoração *Drop Column* foi definida. Uma vez que o objetivo deste padrão é, quando ajustado o período transicional, remover uma coluna de uma tabela e armazená-la em uma tabela de histórico, uma ou mais colunas devem ser definidas.

Split Table

24. Verifica a definição de um nome para a tabela produzida pelo particionamento da relação. Deve-se inserir um nome para a tabela produto da partição na propriedade *Name*.

25. Checa se a propriedade *Distinct Column* foi ajustada. Para garantir a correta migração dos dados, este atributo deve ser inserido. O objetivo é extrair para dentro dele a chave primária da tabela produto e definir uma chave estrangeira na tabela particionada.

26. Verifica se foi especificado um novo nome para a tabela produzida pelo particionamento. Para isto, ajusta-se a propriedade *Target Table Name*.

4.5 CONSIDERAÇÕES FINAIS

Neste capítulo apresenta-se a DSML *EvoDB* detalhando-se as seguintes características: 1) a sua sintaxe concreta, exibindo os construtores e notação; 2) a sua sintaxe abstrata, por meio da descrição dos elementos do metamodelo e sua estrutura; e 3) a sua semântica estática, descrevendo-se as regras de boa formação na construção das refatorações. Junto com as metaclasses estendidas do metamodelo RMM, estes elementos permitem a formação de construções que representem o esquema de um banco de dados, viabilizando a engenharia reversa, modelagem e a engenharia direta de refatorações estruturais.

No próximo capítulo serão exibidos os detalhes da ferramenta *REvolutionCASE* e como seus construtores podem ser utilizados para refatorar uma base de dados. Além disso, exibem-se os exemplos de uso que demonstram a expressividade da DSML proposta.

5 FERRAMENTA REVOLUTIONCASE: AVALIAÇÃO DA DSML PROPOSTA

Neste capítulo, apresentam-se os construtores que compõem DSML proposta, bem como as facilidades entregues pela ferramenta *REvolutionCASE*. A infraestrutura sobre a qual a DSML foi construída é detalhada. Além disso, avalia-se a expressividade dos construtores da linguagem, os quais visam permitir que somente construções válidas sejam efetuadas na ferramenta de modelagem. Por fim, encerra-se com a apresentação das considerações finais e uma análise comparativa com os trabalhos relacionados presentes no chp:related.

5.1 IMPLEMENTAÇÃO DA FERRAMENTA

A ferramenta *REvolutionCASE*, do inglês *Refactoring for Evolution by Computer-Aided Software Engineering*, busca entregar um conjunto de construtores para representar as refatorações estruturais propostas por Ambler Scott W. e Sadalage (2006). Neste sentido, propõe-se uma DSML concebida como resultado do processo descrito por Cho, Gray e Syriani (2012). Além disso, o desenvolvimento desta ferramenta orienta-se pelo paradigma Model Driven Development (MDD) (BRAMBILLA; CABOT; WIMMER, 2012), o qual define que uma linguagem de modelagem deve ser especificada a partir de um metamodelo (i.e., sintaxe abstrata). A titulo de implementação, utiliza-se o *Graphical Modeling Framework* (GMF)[1] em conjunto com o *Eclipse Modeling Framework* (EMF).

As linguagens *Emfatic*, EVL, EGL e EOL, presentes no *Epsilon*, auxiliam na definição do metamodelo, na transformação, na validação e na execução do modelo. Além disso, utiliza-se a linguagem Java, junto com a biblioteca de abstração de objetos de bancos de dados relacionais JDBC[2], para compor a solução de Engenharia Reversa. Ainda que o algoritmo definido para esta DSML permita a Engenharia Reversa total do esquema escolhido, por questões de conformidade com a abordagem para aplicação das Refatorações definidas por Ambler Scott W. e Sadalage (2006), somente as Relações que fazem parte da refatoração são apresentadas na área de desenho.

Para garantir a boa formação do modelo, aplicam-se instruções na linguagem EVL (cf., Apêndice C) para que regras de boa formação garantam a corretude do modelo. Uma vez que a modelagem esteja em conformidade com as regras de boa formação, o modelo gera o código de acordo com o diagrama presente na área de desenho. Para gerar este artefato, utiliza-se a linguagem EGL, que contém um conjunto de regras de transformação (c.f., Apêdice D) definidas com o intuito de viabilizar a transformação do modelo para texto.

Na seção seguinte, são apresentadas as principais características de uso da ferramenta *REvolutionCASE*.

[1] http://wiki.eclipse.org/Graphical$_{M}odeling_{F}ramework/Documentation$
[2] https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

5.2 UTILIZAÇÃO DA FERRAMENTA

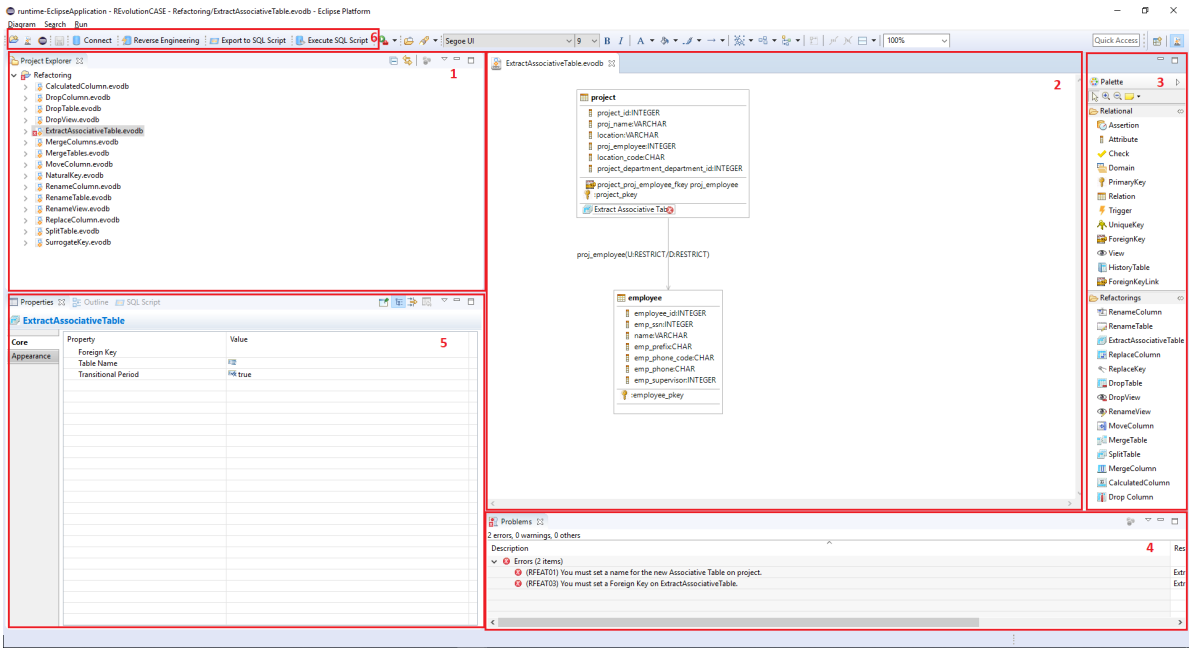
A ferramenta *REvolutionCASE* é formada por seis áreas distintas, onde cada região possui uma utilidade, como pode ser visto na Figura 19. Na área “um” exibem-se os projetos, que mantém os artefatos gerados na modelagem das refatorações. Na área “dois”, encontra-se a área de desenho, na qual é possível modelar a solução desejada. Na área “três”, mostra-se a paleta que contém os elementos que podem ser utilizados na modelagem das soluções. Para isto, apresenta-se o compartimento "*Refactorings*", pois o foco da ferramenta está na aplicação dos padrões representados pelos pictogramas presentes na paleta que contem as Refatorações. Na área “quatro”, apresentam-se os avisos ou erros de modelagem detectados pela ferramenta. Como disposto na Figura 19, uma vez que são necessárias as propriedades *"Foreign Key"* e *"Name"*, apresenta-se um erro que pode ser visto na área quatro. Este erro é produto das validações impostas pelas regras *EVLs* definidas para cada elemento do modelo e dispõe-se visualmente no formato de um pictograma característico de um erro em cor vermelha contendo um "X"em seu centro. Na área “cinco”, destacam-se as propriedades dos objetos selecionados na área de desenho. Na área cinco também é possível observar a aba *"SQL Script"*, que tem como função permitir a visualização do código gerado pela modelagem. Com isto, busca-se entregar para o usuário um conjunto de propriedades contextuais de cada elemento onde os ajustes necessários devem ser feitos para que o modelo possa estar de acordo com as regras estruturais e de boa formação, resultando no produto esperado pelo projetista. Por fim, na área “seis”, encontram-se os botões: 1) *"Connect"*, para permitir que uma conexão com o banco de dados seja estabelecida; 2) *"Reverse Engineering"*, que viabiliza a engenharia reversa das tabelas relacionadas à refatoração; 3) *"Export to SQL Script"*, responsável pela geração do código que será apresentado na aba *"SQL Script"*; e 4) *"Execute SQL Script"*, que submete as declarações *SQL* geradas na etapa anterior para a base de dados.

A conexão com o banco de dados é realizada por meio do ajuste do *login*, *password* e *URL* presentes nas propriedades da área de desenho, como disposto na Figura 20. Uma vez definidas as propriedades de conexão e efetuado o clique no botão *"Connect"*, as tabelas e visões presentes no banco de dados informado na *URL* estarão presentes na propriedade *"Database Object"*. Após a definição da relação e do clique no botão *"Reverse Engineering"*, a tabela escolhida e suas tabelas relacionadas serão trazidas para a área de desenho. O mesmo procedimento pode ser executado para as visões.

A geração de código referente à modelagem da refatoração disposta na área de desenho ocorre por meio do clique no botão *"Export SQL Script"*. Já a engenharia direta realiza-se por intermédio do botão *"Execute SQL Script"*. Na Figura 21, pode-se observar a aba que exibe o código que representa a refatoração.

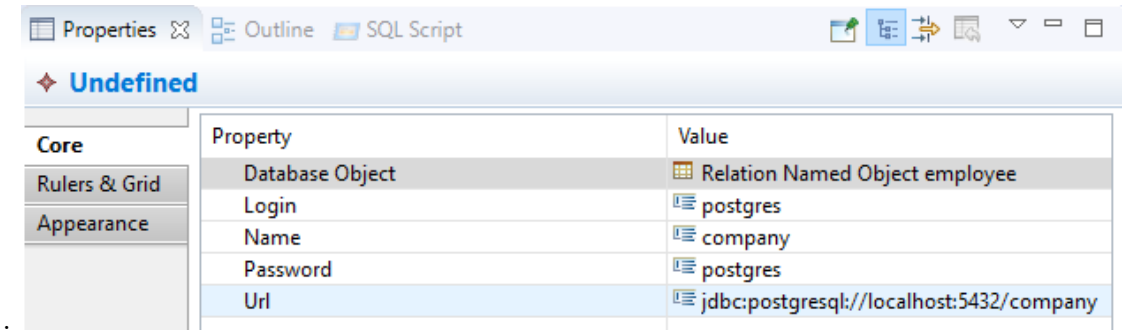
Na próxima Seção, detalha-se o cenário de avaliação, o esquema de banco de dados utilizado e os exemplos de uso definidos para cada refatoração suportada pela ferramenta *REvolutionCASE*.

Figura 19 – Workbench da ferramenta *REvolutionCASE*.



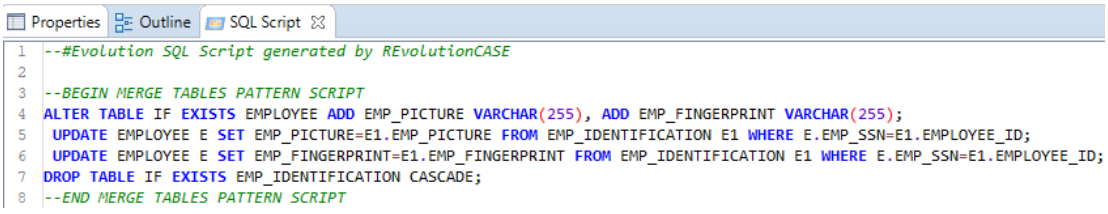
Fonte: (Autor, 2018)

Figura 20 – Propriedades de conexão com banco de dados



Fonte: (Autor, 2018)

Figura 21 – Aba que apresenta o código gerado pela refatoração na ferramenta *REvolutionCASE*



Fonte: (Autor, 2018)

5.3 CENÁRIO DE AVALIAÇÃO

Cada exemplo de uso presente nesta seção tem como principal objetivo demonstrar a expressividade da abordagem, elencando cada um dos construtores suportados pela linguagem *EvoDB* presente na ferramenta *REvolutionCASE*.

Na Figura 22, mostra-se a Engenharia Reversa do modelo de banco de dados utilizado para aplicação das refatorações. A base de dados trazida se encontra com as tabelas devidamente preenchidas com valores entre 500 e 1000 linhas cada. Estes dados foram gerados de forma automática por meio da ferramenta *web Mockaroo*[3]. Ressalta-se que este modelo é adaptado do modelo conceitual apresentado na Figura 23, o qual é amplamente conhecido e completo suficiente para exemplificar todas as refatorações estruturais presentes na DSML.

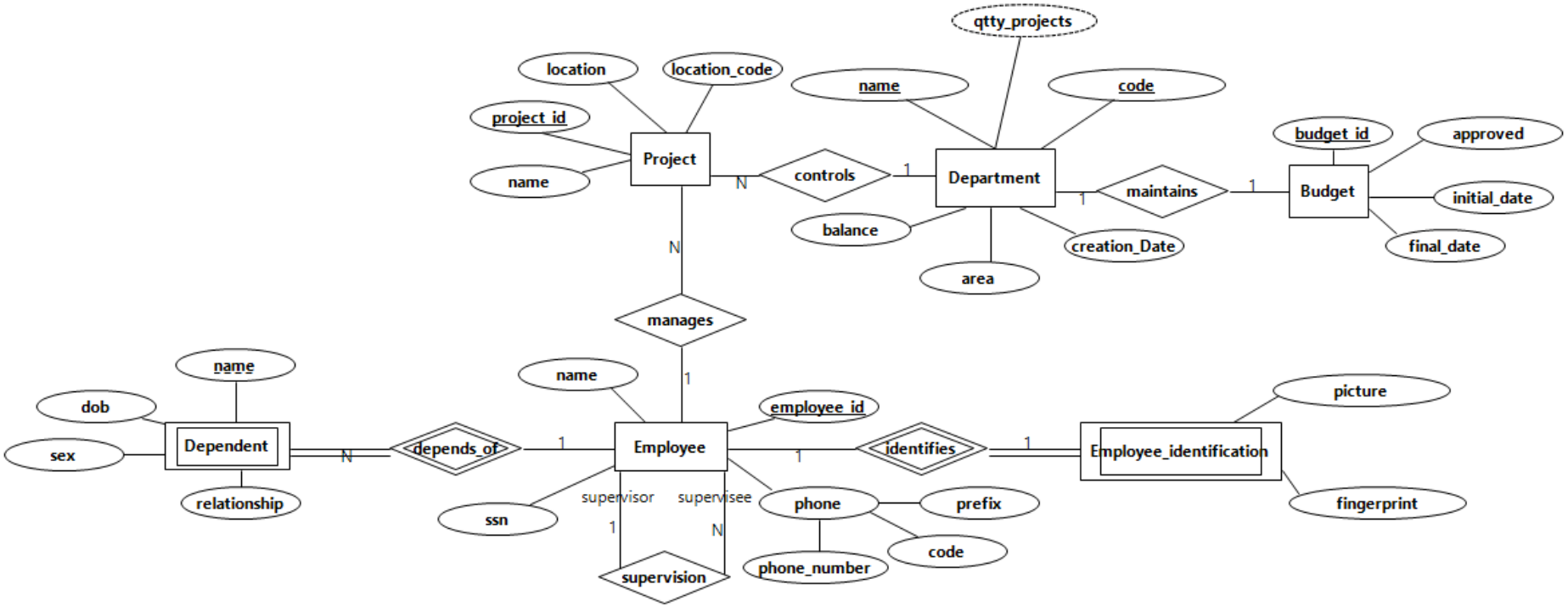
Ao longo dos cenários de avaliação apresentados, são detectados e resolvidos problemas de *design*, erros de modelagem e de tradução do modelo ER para Relacional. Os problemas de normalização, falha na padronização de Tabelas, Colunas e *Views*, assim como tabelas que não são mais utilizadas, são alguns dos erros detectados e corrigidos. Nestes cenários, são efetuadas as devidas evoluções no modelo por meio da modelagem das Refatorações Estruturais presentes na DSML proposta visando trazer corretude para o esquema. Vale ressaltar que: 1) para todos os cenários de avaliação, considera-se a Engenharia Reversa e Engenharia Direta do esquema, isentando-se a descrição destes passos no detalhamento da implementação da Refatoração; 2) no caso do Período Transicional, mostra-se o resultado somente no primeiro cenário (i.e., *Move Column*) devido o excesso de linhas de código gerado pela modelagem (cf. Listagem 5.1). Nos demais cenários, apresenta-se o código retornado sem esta característica; e 3) como supramencionado, o destaque dos construtores do modelo relacional em vermelho é um artifício da notação para identificar, visualmente, as entradas de cada refatoração (cf. Seção 4.2). Portanto, esta característica está presente em todos os cenários descritos nesta seção.

[3] https://mockaroo.com/

Figura 22 – Representação Relacional do Banco de Dados *COMPANY*

Fonte: (Autor, 2018)

Figura 23 – Modelo Entidade-Relacionamento do domínio *COMPANY*

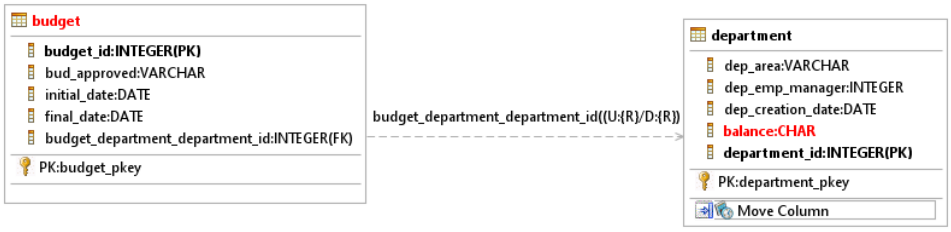


Fonte: Adaptado de Elmasri Ramez e Navathe (2010))

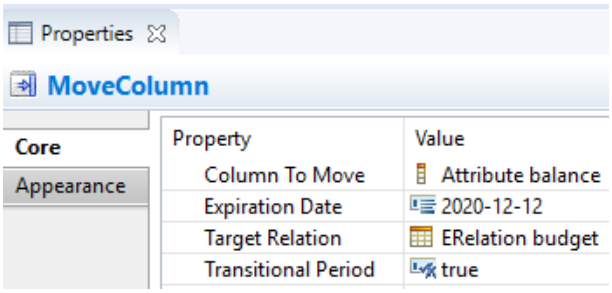
Cenário 1: Move Column

Motivação: Deseja-se mover a coluna *BALANCE* da tabela *DEPARTMENT* para a tabela *BUDGET* para corrigir um erro de modelagem e melhorar a semântica do esquema.

Figura 24 – Modelagem da Refatoração *Move Column* para mover coluna *BALANCE* de *BUDGET* para *DEPARTMENT*.



(a) Modelagem da Refatoração *Move Column*



(b) Propriedades do construtor *Move Column*

Fonte: (Autor, 2018)

Implementação: A aplicação do *Move Column* ocorre por meio da inserção deste construtor no compartimento da relação fonte. Após isto ajustam-se as propriedades: 1) *Column To Move*, que define a coluna que será movida (i.e., *BALANCE*); e 2) *Target Relation*, que especifica a relação alvo, ou seja, a representação da tabela que recebe a coluna (i.e., *BUDGET*). Também define-se o Período Transicional por meio das propriedades *Expiration Date* e *Transitional Period*, que recebem os valores "2019-10-01"e *"true"*, nesta ordem.

Estrutura: Na Figura 24a mostra-se a modelagem desta refatoração, junto com suas propriedades (Figura 24b).

Listagem 5.1 – Código gerado pela modelagem da refatoração *Move Column* com suporte ao Período Transicional

```
1 ALTER TABLE BUDGET ADD BALANCE CHAR(12);
2 UPDATE BUDGET SET BALANCE = (SELECT BALANCE FROM DEPARTMENT WHERE DEPARTMENT.DEPARTMENT_ID
      = BUDGET.BUDGET_DEPARTMENT_DEPARTMENT_ID);
3 CREATE OR REPLACE FUNCTION SYNCDEPARTMENTBALANCE() RETURNS TRIGGER AS
      $SYNCDEPARTMENTBALANCE$
```

```
BEGIN
DECLARE COUNTER INTEGER;
BEGIN
IF(TG_OP = 'UPDATE') THEN
IF(NEW.BALANCE IS NOT NULL AND NEW.BALANCE!=OLD.BALANCE) THEN
SELECT COUNT(*) INTO COUNTER FROM BUDGET B WHERE B.BUDGET_DEPARTMENT_DEPARTMENT_ID = NEW.
    DEPARTMENT_ID;
IF(COUNTER >0) THEN
UPDATE BUDGET B SET BALANCE = NEW.BALANCE WHERE B.BUDGET_DEPARTMENT_DEPARTMENT_ID = NEW.
    DEPARTMENT_ID;
END IF;
END IF;
RETURN NEW;
END IF;
IF(TG_OP = 'INSERT') THEN
IF(NEW.BALANCE IS NOT NULL) THEN
SELECT COUNT(*) INTO COUNTER FROM BUDGET B WHERE NEW.DEPARTMENT_ID;
IF(COUNTER >0) THEN
UPDATE BUDGET B SET BALANCE = NEW.BALANCE WHERE NEW.DEPARTMENT_ID;
END IF;
END IF;
RETURN NEW;
END IF;
END;
END;
$SYNCDEPARTMENTBALANCE$LANGUAGE plpgsql;
DROP TRIGGER IF EXISTS SYNCDEPARTMENTBALANCE ON DEPARTMENT;
CREATE TRIGGER SYNCDEPARTMENTBALANCE
AFTER INSERT OR UPDATE OF BALANCE ON DEPARTMENT
FOR EACH ROW
EXECUTE PROCEDURE SYNCDEPARTMENTBALANCE();
CREATE OR REPLACE FUNCTION SYNCDELETEBUDGETBALANCE()RETURNS TRIGGER AS
    $SYNCDELETEBUDGETBALANCE$
BEGIN
BEGIN
IF (TG_OP = 'DELETE') THEN
DELETE FROM DEPARTMENT D WHERE D.DEPARTMENT_ID = OLD.BUDGET_DEPARTMENT_DEPARTMENT_ID;
RETURN OLD;
END IF;
END;
END;
$SYNCDELETEBUDGETBALANCE$ LANGUAGE plpgsql;
DROP TRIGGER IF EXISTS SYNCDELETEBUDGETBALANCE ON DEPARTMENT;
CREATE TRIGGER SYNCDELETEBUDGETBALANCE
BEFORE DELETE ON DEPARTMENT
FOR EACH ROW
EXECUTE PROCEDURE SYNCDELETEBUDGETBALANCE();
```

```
48 CREATE OR REPLACE FUNCTION SYNCBUDGETBALANCE() RETURNS TRIGGER AS $SYNCBUDGETBALANCE$
49 BEGIN
50 DECLARE COUNTER INTEGER;
51 BEGIN
52 IF(TG_OP = 'UPDATE') THEN
53 SELECT COUNT(*) INTO COUNTER FROM DEPARTMENT D WHERE D.DEPARTMENT_ID = NEW.
       BUDGET_DEPARTMENT_DEPARTMENT_ID;
54 IF(COUNTER >0) THEN
55 UPDATE DEPARTMENT D SET BALANCE = NEW.BALANCE WHERE D.DEPARTMENT_ID = NEW.
       BUDGET_DEPARTMENT_DEPARTMENT_ID;
56 COUNTER:=0;
57 END IF;
58 RETURN NEW;
59 END IF;
60 IF(TG_OP = 'INSERT') THEN
61 IF(NEW.BALANCE IS NOT NULL) THEN
62 SELECT COUNT(*) INTO COUNTER FROM DEPARTMENT D WHERE D.DEPARTMENT_ID = NEW.
       BUDGET_DEPARTMENT_DEPARTMENT_ID;
63 IF(COUNTER >0) THEN
64 UPDATE DEPARTMENT D SET BALANCE = NEW.BALANCE WHERE D.DEPARTMENT_ID = NEW.
       BUDGET_DEPARTMENT_DEPARTMENT_ID;
65 COUNTER:=0;
66 END IF;
67 END IF;
68 RETURN NEW;
69 END IF;
70 END;
71 END;
72 $SYNCBUDGETBALANCE$ LANGUAGE plpgsql;
73 DROP TRIGGER IF EXISTS SYNCBUDGETBALANCE ON BUDGET;
74 CREATE TRIGGER SYNCBUDGETBALANCE
75 AFTER INSERT OR UPDATE OF BALANCE ON BUDGET
76 FOR EACH ROW
77 EXECUTE PROCEDURE SYNCBUDGETBALANCE();
```

Resultado: Na Listagem 5.1, exibe-se o código *SQL* gerado como resultado da modelagem. Nota-se, na linha 1, que a coluna *BALANCE* é inserida na tabela *BUDGET*. Apos isto atualizam-se os dados desta coluna por meio da declaração presente na linha 2. Também mostra-se a criação dos gatilhos (i.e., *triggers*) responsáveis por sincronizar os dados da coluna *BALANCE* tanto na tabela *BUDGET* quanto em *DEPARTMENT* durante o Período Transitional. O gatilho detalhado da linha 3 à 32 é acionado após a ocorrência de uma inserção ou atualização na coluna *BALANCE* na tabela *DEPARTMENT*. Também define-se o gatilho (linha 33 à linha 47) que é acionado antes da remoção da coluna *BALANCE* da tabela *BUDGET*. Isto se justifica pelo fato que a tabela *BUDGET* é a tabela correta para *BALANCE* após o Período Transicional. Já o gatilho que vai da linha 48 à

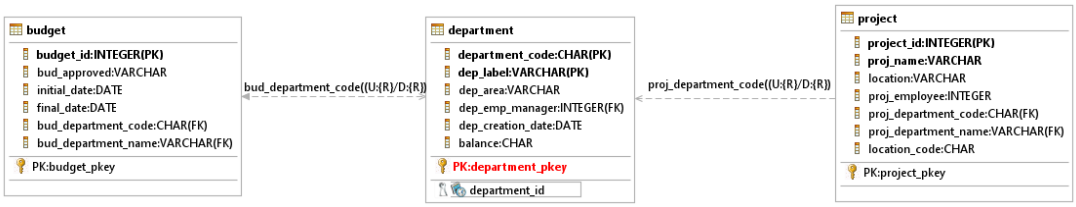
77, é executado após a ocorrência de uma inserção ou atualização na coluna *BALANCE* da tabela *BUDGET*.

Cenário 2: Introduce Surrogatey Key

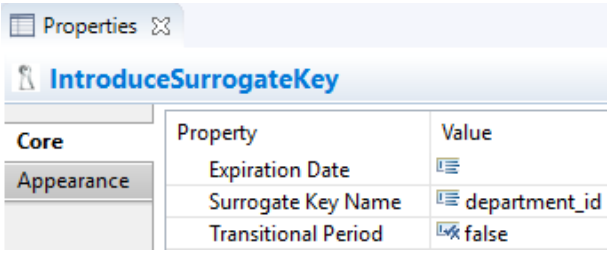
Motivação: Pretende-se substituir a chave primária composta da tabela *DEPARTMENT* por uma chave primária simples.

Implementação: Deve-se introduzir o construtor *Introduce Surrogate Key* na tabela *DEPARTMENT* e ajustar o nome da coluna que representa a chave substituta (i.e., *DEPARTMENT_ID*) na propriedade *Surrogate Key Name*. Ressalta-se que a coluna resultante é do tipo inteiro e recebe os valores de uma nova *SEQUENCE*.

Figura 25 – Modelagem da Refatoração *Introduce Surrogate Key*.



(a) Modelagem da Refatoração *Introduce Surrogate Key*



(b) Propriedades do construtor *Introduce Surrogate Key*

Fonte: (Autor, 2018)

Estrutura: Na Figura 25a exibe-se a modelagem da Refatoração *Introduce Surrogate Key* na ferramenta CASE. Mostra-se, também, as propriedades do construtor na Figura 25b.

Resultado: Na Listagem 5.2 pode-se observar que, na linha 1, é inserida uma nova coluna na tabela *DEPARTMENT*. Além disso, define-se uma sequência de valores únicos para esta coluna, o que pode ser visto nos comandos dispostos da linha 2 à linha 4. Na linha 5, remove-se a restrição de chave primária desta tabela de forma cascateada, ou seja, remove-se também as chaves estrangeiras que a referenciam. Já na linha 6, define-se a nova chave primária da tabela *DEPARTMENT*. Da linha 7 à 10 são gerados novos atributos e novas chaves estrangeiras para todas as tabelas que referenciam a tabela *DEPARTMENT*. Neste caso, as tabelas *PROJECT* e *BUDGET* recebem os novos valores da nova chave

primária de *DEPARTMENT* (i.e., *DEPARTMENT_ID*). Vale ressaltar que estes dados são migrados com base no relacionamento preexistente, como pode ser visto nas linhas 11 e 12. Uma vez que não são mais necessárias, as colunas anteriores são removidas, tanto da tabela refatorada quanto das tabelas que a referenciam, o que pode ser visto a partir da linha 13.

Listagem 5.2 – Código gerado pela modelagem da refatoração *Introduce Surrogate Key*.

```
1  ALTER TABLE IF EXISTS DEPARTMENT ADD DEPARTMENT_ID INTEGER;
2  CREATE SEQUENCE SEQ_DEPARTMENT;
3  ALTER TABLE IF EXISTS DEPARTMENT ALTER COLUMN DEPARTMENT_ID SET DEFAULT NEXTVAL('
       SEQ_DEPARTMENT');
4  UPDATE DEPARTMENT SET DEPARTMENT_ID=NEXTVAL('SEQ_DEPARTMENT');
5  ALTER TABLE IF EXISTS DEPARTMENT DROP CONSTRAINT IF EXISTS DEPARTMENT_PKEY CASCADE;
6  ALTER TABLE IF EXISTS DEPARTMENT ADD PRIMARY KEY(DEPARTMENT_ID);
7  ALTER TABLE IF EXISTS PROJECT ADD PROJECT_DEPARTMENT_DEPARTMENT_ID INTEGER;
8  ALTER TABLE IF EXISTS BUDGET ADD BUDGET_DEPARTMENT_DEPARTMENT_ID INTEGER;
9  ALTER TABLE IF EXISTS PROJECT ADD FOREIGN KEY(PROJECT_DEPARTMENT_DEPARTMENT_ID ) REFERENCES
       DEPARTMENT(DEPARTMENT_ID) ON UPDATE CASCADE;
10 ALTER TABLE IF EXISTS BUDGET ADD FOREIGN KEY(BUDGET_DEPARTMENT_DEPARTMENT_ID ) REFERENCES
       DEPARTMENT(DEPARTMENT_ID) ON UPDATE CASCADE;
11 UPDATE PROJECT SET PROJECT_DEPARTMENT_DEPARTMENT_ID=DEPARTMENT.DEPARTMENT_ID FROM
       DEPARTMENT WHERE DEPARTMENT.DEPARTMENT_CODE = PROJECT.PROJ_DEPARTMENT_CODE AND PROJECT.
       PROJ_DEPARTMENT_NAME = DEPARTMENT.DEP_LABEL;
12 UPDATE BUDGET SET BUDGET_DEPARTMENT_DEPARTMENT_ID=DEPARTMENT.DEPARTMENT_ID FROM DEPARTMENT
       WHERE DEPARTMENT.DEPARTMENT_CODE = BUDGET.BUD_DEPARTMENT_CODE AND BUDGET.
       BUD_DEPARTMENT_NAME = DEPARTMENT.DEP_LABEL;
13 ALTER TABLE IF EXISTS DEPARTMENT DROP COLUMN IF EXISTS DEPARTMENT_CODE CASCADE , DROP
       COLUMN IF EXISTS DEP_LABEL CASCADE ;
14 ALTER TABLE IF EXISTS PROJECT DROP COLUMN IF EXISTS PROJ_DEPARTMENT_CODE CASCADE;
15 ALTER TABLE IF EXISTS PROJECT DROP COLUMN IF EXISTS PROJ_DEPARTMENT_NAME CASCADE;
16 ALTER TABLE IF EXISTS BUDGET DROP COLUMN IF EXISTS BUD_DEPARTMENT_CODE CASCADE;
17 ALTER TABLE IF EXISTS BUDGET DROP COLUMN IF EXISTS BUD_DEPARTMENT_NAME CASCADE;
```

Cenário 3: Replace Surrogate Key With Natural Key

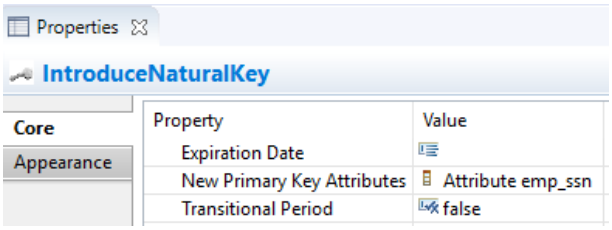
Motivação: Deseja-se permutar a chave substituta *EMPLOYEE_ID* da tabela *EMPLOYEE* pela coluna *EMP_SSN*.

Implementação: A modelagem desta Refatoração dá-se pela adição do construtor *Introduce Natural Key* no compartimento de Refatorações de *ERelation*. Além disso, especifica-se a propriedade *New Primary Key Attributes* para indicar qual atributo representa a chave natural (i.e., *EMP_SSN*).

Estrutura: Na Figura 26a apresenta-se a modelagem da Refatoração *Replace Surrogate Key with Natural Key*, bem como suas propriedades (Figura 26b).

Figura 26 – Modelagem da Refatoração *Replace Surrogate Key With Natural Key* para definir a coluna *EMP_SSN* chave primária natural da tabela *EMPLOYEE*

(a) Modelagem da Refatoração *Replace Surrogate Key With Natural Key*



(b) Propriedades do construtor *Introduce Natural Key*

Fonte: (Autor, 2018)

Resultado: Como resultado, tem-se a remoção da restrição de chave primária da tabela refatorada (i.e., *EMPLOYEE*) em cascata e a definição de uma nova restrição de chave sobre coluna *EMP_SSN*. Isto ocorre nas linhas 1 e 2. Da linha 3 à linha 12, criam-se novas chaves estrangeiras nas tabelas que referenciam *EMPLOYEE* as quais têm seus valores atualizados para refletir estas mudanças. Por fim, na linha 13, remove-se a antiga coluna pertencente à chave primária de *EMPLOYEE* (i.e., *EMPLOYEE_ID*). Isto pode ser visto na Listagem 5.3.

Listagem 5.3 – Código gerado pela modelagem da refatoração *Introduce Natural Key*.

```
1 ALTER TABLE IF EXISTS EMPLOYEE DROP CONSTRAINT IF EXISTS EMPLOYEE_PKEY CASCADE;
2 ALTER TABLE EMPLOYEE ADD PRIMARY KEY(EMP_SSN);
3 UPDATE EMP_IDENTIFICATION E SET EMPLOYEE_ID=E1.EMP_SSN FROM EMPLOYEE E1 WHERE E.EMPLOYEE_ID
      =E1.EMPLOYEE_ID;
4 ALTER TABLE EMP_IDENTIFICATION ADD FOREIGN KEY(EMPLOYEE_ID) REFERENCES EMPLOYEE(EMP_SSN);
5 UPDATE DEPARTMENT D SET DEP_EMP_MANAGER=E.EMP_SSN FROM EMPLOYEE E WHERE D.DEP_EMP_MANAGER=E
      .EMPLOYEE_ID;
```

```
6  ALTER TABLE DEPARTMENT ADD FOREIGN KEY(DEP_EMP_MANAGER) REFERENCES EMPLOYEE(EMP_SSN);
7  UPDATE DEPENDENT D SET EMPLOYEE_ID=E.EMP_SSN FROM EMPLOYEE E WHERE D.EMPLOYEE_ID=E.
       EMPLOYEE_ID;
8  ALTER TABLE DEPENDENT ADD FOREIGN KEY(EMPLOYEE_ID) REFERENCES EMPLOYEE(EMP_SSN);
9  UPDATE EMPLOYEE E SET EMP_SUPERVISOR=E1.EMP_SSN FROM EMPLOYEE E1 WHERE E.EMP_SUPERVISOR=E1.
       EMPLOYEE_ID;
10 ALTER TABLE EMPLOYEE ADD FOREIGN KEY(EMP_SUPERVISOR) REFERENCES EMPLOYEE(EMP_SSN);
11 UPDATE PROJECT P SET PROJ_EMPLOYEE=E.EMP_SSN FROM EMPLOYEE E WHERE P.PROJ_EMPLOYEE=E.
       EMPLOYEE_ID;
12 ALTER TABLE PROJECT ADD FOREIGN KEY(PROJ_EMPLOYEE) REFERENCES EMPLOYEE(EMP_SSN);
13 ALTER TABLE EMPLOYEE DROP COLUMN EMPLOYEE_ID;
```

Cenário 4: Introduce Calculated Column

Motivação: Deseja-se inserir uma coluna derivada na tabela *DEPARTMENT* com o quantitativo de projetos por departamento.

Implementação: Deve-se adicionar o construtor na tabela desejada (i.e., *DEPARTMENT*) e definir as seguintes propriedades: *"Column Name"* - que tem seu valor especificado para *QTTY_PROJECT*; *Data Type* - que define o tipo de dado para *NUMERIC*; *SQL Function* - que descreve a função *COUNT(PROJECT_ID)*; e *Target Table Column* - que determina a coluna *PROJECT_ID*. Ressalta-se que esta propriedade deve ser especificada para garantir que há uma relação entre a tabela que mantém a coluna calculada e a tabela que possui a coluna que compõe a função *SQL*.

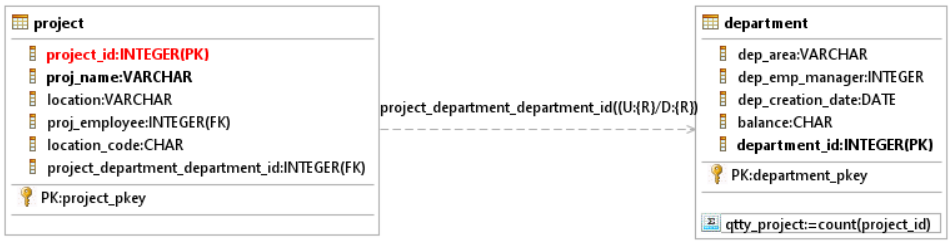
Estrutura: Na Figura 27a, exibe-se a modelagem da Refatoração *Introduce Calculated Column*. As propriedades do construtor estão presentes na Figura 27b.

Resultado: Na Listagem 5.9 pode-se observar o código gerado referente ao modelo. Na linha 1, adiciona-se a coluna calculada (i.e., *QTTY_PROJECT*) e a *trigger* que sincroniza os valores desta coluna (linha 2 a 22). Este gatilho tem como objetivo atualizar os valores das colunas sempre que houver uma inserção ou atualização na tabela que contém a coluna que serve como parâmetro da função definida pelo usuário. Neste caso, se houver uma nova inserção em *PROJECT*, isto é propagado para a coluna *QTTY_PROJECT*. Em caso da remoção da coluna, aciona-se o gatilho especificado da linha 23 à linha 43. Por fim, para que *QTTY_PROJECT* tenha seus valores atualizados após sua concepção, executa-se um comando de atualização encadeado com uma seleção que contém a função de agregação definida pelo usuário (linha 44).

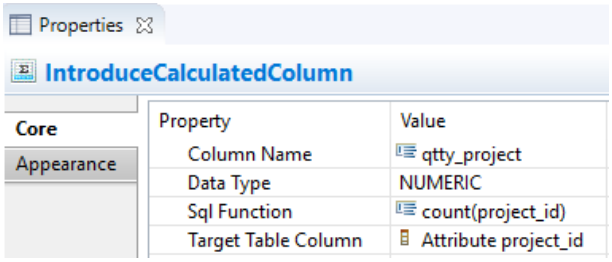
Listagem 5.2 – Código gerado pela refatoração *Introduce Calculated Column*.

```
1  ALTER TABLE DEPARTMENT ADD QTTY_PROJECT NUMERIC;
2  CREATE OR REPLACE FUNCTION ADDCALCULATEDCOLUMNFORQTTY_PROJECT() RETURNS TRIGGER AS
       $ADDCALCULATEDCOLUMNFORQTTY_PROJECT$
3  BEGIN
4  DECLARE
```

Figura 27 – Modelagem da Refatoração *Introduce Calculated Column* para introduzir uma coluna derivada na tabela *DEPARTMENT*.



(a) Modelagem da Refatoração *Introduce Calculated Column*



(b) Propriedades do Construtor *Calculated Column*

Fonte: (Autor, 2018)

```
5  DEPARTMENT_DEPARTMENT_ID_PK INTEGER;
6  CALC_QTTY_PROJECT NUMERIC;
7  BEGIN
8  IF(TG_OP = 'INSERT' OR TG_OP = 'UPDATE' ) THEN
9  DEPARTMENT_DEPARTMENT_ID_PK:=NEW.PROJECT_DEPARTMENT_DEPARTMENT_ID;
10 SELECT INTO CALC_QTTY_PROJECT COUNT(PROJECT_ID) FROM PROJECT WHERE
       PROJECT_DEPARTMENT_DEPARTMENT_ID=NEW.PROJECT_DEPARTMENT_DEPARTMENT_ID;
11 UPDATE DEPARTMENT SET QTTY_PROJECT = CALC_QTTY_PROJECT WHERE DEPARTMENT_ID=
       DEPARTMENT_DEPARTMENT_ID_PK;
12 RETURN NEW;
13 CALC_QTTY_PROJECT:=0;
14 END IF;
15 END;
16 END;
17 $ADDCALCULATEDCOLUMNFORQTTY_PROJECT$ LANGUAGE plpgsql;
18 DROP TRIGGER IF EXISTS ADDCALCULATEDCOLUMNFORQTTY_PROJECT ON PROJECT;
19 CREATE TRIGGER ADDCALCULATEDCOLUMNFORQTTY_PROJECT
20 AFTER INSERT OR UPDATE ON PROJECT
21 FOR EACH ROW
22 EXECUTE PROCEDURE ADDCALCULATEDCOLUMNFORQTTY_PROJECT();
23 CREATE OR REPLACE FUNCTION ADDCALCULATEDCOLUMN_ON_DELETE_FORQTTY_PROJECT() RETURNS TRIGGER
       AS $ADDCALCULATEDCOLUMNONDELETEFORQTTY_PROJECT$
24 BEGIN
25 DECLARE
```

```
26 DEPARTMENT_DEPARTMENT_ID_PK INTEGER;
27 CALC_QTTY_PROJECT NUMERIC;
28 BEGIN
29 IF(TG_OP = 'DELETE' ) THEN
30 DEPARTMENT_DEPARTMENT_ID_PK:=OLD.PROJECT_DEPARTMENT_DEPARTMENT_ID;
31 SELECT INTO CALC_QTTY_PROJECT COUNT(PROJECT_ID) FROM PROJECT WHERE
       PROJECT_DEPARTMENT_DEPARTMENT_ID=OLD.PROJECT_DEPARTMENT_DEPARTMENT_ID;
32 UPDATE DEPARTMENT SET QTTY_PROJECT = CALC_QTTY_PROJECT WHERE DEPARTMENT_ID=
       DEPARTMENT_DEPARTMENT_ID_PK;
33 RETURN OLD;
34 CALC_QTTY_PROJECT:=0;
35 END IF;
36 END;
37 END;
38 $ADDCALCULATEDCOLUMNONDELETEFORQTTY_PROJECT$ LANGUAGE plpgsql;
39 DROP TRIGGER IF EXISTS ADDCALCULATEDCOLUMN_ON_DELETE_FORQTTY_PROJECT ON PROJECT;
40 CREATE TRIGGER ADDCALCULATEDCOLUMN_ON_DELETE_FORQTTY_PROJECT
41 AFTER DELETE ON PROJECT
42 FOR EACH ROW
43 EXECUTE PROCEDURE ADDCALCULATEDCOLUMN_ON_DELETE_FORQTTY_PROJECT();
44 UPDATE DEPARTMENT SET QTTY_PROJECT=(SELECT COUNT(PROJECT_ID) FROM PROJECT WHERE DEPARTMENT.
       DEPARTMENT_ID=PROJECT.PROJECT_DEPARTMENT_DEPARTMENT_ID);
```

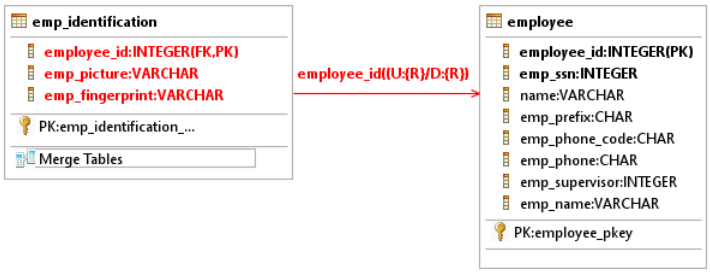
Cenário 5: Merge Tables

Motivação: Eliminar a operação de junção por meio da aglutinação das tabelas *EMPLOYEE* e *EMPLOYEE_IDENTIFICATION*, uma vez que torna-se comum o uso dos dados das duas tabelas sempre que os dados de *EMPLOYEE* são consumidos.

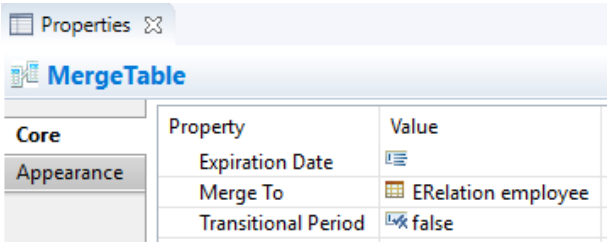
Implementação: Deve-se inserir o construtor *Merge Tables* na relação que mantém a chave estrangeira (i.e., *EMP_IDENTIFICATION*) e ajustar a propriedade *Merge To*, que deve apontar para a relação a qual deseja-se efetuar a fusão (i.e., *EMPLOYEE*).

Estrutura: Na Figura 28a, pode-se visualizar a modelagem da Refatoração *Merge Tables*, bem como as propriedades do seu construtor na Figura 28b.

Figura 28 – Modelagem da Refatoração *Merge Tables* para unir as tabelas *EMP_IDENTIFICATION* e *EMPLOYEE*.



(a) Modelagem da Refatoração *Merge Tables*



(b) Propriedades do construtor *Merge Table*

Fonte: (Autor, 2018)

Resultado: Na Listagem 5.4 exibe-se o código referente ao diagrama presente na Figura 28a. Mostra-se que as colunas *EMP_PICTURE* e *EMP_FINGERPRINT* da tabela *EMP_IDENTIFICATION* são criadas (linha 1) e migradas (linhas 2 e 3) para a tabela *EMPLOYEE*. Na linha 4, vê-se a declaração que remove *EMP_IDENTIFICATION* do esquema.

Listagem 5.3 – Código gerado pela modelagem da refatoração *Merge Tables*.

```
1 ALTER TABLE IF EXISTS EMPLOYEE ADD EMP_PICTURE VARCHAR(255), ADD EMP_FINGERPRINT VARCHAR
      (255);
2 UPDATE EMPLOYEE SET EMP_PICTURE=EMP_IDENTIFICATION.EMP_PICTURE FROM EMP_IDENTIFICATION
      WHERE EMPLOYEE.EMPLOYEE_ID=EMP_IDENTIFICATION.EMPLOYEE_ID;
3 UPDATE EMPLOYEE SET EMP_FINGERPRINT=EMP_IDENTIFICATION.EMP_FINGERPRINT FROM
      EMP_IDENTIFICATION WHERE EMPLOYEE.EMPLOYEE_ID=EMP_IDENTIFICATION.EMPLOYEE_ID;
4 DROP TABLE IF EXISTS EMP_IDENTIFICATION CASCADE;
```

Cenário 6: Merge Columns

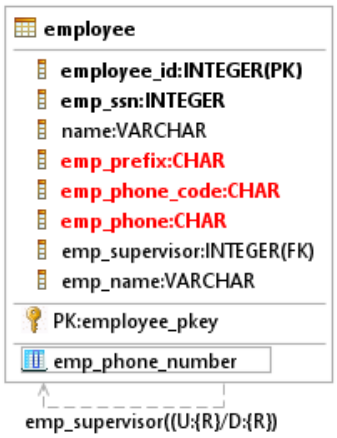
Motivação: Deseja-se unir as colunas *EMP_PHONE_PREFIX*, *EMP_PHONE_CODE* e *EMP_PHONE* em uma única coluna, uma vez que são mantidas em conjunto.

Implementação: Adiciona-se o construtor *Merge Columns* na relação que mantém as colunas e definem-se as seguintes propriedades: *Columns* - que recebe os atributos que

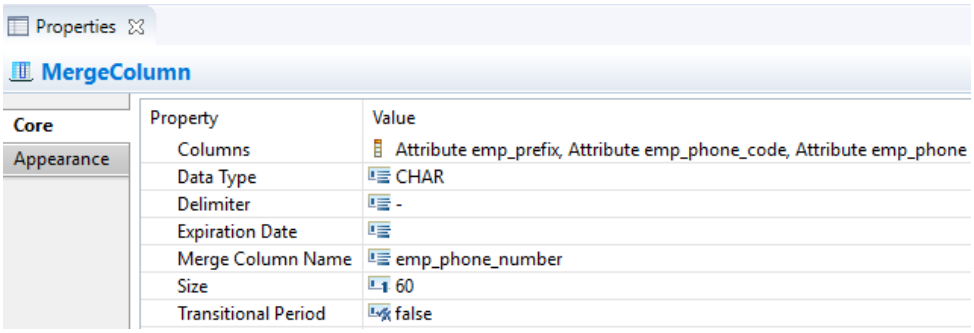
serão unificados, ou seja, *EMP_PHONE_PREFIX*, *EMP_PHONE_CODE* e *EMP_-PHONE*; *Data Type* - que tem seu tipo definido para *CHAR*; *Delimiter* - que especifica um traço ("-") como separador ; *Merge Column Name* - que especifica o nome *EMP_-PHONE_NUMBER*; e *Size* - que define o tamanho da nova coluna.

Estrutura: Na Figura 29 apresenta-se a modelagem da Refatoração *Merge Columns*, junto com as propriedades do construtor.

Figura 29 – Modelagem da Refatoração *Merge Columns* para criar a coluna *EMP_-PHONE_NUMBER* na tabela *EMPLOYEE* baseada nos valores de outras colunas.



(a) Modelagem da Refatoração *Merge Columns*



(b) Propriedades do construtor *Merge Columns*

Fonte: (Autor, 2018)

Resultado: Nas linhas 1 e 2 da Listagem 5.5, observa-se que a coluna alfanumérica *EMP_PHONE_NUMBER* é inserida na tabela *EMPLOYEE* e seus valores são atualizados com a fusão das colunas definidas na propriedade *Columns* do construtor. Nota-se que isto ocorre por meio da função *CONCAT*. Já na linha 3, removem-se as colunas que participam da fusão.

Listagem 5.4 – Código gerado pela modelagem da refatoração *Merge Columns*.

```
1 ALTER TABLE EMPLOYEE ADD EMP_PHONE_NUMBER CHAR(60);
2  UPDATE EMPLOYEE E1 SET EMP_PHONE_NUMBER = CAST(CONCAT(E2.EMP_PREFIX,'-',E2.EMP_PHONE_CODE,
       '-',E2.EMP_PHONE) AS CHAR (60) FROM EMPLOYEE E2 WHERE E1.EMPLOYEE_ID = E2.EMPLOYEE_ID;
3 ALTER TABLE EMPLOYEE DROP COLUMN EMP_PREFIX, DROP COLUMN EMP_PHONE_CODE, DROP COLUMN
      EMP_PHONE;
```

Cenário 7: Replace One-To-Many With Associative Table

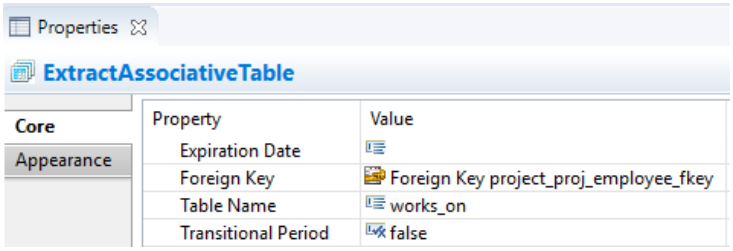
Motivação: Deseja-se transformar um relacionamento 1:N em um relacionamento N:N. Por meio da Refatoração *Replace One-To-Many With Associative Table* é possível gerar um modelo intermediário, permitindo esta migração de forma transparente. Neste caso, deseja-se extrair uma tabela do relacionamento entre as tabelas *PROJECT* e *EMPLOYEE*.

Implementação: Deve-se inserir o construtor *Extract Associative Table* na relação que contém a chave estrangeira e efetuar os seguintes ajustes: 1) definir o nome da tabela associativa na propriedade *"Table Name"* ; e 2) especificar a partir de qual chave estrangeira deve-se gerar a nova tabela associativa, já que a relação em questão pode participar de mais de um relacionamento.

Estrutura: Exemplifica-se este padrão utilizando-se a modelagem e as propriedades do construtor presente nas Figuras 30a e 30b, nesta ordem.

Figura 30 – Modelagem da Refatoração *Replace One-To-Many With Associative Table* para criar a tabela associativa *WORKS_ON*.

(a) Modelagem da Refatoração *Replace One-To-Many With Associative Table*



(b) Propriedades do construtor *Extract Associative Table*

Fonte: (Autor, 2018)

Resultado: Na Listagem 5.6 exibe-se o código gerado a partir do modelo. Na linha 1, define-se a tabela *WORKS_ON*. Nas linhas 2 e 3, especificam-se as seguintes restrições: *NOT NULL* - para as colunas *PROJECT_ID* e *PROJECT_EMPLOYEE*; e UNIQUE - para a coluna *PROJECT_ID_KEY*. Após isto, os dados são migrados da tabela *PROJECT* para WORKS_ON (linha 5) e esta última tem suas chaves estrangeiras definidas (linhas 6 e 7). Por fim, na linha 8, remove-se a chave estrangeira da tabela *PROJECT*.

Listagem 5.5 – Código gerado pela modelagem da refatoração *Replace One-To-Many With Associative Table*.

```
1 CREATE TABLE IF NOT EXISTS WORKS_ON(PROJECT_ID INTEGER, PROJ_EMPLOYEE INTEGER);
2 ALTER TABLE WORKS_ON ALTER COLUMN PROJECT_ID SET NOT NULL;
3 ALTER TABLE WORKS_ON ADD CONSTRAINT PROJECT_ID_KEY UNIQUE(PROJECT_ID);
4 ALTER TABLE WORKS_ON ALTER COLUMN PROJ_EMPLOYEE SET NOT NULL;
5 INSERT INTO WORKS_ON(PROJECT_ID, PROJ_EMPLOYEE) SELECT PROJECT_ID, PROJ_EMPLOYEE FROM
      PROJECT;
6 ALTER TABLE WORKS_ON ADD CONSTRAINT WORKS_ON_EMPLOYEE_FKEY FOREIGN KEY(PROJ_EMPLOYEE)
      REFERENCES EMPLOYEE(EMPLOYEE_ID);
7 ALTER TABLE WORKS_ON ADD CONSTRAINT WORKS_ON_PROJECT_FKEY FOREIGN KEY(PROJECT_ID)
      REFERENCES PROJECT(PROJECT_ID);
8 ALTER TABLE PROJECT DROP PROJ_EMPLOYEE CASCADE;
```

Cenário 8: Rename Column

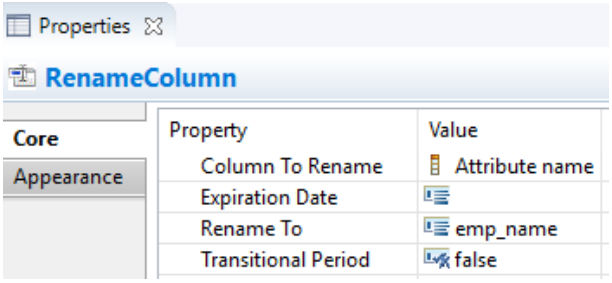
Motivação: Renomear a coluna *NAME* para manter a consistência entre os nomes das colunas da tabela *EMPLOYEE* que são prefixadas.

Implementação: Adiciona-se construtor *Rename Column* no compartimento de Refatorações da Tabela e ajusta-se a propriedade *Column to Rename*, que serve para apontar o nome da coluna que será renomeada. Também deve-se informar o nome da nova coluna utilizando a propriedade *Rename To*, que neste caso renomeia-se para *EMP_NAME*.

Estrutura: Na Figura 31a ilustra-se como ocorre a modelagem da Refatoração *Rename Column*. Já na Figura 31b, apresentam-se as propriedades do construtor.

Figura 31 – Modelagem da Refatoração *Rename Column* para mudar o nome da coluna *NAME* para *EMP_NAME*

(a) Modelagem da Refatoração *Rename Column*



(b) Propriedades do construtor *Rename Column*

Fonte: (Autor, 2018)

Resultado: O modelo presente na área de desenho refere-se ao código descrito na Listagem 5.7. Na linha 1, adiciona-se a nova coluna *EMP_NAME* na tabela *EMPLOYEE* e os dados da coluna *NAME* são inseridos na nova coluna *EMP_NAME* por meio da declaração de atualização disposta na linha 2. Por fim, na linha 3, remove-se a coluna renomeada.

Listagem 5.6 – Código gerado pela modelagem da refatoração *Rename Column*.

```
1 ALTER TABLE EMPLOYEE ADD EMP_NAME VARCHAR(150);
2 UPDATE EMPLOYEE SET EMP_NAME=NAME WHERE EMPLOYEE_ID=EMPLOYEE_ID;
3 ALTER TABLE EMPLOYEE DROP COLUMN NAME CASCADE;
```

Cenário 9: Replace Column

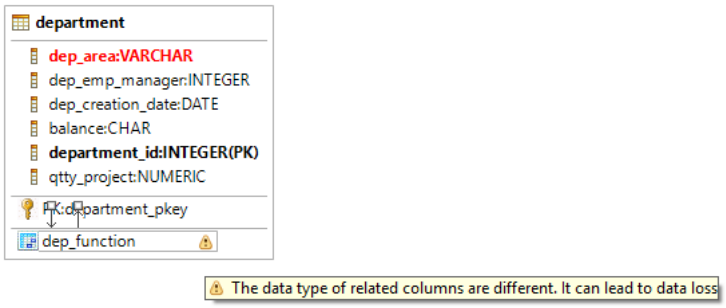
Motivação: Substituir o nome e tipo da coluna *DEP_AREA* para *DEP_FUNCTION* na tabela *DEPARTMENT*. Com isto, alinha-se à nomenclatura padrão do esquema e define-se um tipo com largura mais adequada à coluna.

Implementação: Deve-se inserir o construtor *Replace Column* no compartimento de Refatorações da relação e ajustar as propriedades: *Column To Replace* - que recebe a coluna *DEP_AREA*; *Replace To* - que especifica *DEP_FUNCTION* como o nome da nova coluna; *Replace Column Data Type* - que determina o tipo de dado da nova coluna; e *Replace Column Size* - que especifica um tamanho para a coluna. Ressalta-se que uma crítica será apresentada nos casos onde tanto o tipo de dado seja diferente da coluna substituída quanto tamanho da coluna especificada seja menor que a coluna original.

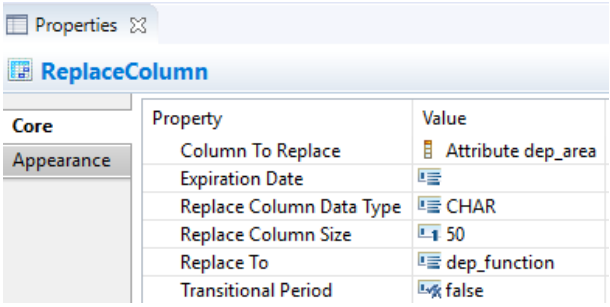
Estrutura: Nas Figuras 32a e 32b visualizam-se a modelagem da Refatoração *Replace Column* e as propriedades do construtor na DSML, nesta ordem. Nota-se que o diagrama expõe um ícone e uma mensagem informando sobre o risco da perda de dados, uma vez que os tipos de dados da coluna especificada e substituída são distintos.

Resultado: Como pode ser visto na Listagem 5.8, código referente à refatoração resulta na especificação da coluna *DEP_FUNCTION*. Esta coluna recebe os valores da coluna substituída *DEP_AREA*. Isto pode ser visto nas linhas 1 e 2. Já na linha 3, remove-se a coluna substituída.

Figura 32 – Modelagem da Refatoração *Replace Column* para substituir a coluna *DEP_AREA* por *DEP_FUNCTION* e propriedades do construtor.



(a) Modelagem da Refatoração *Replace Column*



(b) Propriedades do construtor *Replace Column*

Fonte: (Autor, 2018)

Listagem 5.7 – Código gerado pela modelagem da refatoração *Replace Column*.

```
1 ALTER TABLE DEPARTMENT ADD DEP_FUNCTION CHAR(50);
2 UPDATE DEPARTMENT SET DEP_FUNCTION=DEP_AREA WHERE DEPARTMENT_ID=DEPARTMENT_ID;
3 ALTER TABLE DEPARTMENT DROP COLUMN DEP_AREA CASCADE;
```

Cenário 10: Rename Table

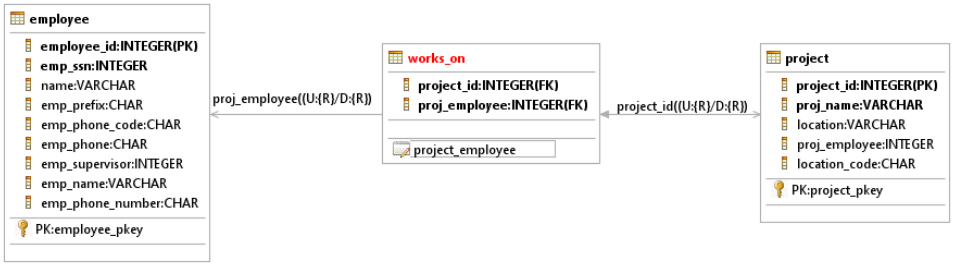
Motivação: Deseja-se mudar o nome da tabela para manter a coerência de nomenclatura na ocorrência de relacionamentos de grau maior que 1 (i.e., enários). Neste caso, renomeia-se a tabela *WORKS_ON* para a fusão dos nomes das tabelas que participam do relacionamento, ou seja, *PROJECT_EMPLOYEE*.

Implementação: Adiciona-se o elemento *Rename Table* no compartimento de refatoração e ajusta-se a propriedade *Rename To* com o valor *PROJECT_EMPLOYEE*.

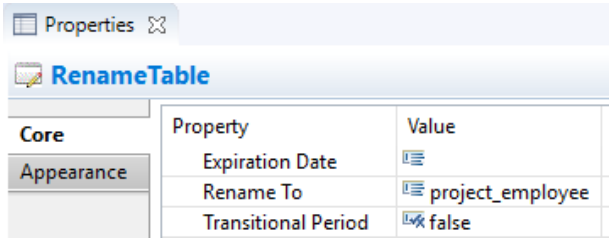
Estrutura: Na Figura 33a, exibe-se a modelagem da Refatoração *Rename Table* sem Período Transicional. As propriedades podem ser vistas na Figura 31b.

Resultado: Na única linha da Listagem 5.9 exibe-se o o código gerado referente à modelagem desta refatoração.

Figura 33 – Modelagem da Refatoração *Rename Table* para mudar o nome de *WORKS_ON* para *PROJECT_EMPLOYEE*



(a) Modelagem da Refatoração *Rename Table*



(b) Propriedades do construtor *Rename Table*

Fonte: (Autor, 2018)

Listagem 5.8 – Código gerado pela modelagem da refatoração *Rename Table*.

```
1 ALTER TABLE WORKS_ON RENAME TO PROJECT_EMPLOYEE;
```

Cenário 11: Rename View

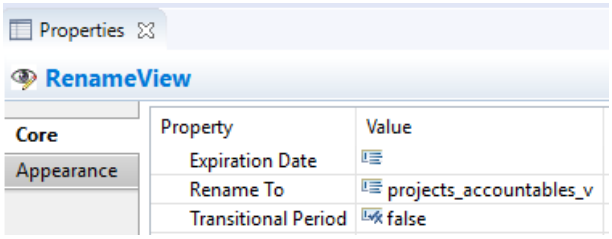
Motivação: Tornar o nome da Visão auto-descritivo, além de seguir o padrão de nomenclatura das demais *Views* do esquema.

Implementação: Adiciona-se o construtor *Rename View* no compartimento de refatoração do construtor visão e ajusta-se seu novo nome(i.e., *PROJECTS_ACCOUNTABLES_V*) por meio da propriedade *Rename To*.

Estrutura: Nas Figuras 34a e 34b exibem-se o modelo da Refatoração *Rename View* e as suas propriedades, respectivamente.

Figura 34 – Modelagem da Refatoração *Rename View* para renomear a Visão *EMPLOYEE_PROJECT_LOCATION para ACCOUNTABLE_FOR_PROJECT_V*.

(a) Modelagem da Refatoração *Rename View*



(b) Propriedades do construtor *Rename View*

Fonte: (Autor, 2018)

Resultado: O código referente ao modelo resulta na remoção da Visão *EMPLOYEE_PROJECT_LOCATION* (linha 1), e na criação de uma nova *View* com o nome definido na propriedade *Rename To* do construtor, que neste caso é *ACCOUNTABLES_FOR_PROJECT_V*. A definição desta nova *View* vai da linha 2 à linha 7.

Listagem 5.9 – Código gerado pela modelagem da refatoração *Rename View*.

```
1 DROP VIEW IF EXISTS EMPLOYEE_PROJECT_LOCATION;
2 CREATE VIEW PROJECTS_ACCOUNTABLES_V AS SELECT P.PROJ_NAME,
3     E.NAME,
4     L.LOCATION
5    FROM ((PROJECT P
6      JOIN EMPLOYEE E ON ((P.PROJ_EMPLOYEE = E.EMP_SUPERVISOR)))
7      JOIN PROJECT L ON ((P.LOCATION_CODE = L.LOCATION_CODE)));
```

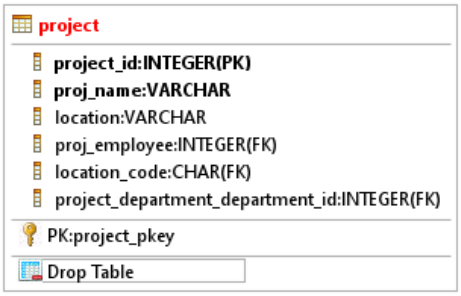
Cenário 12: Drop Table

Motivação: Pretende-se remover a tabela *PROJECT*, porém deve-se manter o histórico dos dados.

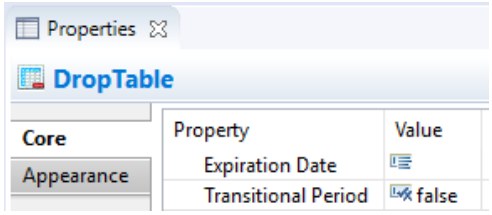
Implementação: Para modelar esta Refatoração, basta inserir o construtor *"Drop Table"* no compartimento de refatorações da tabela *PROJECT*.

Estrutura: Na Figura 35a, ilustra-se a refatoração *Drop Table*. Na Figura 35b, mostram-se as propriedades do construtor.

Figura 35 – Modelagem da Refatoração *Drop Table* para remover a tabela *PROJECT*



(a) Modelagem da Refatoração *Drop Table*



(b) Propriedades do construtor *Drop Table*

Fonte: (Autor, 2018)

Resultado: Nota-se, na Listagem 5.11, que a tabela *PROJECT* é removida como resultado da modelagem da refatoração. Caso defina-se um período transicional, cria-se uma tabela com a mesma especificação para manter o histórico.

Listagem 5.10 – Código gerado pela modelagem da refatoração *Drop Table*.

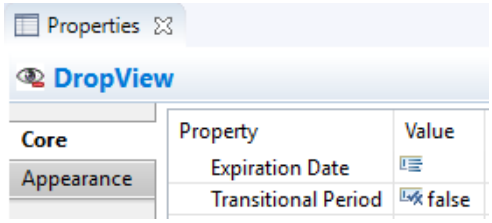
```
1 DROP TABLE PROJECT CASCADE;
```

Cenário 13: Drop View

Motivação: Busca-se remover a *View DEPARTMENTS_BUDGET*, que não é mais requerida.

Figura 36 – Modelagem da Refatoração *Drop View* para remover a Visão *DEPARTMENTS_BUDGET*

(a) Modelagem da Refatoração *Drop View*



(b) Propriedades do construtor *Drop View*

Fonte: (Autor, 2018)

Implementação: Para aplicar esta Refatoração, deve-se inserir o construtor *"Drop View"* no compartimento de refatorações do elemento visual *View*.

Estrutura: Na Figura 36a apresenta-se modelagem da Refatoração *Drop View*. Suas propriedades podem ser vistas na Figura 36b.

Listagem 5.11 – Código gerado pela modelagem da refatoração *Drop View*.

```
1 DROP VIEW IF EXISTS DEPARTMENTS_BUDGET_V;
```

Resultado: Como disposto na Listagem 5.12, onde encontra-se o código *SQL*, executa-se uma declaração para remoção da Visão.

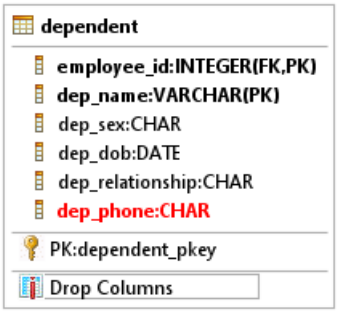
Cenário 14: Drop Column

Motivação: Deseja-se remover a coluna *DEP_PHONE* da tabela *DEPENDENT*, porém o histórico desta coluna deve ser mantido.

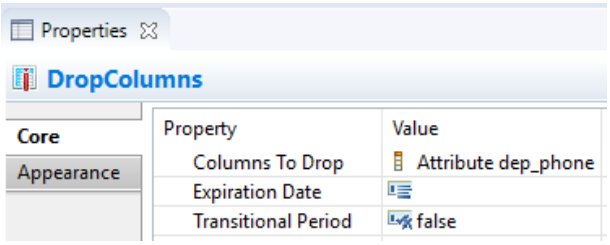
Implementação: Para aplicar esta refatoração deve-se inserir o construtor *Drop Column* no compartimento de refatoração da tabela que contém a coluna que deve ser removida. Vale ressaltar que mais de uma coluna pode ser removida. Isto ocorre por intermédio da propriedade *Columns to Drop*. Neste caso, esta propriedade recebeu a coluna *DEP_PHONE*. Alternativamente, pode-se ajustar o período transicional, o que leva à criação de uma tabela de histórico para manter as colunas removidas, junto com seu determinante na tabela (i.e., chave primária). Isto permite que a ação possa ser desfeita, caso necessário.

Estrutura: Na Figura 37a apresenta-se a modelagem desta refatoração, bem como as propriedades do construtor *"Drop Column"* (Figura 37b).

Figura 37 – Modelagem da Refatoração *Drop Column* para remover a coluna *DEP_PHONE* com tabela de histórico.



(a) Modelagem da Refatoração *Drop Column*



(b) Propriedades do construtor *Drop Column*

Fonte: (Autor, 2018)

Resultado: Como apresentado na Listagem 5.13 remove-se a coluna *DEP_PHONE* da tabela *DEPENDENT*.

Listagem 5.12 – Código gerado pela modelagem da refatoração *Drop Table*.

```
1 ALTER TABLE DEPENDENT DROP COLUMN DEP_PHONE;
```

Cenário 15: Split Table

Motivação: Pretende-se mover as colunas *LOCATION* e *LOCATION_CODE* para uma nova tabela. Com isto, elimina-se as anomalias trazidas por uma tabela aninhada.

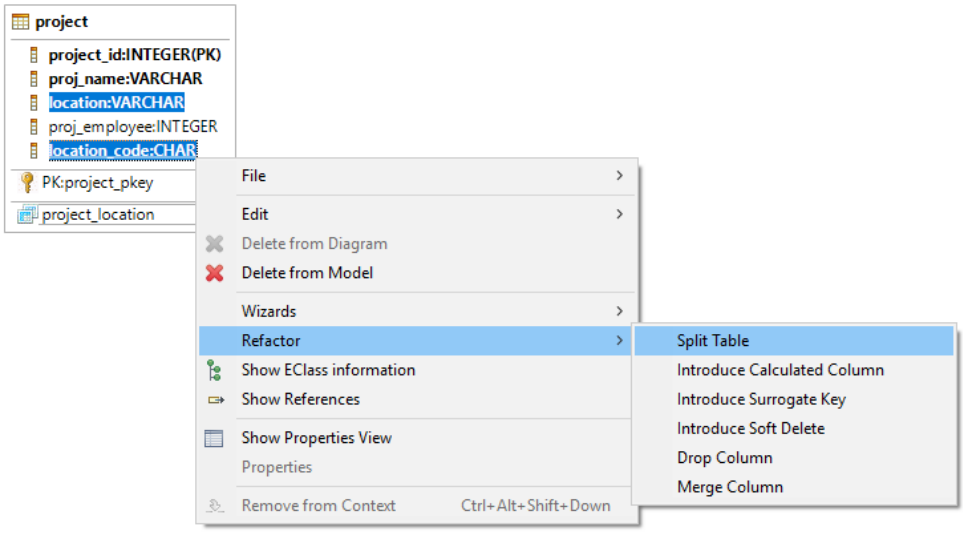
Implementação: Adiciona-se o construtor *Split Table* no compartimento de refatorações da tabela *PROJECT* e definem-se as colunas que irão compor a nova tabela, que neste caso são *LOCATION* e *LOCATION_CODE*. Após isso, ajustam-se as propriedade: *Distinct Columns* - para indicar qual coluna será a chave primária da tabela gerada. Neste caso, aponta-se a coluna LOCATION_CODE; e *Target Table Name* - para especificar o nome da nova tabela gerada (i.e., *PROJECT_LOCATION*) a partir de *PROJECT*. Alternativamente, pode-se aplicar a Refatoração *Split Table* por meio da seleção das colunas e escolha da refatoração no menu de contexto. Este menu é acessado ao clicar com o botão

direito do *mouse* sobre as colunas que compõem a nova tabela. Neste caso, com exceção do nome da nova tabela, as demais propriedades são preenchidas de forma automática.

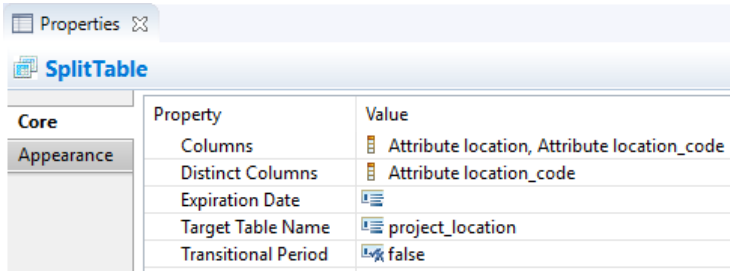
Estrutura: Na modelagem presenta na Figura 38a pode-se observar o menu de contexto que contém a Refatoração *Split Table* e as colunas envolvidas na refatoração. Nota-se, também, o nome da nova tabela no rótulo do construtor. Já na Figura 38b, apresentam-se as propriedades do construtor.

Figura 38 – Modelagem da Refatoração *Split Table* para definir uma nova tabela PROJECT_LOCATION baseada nas colunas LOCATION_CODE e LOCATION da tabela PROJECT

.



(a) Modelagem da Refatoração *Split Table*



(b) Propriedades do construtor *Split Table*

Fonte: (Autor, 2018)

Resultado: Na Listagem 5.14 (linhas 1 e 2), pode-se visualizar a criação da tabela *PROJECT_LOCATION* e a migração das colunas *LOCATION* e *LOCATION_CODE* presentes na tabela *PROJECT*. Nas linhas 3 e 4, observa-se a criação da chave primária de *PROJECT_LOCATION* e chave a estrangeira de *PROJECT*, respectivamente. Por fim, na linha 5, vê-se a remoção da coluna que não faz parte da chave estrangeira de *PROJECT* que referencia *PROJECT_LOCATION*.

Listagem 5.13 – Código gerado pela modelagem da refatoração *Split Table*.

```
1 CREATE TABLE PROJECT_LOCATION(LOCATION VARCHAR(50),LOCATION_CODE CHAR(3));
2 INSERT INTO PROJECT_LOCATION(LOCATION_CODE,LOCATION) SELECT DISTINCT LOCATION_CODE,LOCATION
      FROM PROJECT WHERE LOCATION_CODE IS NOT NULL;
3 ALTER TABLE PROJECT_LOCATION ADD CONSTRAINT PROJECT_LOCATION_PKEY PRIMARY KEY(LOCATION_CODE
    );
4 ALTER TABLE PROJECT ADD CONSTRAINT PROJECT_PROJECT_LOCATION_FKEY FOREIGN KEY(LOCATION_CODE)
      REFERENCES PROJECT_LOCATION(LOCATION_CODE) ON UPDATE CASCADE ON DELETE RESTRICT;
5 ALTER TABLE PROJECT DROP LOCATION;
```

Os exemplos de uso presentes nos cenários permitem avaliar a expressividade e abstração de cada construtor visando demonstrar a redução da complexidade inerente às refatorações estruturais de bancos de dados relacionais. Consequentemente, o esforço também é reduzido, permitindo que não só profissionais de dados executem mudanças no esquema de forma eficiente e eficaz, mas também outros profissionais menos especializados.

5.4 ANÁLISE CONSOLIDADA

No Quadro 1, exibe-se um comparativo das características presentes nos trabalhos avaliados no Capítulo 3 com os atributos da abordagem apresentada neste trabalho. Com esta analise observa-se que a ferramenta *REvolutionCASE* em comparação com as outras soluções: 1) não da suporte as características (i.e., *Replace LOB with Table* e *Split Column*), as quais também não são suportadas por nenhuma outra solução; 2) possui quase o dobro de expressividade (i.e., 88%) quando comparado com a solução concorrente mais expressiva (i.e., 48%); e 3) suporta com exclusividade as refatorações *Drop View*, *Introduce Calculated Column*, *Introduce Surrogate Key*, *Merge Columns*, *Replace One-to-Many with Associative Table*, *Replace Surrogate Key with Natural Key*.

Quadro 1 – Comparativo da ferramenta *SQL Refactor Studio* e dos trabalhos acadêmicos relacionados dispostos no Capitulo 3 com a feramenta *REvolutionCASE*.

Característica	Aboulsamh & Davies	Garcia et al.	Milovanovic & Milicev	*MSSQL Refactor Studio*	Curino et. Al.	REvolutionCASE	
Drop Column	SIM	SIM	SIM	NÃO	SIM	SIM	85%
Drop Table	SIM	SIM	SIM	NÃO	SIM	SIM	85%
Drop View	NÃO	NÃO	NÃO	NÃO	NÃO	SIM	17%
Introduce Calculated Column	NÃO	NÃO	NÃO	NÃO	NÃO	SIM	17%
Introduce Surrogate Key	NÃO	NÃO	NÃO	NÃO	NÃO	SIM	17%
Merge Columns	NÃO	NÃO	NÃO	NÃO	NÃO	SIM	17%
Merge Tables	NÃO	NÃO	NÃO	NÃO	SIM	SIM	34%
Move Column	SIM	SIM	SIM	SIM	SIM	SIM	100%
Rename Column	SIM	SIM	SIM	SIM	SIM	SIM	100%
Rename Table	SIM	SIM	SIM	SIM	SIM	SIM	100%
Rename View	NÃO	NÃO	NÃO	SIM	NÃO	SIM	34%
Replace LOB with Table	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO	0%
Replace Column	SIM	SIM	SIM	NÃO	SIM	SIM	85%
Replace One-to-Many with Associative Table	NÃO	NÃO	NÃO	NÃO	NÃO	SIM	17%
Replace Surrogate Key with Natural Key	NÃO	NÃO	NÃO	NÃO	NÃO	SIM	17%
Split Column	NÃO	NÃO	NÃO	NÃO	NÃO	NÃO	0%
Split Table	SIM	SIM	NÃO	NÃO	SIM	SIM	68%
Periodo Transicional	NÃO	NÃO	NÃO	SIM	NÃO	SIM	34%
Sintaxe Abstrata(Metamodelo)	SIM	SIM	SIM	NÃO	NÃO	SIM	68%
Sintaxe Concreta(Notação)	SIM	NÃO	SIM	NÃO	NÃO	SIM	51%
Semântica Estática	SIM	NÃO	NÃO	NÃO	NÃO	SIM	34%
Engenharia Reversa	NÃO	SIM	NÃO	SIM	NÃO	SIM	51%
Engenharia Direta	SIM	SIM	SIM	SIM	SIM	SIM	100%
Migração de Dados	SIM	SIM	SIM	SIM	SIM	SIM	100%
Cobertura	48%	44%	40%	32%	40%	88%	

Fonte: (Autor, 2018)

5.5 COONSIDERAÇÕES FINAIS

No presente capítulo efetua-se uma avaliação da DSML proposta por meio da ferramenta *REvolutionCASE* para modelagem de Refatorações Estruturais de bancos de dados. Exibe-se, também, o uso da ferramenta, bem como sua estrutura e utilização.

Por meio dos exemplos de uso, apresentam-se cenários no intuito de avaliar as evoluções suportadas utilizando-se os construtores providos pela DSML. No Quadro 1, apresenta-se um comparativo entre as evoluções que seriam alcançadas utilizando os caminhos definidos pelas ferramentas avaliadas no Capítulo 3 com a abordagem presente neste trabalho. No Quadro 1 também mostra-se que a abordagem aqui proposta apresenta duas fraquezas, que são as Refatorações *Replace LOB with Table* e *Split Column*. Por outro lado, apresenta-se algumas características inéditas no ramo da evolução de esquemas de bancos de dados como: 1) uma DSML para representar as Refatorações Estruturais; 2) uma família de

validações estruturais e regras de boa formação para modelagem das Refatorações; 3) um metamodelo que abstrai os conceitos das Refatorações Estruturais; e 4) o suporte ao Período Transicional para sincronização, viabilizando a manutenção do esquema em duas versões até que outros artefatos que utilizam este esquema se ajustem às mudanças.

No capítulo seguinte serão apresentadas as conclusões, bem como as contribuições e trabalhos futuros.

6 CONCLUSÃO

Este capítulo apresenta as considerações finais, contribuições e trabalhos futuros relacionados a este trabalho.

6.1 CONSIDERAÇÕES FINAIS

Neste trabalho, apresenta-se uma nova perspectiva à evolução de bancos de dados. Por meio da DSML presente na ferramenta *REvolutionCASE*, possibilita-se a modelagem de Refatorações Estruturais de bancos de dados. Esta proposta baseia-se no arcabouço definido por Brambilla, Cabot e Wimmer (2012) para especificação dos elementos que compõem a DSML proposta. Além disso, orienta-se pelo processo de desenvolvimento de DSML definido por Cho, Gray e Syriani (2012). Com isto, torna-se possível capturar os elementos do domínio presentes nas refatorações estruturais propostas por Ambler Scott W. e Sadalage (2006) e transformá-las nos construtores da DSML de forma correta. Este processo de desenvolvimento consiste na definição de uma sintaxe concreta (i.e., notação), sintaxe abstrata (i.e., metamodelo) e da semântica estática (i.e., regras de boa formação) da linguagem. Soma-se ao processo a etapa de análise de domínio para identificar os elementos do domínio que serão abstraídos pelos construtores da linguagem.

A abordagem proposta neste trabalho possui como principais características : 1) a Engenharia Reversa do esquema, que pode ser feita independente de aplicação de Refatoração; 2) a Geração de código e Engenharia Direta da refatoração modelada; 3) a cobertura à característica do Período Transicional, que permite a evolução de esquemas em ambiente multi-aplicação. Ainda que as refatorações *Introduce Surrogate Key*, *Introduce Natural Key* e *Merge Tables* não suportem o período transicional, em futuras iterações esta característica estará presente na sua totalidade; 4) a representação simplificada pra operações complexas. Evoluir bancos de dados é uma tarefa não trivial, porém com apenas alguns ajustes nos construtores da linguagem, junto com suas regras de boa formação, é possível abstrair toda complexidade da tarefa; e 5) a utilização de MDD como paradigma de desenvolvimento, o que permite a abstração dos conceitos do domínio baseando-se em um metamodelo.

Para avaliar a DSML proposta neste trabalho, implementou-se a ferramenta *REvolutionCASE* no intuito de disponibilizar os construtores desta linguagem. A partir destes construtores, pode-se modelar cenários de Refatoração e efetuar transformações M2T, ou seja, gerar código executável. Com isto, definiu-se cenários de avaliação para cada Refatoração coberta na proposta no sentido de avaliar a expressividade da abordagem e o nível de abstração de cada construtor por meio de exemplos de uso.

6.2 LIMITAÇÕES

- **Não realizou-se uma avaliação da DSML proposta e da ferramenta *REvolutionCASE* junto com alunos e/ou especialistas da área de bancos de dados.**
  Esta avaliação pode auxiliar no processo de adoção da ferramenta, além de ajudar na identificação forças e fraquezas não contempladas pela abordagem, mas que são provenientes tanto do ambiente acadêmico quanto do setor industrial.

- **Nem todas as regras de boa formação foram capturadas.** Apesar de cobrir uma vasta gama de regras de boa formação que validem o modelo, pode haver regras que ainda não estão presentes na abordagem.

- **As Refatorações *Replace LOB with Table* e *Split Column* ainda não são suportadas pela DSML.** Como visto no Quadro 1, estas refatorações ainda não são suportadas pela ferramenta *REvolutionCASE*.

- **As Refatorações *Introduce Surrogate Key*, *Introduce Natural Key* e *Merge Tables* ainda não possuem suporte ao Período Transicional.** Estas refatorações se limitam à solução dos problemas onde o Período Transicional não é requerido.

- **A ferramenta *REvolutionCASE* possui suporte somente ao SGBD PostgreSQL.**
  Isto acaba impossibilitando a adoção da DSML em ambientes onde outro SGBD(e.g., Oracle.) seja utilizado.

6.3 CONTRIBUIÇÕES

As principais contribuições deste trabalho são:

- **Uma DSML para Refatorações Estruturais de Bancos de Dados Relacionais.** O conceito de modelo executável entregue pela abordagem MDD foi aplicado na construção da linguagem de modelagem proposta para permitir que o usuário possa executar alterações não triviais nos bancos de dados por meio da refatoração. Os construtores da DSML apresentada neste trabalho foi construída de acordo com os padrões de refatorações estruturais definidos por Ambler Scott W. e Sadalage (2006). Apesar de não cobrir as refatorações *Split Column* e *Replace LOB With Table* , as 15 refatorações estruturais restantes estão presentes da DSML, junto com o Período Transicional e a Migração de Dados.

- **Um metamodelo que viabiliza a implementação da DSML proposta.**
  O metamodelo definido neste trabalho pode servir de base não só para a extensão da DSML por meio da adição de outras refatorações de outras categorias, mas também

como ponto de partida para outras aplicações como coevolução e versionamento de esquemas, por exemplo. Consequentemente, isto pode auxiliar o processo de pesquisas relacionadas às evoluções de bancos de dados que utilizam Refatoração.

- **Uma ferramenta para modelagem e evolução de banco de dados Relacional.** Para prover construções dos elementos do Modelo Relacional, o metamodelo *RMM* é estendido pelo *EvoDB*. Estes metamodelos serviram como alicerce para a ferramenta *REvolutionCASE* que além de permitir a modelagem de Refatorações possui as seguintes características: 1) permite a Engenharia Reversa do banco de dados; 2) efetua validações (i.e., estruturais e de boa formação) das construções para garantir a qualidade do modelo; 3) promove a geração de código *SQL* da Refatoração na área de código; e 4) realiza a Engenharia Direta do modelo para o banco de dados que se aplica Refatoração.

6.4 TRABALHOS FUTUROS

- **Construir e integrar metamodelos que abstraem as refatorações de outras categorias.** Metamodelos podem ser integrados de forma simples e transparente. Sendo assim, entende-se que a abordagem presente neste trabalho serve como bloco de construção para a adição de novas refatorações de forma a obter não só os benefícios entrega pela abordagem, mas também aproveitar-se da arquitetura proposta para tornar-se uma solução às diversas refatorações que podem ser abstraídas pela ferramenta.

- **Estender o dialeto suportado pela ferramenta *REvolutionCASE*.** Adicionar suporte à outros bancos de dados como *MariaDB*, *Oracle* e etc.

- **Avaliar a DSML no ramo acadêmico e de mercado.** Aplicar a ferramenta com participação de alunos de bancos de dados para ensinar os conceitos de refatoração e evolução de bancos de dados. Avaliar, também, junto com especialistas no sentido de aprimorar a solução.

- **Efetuar experimentos para:** 1) identificar o nível de efetividade e deficiências da DSML; 2) observar seu comportamento em ambientes distintos; e 3) identificar novos padrões que possam convergir para a DSML proposta.

REFERÊNCIAS

ABOULSAMH MOHAMMED A. E DAVIES, J. A metamodel-based approach to information systems evolution and data migration. In: HALL, J.; KAINDL, H.; LAVAZZA, L.; BUCHGEHER, G.; TAKAKI, O. (Ed.). *ICSEA*. [S.l.]: IEEE Computer Society, 2010. p. 155–161. ISBN 978-0-7695-4144-0.

AMBLER SCOTT W. E SADALAGE, P. J. *Refactoring Databases: Evolutionary Database Design*. [S.l.]: Addison-Wesley, 2006. ISBN 0-321-29353-3.

ATKINSON, C.; KUEHNE, T. Model-driven development: A metamodeling foundation. *IEEE Software*, v. 20, n. 5, p. 36–41, 2003.

BRAMBILLA, M.; CABOT, J.; WIMMER, M. *Model-Driven Software Engineering in Practice*. [S.l.]: Morgan Claypool Publishers, 2012. (Synthesis Lectures on Software Engineering).

CHO, H.; GRAY, J. G.; SYRIANI, E. Creating visual domain-specific modeling languages from end-user demonstration. In: ATLEE, J. M.; BAILLARGEON, R.; FRANCE, R. B.; GEORG, G.; MOREIRA, A.; RUMPE, B.; ZSCHALER, S. (Ed.). *MiSE*. [S.l.]: IEEE Computer Society, 2012. p. 22–28. ISBN 978-1-4673-1757-3.

CURINO, C.; MOON, H. J.; DEUTSCH, A.; ZANIOLO, C. Automating the database schema evolution process. *VLDB J.*, v. 22, n. 1, p. 73–98, 2013.

CURINO CARLO E MOON, H. J. e. Z. C. Automating database schema evolution in information system upgrades. In: DUMITRAS, T.; NEAMTIU, I.; TILEVICH, E. (Ed.). *HotSWUp*. [S.l.]: ACM, 2009. ISBN 978-1-60558-723-3.

ELMASRI RAMEZ E NAVATHE, S. *Fundamentals of Database Systems*. 6. ed. [S.l.]: Prentice Hall International, 2010.

EVA; MOONEN, L. Java quality assurance by detecting code smells. In: . [S.l.]: IEEE Press, 2002. p. 97–107.

FERNANDES, L. A. *Uma Linguagem de Modelagem e uma Ferramenta CASE para apoiar o Projeto Lógico de Banco de Dados Relacionais*. Dissertação (Mestrado) — Universidade Federal de Pernambuco, Brazil, 2017.

FOWLER, M. *Refactoring - Improving the Design of Existing Code*. [S.l.]: Addison-Wesley, Reading/Massachusetts, 1999.

GARCíA, J.; DíAZ OSCAR E CABOT, J. An adapter-based approach to co-evolve generated sql in model-to-text transformations. In: JARKE, M.; MYLOPOULOS, J.; QUIX, C.; ROLLAND, C.; MANOLOPOULOS, Y.; MOURATIDIS, H.; HORKOFF, J. (Ed.). *CAiSE*. [S.l.]: Springer, 2014. (Lecture Notes in Computer Science, v. 8484), p. 518–532. ISBN 978-3-319-07880-9.

HUTCHINSON, J.; WHITTLE, J.; ROUNCEFIELD, M. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. *Sci. Comput. Program.*, v. 89, p. 144–161, 2014.

KELLY, S.; TOLVANEN, J.-P. *Domain-Specific Modeling: Enabling Full Code Generation*. [S.l.]: Wiley, 2008. ISBN 978-0-470-03666-2.

KOLOVOS, D.; ROSE, L.; GARCIA-DOMINGUEZ, A.; PAIGE, R. *The Epsilon Book (2010)*. 2012.

MARIANI, T.; VERGILIO, S. R. A systematic review on search-based refactoring. *Information Software Technology*, v. 83, p. 14–34, 2017.

MENS, T.; DEMEYER, S. *Software Evolution*. 1. ed. [S.l.]: Springer Publishing Company, Incorporated, 2008. ISBN 3540764399, 9783540764397.

MERNIK, M.; HEERING, J.; SLOANE, A. M. When and how to develop domain-specific languages. *ACM Comput. Surv.*, ACM Press, New York, NY, USA, v. 37, n. 4, p. 316–344, 2005. ISSN 0360-0300.

MILOVANOVIC VUKASIN E MILICEV, D. An interactive tool for uml class model evolution in database applications. *Software and System Modeling*, v. 14, n. 3, p. 1273–1295, 2015.

MOORE, B. *Eclipse Development: Using the Graphical Editing Framework and the Eclipse Modeling Framework*. [S.l.]: IBM Corporation, International Technical Support Organization, 2004.

NEIGHBORS, J. M. *Software construction using components*. Tese (Doutorado) — University of California, Irvine, 1980.

PEFFERS, K.; TUUNANEN, T.; ROTHENBERGER, M. A.; CHATTERJEE, S. A design science research methodology for information systems research. *Journal of Management Information Systems*, v. 24, n. 3, p. 45–77, 2007.

PICEK, R.; STRAHONJA, V. Model driven development - future or failure of software development. In: *In IIS'07: 18th International Conference on Information and Intelligent Systems*. [S.l.: s.n.], 2007.

PRIETO-DIAZ, R. Domain analysis: An introduction. *ACM SIGSoft Software Engineering Notes*, v. 15, n. 2, p. 47–54, April 1990.

QIU, D.; LI, B.; SU, Z. An empirical analysis of the co-evolution of schema and code in database applications. In: MEYER, B.; BARESI, L.; MEZINI, M. (Ed.). *ESEC/SIGSOFT FSE*. [S.l.]: ACM, 2013. p. 125–135. ISBN 978-1-4503-2237-9.

RATZINGER, J.; FISCHER, M.; GALL, H. C. Improving evolvability through refactoring. *ACM SIGSOFT Software Engineering Notes*, v. 30, n. 4, p. 1–5, 2005.

SCHMIDT, D. C. Model Driven Engineering. *IEEE Computer*, v. 39, n. 2, p. 25–31, fev. 2006.

SELIC, B. The pragmatics of model-driven development. *IEEE Software*, v. 20, n. 5, p. 19–25, 2003.

SHATALIN, A.; TIKHOMIROV, A. Graphical modeling framework architecture overview. In: *Eclipse modeling symposium*. [S.l.: s.n.], 2006.

SIEGEL, J. Object management group model driven architecture (mda) mda guide rev. 2.0. 2014. Disponível em: <http://www.omg.org/cgi-bin/doc?ormsc/14-06-01>.

TOPCU, O.; DURAK, U.; OGUZTüZüN, H.; YILMAZ, L. *Distributed Simulation - A Model Driven Engineering Approach*. [S.l.]: Springer, 2016. 3-270 p. (Simulation Foundations, Methods and Applications). ISBN 978-3-319-03050-0.

VOELTER, M.; BENZ, S.; DIETRICH, C.; ENGELMANN, B.; HELANDER, M.; KATS, L. C. L.; VISSER, E.; WACHSMUTH, G. *DSL Engineering - Designing, Implementing and Using Domain-Specific Languages.* [S.l.]: dslbook.org, 2013. 1-558 p. ISBN 978-1-4812-1858-0.

WHITTLE, J.; HUTCHINSON, J.; ROUNCEFIELD, M. The state of practice in model driven engineering. *IEEE Software*, v. 31, n. 3, p. 79–85, 2014.

APÊNDICE A – REFATORAÇÕES ESTRUTURAIS

A seguir são apresentados os 17 padrões estruturais definidos por Ambler Scott W. e Sadalage (2006):

1 Split Table

Objetivo: Particionar, por colunas, uma tabela existente em uma ou mais tabelas.
Motivação: Melhorar performance; Eliminar redundância de dados.
Potenciais Tradeoffs: Queda da performance ao produzir uma nova tabela e uma nova junção.
Mecânica de Atualização do *Schema*: (I) Criar nova tabela; (II) Mover colunas da tabela antiga para a nova tabela; (III) Definir chave primária da nova tabela; (IV) Definir chave estrangeira estrangeira para nova tabela; e (V) Criação de *trigger* para sincronização de dados das colunas antigas e novas.
Mecânica de migração de dados:Inserir campos da tabela antiga na nova tabela, obedecendo a integridade referencial e dependência funcional dos dados.
Período Transicional: Sim.

2 Drop Column

Objetivo: Remover uma coluna da tabela.
Motivação: A coluna não é mais utilizada.
Potenciais *Tradeoffs*: Pode levar a perda de dados relevantes.
Mecânica de Atualização do *Schema*: (I) Criar tabela de histórico para armazenar coluna a ser removida.
Mecânica de migração de dados:Inserção da coluna a ser removida na tabela de histórico. Garantir a dependência funcional da coluna na tabela de histórico para viabilizar a operação inversa,caso necessário.
Período Transicional: Sim.

3 Drop Table

Objetivo: Remover uma tabela existente do banco de dados.
Motivação: A tabela não é mais utilizada; A tabela foi substituída por uma view ou outra fonte de dados.
Potenciais Tradeoffs: Perca dos dados da tabela e risco de problemas com tabelas órfãs.
Mecânica de Atualização do *Schema*: (I) Criar tabela de histórico para armazenar coluna a ser removida.

Mecânica de migração de dados:Inserção dos campos da tabela a ser removida na tabela de histórico. Com isto torna-se possível reverter a remoção.
Período Transicional: Sim.

4 Drop View

Objetivo: Remover uma *View* existente do banco de dados.
Motivação: A *View* não é mais requerida.
Potenciais Tradeoffs: Não remove dados do banco de dados, mas pode levar a aplicação que consome a *View* a falha.
Mecânica de Atualização do *Schema*: Não possui.
Mecânica de migração de dados: Não há necessidade de migração de dados.
Período Transicional: Sim.

5 Introduce Calculated Column

Objetivo: Introduz uma nova coluna baseada no cálculo de valores de uma ou mais colunas.
Motivação: Melhoria da performance da aplicação.
Potenciais Tradeoffs:Perca de sincronização com os dados atuais das colunas calculadas.
Mecânica de Atualização do *Schema*: (I)Adicionar coluna calculada; (II) Determinar cálculo da coluna; (III) Determinar quais colunas estarão presentas na função matemática; (IV) Criação de *trigger* para atualizar a coluna calculada com o resultado da função.
Mecânica de migração de dados:Não há necessidade de migração de dados.
Período Transicional: Não.

6 Introduce Surrogate Key

Objetivo: Substituir uma chave natural por uma chave substituta.
Motivação: Melhorar a performance do banco de dados, já que algumas chaves naturais podem ser compostas e/ou ter seu valor repleto de caracteres.
Potenciais Tradeoffs:Alteração das chaves estrangeiras nas tabelas filhas pode ser custosa e propensa a erros.
Mecânica de Atualização do *Schema*: (I) Adicionar nova chave primaria; (II) Adicionar nova chave estrangeira nas tabelas filhas; (III) Atualizar integridades referenciais; (IV) Criação de *trigger* para sincronização, caso o ambiente seja multi-aplicação.
Mecânica de migração de dados:Inserir novos valores para chave primária e respectivas chaves estrangeiras nas tabelas filhas.

Período Transicional: Sim.

7 Merge Columns

Objetivo: Fundir duas ou mais colunas na mesma tabela.
Motivação: Corrigir erros de design e melhorar o nível de granularidade da coluna.
Potenciais Tradeoffs: Pode levar à perda de precisão de dados.
Mecânica de Atualização do *Schema*: (I) Adicionar nova coluna;(II) Remover colunas fundidas.(III) Criação de *trigger* para sincronização, caso o ambiente seja multi-aplicação.
Mecânica de migração de dados: Inserção dos valores das colunas fundidas na nova coluna de fusão.
Período Transicional: Sim.

8 Merge Table

Objetivo: Fundir duas ou mais tabelas.
Motivação: Corrigir erros de design e granularidade da tabela.
Potenciais Tradeoffs: Perca da precisão dos dados quando aplicado em um contexto inadequado.
Mecânica de Atualização do *Schema*: (I) Adição de nova tabela; (II) Adição das colunas das tabelas fundidas;(III) Definição da chave primária; (IV) Criação de *trigger* para sincronização, caso o ambiente seja multi-aplicação.
Mecânica de migração de dados: Inserção dos valores das tabelas fundidas na nova tabela.
Período Transicional: Sim.

9 Move Column

Objetivo: Migrar uma coluna e seus dados para outra tabela existente.
Motivação: Normalização ou desnormalização de dados.
Potenciais Tradeoffs: Mover colunas para obter normalização pode reduzir a performance do banco de dados.
Mecânica de Atualização do *Schema*: (I) Adicionar nova coluna na tabela alvo; (II) Criação de *trigger* para sincronização, caso o ambiente seja multi-aplicação.
Mecânica de migração de dados: Inserção dos valores da coluna que está sendo migrada da tabela fonte para a tabela destino.
Período Transicional: Sim.

10 Rename Column

Objetivo: Renomear uma coluna existente.

Motivação: Viabilizar portabilidade do esquema; Melhorar a leitura do esquema; Adequação a algum determinado padrão.
Potenciais Tradeoffs: Custo da refatoração do banco de dados pode ser maior que o ganho com sua utilização, ou seja, esta evolução pode requerer uma série de modificações nas aplicações externas. Como por exemplo a mudança nos nomes dos atributos de uma classe e/ou substituição dos nomes das colunas nas consultas.
Mecânica de Atualização do *Schema*: (I) Adicionar nova coluna (II) Remover a coluna anterior. (III) (II) Criação de *trigger* para sincronização das colunas, caso seja necessário um período transicional.
Mecânica de migração de dados: Inserção dos valores da coluna anterior na nova coluna.
Período Transicional: Sim.

11 Rename Table

Objetivo: Renomear uma tabela existente.
Motivação: Melhorar o significado da tabela em relação ao esquema.
Potenciais Tradeoffs: Custo da refatoração do banco de dados pode ser maior que o ganho com a aplicação do padrão. Isto significa que a modificação pode propagar uma série de alterações para as aplicações externas. Um exemplo seria a mudança em todas as consultas que consomem a tabela renomeada.
Mecânica de Atualização do *Schema*: (I) Adição de nova tabela; ou (I) Renomeação da tabela; (II) Criação de uma *View* para sincronização, caso o ambiente seja multi-aplicação; (III) Remoção da tabela anterior.
Mecânica de migração de dados: Efetuar a inserção, na nova tabela, de todas as colunas da tabela anterior.
Período Transicional: Sim.

12 Rename View

Objetivo: Renomear uma *View* existente.
Motivação: Melhorar o significado da *View* em relação ao esquema.
Potenciais Tradeoffs: O custo das mudanças nas aplicações externas que consomem a *View* pode ser maior que o benefício trazido pela evolução. O custo de mudar o código, parar e redistribuir a aplicação pode ser um limitador para esta refatoração.
Mecânica de Atualização do *Schema*: (I) Criação da *View* com novo nome; (II) Remoção da *View* anterior.
Mecânica de migração de dados: Não há necessidade de migração de dados.
Período Transicional: Sim.

13 Replace LOB With Table

Objetivo: Substituir um *Large Object*(lob) contido em uma coluna por uma nova tabela.
Motivação: Tratar partes de um *Large Object* como elementos distintos de dados.
Potenciais *Tradeoffs*: Estruturas de dados contidas em *Large Objects* são bastantes complexa. Por isso, demandam mais tempo para tratamento dos dados, o que pode gerar um alto custo para implementação deste padrão.
Mecânica de Atualização do schema: (I) Adicionar nova(s) tabela(s); (II) Definição das restrições de dependências funcionais e integridades Referenciais.
Mecânica de migração de dados: Extrair dados do *lob* e inserir na(s) nova(s) tabela(s)
Período Transicional: Sim.

14 Replace Column

Objetivo: Substituir uma coluna não chave por uma nova coluna.
Motivação: Mudanças no uso da coluna ou nos requisitos podem demandar uma modificação no tipo de dado. Por exemplo a troca de uma coluna numérica para um tipo alfanumérico para acomodar um caractere especial.
Potenciais Tradeoffs: Perca na transferência de dados. Por exemplo, CHAR para NUMERIC.
Mecânica de Atualização do schema: (I) Adição de coluna; (II) Remoção da coluna substituída; (III) Criação de trigger para sincronização, caso o ambiente seja multi-aplicação.
Mecânica de migração de dados: Inserção dos dados da coluna substituída na nova coluna.
Período Transicional: Sim.

15 Replace One-To-Many With Associative Table

Objetivo: Substitui um relacionamento 1:N entre duas tabelas por uma tabela associativa.
Motivação: Preparar as tabelas para receber um potencial relacionamento N:M.
Potenciais Tradeoffs: Aumento na quantidade de junções e, consequentemente, redução da performance.
Mecânica de Atualização do schema: (I) Adição da tabela associativa (II) Definição das restrições de dependências funcionais e integridades Referenciais. (III) Criação de trigger para sincronização, caso o ambiente seja multi-aplicação.
Mecânica de migração de dados: Inserção dos dados das restrições de dependência funcional e integridade referencial na tabela associativa.
Período Transicional: Sim.

16 Replace Surrogate Key With Natural Key

Objetivo: Substituir uma chave substituta por uma chave natural existente. Oposto ao Introduce Surrogate Key.
Motivação: Consolidar a estratégia de chaves.
Potenciais Tradeoffs: Alteração das chaves estrangeiras nas tabelas filhas pode ser custosa e propensa a erros.
Mecânica de Atualização do schema: (I) Remover restrições anteriores (II) Criar novas restrições de dependências funcionais e integridades Referenciais.
Mecânica de migração de dados: Não há necessidade de migração de dados.
Período Transicional: Sim.

17 Split Column

Objetivo: Distribuir uma coluna em uma ou mais colunas dentro da mesma tabela.
Motivação: Redução do nível de granularidade de uma coluna.
Potenciais Tradeoffs: Redução do nível de granularidade de uma coluna é uma tarefa bastante complexa e geralmente requer mais de uma aplicação para tratamento dos dados.
Mecânica de Atualização do schema: (I) Adicionar novas colunas; (II) Definição das restrições de dependências funcionais e integridades.
Mecânica de migração de dados: Extrair dados da coluna com alto nível de granularidade e inserir nas novas colunas.
Período Transicional: Sim.

APÊNDICE B – CÓDIGOS EGL

Este apêndice contém as regras EGL para transformação M2T.

```
1
2  for(element in DropColumns.allInstances()) {
3  if(element.transitionalPeriod==false){
4  out.println("--BEGIN DROP COLUMN PATTERN SCRIPT");
5
6  out.println();
7  out.print("ALTER TABLE ");
8  out.print(element.eContainer().name.toUpperCase());
9  for(attribute in element.columnsToDrop){
10
11 if(element.columnsToDrop.last().equals(attribute)){
12 out.print(" DROP COLUMN ");
13 out.print(attribute.name.toUpperCase());
14 }else{
15 out.print(" DROP COLUMN ");
16 out.print(attribute.name.toUpperCase()+",");
17 }
18 }
19 out.println(";");
20 }else{
21 out.println("--BEGIN DROP COLUMN PATTERN SCRIPT");
22
23 out.print("CREATE TABLE ");
24 out.print(element.eContainer().name.toUpperCase()+"_HISTORY");
25 out.println("(");
26
27 for(attribute in element.columnsToDrop){
28 if(element.columnsToDrop.last().equals(attribute)){
29   out.print(attribute.name.toUpperCase()+" "+attribute.getDataType(attribute));
30   if(attribute.defaultValue.isDefined()){
31   out.print(" DEFAULT "+attribute.defaultValue);
32   }
33   if(attribute.isNotNull = true){
34   out.print(" NOT NULL ");
35   }
36   }else{
37  out.print(attribute.name.toUpperCase()+" "+attribute.getDataType(attribute));
38   if(attribute.defaultValue.isDefined()){
39   out.print(" DEFAULT "+attribute.defaultValue);
40   }
41   if(attribute.isNotNull = true){
42   out.print(" NOT NULL ");
```

```
43   }
44   out.println(",");
45   }
46 }
47 out.println(");");
48 out.println();
49 out.print("ALTER TABLE "+element.eContainer().name.toUpperCase()+"_HISTORY");
50 for(constraint in element.eContainer().relationConstraints->select(c|c.isKindOf(PrimaryKey)
       )){
51 for(sAttribute in constraint.attributes){
52 if(sAttribute.equals(getPrimaryKeyAttributes(element.eContainer()).last())){
53 out.print(" ADD "+sAttribute.name.toUpperCase()+" "+sAttribute.getDataType(sAttribute));
54 }else{
55 out.print(" ADD "+sAttribute.name.toUpperCase()+" "+sAttribute.getDataType(sAttribute)+",")
       ;
56 }}}
57 out.println(";");
58 out.println();
59 out.print("ALTER TABLE "+element.eContainer().name.toUpperCase()+"_HISTORY");
60 out.print(" ADD CONSTRAINT PK_"+element.eContainer().name.toUpperCase()+"_HISTORY");
61 out.print(" PRIMARY KEY(");
62 for(constraint in element.eContainer().relationConstraints->select(c|c.isKindOf(PrimaryKey)
       )){
63 for(sAttribute in constraint.attributes){
64 if(sAttribute.equals(getPrimaryKeyAttributes(element.eContainer()).last())){
65 out.print(sAttribute.name.toUpperCase());
66 }else{
67 out.print(sAttribute.name.toUpperCase()+",");
68 }}}
69 out.println(");");
70 out.println();
71 out.print("INSERT INTO ");
72 out.print(element.eContainer().name.toUpperCase()+"_HISTORY");
73 out.print("(");
74 for(constraint in element.eContainer().relationConstraints->select(c|c.isKindOf(PrimaryKey)
       )){
75 for(sAttribute in constraint.attributes){
76 if(sAttribute.equals(getPrimaryKeyAttributes(element.eContainer()).last())){
77 if(element.columnsToDrop.size() >1){
78 out.print(sAttribute.name.toUpperCase()+",");
79 }else{
80 out.print(sAttribute.name.toUpperCase());
81 }
82 }else{
83 out.print(sAttribute.name.toUpperCase()+",");
84 }}}
85 for(attribute in element.columnsToDrop){
```

```
86 if(element.columnsToDrop.last().equals(attribute)){
87 out.print(attribute.name.toUpperCase());
88 }else{
89 out.print(attribute.name.toUpperCase()+",");
90 }}
91 out.print(")");
92 out.print(" SELECT ");
93 for(constraint in element.eContainer().relationConstraints->select(c|c.isKindOf(PrimaryKey)
       )){
94 for(sAttribute in constraint.attributes){
95 if(sAttribute.equals(getPrimaryKeyAttributes(element.eContainer()).last())){
96 if(element.columnsToDrop.size() >1){
97 out.print(sAttribute.name.toUpperCase()+",");
98 }else{
99 out.print(sAttribute.name.toUpperCase());
100 }
101 }else{
102 out.print(sAttribute.name.toUpperCase()+",");
103 }}}
104 for(attribute in element.columnsToDrop){
105 if(element.columnsToDrop.last().equals(attribute)){
106 out.print(attribute.name.toUpperCase());
107 }else{
108 out.print(attribute.name.toUpperCase()+",");
109 }}
110 out.print(" FROM ");
111 out.println(element.eContainer().name.toUpperCase()+";");
112 out.println();
113 out.print("ALTER TABLE ");
114 out.print(element.eContainer().name.toUpperCase());
115
116 for(attribute in element.columnsToDrop){
117 if(element.columnsToDrop.last().equals(attribute)){
118 out.print(" DROP COLUMN ");
119 out.print(attribute.name.toUpperCase());
120 }else{
121 out.print(" DROP COLUMN ");
122 out.print(attribute.name.toUpperCase()+",");
123 }}
124 out.println(";");
125 out.print("COMMENT ON TABLE ");
126 out.print(element.eContainer().name.toUpperCase()+"_HISTORY");
127
128 out.print(" IS ");
129
130 if(element.expirationDate.isDefined()){
131 out.print("'Run the following commands by the date ");
```

```
132 out.print(element.expirationDate.asString()+"'");
133 }else{
134 out.print("'Date not defined!'");
135 }
136 out.println(";");
137 out.print("--DROP TABLE IF EXISTS ");
138 out.print(element.eContainer().name.toUpperCase()+"_HISTORY");
139 out.println(";");
140 }
141 out.println("--END DROP COLUMN PATTERN SCRIPT");
142 }
```

Listing B.1 – Regras de transformação M2T referente ao construtor *Drop Column*

```
1
2 for(element in CalculatedColumn.allInstances()) {
3 var primaryKey:PrimaryKey;
4 var element_id = element.eContainer().name.toUpperCase()+"_ID";
5 out.print("ALTER TABLE "+element.eContainer().name.toUpperCase()+" ADD ");
6 out.print(element.name.toUpperCase()+" "+element.getDataType(element));
7 out.println(";");
8 out.println();
9 if(element.columns->select(c | c.eContainer() = element.eContainer()).size() = element.
      columns.size()){
10 out.print("UPDATE "+element.eContainer().name.toUpperCase()+" SET ");
11 out.print(element.name.toUpperCase()+"=");
12 var calcPattern:String = element.calculationPattern.trim().toUpperCase();
13 for(attribute in element.columns){
14 if(attribute.name.toUpperCase().trim().isSubstringOf(calcPattern)){
15 calcPattern:= calcPattern.replace(attribute.name.toUpperCase().trim(),attribute.name.
      toUpperCase().trim());}}
16 out.print(calcPattern);
17 out.print(" WHERE ");
18 primaryKey := element.eContainer().constraints->select(p | p.isKindOf(PrimaryKey)).last();
19 for(attribute in primaryKey.attributes){
20 if(primaryKey.attributes.last().equals(attribute)){
21 out.print(attribute.name.toUpperCase()+"=");
22 out.print(attribute.name.toUpperCase());
23 }else{
24 out.print(attribute.name.toUpperCase()+"=");
25 out.print(attribute.name.toUpperCase()+",");
26 }}
27 out.println(";");
28 out.println();
29 out.println("CREATE OR REPLACE FUNCTION ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase()+
      "() RETURNS TRIGGER AS $"+"ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase()+"$");
30 out.println("BEGIN");
31 out.println("DECLARE");
```

```
32 out.println(element_id+" INTEGER;");
33 out.println("BEGIN");
34 primaryKey := element.eContainer().constraints->select(p | p.isKindOf(PrimaryKey)).last();
35 for(attribute in primaryKey.attributes){
36 if(primaryKey.attributes.last().equals(attribute)){
37 out.print(element_id+"=");
38 out.print("NEW."+attribute.name.toUpperCase());
39 }else{
40 out.print(element_id+"=");
41 out.print("NEW."+attribute.name.toUpperCase()+",");
42 }}
43 out.println(";");
44 out.println("IF(TG_OP = 'INSERT') THEN");
45 out.print("UPDATE "+element.eContainer().name.toUpperCase()+" SET ");
46 out.print(element.name.toUpperCase()+"=");
47 var calcPattern:String = element.calculationPattern.trim().toUpperCase();
48 for(attribute in element.columns){
49 if(attribute.name.toUpperCase().trim().isSubstringOf(calcPattern)){
50 calcPattern:= calcPattern.replace(attribute.name.toUpperCase().trim(),"NEW."+attribute.name
       .toUpperCase().trim());
51 }}
52 out.print(calcPattern);
53 out.print(" WHERE ");
54 primaryKey := element.eContainer().constraints->select(p | p.isKindOf(PrimaryKey)).last();
55 for(attribute in primaryKey.attributes){
56 if(primaryKey.attributes.last().equals(attribute)){
57 out.print(attribute.name.toUpperCase()+"=");
58 out.print(element_id);
59 }else{
60 out.print(attribute.name.toUpperCase()+"=");
61 out.print(element_id+",");
62 }}
63 out.println(";");
64 out.println("END IF;");
65 out.println("RETURN NEW;");
66 out.println("END;");
67 out.println("END;");
68 out.print("$"+"ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase()+"$");
69 out.println(" LANGUAGE plpgsql;");
70 out.println();
71 out.println();
72 out.print("DROP TRIGGER IF EXISTS ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase());
73 out.println(" ON "+element.eContainer().name.toUpperCase()+";");
74 out.println();
75 out.println("CREATE TRIGGER ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase());
76 out.println("AFTER INSERT ON "+element.eContainer().name.toUpperCase());
77 out.println("FOR EACH ROW");
```

```
78 out.println("EXECUTE PROCEDURE ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase()+"();");
79 out.println();
80 out.println("CREATE OR REPLACE FUNCTION ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase()+
       "UPDATE() RETURNS TRIGGER AS $"+"ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase()+"
       UPDATE$");
81 out.println("BEGIN");
82 out.println("DECLARE");
83 out.println("BEGIN");
84 out.print("IF(TG_OP = 'UPDATE' AND ");
85 primaryKey := element.eContainer().constraints->select(p | p.isKindOf(PrimaryKey)).last();
86 for(attribute in primaryKey.attributes){
87 if(primaryKey.attributes.last().equals(attribute)){
88 out.print("NEW."+attribute.name.toUpperCase()+"=");
89 out.print("OLD."+attribute.name.toUpperCase());
90 }else{
91 out.print("NEW."+attribute.name.toUpperCase()+"=");
92 out.print("OLD."+attribute.name.toUpperCase()+" AND ");
93 }}
94 out.println(") THEN");
95 out.print("NEW."+element.name.toUpperCase()+":=");
96 var calcPatternUpdate:String = element.calculationPattern.trim().toUpperCase();
97 for(attribute in element.columns){
98 if(attribute.name.toUpperCase().trim().isSubstringOf(calcPatternUpdate)){
99 calcPatternUpdate:= calcPatternUpdate.replace(attribute.name.toUpperCase().trim(),"NEW."+
       attribute.name.toUpperCase().trim());
100 }}
101 out.print(calcPatternUpdate);
102 out.println(";");
103 out.println("END IF;");
104 out.println("RETURN NEW;");
105 out.println("END;");
106 out.println("END;");
107 out.print("$"+"ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase()+"UPDATE$");
108 out.println(" LANGUAGE plpgsql;");
109 out.println();
110 out.println();
111 out.print("DROP TRIGGER IF EXISTS ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase()+"
       UPDATE");
112 out.println(" ON "+element.eContainer().name.toUpperCase()+";");
113 out.println();
114 out.println("CREATE TRIGGER ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase()+"UPDATE");
115 out.println("BEFORE UPDATE ON "+element.eContainer().name.toUpperCase());
116 out.println("FOR EACH ROW");
117 out.println("EXECUTE PROCEDURE ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase()+"UPDATE()
       ;");
118 }else{
119 var sourceRelation:ERelation;
```

```
120  for(ERelation in ERelations){
121    if(ERelation.constraints->select(c | c.isKindOf(ForeignKey)).size()>0){
122          for(constraint in ERelation.constraints->select(c | c.isKindOf(ForeignKey))){
123         if(element.eContainer().equals(constraint.foreignRelation)){
124       sourceRelation=ERelation;
125        } }}}
126 out.println("CREATE OR REPLACE FUNCTION ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase()+
        "() RETURNS TRIGGER AS $"+"ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase()+"$");
127 out.println("BEGIN");
128 out.println("DECLARE");
129 for(constraint in element.eContainer().constraints->select(c|c.isKindOf(PrimaryKey))){
130 for(tAttribute in constraint.attributes){
131 if(tAttribute.equals(getPrimaryKeyAttributes(tAttribute.eContainer()).last())){
132 out.print(tAttribute.eContainer().name.toUpperCase()+"_"+tAttribute.name.toUpperCase()+"_PK
        ");
133 out.println(" "+tAttribute.getDataType(tAttribute)+";");
134 }else{
135 out.print(tAttribute.eContainer().name.toUpperCase()+"_"+tAttribute.name.toUpperCase()+"_PK
        ");
136 out.println(" "+tAttribute.getDataType(tAttribute)+";");
137 }}}
138 out.print("CALC_"+element.name.toUpperCase());
139 out.println(" "+element.getDataType(element)+";");
140 out.println("BEGIN");
141 out.println("IF(TG_OP = 'INSERT' OR TG_OP = 'UPDATE' ) THEN");
142
143 //Pattern for composite key on Calculated Column
144 for(tConstraint in sourceRelation.constraints){
145 if(tConstraint.isKindOf(ForeignKey)){
146 for(sConstraint in element.eContainer().constraints){
147 if(sConstraint.isKindOf(PrimaryKey)){
148 for(tAttribute in tConstraint.attributes){
149 for(sAttribute in sConstraint.attributes){
150 if(tAttribute.equals(tConstraint.attributes.last()) and sAttribute.equals(sConstraint.
        attributes.last())){
151 out.print(sAttribute.eContainer().name.toUpperCase()+"_"+sAttribute.name.toUpperCase()+"_PK
        "+":=");
152 out.println("NEW."+tAttribute.name.toUpperCase()+";");
153 }else{
154      if(not tAttribute.equals(tConstraint.attributes.last()) and not sAttribute.equals(
            sConstraint.attributes.last())){
155      out.print(sAttribute.eContainer().name.toUpperCase()+"_"+sAttribute.name.toUpperCase()
            +"_PK"+":=");
156      out.println("NEW."+tAttribute.name.toUpperCase()+";");
157      }}}}}}}}
158 out.print("SELECT INTO ");
159 out.print("CALC_"+element.name.toUpperCase()+" ");
```

```
160 out.print(element.calculationPattern.toUpperCase());
161 out.print(" FROM ");
162 out.print(sourceRelation.name.toUpperCase());
163 out.print(" WHERE ");
164 for(tConstraint in element.eContainer().constraints){
165 if(tConstraint.isKindOf(PrimaryKey)){
166 for(sConstraint in sourceRelation.constraints){
167 if(sConstraint.isKindOf(ForeignKey)){
168 for(sAttribute in sConstraint.attributes){
169 if(sAttribute.equals(sConstraint.attributes.last())){
170 out.print(sAttribute.name.toUpperCase()+"=");
171 for(tAttribute in tConstraint.attributes){
172 if(tAttribute.equals(tConstraint.attributes.last())){
173 out.print("NEW."+sAttribute.name.toUpperCase());
174 }}
175 }else{
176 out.print(sAttribute.name.toUpperCase()+"=");
177 out.print("NEW."+sAttribute.name.toUpperCase()+" AND ");
178 }}}}}}
179 out.println(";");
180 out.print("UPDATE ");
181 out.print(element.eContainer().name.toUpperCase());
182 out.print(" SET ");
183 out.print(element.name.toUpperCase()+" = ");
184 out.print("CALC_"+element.name.toUpperCase()+" ");
185 out.print(" WHERE ");
186 for(tConstraint in sourceRelation.constraints){
187 if(tConstraint.isKindOf(ForeignKey)){
188 for(sConstraint in element.eContainer().constraints){
189 if(sConstraint.isKindOf(PrimaryKey)){
190 for(tAttribute in tConstraint.attributes){
191 for(sAttribute in sConstraint.attributes){
192 if(tAttribute.equals(tConstraint.attributes.last()) and sAttribute.equals(sConstraint.
        attributes.last())){
193 out.print(sAttribute.name.toUpperCase()+"=");
194 out.print(sAttribute.eContainer().name.toUpperCase()+"_"+sAttribute.name.toUpperCase()+"_PK
        ");
195 }else{
196     if(not tAttribute.equals(tConstraint.attributes.last()) and not sAttribute.equals(
            sConstraint.attributes.last())){
197 out.print(sAttribute.name.toUpperCase()+"=");
198 out.print(sAttribute.eContainer().name.toUpperCase()+"_"+sAttribute.name.toUpperCase()+"_PK
        "+" AND ");
199     }}}}}}}}
200 out.println(";");
201 out.println("RETURN NEW;");
202 out.println("CALC_"+element.name.toUpperCase()+":=0;");
```

```
203 out.println("END IF;");
204 out.println("END;");
205 out.println("END;");
206
207 out.print("$"+"ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase()+"$");
208 out.println(" LANGUAGE plpgsql;");
209 out.println();
210 out.print("DROP TRIGGER IF EXISTS ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase());
211 out.println(" ON "+sourceRelation.name.toUpperCase()+";");
212 out.println();
213 out.println("CREATE TRIGGER ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase());
214 out.println("AFTER INSERT OR UPDATE OR DELETE ON "+sourceRelation.name.toUpperCase());
215 out.println("FOR EACH ROW");
216 out.println("EXECUTE PROCEDURE ADDCALCULATEDCOLUMNFOR"+element.name.toUpperCase()+"();");
217 out.println();
218 out.print("UPDATE ");
219 out.print(element.eContainer().name.toUpperCase());
220 out.print(" SET ");
221 out.print(element.name.toUpperCase());
222 out.print("=");
223 out.print("(SELECT ");
224 out.print(element.calculationPattern.toUpperCase());
225 out.print(" FROM ");
226 out.print(sourceRelation.name.toUpperCase());
227 out.print(" WHERE ");
228 for(tConstraint in sourceRelation.constraints){
229 if(tConstraint.isKindOf(ForeignKey)){
230 for(sConstraint in element.eContainer().constraints){
231 if(sConstraint.isKindOf(PrimaryKey)){
232 for(tAttribute in tConstraint.attributes){
233 for(sAttribute in sConstraint.attributes){
234 if(tAttribute.equals(tConstraint.attributes.last()) and sAttribute.equals(sConstraint.
    attributes.last())){
235 out.print(sAttribute.eContainer().name.toUpperCase()+"."+sAttribute.name.toUpperCase()+"=")
    ;
236 out.print(tAttribute.eContainer().name.toUpperCase()+"."+tAttribute.name.toUpperCase());
237 }else{
238     if(not tAttribute.equals(tConstraint.attributes.last())
239     and not
240     sAttribute.equals(sConstraint.attributes.last())){
241 out.print(sAttribute.eContainer().name.toUpperCase()+"."+sAttribute.name.toUpperCase()+"=")
    ;
242 out.print(tAttribute.eContainer().name.toUpperCase()+"."+tAttribute.name.toUpperCase()+"
    AND ");
243     }}}}}}}}
244 out.println(");");
245 out.println();
```

```
246 out.println();
247 }
248 }
```

Listing B.2 – Regras de transformação M2T referente ao construtor *Calculated Column*

```
1  for(element in ReplaceKey.allInstances()) {
2    if(element.replaceStrategy = ReplaceKeyStrategy#SurrogateKey){
3         out.println("--BEGIN INTRODUCE SURROGATE KEY PATTERN SCRIPT");
4    var i:Integer=0;
5    out.print("ALTER TABLE IF EXISTS ");
6    out.print(element.eContainer().name.toUpperCase());
7    out.print(" ADD ");
8        for(attribute in element.newPrimaryKeyAttributes){
9        if(element.newPrimaryKeyAttributes.last().equals(attribute)){
10        out.print(attribute.name.toUpperCase()+" ");
11        out.print(attribute.getDataType(attribute));
12     }else{
13     out.print(attribute.name.toUpperCase()+" ");
14     out.print(attribute.getDataType(attribute)+",");
15     }
16   }
17   out.println(";");
18   out.print("CREATE SEQUENCE ");
19   out.println("SEQ_"+element.eContainer().name.toUpperCase()+";");
20   out.print("ALTER TABLE IF EXISTS "+element.eContainer().name.toUpperCase()+" ");
21    out.print(" ALTER COLUMN ");
22      for(attribute in element.newPrimaryKeyAttributes){
23      if(element.newPrimaryKeyAttributes.last().equals(attribute)){
24      out.print(attribute.name.toUpperCase()+" ");
25      out.print("SET DEFAULT NEXTVAL('SEQ_"+element.eContainer().name.toUpperCase()+"')");
26      }else{
27      out.print(attribute.name.toUpperCase()+" ");
28      out.print("SET DEFAULT NEXTVAL('SEQ_"+element.eContainer().name.toUpperCase()+"'),");
29      }
30   }
31   out.println(";");
32   out.print("UPDATE ");
33   out.print(element.eContainer().name.toUpperCase());
34   out.print(" SET ");
35       for(attribute in element.newPrimaryKeyAttributes){
36      if(element.newPrimaryKeyAttributes.last().equals(attribute)){
37
38      out.print(attribute.name.toUpperCase()+"=");
39      out.print("NEXTVAL('SEQ_"+element.eContainer().name.toUpperCase()+"')");
40      }else{
41        out.print(attribute.name.toUpperCase()+" ");
42      }
```

```
43   }
44   out.println(";");
45   out.print("ALTER TABLE IF EXISTS ");
46   out.print(element.eContainer().name.toUpperCase());
47   out.print(" DROP CONSTRAINT IF EXISTS ");
48 out.print(element.eContainer().constraints->select(p | p.isKindOf(PrimaryKey)).last().name.
     toUpperCase());
49   out.println(" CASCADE;");
50   out.print("ALTER TABLE IF EXISTS ");
51   out.print(element.eContainer().name.toUpperCase());
52   out.print(" ADD PRIMARY KEY(");
53     for(attribute in element.newPrimaryKeyAttributes){
54       if(element.newPrimaryKeyAttributes.last().equals(attribute)){
55        out.print(attribute.name.toUpperCase());
56     }else{
57     out.print(attribute.name.toUpperCase()+",");
58     }
59   }
60   out.println(");");
61      for(relation in getChildrensforERelation(element.eContainer())){
62        out.print("ALTER TABLE IF EXISTS ");
63   out.print(relation.name.toUpperCase());
64   out.print(" ADD ");
65   for(attribute in element.newPrimaryKeyAttributes){
66   i++;
67      if(element.newPrimaryKeyAttributes.last().equals(attribute)){
68        out.print(relation.name.toUpperCase()+"_"+element.eContainer().name.toUpperCase()+"_
             "+element.newPrimaryKeyAttributes.last().name.toUpperCase()+" ");
69       out.print(attribute.getDataType(attribute));
70      }else{
71       out.print(attribute.name.toUpperCase()+" ");
72       out.print(attribute.getDataType(attribute)+",");
73      }
74   }
75   out.println(";");
76      }
77      for(relation in getChildrensforERelation(element.eContainer())){
78        out.print("ALTER TABLE IF EXISTS ");
79        out.print(relation.name.toUpperCase());
80        out.print(" ADD ");
81        out.print("FOREIGN KEY(");
82   for(attribute in element.newPrimaryKeyAttributes){
83   i++;
84      if(element.newPrimaryKeyAttributes.last().equals(attribute)){
85       out.print(relation.name.toUpperCase()+"_"+element.eContainer().name.toUpperCase()+"_"
           +element.newPrimaryKeyAttributes.last().name.toUpperCase()+" ");
86      }else{
```

```
87      out.print(attribute.name.toUpperCase()+" ");
88     }
89  }      out.print(")");
90      out.print(" REFERENCES ");
91      out.print(element.eContainer().name.toUpperCase()+"(");
92      for(attribute in element.newPrimaryKeyAttributes){
93     if(element.newPrimaryKeyAttributes.last().equals(attribute)){
94      out.print(attribute.name.toUpperCase());
95    }else{
96    out.print(attribute.name.toUpperCase()+",");
97    }
98  }
99  out.print(")");
100      out.println(" ON UPDATE CASCADE;");
101     }
102          for(relation in getChildrensforERelation(element.eContainer())){
103     i++;
104     out.print("UPDATE ");
105      out.print(relation.name.toUpperCase()+" "+relation.name.toUpperCase().charAt(0));
106  out.print(" SET ");
107        for(attribute in element.newPrimaryKeyAttributes){
108     if(element.newPrimaryKeyAttributes.last().equals(attribute)){
109      out.print(relation.name.toUpperCase()+"_"+element.eContainer().name.toUpperCase()+"_
            "+element.newPrimaryKeyAttributes.last().name.toUpperCase());
110     out.print("="+element.eContainer().name.toUpperCase().charAt(0)+".");
111      out.print(attribute.name.toUpperCase());
112    }else{
113    out.print(relation.name.toUpperCase()+"_"+element.eContainer().name.toUpperCase()+"_"+
        element.newPrimaryKeyAttributes.last().name.toUpperCase());
114    out.print("="+element.eContainer().name.toUpperCase().charAt(0)+".");
115    out.print(attribute.name.toUpperCase()+",");
116    } }
117  out.print(" FROM ");
118  out.print(element.eContainer().name.toUpperCase()+" "+element.eContainer().name.
       toUpperCase().charAt(0));
119  out.print(" WHERE ");
120       for(targetConstraint in relation.constraints){
121     if(targetConstraint.isKindOf(ForeignKey) ){
122      for(sourceConstraint in element.eContainer().constraints){
123       if(sourceConstraint.isKindOf(PrimaryKey)){
124         for(sourceAttribute in sourceConstraint.attributes){
125          if(sourceAttribute.equals(sourceConstraint.attributes.last())){
126     out.print(element.eContainer().name.toUpperCase().charAt(0)+"."+sourceAttribute.name.
          toUpperCase());
127     }else{
128     out.print(element.eContainer().name.toUpperCase().charAt(0)+"."+sourceAttribute.name.
          toUpperCase()+" = ");
```

```
129       for(targetAttribute in targetConstraint.attributes){
130      if(targetAttribute.equals(targetConstraint.attributes.last())){
131      out.print(relation.name.toUpperCase().charAt(0)+"."+targetAttribute.name.toUpperCase()
             +" = ");
132      }else{
133      out.print(relation.name.toUpperCase().charAt(0)+"."+targetAttribute.name.toUpperCase()
             +" AND ");
134      } }}}}}}
135      out.println(";");
136    }
137    out.print("ALTER TABLE IF EXISTS ");
138    out.print(element.eContainer().name.toUpperCase());
139   for(constraint in element.eContainer().constraints){
140   if(constraint.isKindOf(PrimaryKey)){
141    for(attribute in constraint.attributes){
142     if(constraint.attributes.last().equals(attribute)){
143         out.print(" DROP COLUMN IF EXISTS ");
144        out.print(attribute.name.toUpperCase());
145        out.print(" CASCADE ");
146     }else{
147      out.print(" DROP COLUMN IF EXISTS ");
148        out.print(attribute.name.toUpperCase());
149              out.print(" CASCADE "+",");
150     }}}}
151     out.println(";");
152   for(relation in getChildrensforERelation(element.eContainer())){
153   var i :Integer=0;
154     for(constraint in relation.constraints){
155     if(constraint.isKindOf(ForeignKey) and constraint.foreignRelation.equals(element.
           eContainer())){
156      for(attribute in constraint.attributes){
157       out.print("ALTER TABLE IF EXISTS ");
158       out.print(relation.name.toUpperCase());
159       out.print(" DROP COLUMN IF EXISTS ");
160       out.print(attribute.name.toUpperCase());
161       out.println(" CASCADE;");
162      }}} }
163   }else if(element.replaceStrategy = ReplaceKeyStrategy#NaturalKey){
164   out.print("ALTER TABLE IF EXISTS ");
165   out.print(element.eContainer().name.toUpperCase());
166   out.print(" DROP CONSTRAINT IF EXISTS ");
167   out.print(element.eContainer().constraints->select(p | p.isKindOf(PrimaryKey)).last().
        name.toUpperCase());
168    out.println(" CASCADE;");
169   out.print("ALTER TABLE "+element.eContainer().name.toUpperCase());
170   out.print(" ADD PRIMARY KEY(");
171       for(attribute in element.newPrimaryKeyAttributes){
```

```
172     if(element.newPrimaryKeyAttributes.last().equals(attribute)){
173      out.print(attribute.name.toUpperCase());
174    }else{
175    out.print(attribute.name.toUpperCase()+",");
176    }
177  }
178   out.println(");");
179              for(relation in getChildrensforERelation(element.eContainer())){
180           var relationToken:String;
181
182     out.print("UPDATE ");
183       for(attribute in element.key.attributes){
184
185     if(element.key.attributes.last().equals(attribute)){
186      relationToken=element.eContainer().name.toUpperCase().charAt(0).asString();
187      out.print(relation.name.toUpperCase()+" "+relation.name.toUpperCase().charAt(0));
188     }else{
189      out.print(attribute.name.toUpperCase()+" ");
190     }
191  }
192  out.print(" SET ");
193   for(constraint in relation.constraints){
194   if(constraint.isKindOf(ForeignKey)){
195     for(attribute in constraint.attributes){
196       if(constraint.attributes.last().equals(attribute)){
197         out.print(attribute.name.toUpperCase());
198       }else{
199       out.print(attribute.name.toUpperCase()+",");
200       }
201     }
202     }
203   }
204      if(element.eContainer().name.toUpperCase().charAt(0).asString().equals(relation.name.
           toUpperCase().charAt(0).asString())){
205   relationToken=relation.name.toUpperCase().charAt(0).asString()+"1";
206  out.print("="+relationToken+".");
207  }else{
208   out.print("="+relationToken+".");
209    }
210      for(attribute in element.newPrimaryKeyAttributes){
211      if(element.newPrimaryKeyAttributes.last().equals(attribute)){
212       out.print(attribute.name.toUpperCase());
213    }else{
214    out.print(attribute.name.toUpperCase()+",");
215    }
216  }
217  out.print(" FROM ");
```

```
218   out.print(element.eContainer().name.toUpperCase());
219   if(element.eContainer().name.toUpperCase().charAt(0).asString().equals(relation.name.
          toUpperCase().charAt(0).asString())){
220
221   out.print(" "+relationToken);
222   }else{
223    out.print(" "+element.eContainer().name.toUpperCase().charAt(0));
224     }
225   out.print(" WHERE ");
226  out.print(relation.name.toUpperCase().charAt(0)+".");
227    for(constraint in relation.constraints){
228   if(constraint.isKindOf(ForeignKey)){
229    for(attribute in constraint.attributes){
230     if(constraint.attributes.last().equals(attribute)){
231        out.print(attribute.name.toUpperCase());
232     }else{
233     out.print(attribute.name.toUpperCase()+",");
234     }
235    }
236    }
237
238   }
239     for(constraint in element.eContainer().constraints){
240   if(constraint.isKindOf(PrimaryKey)){
241    for(attribute in constraint.attributes){
242     if(constraint.attributes.last().equals(attribute)){
243     out.print("="+relationToken+".");
244       out.print(attribute.name.toUpperCase());
245     }else{
246     out.print("="+relationToken+"."+attribute.name.toUpperCase()+",");
247     }
248    }
249    }
250   }
251   out.println(";");
252   out.print("ALTER TABLE ");
253   out.print(relation.name.toUpperCase());
254   out.print(" ADD ");
255   out.print("FOREIGN KEY(");
256    for(constraint in relation.constraints){
257    if(constraint.isKindOf(ForeignKey)){
258      for(attribute in constraint.attributes){
259        if(constraint.attributes.last().equals(attribute)){
260          out.print(attribute.name.toUpperCase());
261        }else{
262        out.print(attribute.name.toUpperCase()+",");
263        }
```

```
264      }
265      }
266    }
267    out.print(")");
268    out.print(" REFERENCES ");
269    out.print(element.eContainer().name.toUpperCase());
270    out.print("(");
271       for(attribute in element.newPrimaryKeyAttributes){
272       if(element.newPrimaryKeyAttributes.last().equals(attribute)){
273        out.print(attribute.name.toUpperCase());
274     }else{
275     out.print(attribute.name.toUpperCase()+",");
276     }
277   }
278    out.println(");");
279    }}}
```

Listing B.3 – Regras de transformação M2T referente à refatoração *Replace Key*

```
1
2  for(element in MergeTable.allInstances()) {
3  var attrib:Bag = element.source.constraints->select(p | p.isKindOf(KeyConstraint)).
       attributes;
4  out.print("ALTER TABLE IF EXISTS ");
5  out.print(element.target.name.toUpperCase());
6
7
8  for(attribute in element.source.attributes){
9  if(not attrib->exists(a|a.first().name.toUpperCase().equals(attribute.name.toUpperCase()))
       {
10 if(element.source.attributes.last().equals(attribute)){
11 out.print(" ADD ");
12 out.print(attribute.name.toUpperCase()+" ");
13 out.print(attribute.getDataType(attribute));
14 }else{
15 out.print(" ADD ");
16 out.print(attribute.name.toUpperCase()+" ");
17 out.print(attribute.getDataType(attribute)+",");
18 }}}
19 out.println(";");
20
21 var mergeTablerelationToken:String;
22 for(attribute in element.source.attributes){
23 mergeTablerelationToken=element.target.name.toUpperCase().charAt(0).asString()+"1";
24 if(not attrib->exists(a|a.first().name.toUpperCase().equals(attribute.name.toUpperCase()))
       {
25 attribute.println();
26
```

```
27 out.print(" UPDATE ");
28 out.print(element.target.name.toUpperCase()+" "+element.target.name.toUpperCase().charAt(0)
       );
29 out.print(" SET ");
30 out.print(attribute.name.toUpperCase());
31 out.print("="+mergeTablerelationToken+".");
32 out.print(attribute.name.toUpperCase());
33 out.print(" FROM ");
34 out.print(element.source.name.toUpperCase()+" "+mergeTablerelationToken);
35 out.print(" WHERE ");
36 for(constraint in element.target.constraints){
37    if(constraint.isKindOf(PrimaryKey)){
38      for(attribute in constraint.attributes){
39        if(constraint.attributes.last().equals(attribute)){
40        out.print(element.target.name.toUpperCase().charAt(0)+".");
41        out.print(attribute.name.toUpperCase());
42        }else{
43        out.print(element.target.name.toUpperCase().charAt(0)+".");
44        out.print(attribute.name.toUpperCase()+" AND ");
45        } } } }
46 for(constraint in element.source.constraints){
47    if(constraint.isKindOf(ForeignKey)){
48      for(attribute in constraint.attributes){
49        if(constraint.attributes.last().equals(attribute)){
50        out.print("="+mergeTablerelationToken+".");
51        out.print(attribute.name.toUpperCase());
52        }else{
53        out.print("="+mergeTablerelationToken+".");
54        out.print(attribute.name.toUpperCase()+" AND ");
55        } } } }
56 out.println(";");
57 }
58 out.print("DROP TABLE IF EXISTS ");
59 out.print(element.source.name.toUpperCase());
60  out.println(" CASCADE;");
61  out.println();
62 }
```

Listing B.4 – Regras de transformação M2T referente ao construtor *Merge Table*

```
1 for(element in MoveColumn.allInstances()) {
2 if(element.transitionalPeriod = false){
3 out.println("ALTER TABLE "+element.target.name.toUpperCase()+" ADD "+element.columnToMove.
      name.toUpperCase()+" "+element.columnToMove.getDataType(element.columnToMove)+";");
4 out.println();
5 out.print("UPDATE "+element.target.name.toUpperCase()+" SET "+element.columnToMove.name.
      toUpperCase()+" = ");
```

```
6  out.print("("+"SELECT "+element.columnToMove.name.toUpperCase()+" FROM "+element.source.
       name.toUpperCase()+" WHERE ");
7      //Pattern for composite key problem
8      for(targetConstraint in element.target.constraints){
9       if(targetConstraint.isKindOf(ForeignKey) ){
10       for(sourceConstraint in element.source.constraints){
11        if(sourceConstraint.isKindOf(PrimaryKey)){
12          for(sourceAttribute in sourceConstraint.attributes){
13           if(sourceAttribute.equals(sourceConstraint.attributes.last())){
14       out.print(element.source.name.toUpperCase()+"."+sourceAttribute.name.toUpperCase());
15       }else{
16       out.print(element.source.name.toUpperCase()+"."+sourceAttribute.name.toUpperCase()+" =
              ");
17        for(targetAttribute in targetConstraint.attributes){
18       if(targetAttribute.equals(targetConstraint.attributes.last())){
19       out.println(element.target.name.toUpperCase()+"."+targetAttribute.name.toUpperCase()+"
              = ");
20       }else{
21       out.print(element.target.name.toUpperCase()+"."+targetAttribute.name.toUpperCase()+"
             AND ");
22       } } }}} } } }
23       out.println(");");
24  out.println();
25  out.println("ALTER TABLE "+element.source.name.toUpperCase()+" DROP COLUMN "+element.
       columnToMove.name.toUpperCase()+";");
26  }else{
27  out.println();
28  out.println();
29  out.println("ALTER TABLE "+element.target.name.toUpperCase()+" ADD "+element.columnToMove.
       name.toUpperCase()+" "+element.columnToMove.getDataType(element.columnToMove)+";");
30  out.println();
31  out.print("UPDATE "+element.target.name.toUpperCase()+" SET "+element.columnToMove.name.
       toUpperCase()+" = ");
32  out.print("("+"SELECT "+element.columnToMove.name.toUpperCase()+" FROM "+element.source.
       name.toUpperCase()+" WHERE ");
33  for(targetConstraint in element.target.constraints){
34  if(targetConstraint.isKindOf(ForeignKey)){
35  for(sourceConstraint in element.source.constraints){
36  if(sourceConstraint.isKindOf(PrimaryKey)){
37  for(sourceAttribute in sourceConstraint.attributes){
38  for(targetAttribute in targetConstraint.attributes){
39  if(sourceAttribute.equals(sourceConstraint.attributes.last())and targetAttribute.equals(
       targetConstraint.attributes.last())){
40  out.print(element.source.name.toUpperCase()+"."+sourceAttribute.name.toUpperCase()+" = ");
41  out.print(element.target.name.toUpperCase()+"."+targetAttribute.name.toUpperCase());
42  }else{
43       if(not sourceAttribute.equals(sourceConstraint.attributes.last()) and not
```

```
        targetAttribute.equals(targetConstraint.attributes.last())){
44          out.print(element.source.name.toUpperCase()+"."+sourceAttribute.name.toUpperCase()
                +" = ");
45          out.print(element.target.name.toUpperCase()+"."+targetAttribute.name.toUpperCase()
                +" AND ");
46      }}}}}}}}
47      out.println(");");
48      out.println();
49  out.print("CREATE OR REPLACE FUNCTION SYNC"+element.source.name.toUpperCase()+element.
        columnToMove.name.toUpperCase()+"()");
50  out.println(" RETURNS TRIGGER AS $SYNC"+element.source.name.toUpperCase()+element.
        columnToMove.name.toUpperCase()+"$");
51  out.println("BEGIN");
52  out.println("DECLARE COUNTER INTEGER;");
53  out.println("BEGIN");
54  out.println("IF(TG_OP = 'UPDATE') THEN");
55  out.println("IF(NEW."+element.columnToMove.name.toUpperCase()+" IS NOT NULL AND "+"NEW."+
        element.columnToMove.name.toUpperCase()+"!=OLD."+element.columnToMove.name.toUpperCase
        ()+") THEN");
56  out.print("SELECT COUNT(*) INTO COUNTER FROM ");
57  out.print(element.target.name.toUpperCase()+" "+element.target.name.toUpperCase().charAt(0)
        );
58  out.print(" WHERE ");
59  for(targetConstraint in element.target.constraints){
60  if(targetConstraint.isKindOf(ForeignKey)){
61  for(sourceConstraint in element.source.constraints){
62  if(sourceConstraint.isKindOf(PrimaryKey)){
63  for(sourceAttribute in sourceConstraint.attributes){
64  for(targetAttribute in targetConstraint.attributes){
65  if(sourceAttribute.equals(sourceConstraint.attributes.last())and targetAttribute.equals(
        targetConstraint.attributes.last())){
66  out.print(element.target.name.toUpperCase().charAt(0)+"."+targetAttribute.name.toUpperCase
        ()+" = ");
67  out.print("NEW."+sourceAttribute.name.toUpperCase());
68
69  }else{
70      if(not sourceAttribute.equals(sourceConstraint.attributes.last()) and not
            targetAttribute.equals(targetConstraint.attributes.last())){
71          out.print(element.target.name.toUpperCase().charAt(0)+"."+targetAttribute.name.
                toUpperCase()+" = ");
72          out.print("NEW."+sourceAttribute.name.toUpperCase()+" AND ");
73      }
74  }}}}}}}
75  out.println(";");
76  out.println("IF(COUNTER >0) THEN");
77  out.print("UPDATE "+element.target.name.toUpperCase()+" "+element.target.name.toUpperCase()
        .charAt(0)+" SET "+element.columnToMove.name.toUpperCase()+" = NEW."+element.
```

```
    columnToMove.name.toUpperCase());
out.print(" WHERE ");
for(targetConstraint in element.target.constraints){
if(targetConstraint.isKindOf(ForeignKey)){
for(sourceConstraint in element.source.constraints){
if(sourceConstraint.isKindOf(PrimaryKey)){
for(sourceAttribute in sourceConstraint.attributes){
for(targetAttribute in targetConstraint.attributes){
if(sourceAttribute.equals(sourceConstraint.attributes.last())and targetAttribute.equals(
    targetConstraint.attributes.last())){
out.print(element.target.name.toUpperCase().charAt(0)+"."+targetAttribute.name.toUpperCase
    ()+" = ");
out.print("NEW."+sourceAttribute.name.toUpperCase());
}else{
     if(not sourceAttribute.equals(sourceConstraint.attributes.last()) and not
         targetAttribute.equals(targetConstraint.attributes.last())){
          out.print(element.target.name.toUpperCase().charAt(0)+"."+targetAttribute.name.
              toUpperCase()+" = ");
          out.print("NEW."+sourceAttribute.name.toUpperCase()+" AND ");
     }
     }}}}}}}
out.println(";");
out.println("END IF;");
out.println("END IF;");
out.println("RETURN NEW;");
out.println("END IF;");
out.println("IF(TG_OP = 'INSERT') THEN");
out.println("IF(NEW."+element.columnToMove.name.toUpperCase()+" IS NOT NULL) THEN");
out.print("SELECT COUNT(*) INTO COUNTER FROM ");
out.print(element.target.name.toUpperCase()+" "+element.target.name.toUpperCase().charAt(0)
    );
out.print(" WHERE ");
 for(targetConstraint in element.target.constraints){
     if(targetConstraint.isKindOf(ForeignKey) ){
      for(sourceConstraint in element.source.constraints){
       if(sourceConstraint.isKindOf(PrimaryKey)){
         for(sourceAttribute in sourceConstraint.attributes){
          if(sourceAttribute.equals(sourceConstraint.attributes.last())){
     out.print("NEW."+sourceAttribute.name.toUpperCase());
     }else{
     out.print("NEW."+sourceAttribute.name.toUpperCase()+" = ");
       for(targetAttribute in targetConstraint.attributes){
     if(targetAttribute.equals(targetConstraint.attributes.last())){
     out.print(element.target.name.toUpperCase().charAt(0)+"."+targetAttribute.name.
         toUpperCase()+" = ");
     }else{
     out.print(element.target.name.toUpperCase().charAt(0)+"."+targetAttribute.name.
```

```
        toUpperCase()+" AND ");
118     } } } } }}} }
119 out.println(";");
120 out.println("IF(COUNTER >0) THEN");
121 out.print("UPDATE "+element.target.name.toUpperCase()+" "+element.target.name.toUpperCase()
    .charAt(0)+" SET "+element.columnToMove.name.toUpperCase()+" = NEW."+element.
    columnToMove.name.toUpperCase());
122 out.print(" WHERE ");
123  for(targetConstraint in element.target.constraints){
124     if(targetConstraint.isKindOf(ForeignKey) ){
125      for(sourceConstraint in element.source.constraints){
126       if(sourceConstraint.isKindOf(PrimaryKey)){
127         for(sourceAttribute in sourceConstraint.attributes){
128          if(sourceAttribute.equals(sourceConstraint.attributes.last())){
129     out.print("NEW."+sourceAttribute.name.toUpperCase());
130     }else{
131     out.print("NEW."+sourceAttribute.name.toUpperCase()+" = ");
132       for(targetAttribute in targetConstraint.attributes){
133     if(targetAttribute.equals(targetConstraint.attributes.last())){
134     out.print(element.target.name.toUpperCase().charAt(0)+"."+targetAttribute.name.
        toUpperCase()+" = ");
135     }else{
136     out.print(element.target.name.toUpperCase().charAt(0)+"."+targetAttribute.name.
        toUpperCase()+" AND ");
137     } } } } } } } }
138 out.println(";");
139 out.println("END IF;");
140 out.println("END IF;");
141 out.println("RETURN NEW;");
142 out.println("END IF;");
143 out.println("END;");
144 out.println("END;");
145 out.print("$SYNC"+element.source.name.toUpperCase()+element.columnToMove.name.toUpperCase()
    +"$");
146 out.println("LANGUAGE plpgsql;");
147 out.println();
148 out.print("DROP TRIGGER IF EXISTS SYNC"+element.source.name.toUpperCase()+element.
    columnToMove.name.toUpperCase());
149 out.println(" ON "+element.source.name.toUpperCase()+";");
150 out.println();
151 out.println("CREATE TRIGGER SYNC"+element.source.name.toUpperCase()+element.columnToMove.
    name.toUpperCase());
152 out.println("AFTER INSERT OR UPDATE OF "+element.columnToMove.name.toUpperCase()+" ON "+
    element.source.name.toUpperCase());
153 out.println("FOR EACH ROW");
154 out.println("EXECUTE PROCEDURE SYNC"+element.source.name.toUpperCase()+element.columnToMove
    .name.toUpperCase()+"();");
```

```
155 out.println();
156 out.print("CREATE OR REPLACE FUNCTION SYNC"+element.target.name.toUpperCase()+element.
        columnToMove.name.toUpperCase()+"()");
157 out.println(" RETURNS TRIGGER AS $SYNC"+element.target.name.toUpperCase()+element.
        columnToMove.name.toUpperCase()+"$");
158 out.println("BEGIN");
159 out.println("DECLARE COUNTER INTEGER;");
160 out.println("BEGIN");
161 out.println("IF(TG_OP = 'UPDATE') THEN");
162 out.print("SELECT COUNT(*) INTO COUNTER FROM ");
163 out.print(element.source.name.toUpperCase()+" "+element.source.name.toUpperCase().charAt(0)
        );
164 out.print(" WHERE ");
165 for(targetConstraint in element.target.constraints){
166 if(targetConstraint.isKindOf(ForeignKey)){
167 for(sourceConstraint in element.source.constraints){
168 if(sourceConstraint.isKindOf(PrimaryKey)){
169 for(sourceAttribute in sourceConstraint.attributes){
170 for(targetAttribute in targetConstraint.attributes){
171 if(sourceAttribute.equals(sourceConstraint.attributes.last())and targetAttribute.equals(
        targetConstraint.attributes.last())){
172 out.print(element.source.name.toUpperCase().charAt(0)+"."+sourceAttribute.name.toUpperCase
        ()+" = ");
173 out.print("NEW."+targetAttribute.name.toUpperCase());
174 }else{
175      if(not sourceAttribute.equals(sourceConstraint.attributes.last()) and not
            targetAttribute.equals(targetConstraint.attributes.last())){
176          out.print(element.source.name.toUpperCase().charAt(0)+"."+sourceAttribute.name.
                toUpperCase()+" = ");
177          out.print("NEW."+targetAttribute.name.toUpperCase()+" AND ");
178      }}}}}}}}
179 out.println(";");
180 out.println("IF(COUNTER >0) THEN");
181 out.print("UPDATE "+element.source.name.toUpperCase()+" "+element.source.name.toUpperCase()
        .charAt(0)+" SET "+element.columnToMove.name.toUpperCase()+" = NEW."+element.
        columnToMove.name.toUpperCase());
182 out.print(" WHERE ");
183 for(targetConstraint in element.target.constraints){
184 if(targetConstraint.isKindOf(ForeignKey)){
185 for(sourceConstraint in element.source.constraints){
186 if(sourceConstraint.isKindOf(PrimaryKey)){
187 for(sourceAttribute in sourceConstraint.attributes){
188 for(targetAttribute in targetConstraint.attributes){
189 if(sourceAttribute.equals(sourceConstraint.attributes.last())and targetAttribute.equals(
        targetConstraint.attributes.last())){
190 out.print(sourceAttribute.eContainer().name.toUpperCase().charAt(0)+"."+sourceAttribute.
        name.toUpperCase()+" = ");
```

```
191 out.print("NEW."+targetAttribute.name.toUpperCase());
192 }else{
193     if(not sourceAttribute.equals(sourceConstraint.attributes.last()) and not
            targetAttribute.equals(targetConstraint.attributes.last())){
194     out.print(sourceAttribute.eContainer().name.toUpperCase().charAt(0)+"."+
            sourceAttribute.name.toUpperCase()+" = ");
195     out.print("NEW."+targetAttribute.name.toUpperCase()+" AND ");
196     }
197 }}}}}}}
198 out.println(";");
199 out.println("COUNTER:=0;");
200 out.println("END IF;");
201 out.println("RETURN NEW;");
202 out.println("END IF;");
203 out.println("IF(TG_OP = 'INSERT') THEN");
204 out.println("IF(NEW."+element.columnToMove.name.toUpperCase()+" IS NOT NULL) THEN");
205 out.print("SELECT COUNT(*) INTO COUNTER FROM ");
206 out.print(element.source.name.toUpperCase()+" "+element.source.name.toUpperCase().charAt(0)
    );
207 out.print(" WHERE ");
208 for(targetConstraint in element.target.constraints){
209 if(targetConstraint.isKindOf(ForeignKey)){
210 for(sourceConstraint in element.source.constraints){
211 if(sourceConstraint.isKindOf(PrimaryKey)){
212 for(sourceAttribute in sourceConstraint.attributes){
213 for(targetAttribute in targetConstraint.attributes){
214 if(sourceAttribute.equals(sourceConstraint.attributes.last())and targetAttribute.equals(
    targetConstraint.attributes.last())){
215 out.print(sourceAttribute.eContainer().name.toUpperCase().charAt(0)+"."+sourceAttribute.
    name.toUpperCase()+" = ");
216 out.print("NEW."+targetAttribute.name.toUpperCase());
217 }else{
218     if(not sourceAttribute.equals(sourceConstraint.attributes.last()) and not
            targetAttribute.equals(targetConstraint.attributes.last())){
219     out.print(sourceAttribute.eContainer().name.toUpperCase().charAt(0)+"."+
            sourceAttribute.name.toUpperCase()+" = ");
220     out.print("NEW."+targetAttribute.name.toUpperCase()+" AND ");
221     }
222 }}}}}}}
223 out.println(";");
224 out.println("IF(COUNTER >0) THEN");
225 out.print("UPDATE "+element.source.name.toUpperCase()+" "+element.source.name.toUpperCase()
    .charAt(0)+" SET "+element.columnToMove.name.toUpperCase()+" = NEW."+element.
    columnToMove.name.toUpperCase());
226 out.print(" WHERE ");
227 for(targetConstraint in element.target.constraints){
228 if(targetConstraint.isKindOf(ForeignKey)){
```

```
for(sourceConstraint in element.source.constraints){
if(sourceConstraint.isKindOf(PrimaryKey)){
for(sourceAttribute in sourceConstraint.attributes){
for(targetAttribute in targetConstraint.attributes){
if(sourceAttribute.equals(sourceConstraint.attributes.last())and targetAttribute.equals(
    targetConstraint.attributes.last())){
out.print(sourceAttribute.eContainer().name.toUpperCase().charAt(0)+"."+sourceAttribute.
    name.toUpperCase()+" = ");
out.print("NEW."+targetAttribute.name.toUpperCase());
}else{
     if(not sourceAttribute.equals(sourceConstraint.attributes.last()) and not
         targetAttribute.equals(targetConstraint.attributes.last())){
     out.print(sourceAttribute.eContainer().name.toUpperCase().charAt(0)+"."+
         sourceAttribute.name.toUpperCase()+" = ");
     out.print("NEW."+targetAttribute.name.toUpperCase()+" AND ");
     }
}}}}}}}
out.println(";");
out.println("COUNTER:=0;");
out.println("END IF;");
out.println("END IF;");
out.println("RETURN NEW;");
out.println("END IF;");
out.println("END;");
out.println("END;");
out.print("$SYNC"+element.target.name.toUpperCase()+element.columnToMove.name.toUpperCase()
    +"$");
out.println("LANGUAGE plpgsql;");
out.println();
out.print("DROP TRIGGER IF EXISTS SYNC"+element.target.name.toUpperCase()+element.
    columnToMove.name.toUpperCase());
out.println(" ON "+element.target.name.toUpperCase()+";");
out.println();
out.println("CREATE TRIGGER SYNC"+element.target.name.toUpperCase()+element.columnToMove.
    name.toUpperCase());
out.println("AFTER INSERT OR UPDATE OF "+element.columnToMove.name.toUpperCase()+" ON "+
    element.target.name.toUpperCase());
out.println("FOR EACH ROW");
out.println("EXECUTE PROCEDURE SYNC"+element.target.name.toUpperCase()+element.columnToMove
    .name.toUpperCase()+"();");
out.println();
}
}
```

Listing B.5 – Regras de transformação M2T referente à refatoração *Move Column*

```
for(element in RenameColumn.allInstances()) {
if(element.transitionalPeriod = false){
```

```
3  out.print("ALTER TABLE ");
4  out.print(element.eContainer().name.toUpperCase());
5  out.print(" ADD ");
6  out.print(element.renameTo.toUpperCase()+" ");
7  out.print(element.columnToRename.getDataType(element.columnToRename));
8  out.println(";");
9  out.println();
10 out.print("UPDATE ");
11 out.print(element.eContainer().name.toUpperCase());
12 out.print(" SET ");
13 out.print(element.renameTo.toUpperCase()+"=");
14 out.print(element.columnToRename.name.toUpperCase());
15 out.print(" WHERE ");
16 for(constraint in element.eContainer().constraints->select(c|c.isKindOf(PrimaryKey))){
17 for(attribute in constraint.attributes){
18 if(attribute.equals(constraint.attributes.last())){
19 out.print(attribute.name.toUpperCase()+"=");
20 out.print(attribute.name.toUpperCase());
21
22 }else{
23
24 out.print(attribute.name.toUpperCase()+"=");
25 out.print(attribute.name.toUpperCase()+" AND ");
26
27 }}}
28 out.println(";");
29 out.println();
30 out.print("ALTER TABLE ");
31 out.print(element.eContainer().name.toUpperCase());
32 out.print(" DROP COLUMN ");
33 out.print(element.columnToRename.name.toUpperCase());
34 out.println(" CASCADE;");
35 }else{
36 out.print("ALTER TABLE ");
37 out.print(element.eContainer().name.toUpperCase());
38 out.print(" ADD ");
39 out.print(element.renameTo.toUpperCase()+" ");
40 out.print(element.columnToRename.getDataType(element.columnToRename));
41 out.println(";");
42 out.println();
43 out.print("UPDATE ");
44 out.print(element.eContainer().name.toUpperCase());
45 out.print(" SET ");
46 out.print(element.renameTo.toUpperCase()+"=");
47 out.print(element.columnToRename.name.toUpperCase());
48 out.print(" WHERE ");
49 for(constraint in element.eContainer().constraints->select(c|c.isKindOf(PrimaryKey))){
```

```
50 for(attribute in constraint.attributes){
51 if(attribute.equals(constraint.attributes.last())){
52 out.print(attribute.name.toUpperCase()+"=");
53 out.print(attribute.name.toUpperCase());
54
55 }else{
56
57 out.print(attribute.name.toUpperCase()+"=");
58 out.print(attribute.name.toUpperCase()+" AND ");
59 }}}
60 out.println(";");
61 out.println();
62 out.println("CREATE OR REPLACE FUNCTION SYNC_"+element.renameTo.toUpperCase()+"()"+"
       RETURNS TRIGGER AS $SYNC_"+element.renameTo.toUpperCase()+"$");
63 out.println("BEGIN");
64 out.println("BEGIN");
65 out.println("IF(TG_OP='INSERT') THEN");
66 out.println("IF(NEW."+element.columnToRename.name.toUpperCase()+" IS NULL) THEN");
67
68 out.print("NEW."+element.columnToRename.name.toUpperCase());
69 out.println("="+"NEW."+element.renameTo.toUpperCase()+";");
70 out.println("END IF;");
71 out.println("IF(NEW."+element.renameTo.toUpperCase()+" IS NULL) THEN");
72 out.print("NEW."+element.renameTo.toUpperCase());
73 out.println("="+"NEW."+element.columnToRename.name.toUpperCase()+";");
74 out.println("END IF;");
75 out.println("END IF;");
76 out.println("IF(TG_OP='UPDATE') THEN");
77 out.println("IF(NEW."+element.renameTo.toUpperCase()+"!=OLD."+element.renameTo.toUpperCase
       ()+") THEN");
78
79 out.print("NEW."+element.columnToRename.name.toUpperCase());
80 out.println("="+"NEW."+element.renameTo.toUpperCase()+";");
81 out.println("END IF;");
82 out.println("IF(NEW."+element.columnToRename.name.toUpperCase()+"!=OLD."+element.
       columnToRename.name.toUpperCase()+") THEN");
83 out.print("NEW."+element.renameTo.toUpperCase());
84 out.println("="+"NEW."+element.columnToRename.name.toUpperCase()+";");
85 out.println("END IF;");
86 out.println("END IF;");
87 out.println("RETURN NEW;");
88 out.println("END;");
89 out.println("END;");
90 out.print("$SYNC_"+element.renameTo.toUpperCase()+"$");
91 out.println(" language plpgsql;");
92 out.print("DROP TRIGGER IF EXISTS SYNC_"+element.renameTo.toUpperCase());
93 out.println(" ON "+element.eContainer().name.toUpperCase()+";");
```

```
94  out.println();
95  out.println("CREATE TRIGGER SYNC_"+element.renameTo.toUpperCase());
96  out.println("BEFORE INSERT OR UPDATE OR DELETE ON "+element.eContainer().name.toUpperCase()
        );
97  out.println("FOR EACH ROW");
98  out.println("EXECUTE PROCEDURE SYNC_"+element.renameTo.toUpperCase()+"();");
99  out.println("--AFTER TRANSITIONAL PERIOD RUN THE FOLLOWING SCRIPT:");
100 out.print("--ALTER TABLE ");
101 out.print(element.eContainer().name.toUpperCase());
102 out.print(" DROP COLUMN ");
103 out.print(element.columnToRename.name.toUpperCase());
104 out.println(";");
105 }
106 }
```

Listing B.6 – Regras de transformação M2T referente à refatoração *Rename Column*

```
1   for(element in RenameTable.allInstances()) {
2   if(element.transitionalPeriod = false){
3   out.print("CREATE TABLE ");
4   out.print(element.renameTo.toUpperCase());
5   out.print("(");
6      for(attribute in element.eContainer().attributes){
7        if(attribute.isKindOf(Attribute)){
8        if(element.eContainer().attributes.last().equals(attribute)){
9         out.print(attribute.name.toUpperCase());
10        out.print(" "+attribute.getDataType(attribute));
11       }else{
12          out.print(attribute.name.toUpperCase());
13          out.print(" "+attribute.getDataType(attribute)+",");
14        }}}
15       out.println(");");
16  var sPrimaryKey = getPrimaryKey(element.eContainer());
17  if(sPrimaryKey.isDefined() and sPrimaryKey.attributes.size()>0){
18  out.print("ALTER TABLE ");
19  out.print(element.renameTo.toUpperCase());
20  out.print(" ADD PRIMARY KEY(");
21  for(attribute in sPrimaryKey.attributes){
22   if(sPrimaryKey.attributes.last().equals(attribute)){
23  out.print(attribute.name.toUpperCase());
24  }else{
25  out.print(attribute.name.toUpperCase()+",");
26  }}
27  out.println(");");
28  }
29  for(constraint in element.eContainer().constraints){
30   if(constraint.isKindOf(EForeignKey)){
31   out.print("ALTER TABLE ");
```

```
 out.print(element.renameTo.toUpperCase());
 out.print(" ADD FOREIGN KEY(");
 for(attribute in constraint.attributes){
 if(constraint.attributes.last().equals(attribute)){
out.print(attribute.name.toUpperCase());
 }else{
out.print(attribute.name.toUpperCase()+",");
  }
}
out.print(")");
out.print("REFERENCES ");
out.print(constraint.foreignRelation.name.toUpperCase());
out.print("(");
var primaryKey = getPrimaryKey(constraint.foreignRelation);
for(attribute in primaryKey.attributes){
 if(primaryKey.attributes.last().equals(attribute)){
out.print(attribute.name.toUpperCase());
}else{
out.print(attribute.name.toUpperCase()+",");
}
}
out.println(");");
}
}
out.print("INSERT INTO "+element.renameTo.toUpperCase());
out.println(" SELECT * FROM "+element.eContainer().name.toUpperCase()+";");
}else{
out.print("ALTER TABLE IF EXISTS ");
out.print(element.eContainer().name.toUpperCase());
out.print(" RENAME TO ");
out.print(element.renameTo.toUpperCase());
out.println(";");
out.print("CREATE VIEW ");
out.print(element.eContainer().name.toUpperCase());
out.print(" AS ");
out.print("SELECT * FROM ");
out.println(element.renameTo.toUpperCase()+";");
}
out.println("DROP TABLE IF EXISTS "+element.eContainer().name.toUpperCase()+";");

}
```

Listing B.7 – Regras de transformação M2T referente à refatoração *Rename Table*

```


for(element in RenameView.allInstances()) {
out.print("DROP VIEW IF EXISTS ");
```

```
out.println(element.eContainer().name.toUpperCase()+";");
out.print("CREATE VIEW "+element.renameTo.toUpperCase()+" AS ");
out.println(element.eContainer().viewDefinition.toUpperCase());
}
```

Listing B.8 – Regras de transformação M2T referente à refatoração *Rename View*

```
for(element in ReplaceColumn.allInstances()) {
if(element.transitionalPeriod == false){
out.print("ALTER TABLE ");
out.print(element.eContainer().name.toUpperCase());
out.print(" ADD ");
out.print(element.replaceTo.name.toUpperCase()+" ");
out.print(element.replaceTo.dataType+"("+element.replaceTo.size+")");
out.println(";");
out.println();
out.print("UPDATE ");
out.print(element.eContainer().name.toUpperCase());
out.print(" SET ");
out.print(element.replaceTo.name.toUpperCase()+"=");
out.print(element.columnToReplace.name.toUpperCase());
out.print(" WHERE ");
for(constraint in element.eContainer().constraints->select(c|c.isKindOf(PrimaryKey))){
for(attribute in constraint.attributes){
if(attribute.equals(constraint.attributes.last())){
out.print(attribute.name.toUpperCase()+"=");
out.print(attribute.name.toUpperCase());
}else{
out.print(attribute.name.toUpperCase()+"=");
out.print(attribute.name.toUpperCase()+" AND ");
}}}
out.println(";");
out.println();
out.print("ALTER TABLE ");
out.print(element.eContainer().name.toUpperCase());
out.print(" DROP COLUMN ");
out.print(element.columnToReplace.name.toUpperCase());
out.println(" CASCADE;");
}else{
out.print("ALTER TABLE ");
out.print(element.eContainer().name.toUpperCase());
out.print(" ADD ");

out.print(element.replaceTo.name.toUpperCase()+" ");
out.print(element.replaceTo.dataType+"("+element.replaceTo.size+")");
out.println(";");
out.println();
out.print("UPDATE ");
```

```
out.print(element.eContainer().name.toUpperCase());
out.print(" SET ");
out.print(element.replaceTo.name.toUpperCase()+"=");
out.print(element.columnToReplace.name.toUpperCase());
out.print(" WHERE ");
for(constraint in element.eContainer().constraints->select(c|c.isKindOf(PrimaryKey))){
for(attribute in constraint.attributes){
if(attribute.equals(constraint.attributes.last())){
out.print(attribute.name.toUpperCase()+"=");
out.print(attribute.name.toUpperCase());

}else{

out.print(attribute.name.toUpperCase()+"=");
out.print(attribute.name.toUpperCase()+" AND ");

}}}
out.println(";");
out.println();
out.println("CREATE OR REPLACE FUNCTION SYNC_"+element.replaceTo.name.toUpperCase()+"()"+"
    RETURNS TRIGGER AS $SYNC_"+element.replaceTo.name.toUpperCase()+"$");
out.println("BEGIN");
out.println("BEGIN");
out.println("IF(TG_OP='INSERT') THEN");
out.println("IF(NEW."+element.columnToReplace.name.toUpperCase()+" IS NULL) THEN");

out.print("NEW."+element.columnToReplace.name.toUpperCase());
out.println("="+"NEW."+element.replaceTo.name.toUpperCase()+";");
out.println("END IF;");
out.println("IF(NEW."+element.replaceTo.name.toUpperCase()+" IS NULL) THEN");
out.print("NEW."+element.replaceTo.name.toUpperCase());
out.println("="+"NEW."+element.columnToReplace.name.toUpperCase()+";");
out.println("END IF;");
out.println("END IF;");
out.println("IF(TG_OP='UPDATE') THEN");
out.println("IF(NEW."+element.replaceTo.name.toUpperCase()+"!=OLD."+element.replaceTo.name.
    toUpperCase()+") THEN");

out.print("NEW."+element.columnToReplace.name.toUpperCase());
out.println("="+"NEW."+element.replaceTo.name.toUpperCase()+";");
out.println("END IF;");
out.println("IF(NEW."+element.columnToReplace.name.toUpperCase()+"!=OLD."+element.
    columnToReplace.name.toUpperCase()+") THEN");
out.print("NEW."+element.replaceTo.name.toUpperCase());
out.println("="+"NEW."+element.columnToReplace.name.toUpperCase()+";");
out.println("END IF;");
out.println("END IF;");
```

```
86 out.println("RETURN NEW;");
87 out.println("END;");
88 out.println("END;");
89 out.print("$SYNC_"+element.replaceTo.name.toUpperCase()+"$");
90 out.println(" language plpgsql;");
91 out.print("DROP TRIGGER IF EXISTS SYNC_"+element.replaceTo.name.toUpperCase());
92 out.println(" ON "+element.eContainer().name.toUpperCase()+";");
93 out.println();
94 out.println("CREATE TRIGGER SYNC_"+element.replaceTo.name.toUpperCase());
95 out.println("BEFORE INSERT OR UPDATE OR DELETE ON "+element.eContainer().name.toUpperCase()
       );
96 out.println("FOR EACH ROW");
97 out.println("EXECUTE PROCEDURE SYNC_"+element.replaceTo.name.toUpperCase()+"();");
98 out.println("--AFTER TRANSITIONAL PERIOD RUN THE FOLLOWING SCRIPT:");
99 out.print("--ALTER TABLE ");
100 out.print(element.eContainer().name.toUpperCase());
101 out.print(" DROP COLUMN ");
102 out.print(element.columnToReplace.name.toUpperCase());
103 out.println(";");
104 }
105 }
```

Listing B.9 – Regras de transformação M2T referente à refatoração *Replace Column*

```
1  for(element in ExtractAssociativeTable.allInstances()){
2  var attrib:Bag = element.eContainer().constraints->select(p | p.isKindOf(KeyConstraint)).
       attributes;
3  if(element.transitionalPeriod = false){
4  out.print("CREATE TABLE IF NOT EXISTS ");
5  out.print(element.tableName.toUpperCase());
6  out.print("(");
7  for(attribute in attrib){
8  if(attrib.last().equals(attribute)){
9  out.print(attribute.first().name.toUpperCase()+" ");
10 out.print(attribute.first().getDataType(attribute.first()));
11 }else{
12 out.print(attribute.first().name.toUpperCase()+" ");
13 out.print(attribute.first().getDataType(attribute.first())+",");
14 }}
15 out.println(");");
16
17 out.print("ALTER TABLE ");
18 out.print(element.tableName.toUpperCase());
19 out.print(" ALTER COLUMN ");
20 for(constraint in element.eContainer().constraints){
21   if(constraint.isKindOf(PrimaryKey)){
22    for(attribute in constraint.attributes){
23     if(constraint.attributes.last().equals(attribute)){
```

```
24      out.print(attribute.name.toUpperCase());
25     out.print(" SET NOT NULL");
26     }else{
27     out.print(attribute.name.toUpperCase());
28     out.print(" SET NOT NULL,");
29     }}}}
30 out.println(";");
31 
32 out.print("ALTER TABLE ");
33 out.print(element.tableName.toUpperCase());
34 out.print(" ADD CONSTRAINT ");
35 for(constraint in element.eContainer().constraints){
36   if(constraint.isKindOf(PrimaryKey)){
37    for(attribute in constraint.attributes){
38     if(constraint.attributes.last().equals(attribute)){
39      out.print(attribute.name.toUpperCase());
40      out.print("_KEY");
41     }else{
42     out.print(attribute.name.toUpperCase());
43         out.print("_");
44 
45     }}}}
46 out.print(" UNIQUE(");
47 for(constraint in element.eContainer().constraints){
48   if(constraint.isKindOf(PrimaryKey)){
49    for(attribute in constraint.attributes){
50     if(constraint.attributes.last().equals(attribute)){
51      out.print(attribute.name.toUpperCase());
52     }else{
53     out.print(attribute.name.toUpperCase()+",");
54     }}}}
55 out.println(");");
56 
57 out.print("ALTER TABLE ");
58 out.print(element.tableName.toUpperCase());
59 out.print(" ALTER COLUMN ");
60 for(attribute in element.foreignKey.attributes){
61     if(element.foreignKey.attributes.last().equals(attribute)){
62      out.print(attribute.name.toUpperCase());
63     out.print(" SET NOT NULL");
64    }else{
65     out.print(attribute.name.toUpperCase());
66     out.print(" SET NOT NULL,");
67     }}
68 out.println(";");
69 out.print("INSERT INTO ");
70 out.print(element.tableName.toUpperCase()+"(");
```

```
71 for(attribute in attrib){
72 if(attrib.last().equals(attribute)){
73 out.print(attribute.first().name.toUpperCase());
74 }else{
75 out.print(attribute.first().name.toUpperCase()+",");
76 }}
77 out.print(")");
78 out.print(" SELECT ");
79 for(attribute in attrib){
80 if(attrib.last().equals(attribute)){
81 out.print(attribute.first().name.toUpperCase());
82 }else{
83 out.print(attribute.first().name.toUpperCase()+",");
84 }}
85 out.print(" FROM ");
86 out.print(element.eContainer().name.toUpperCase());
87 out.println(";");
88 out.print("ALTER TABLE ");
89 out.print(element.tableName.toUpperCase());
90 out.print(" ADD FOREIGN KEY(");
91 for(constraint in element.eContainer().constraints){
92   if(constraint.isKindOf(PrimaryKey)){
93    for(attribute in constraint.attributes){
94     if(constraint.attributes.last().equals(attribute)){
95      out.print(attribute.name.toUpperCase());
96     }else{
97     out.print(attribute.name.toUpperCase()+",");
98     }}}}
99 out.print(")");
100 out.print(" REFERENCES ");
101 out.print(element.eContainer().name.toUpperCase()+"(");
102 for(constraint in element.eContainer().constraints){
103   if(constraint.isKindOf(PrimaryKey)){
104    for(attribute in constraint.attributes){
105     if(constraint.attributes.last().equals(attribute)){
106      out.print(attribute.name.toUpperCase());
107     }else{
108     out.print(attribute.name.toUpperCase()+",");
109     }}}}
110 out.println(");");
111 out.print("ALTER TABLE ");
112 out.print(element.tableName.toUpperCase());
113 out.print(" ADD FOREIGN KEY(");
114 for(attribute in element.foreignKey.attributes){
115     if(element.foreignKey.attributes.last().equals(attribute)){
116      out.print(attribute.name.toUpperCase());
117    }else{
```

```
118     out.print(attribute.name.toUpperCase()+",");
119     }}
120 out.print(")");
121 out.print(" REFERENCES ");
122 out.print(element.foreignKey.foreignRelation.name.toUpperCase()+"(");
123 var targetAttributes:OrderedSet=getPrimaryKeyAttributes(element.foreignKey.foreignRelation)
        ;
124 for(attribute in targetAttributes){
125     if(targetAttributes.last().equals(attribute)){
126      out.print(attribute.name.toUpperCase());
127    }else{
128     out.print(attribute.name.toUpperCase()+",");
129     }}
130 out.println(");");
131 
132 out.println("--AFTER TRANSITIONAL PERIOD RUN THIS SCRIPT");
133 out.print("--ALTER TABLE ");
134 out.print(element.eContainer().name.toUpperCase());
135 out.print(" DROP ");
136 for(attribute in element.foreignKey.attributes){
137     if(element.foreignKey.attributes.last().equals(attribute)){
138     out.print(attribute.name.toUpperCase());
139     }else{
140     out.print(attribute.name.toUpperCase()+",");
141     }
142 }
143 out.println(";");
144 
145 out.print("--ALTER TABLE ");
146 out.print(element.tableName.toUpperCase());
147 out.print(" ADD FOREIGN KEY(");
148 for(attribute in element.foreignKey.attributes){
149     if(element.foreignKey.attributes.last().equals(attribute)){
150     out.print(attribute.name.toUpperCase());
151     }else{
152     out.print(attribute.name.toUpperCase()+",");
153     }
154 }
155 out.print(")");
156 out.print(" REFERENCES ");
157 out.print(element.foreignKey.foreignRelation.name.toUpperCase()+"(");
158 for(constraint in element.foreignKey.foreignRelation.constraints){
159   if(constraint.isKindOf(PrimaryKey)){
160    for(attribute in constraint.attributes){
161     if(constraint.attributes.last().equals(attribute)){
162     out.print(attribute.name.toUpperCase());
163     }else{
```

```
164     out.print(attribute.name.toUpperCase()+",");
165     }}}}
166 out.println(") ON UPDATE CASCADE ON DELETE CASCADE;");
167 out.print("--ALTER TABLE ");
168 out.print(element.tableName.toUpperCase());
169 out.print(" ADD FOREIGN KEY(");
170 for(constraint in element.eContainer().constraints){
171    if(constraint.isKindOf(PrimaryKey)){
172     for(attribute in constraint.attributes){
173      if(constraint.attributes.last().equals(attribute)){
174        out.print(attribute.name.toUpperCase());
175      }else{
176       out.print(attribute.name.toUpperCase()+",");
177       }}}}
178 out.print(")");
179 out.print(" REFERENCES ");
180 out.print(element.eContainer().name.toUpperCase()+"(");
181 for(constraint in element.eContainer().constraints){
182    if(constraint.isKindOf(PrimaryKey)){
183     for(attribute in constraint.attributes){
184      if(constraint.attributes.last().equals(attribute)){
185        out.print(attribute.name.toUpperCase());
186      }else{
187       out.print(attribute.name.toUpperCase()+",");
188       }}}}
189 out.println(") ON UPDATE CASCADE ON DELETE CASCADE;");
190 }else{
191 out.println("--BEGIN EXTRACT ASSOCIATIVE TABLE PATTERN SCRIPT");
192 out.print("CREATE TABLE IF NOT EXISTS ");
193 out.print(element.tableName.toUpperCase());
194 out.print("(");
195 for(attribute in attrib){
196 if(attrib.last().equals(attribute)){
197 out.print(attribute.first().name.toUpperCase()+" ");
198 out.print(attribute.first().getDataType(attribute.first()));
199 }else{
200 out.print(attribute.first().name.toUpperCase()+" ");
201 out.print(attribute.first().getDataType(attribute.first())+",");
202 }}
203 out.println(");");
204 out.print("ALTER TABLE ");
205 out.print(element.tableName.toUpperCase());
206 out.print(" ALTER COLUMN ");
207 for(constraint in element.eContainer().constraints){
208   if(constraint.isKindOf(PrimaryKey)){
209    for(attribute in constraint.attributes){
210     if(constraint.attributes.last().equals(attribute)){
```

```
211      out.print(attribute.name.toUpperCase());
212     out.print(" SET NOT NULL");
213     }else{
214     out.print(attribute.name.toUpperCase());
215     out.print(" SET NOT NULL,");
216     }}}}
217 out.println(";");
218 
219 out.print("ALTER TABLE ");
220 out.print(element.tableName.toUpperCase());
221 out.print(" ADD CONSTRAINT ");
222 for(constraint in element.eContainer().constraints){
223   if(constraint.isKindOf(PrimaryKey)){
224    for(attribute in constraint.attributes){
225     if(constraint.attributes.last().equals(attribute)){
226      out.print(attribute.name.toUpperCase());
227      out.print("_KEY");
228     }else{
229     out.print(attribute.name.toUpperCase());
230         out.print("_");
231 
232     }}}}
233 out.print(" UNIQUE(");
234 for(constraint in element.eContainer().constraints){
235   if(constraint.isKindOf(PrimaryKey)){
236    for(attribute in constraint.attributes){
237     if(constraint.attributes.last().equals(attribute)){
238      out.print(attribute.name.toUpperCase());
239     }else{
240     out.print(attribute.name.toUpperCase()+",");
241     }}}}
242 out.println(");");
243 
244 out.print("ALTER TABLE ");
245 out.print(element.tableName.toUpperCase());
246 out.print(" ALTER COLUMN ");
247 for(attribute in element.foreignKey.attributes){
248 
249     if(element.foreignKey.attributes.last().equals(attribute)){
250      out.print(attribute.name.toUpperCase());
251     out.print(" SET NOT NULL");
252    }else{
253     out.print(attribute.name.toUpperCase());
254     out.print(" SET NOT NULL,");
255     }
256 }
257 out.println(";");
```

```
258
259 out.print("INSERT INTO ");
260 out.print(element.tableName.toUpperCase()+"(");
261 for(attribute in attrib){
262 if(attrib.last().equals(attribute)){
263 out.print(attribute.first().name.toUpperCase());
264 }else{
265 out.print(attribute.first().name.toUpperCase()+",");
266 }}
267 out.print(")");
268 out.print(" SELECT ");
269 for(attribute in attrib){
270 if(attrib.last().equals(attribute)){
271 out.print(attribute.first().name.toUpperCase());
272 }else{
273 out.print(attribute.first().name.toUpperCase()+",");
274 }}
275 out.print(" FROM ");
276 out.print(element.eContainer().name.toUpperCase());
277 out.println(";");
278 out.print("ALTER TABLE ");
279 out.print(element.tableName.toUpperCase());
280 out.print(" ADD FOREIGN KEY(");
281 for(constraint in element.eContainer().constraints){
282   if(constraint.isKindOf(PrimaryKey)){
283    for(attribute in constraint.attributes){
284     if(constraint.attributes.last().equals(attribute)){
285      out.print(attribute.name.toUpperCase());
286     }else{
287     out.print(attribute.name.toUpperCase()+",");
288     }}}}
289 out.print(")");
290 out.print(" REFERENCES ");
291 out.print(element.eContainer().name.toUpperCase()+"(");
292 for(constraint in element.eContainer().constraints){
293   if(constraint.isKindOf(PrimaryKey)){
294    for(attribute in constraint.attributes){
295     if(constraint.attributes.last().equals(attribute)){
296      out.print(attribute.name.toUpperCase());
297     }else{
298     out.print(attribute.name.toUpperCase()+",");
299     }}}}
300 out.println(");");
301 out.print("ALTER TABLE ");
302 out.print(element.tableName.toUpperCase());
303 out.print(" ADD FOREIGN KEY(");
304 for(attribute in element.foreignKey.attributes){
```

```
305     if(element.foreignKey.attributes.last().equals(attribute)){
306      out.print(attribute.name.toUpperCase());
307    }else{
308     out.print(attribute.name.toUpperCase()+",");
309     }}
310 out.print(")");
311 out.print(" REFERENCES ");
312 out.print(element.foreignKey.foreignRelation.name.toUpperCase()+"(");
313 var targetAttributes:OrderedSet=getPrimaryKeyAttributes(element.foreignKey.foreignRelation)
        ;
314 for(attribute in targetAttributes){
315     if(targetAttributes.last().equals(attribute)){
316      out.print(attribute.name.toUpperCase());
317    }else{
318     out.print(attribute.name.toUpperCase()+",");
319     }}
320 out.println(");");
321 out.println();
322 out.print("CREATE OR REPLACE FUNCTION SYNC_ASSOCIATIVE_");
323 out.print(element.eContainer().name.toUpperCase());
324 out.print("_WITH_"+element.foreignKey.foreignRelation.name.toUpperCase()+"()");
325 out.println(" RETURNS TRIGGER AS "+"$SYNC_ASSOCIATIVE_"+element.eContainer().name.
        toUpperCase()+"_WITH_"+element.foreignKey.foreignRelation.name.toUpperCase()+"$");
326 out.println("BEGIN");
327 out.println("DECLARE COUNTER INTEGER;");
328 out.println("BEGIN");
329 out.println("IF (TG_OP='UPDATE') THEN");
330 out.print("IF(");
331 for(attribute in attrib){
332 if(attrib.last().equals(attribute)){
333 out.print("NEW."+attribute.first().name.toUpperCase()+" IS NOT NULL");
334 }else{
335 out.print("NEW."+attribute.first().name.toUpperCase()+" IS NOT NULL AND ");
336 }}
337 out.println(")THEN");
338 out.print("SELECT COUNT(*) INTO COUNTER FROM ");
339 out.print(element.tableName.toUpperCase()+" "+element.tableName.toUpperCase().charAt(0)+
        element.tableName.toUpperCase().charAt(1));
340 out.print(" WHERE ");
341 for(attribute in attrib){
342 if(attrib.last().equals(attribute)){
343 out.print("NEW."+attribute.first().name.toUpperCase()+"=");
344 out.print(element.tableName.toUpperCase().charAt(0)+element.tableName.toUpperCase().charAt
        (1)+"."+attribute.first().name.toUpperCase());
345 }else{
346 out.print("NEW."+attribute.first().name.toUpperCase()+"=");
347 out.print(element.tableName.toUpperCase().charAt(0)+element.tableName.toUpperCase().charAt
```

```
    (1)+"."+attribute.first().name.toUpperCase()+" OR ");
}}
out.println(";");
out.println("IF(COUNTER >0) THEN");
out.print("UPDATE ");
out.print(element.tableName.toUpperCase());
out.print(" SET ");
for(attribute in attrib){
if(attrib.last().equals(attribute)){
out.print(attribute.first().name.toUpperCase()+"=");
out.print("NEW."+attribute.first().name.toUpperCase());
}else{
out.print(attribute.first().name.toUpperCase()+"=");
out.print("NEW."+attribute.first().name.toUpperCase()+",");
}}
out.print(" WHERE ");
for(attribute in attrib){
if(attrib.last().equals(attribute)){
out.print(attribute.first().name.toUpperCase()+"=");
out.print("OLD."+attribute.first().name.toUpperCase());
}else{
out.print(attribute.first().name.toUpperCase()+"=");
out.print("OLD."+attribute.first().name.toUpperCase()+" AND ");
}}
out.println(";");
out.println("COUNTER:=0;");
out.println("END IF;");
out.println("END IF;");
out.println("RETURN NEW;");
out.println("END IF;");
out.println("IF (TG_OP='INSERT') THEN");
out.print("IF(");
for(attribute in attrib){
if(attrib.last().equals(attribute)){
out.print("NEW."+attribute.first().name.toUpperCase()+" IS NOT NULL");
}else{
out.print("NEW."+attribute.first().name.toUpperCase()+" IS NOT NULL AND ");
}}
out.println(")THEN");
out.print("SELECT COUNT(*) INTO COUNTER FROM ");
out.print(element.tableName.toUpperCase()+" "+element.tableName.toUpperCase().charAt(0)+
    element.tableName.toUpperCase().charAt(1));
out.print(" WHERE ");
for(attribute in attrib){
if(attrib.last().equals(attribute)){
out.print("NEW."+attribute.first().name.toUpperCase()+"=");
out.print(element.tableName.toUpperCase().charAt(0)+element.tableName.toUpperCase().charAt
```

```
    (1)+"."+attribute.first().name.toUpperCase());
}else{
out.print("NEW."+attribute.first().name.toUpperCase()+"=");
out.print(element.tableName.toUpperCase().charAt(0)+element.tableName.toUpperCase().charAt
    (1)+"."+attribute.first().name.toUpperCase()+" AND ");
}}
out.println(";");
out.println("IF(COUNTER=0) THEN");
out.print("INSERT INTO ");
out.print(element.tableName.toUpperCase()+"(");
for(attribute in attrib){
if(attrib.last().equals(attribute)){
out.print(attribute.first().name.toUpperCase());
}else{
out.print(attribute.first().name.toUpperCase()+",");
}}

out.print(") VALUES (");
for(attribute in attrib){
if(attrib.last().equals(attribute)){
out.print("NEW."+attribute.first().name.toUpperCase());
}else{
out.print("NEW."+attribute.first().name.toUpperCase()+",");
}}
out.println(");");
out.println("END IF;");
out.println("END IF;");
out.println("RETURN NEW;");
out.println("END IF;");
out.println("IF (TG_OP='DELETE') THEN");
out.print("DELETE FROM ");
out.print(element.tableName.toUpperCase());
out.print(" WHERE ");
for(attribute in attrib){
if(attrib.last().equals(attribute)){
out.print(attribute.first().name.toUpperCase()+"=");
out.print("OLD."+attribute.first().name.toUpperCase());

}else{
out.print(attribute.first().name.toUpperCase()+"=");
out.print("OLD."+attribute.first().name.toUpperCase()+" AND ");
}}
out.println(";");
out.println("RETURN OLD;");
out.println("END IF;");
out.println("END;");
out.println("END;");
```

```
438 out.println("$SYNC_ASSOCIATIVE_"+element.eContainer().name.toUpperCase()+"_WITH_"+element.
        foreignKey.foreignRelation.name.toUpperCase()+"$"+" LANGUAGE plpgsql;");
439 out.println();
440 out.print("DROP TRIGGER IF EXISTS ");
441 out.print("SYNC_ASSOCIATIVE_"+element.eContainer().name.toUpperCase()+"_WITH_"+element.
        foreignKey.foreignRelation.name.toUpperCase());
442 out.println(" ON "+element.eContainer().name.toUpperCase()+";");
443 out.println();
444 out.print("CREATE TRIGGER ");
445 out.println("SYNC_ASSOCIATIVE_"+element.eContainer().name.toUpperCase()+"_WITH_"+element.
        foreignKey.foreignRelation.name.toUpperCase());
446 out.println("AFTER INSERT OR UPDATE OR DELETE ON "+element.eContainer().name.toUpperCase())
        ;
447 out.println("FOR EACH ROW");
448 out.print("EXECUTE PROCEDURE ");
449 out.println("SYNC_ASSOCIATIVE_"+element.eContainer().name.toUpperCase()+"_WITH_"+element.
        foreignKey.foreignRelation.name.toUpperCase()+"();");
450
451 out.println();
452 out.print("CREATE OR REPLACE FUNCTION SYNC_ASSOCIATIVE_");
453 out.print(element.tableName.toUpperCase()+"()");
454 out.println(" RETURNS TRIGGER AS "+"$SYNC_ASSOCIATIVE_"+element.tableName.toUpperCase()+"$"
        );
455 out.println("BEGIN");
456 out.println("DECLARE COUNTER INTEGER;");
457 out.println("BEGIN");
458 out.println("IF (TG_OP='UPDATE') THEN");
459 out.print("IF(");
460 for(attribute in attrib){
461 if(attrib.last().equals(attribute)){
462 out.print("NEW."+attribute.first().name.toUpperCase()+" IS NOT NULL");
463 }else{
464 out.print("NEW."+attribute.first().name.toUpperCase()+" IS NOT NULL AND ");
465 }}
466 out.println(")THEN");
467 out.print("SELECT COUNT(*) INTO COUNTER FROM ");
468 out.print(element.eContainer.name.toUpperCase()+" "+element.eContainer.name.toUpperCase().
        charAt(0)+element.eContainer.name.toUpperCase().charAt(1));
469 out.print(" WHERE ");
470 for(attribute in attrib){
471 if(attrib.last().equals(attribute)){
472 out.print("NEW."+attribute.first().name.toUpperCase()+"=");
473 out.print(element.eContainer.name.toUpperCase().charAt(0)+element.eContainer.name.
        toUpperCase().charAt(1)+"."+attribute.first().name.toUpperCase());
474 }else{
475 out.print("NEW."+attribute.first().name.toUpperCase()+"=");
476 out.print(element.eContainer.name.toUpperCase().charAt(0)+element.eContainer.name.
```

```
    toUpperCase().charAt(1)+"."+attribute.first().name.toUpperCase()+" OR ");
}}
out.println(";");
out.println("IF(COUNTER >0) THEN");
out.print("UPDATE ");
out.print(element.eContainer.name.toUpperCase());
out.print(" SET ");
for(attribute in attrib){
if(attrib.last().equals(attribute)){
out.print(attribute.first().name.toUpperCase()+"=");
out.print("NEW."+attribute.first().name.toUpperCase());
}else{
out.print(attribute.first().name.toUpperCase()+"=");
out.print("NEW."+attribute.first().name.toUpperCase()+",");
}}
out.print(" WHERE ");
for(attribute in attrib){
if(attrib.last().equals(attribute)){
out.print(attribute.first().name.toUpperCase()+"=");
out.print("OLD."+attribute.first().name.toUpperCase());
}else{
out.print(attribute.first().name.toUpperCase()+"=");
out.print("OLD."+attribute.first().name.toUpperCase()+" AND ");
}}
out.println(";");
out.println("COUNTER:=0;");
out.println("END IF;");
out.println("END IF;");
out.println("RETURN NEW;");
out.println("END IF;");
out.println("IF (TG_OP='INSERT') THEN");
out.print("IF(");
for(attribute in attrib){
if(attrib.last().equals(attribute)){
out.print("NEW."+attribute.first().name.toUpperCase()+" IS NOT NULL");
}else{
out.print("NEW."+attribute.first().name.toUpperCase()+" IS NOT NULL AND ");
}}
out.println(")THEN");
out.print("SELECT COUNT(*) INTO COUNTER FROM ");
out.print(element.eContainer.name.toUpperCase()+" "+element.eContainer.name.toUpperCase().
    charAt(0)+element.eContainer.name.toUpperCase().charAt(1));
out.print(" WHERE ");
for(attribute in attrib){
if(attrib.last().equals(attribute)){
out.print("NEW."+attribute.first().name.toUpperCase()+"=");
out.print(element.eContainer.name.toUpperCase().charAt(0)+element.eContainer.name.
```

```
    toUpperCase().charAt(1)+"."+attribute.first().name.toUpperCase());
}else{
out.print("NEW."+attribute.first().name.toUpperCase()+"=");
out.print(element.eContainer.name.toUpperCase().charAt(0)+element.eContainer.name.
    toUpperCase().charAt(1)+"."+attribute.first().name.toUpperCase()+" AND ");
}}
out.println(";");
out.println("IF(COUNTER=0) THEN");
out.print("INSERT INTO ");
out.print(element.eContainer.name.toUpperCase()+"(");
for(attribute in attrib){
if(attrib.last().equals(attribute)){
out.print(attribute.first().name.toUpperCase());
}else{
out.print(attribute.first().name.toUpperCase()+",");
}}
out.print(") VALUES (");
for(attribute in attrib){
if(attrib.last().equals(attribute)){
out.print("NEW."+attribute.first().name.toUpperCase());
}else{
out.print("NEW."+attribute.first().name.toUpperCase()+",");
}}
out.println(");");
out.println("END IF;");
out.println("END IF;");
out.println("RETURN NEW;");
out.println("END IF;");
out.println("IF (TG_OP='DELETE') THEN");
out.print("DELETE FROM ");
out.print(element.eContainer.name.toUpperCase());
out.print(" WHERE ");
for(attribute in attrib){
if(attrib.last().equals(attribute)){
out.print(attribute.first().name.toUpperCase()+"=");
out.print("OLD."+attribute.first().name.toUpperCase());
}else{
out.print(attribute.first().name.toUpperCase()+"=");
out.print("OLD."+attribute.first().name.toUpperCase()+" AND ");
}}
out.println(";");
out.println("RETURN OLD;");
out.println("END IF;");
out.println("END;");
out.println("END;");
out.println("$SYNC_ASSOCIATIVE_"+element.tableName.toUpperCase()+"$"+" LANGUAGE plpgsql;");
out.println();
```

```
out.print("DROP TRIGGER IF EXISTS ");
out.print("SYNC_ASSOCIATIVE_"+element.tableName.toUpperCase());
out.println(" ON "+element.tableName.toUpperCase()+";");
out.println();
out.print("CREATE TRIGGER ");
out.println("SYNC_ASSOCIATIVE_"+element.tableName.toUpperCase());
out.println("AFTER INSERT OR UPDATE OR DELETE ON "+element.tableName.toUpperCase());
out.println("FOR EACH ROW");
out.print("EXECUTE PROCEDURE ");
out.println("SYNC_ASSOCIATIVE_"+element.tableName.toUpperCase()+"();");
out.println();
out.println("--AFTER TRANSITIONAL PERIOD RUN THIS SCRIPT");
out.print("--ALTER TABLE ");
out.print(element.eContainer().name.toUpperCase());
out.print(" DROP ");
for(attribute in element.foreignKey.attributes){
    if(element.foreignKey.attributes.last().equals(attribute)){
    out.print(attribute.name.toUpperCase());
    }else{
    out.print(attribute.name.toUpperCase()+",");
    }}
out.println(";");
out.print("--ALTER TABLE ");
out.print(element.tableName.toUpperCase());
out.print(" ADD FOREIGN KEY(");
for(attribute in element.foreignKey.attributes){
    if(element.foreignKey.attributes.last().equals(attribute)){
    out.print(attribute.name.toUpperCase());
    }else{
    out.print(attribute.name.toUpperCase()+",");
    }}
out.print(")");
out.print(" REFERENCES ");
out.print(element.foreignKey.foreignRelation.name.toUpperCase()+"(");
for(constraint in element.foreignKey.foreignRelation.constraints){
  if(constraint.isKindOf(PrimaryKey)){
   for(attribute in constraint.attributes){
    if(constraint.attributes.last().equals(attribute)){
    out.print(attribute.name.toUpperCase());
    }else{
    out.print(attribute.name.toUpperCase()+",");
    }}}}
out.println(") ON UPDATE CASCADE ON DELETE CASCADE;");
out.print("--ALTER TABLE ");
out.print(element.tableName.toUpperCase());
out.print(" ADD FOREIGN KEY(");
for(constraint in element.eContainer().constraints){
```

```
614    if(constraint.isKindOf(PrimaryKey)){
615     for(attribute in constraint.attributes){
616      if(constraint.attributes.last().equals(attribute)){
617        out.print(attribute.name.toUpperCase());
618      }else{
619       out.print(attribute.name.toUpperCase()+",");
620       }}}}
621 out.print(")");
622 out.print(" REFERENCES ");
623 out.print(element.eContainer().name.toUpperCase()+"(");
624 for(constraint in element.eContainer().constraints){
625    if(constraint.isKindOf(PrimaryKey)){
626     for(attribute in constraint.attributes){
627      if(constraint.attributes.last().equals(attribute)){
628        out.print(attribute.name.toUpperCase());
629      }else{
630       out.print(attribute.name.toUpperCase()+",");
631       }}}}
632 out.println(") ON UPDATE CASCADE ON DELETE CASCADE;");
633 }}
```

Listing B.10 – Regras de transformação M2T referente à refatoração *Extract Associative Table*

```
1  for(element in SplitTable.allInstances()) {
2  var constraints:Collection;
3  var primaryKey:PrimaryKey;
4  if(element.transitionalPeriod = false){
5  out.println("--BEGIN OF SPLIT TABLE PATTERN SCRIPT");
6  out.print("CREATE TABLE IF NOT EXISTS "+element.targetTableName.toUpperCase()+"(");
7  for(attribute in element.columns){
8  if(element.columns.last().equals(attribute)){
9    out.print(attribute.name.toUpperCase()+" "+attribute.getDataType(attribute));
10   if(attribute.defaultValue.isDefined()){
11   out.print(" DEFAULT "+attribute.defaultValue);
12   }
13   if(attribute.isNotNull = true){
14   out.print(" NOT NULL ");
15   }
16   }else{
17  out.print(attribute.name.toUpperCase()+" "+attribute.getDataType(attribute));
18   if(attribute.defaultValue.isDefined()){
19   out.print(" DEFAULT "+attribute.defaultValue);
20   }
21   if(attribute.isNotNull = true){
22   out.print(" NOT NULL ");
23   }
24   out.println(",");
```

```
25    }}
26  out.println(");");
27
28  out.print("INSERT INTO "+element.targetTableName.toUpperCase()+"(");
29  var distinct:String;
30  distinct:=element.distinctColumns.name.asString();
31  out.print(distinct.toUpperCase()+",");
32  for(attribute in element.columns){
33  if(element.columns.last().equals(attribute)){
34  if(not attribute.name.toUpperCase().equals(distinct.toUpperCase())){
35  out.print(attribute.name.toUpperCase());
36  }
37  }else{
38  if(not attribute.name.toUpperCase().equals(distinct.toUpperCase())){
39  if(element.columns.last().name.toUpperCase().equals(distinct.toUpperCase())){
40  out.print(attribute.name.toUpperCase());
41  }else{
42  out.print(attribute.name.toUpperCase()+",");
43  }}}}
44  out.print(") ");
45  out.print("SELECT DISTINCT ");
46  out.print(distinct.toUpperCase()+",");
47  for(attribute in element.columns){
48  if(element.columns.last().equals(attribute)){
49  if(not attribute.name.toUpperCase().equals(distinct.toUpperCase())){
50  out.print(attribute.name.toUpperCase());
51  }}else{
52  if(not attribute.name.toUpperCase().equals(distinct.toUpperCase())){
53  if(element.columns.last().name.toUpperCase().equals(distinct.toUpperCase())){
54  out.print(attribute.name.toUpperCase());
55  }else{
56  out.print(attribute.name.toUpperCase()+",");
57  }}}}
58  out.print(" FROM "+element.eContainer().name.toUpperCase());
59  out.print(" WHERE "+element.distinctColumns.name.toUpperCase());
60  out.println(" IS NOT NULL;");
61  out.print("ALTER TABLE "+element.name.toUpperCase());
62  out.print(" ADD CONSTRAINT ");
63  primaryKey := element.relationConstraints->select(p | p.isKindOf(PrimaryKey)).last();
64  if(primaryKey.name.isUndefined()){
65  out.print(element.name.toUpperCase()+"_PKEY");
66  }else{
67  out.print(primaryKey.name.toUpperCase());
68  }
69  out.print(" PRIMARY KEY(");
70  for(attribute in primaryKey.attributes){
71  if(primaryKey.attributes.last().equals(attribute)){
```

```
72 out.print(attribute.name.toUpperCase());
73 }else{
74 out.print(attribute.name.toUpperCase()+",");
75 }
76 out.println(");");
77 }
78 constraints := element.eContainer().relationConstraints->select(c | c.isKindOf(ForeignKey))
       ;
79
80 out.print("ALTER TABLE "+element.eContainer().name.toUpperCase());
81 out.print(" ADD CONSTRAINT ");
82
83 for(constraint in constraints){
84  if(constraint.target = element){
85  if(constraint.name.isUndefined()){
86 out.print(element.eContainer().name.toUpperCase()+"_"+element.targetTableName.toUpperCase()
       +"_FKEY");
87
88 }else{
89 out.print(constraint.name.toUpperCase());
90 }
91 out.print(" FOREIGN KEY(");
92 for(attribute in constraint.attributes){
93 if(constraint.attributes.last().equals(attribute)){
94 out.print(attribute.name.toUpperCase());
95 out.print(")");
96 }else{
97 out.print(attribute.name.toUpperCase()+",");
98 }
99 out.print(" REFERENCES ");
100 out.print(element.name.toUpperCase()+"(");
101 for(attribute in primaryKey.attributes){
102 if(primaryKey.attributes.last().equals(attribute)){
103 out.print(attribute.name.toUpperCase());
104 }else{
105 out.print(attribute.name.toUpperCase()+",");
106 }}
107 out.print(")");
108 out.println(" ON UPDATE CASCADE ON DELETE RESTRICT;");
109 }}}
110 out.println();
111 out.print("ALTER TABLE "+element.sourceRelation.name.toUpperCase());
112 for(attribute in element.attributes){
113 if(element.attributes.last().name.toUpperCase().equals(attribute.name.toUpperCase()) and
       not element.attributes.last().name.toUpperCase().equals(element.distinctColumn.name.
       toUpperCase())){
114 out.print(" DROP COLUMN ");
```

```
115 out.print(attribute.name.toUpperCase()+" CASCADE");
116 }else if(not attribute.name.toUpperCase().equals(element.distinctColumn.name.toUpperCase()
    ) ){
117 if(element.attributes.last().name.toUpperCase().equals(distinct.toUpperCase())){
118 out.print(" DROP COLUMN ");
119 out.print(attribute.name.toUpperCase()+" CASCADE");
120 }else{
121 out.print(" DROP COLUMN ");
122 out.print(attribute.name.toUpperCase()+" CASCADE,");
123 }}}
124 out.println(";");
125 }else{
126 out.println("--BEGIN OF SPLIT TABLE PATTERN SCRIPT");
127 out.print("CREATE TABLE "+element.targetTableName.toUpperCase()+"(");
128 for(attribute in element.columns){
129 if(element.columns.last().equals(attribute)){
130   out.print(attribute.name.toUpperCase()+" "+attribute.getDataType(attribute));
131   if(attribute.defaultValue.isDefined()){
132   out.print(" DEFAULT "+attribute.defaultValue);
133   }
134   if(attribute.isNotNull = true){
135   out.print(" NOT NULL ");
136   }
137   }else{
138  out.print(attribute.name.toUpperCase()+" "+attribute.getDataType(attribute));
139   if(attribute.defaultValue.isDefined()){
140   out.print(" DEFAULT "+attribute.defaultValue);
141   }
142   if(attribute.isNotNull = true){
143   out.print(" NOT NULL ");
144   }
145   out.println(",");
146   }}
147 out.println(");");
148 out.print("INSERT INTO "+element.targetTableName.toUpperCase()+"(");
149 var distinct:String;
150 distinct:=element.distinctColumns.last().name.asString();
151 out.print(distinct.toUpperCase()+",");
152 for(attribute in element.columns){
153 if(element.columns.last().equals(attribute)){
154 if(not attribute.name.toUpperCase().equals(distinct.toUpperCase())){
155 out.print(attribute.name.toUpperCase());
156 }
157 }else{
158 if(not attribute.name.toUpperCase().equals(distinct.toUpperCase())){
159 if(element.columns.last().name.toUpperCase().equals(distinct.toUpperCase())){
160 out.print(attribute.name.toUpperCase());
```

```
}else{
out.print(attribute.name.toUpperCase()+",");
}}}}
out.print(") ");
out.print("SELECT DISTINCT ");
out.print(distinct.toUpperCase()+",");
for(attribute in element.columns){
if(element.columns.last().equals(attribute)){
if(not attribute.name.toUpperCase().equals(distinct.toUpperCase())){
out.print(attribute.name.toUpperCase());
}
}else{
if(not attribute.name.toUpperCase().equals(distinct.toUpperCase())){
if(element.columns.last().name.toUpperCase().equals(distinct.toUpperCase())){
out.print(attribute.name.toUpperCase());
}else{
out.print(attribute.name.toUpperCase()+",");
}}}}
out.print(" FROM "+element.eContainer().name.toUpperCase());
out.print(" WHERE "+element.distinctColumns.last().name.toUpperCase());
out.println(" IS NOT NULL;");
out.print("ALTER TABLE "+element.targetTableName.toUpperCase());
out.print(" ADD CONSTRAINT ");
out.print(element.targetTableName.toUpperCase()+"_PKEY");
out.print(" PRIMARY KEY");
out.print("(");
for(attribute in element.distinctColumns){
if(element.distinctColumns.last().equals(attribute)){
out.print(attribute.name.toUpperCase());
}else{
out.print(attribute.name.toUpperCase()+",");
}}
out.println(");");
out.print("ALTER TABLE "+element.eContainer().name.toUpperCase());
out.print(" ADD CONSTRAINT ");
out.print(element.eContainer().name.toUpperCase()+"_"+element.targetTableName.toUpperCase()
    +"_FKEY");
out.print(" FOREIGN KEY");
out.print("(");
for(attribute in element.distinctColumns){
if(element.distinctColumns.last().equals(attribute)){
out.print(attribute.name.toUpperCase());
}else{
out.print(attribute.name.toUpperCase()+",");
}
}
out.print(")");
```

```
207 out.print(" REFERENCES ");
208 out.print(element.targetTableName.toUpperCase());
209 out.print("(");
210 for(attribute in element.distinctColumns){
211 if(element.distinctColumns.last().equals(attribute)){
212 out.print(attribute.name.toUpperCase());
213 }else{
214 out.print(attribute.name.toUpperCase()+",");
215 }
216 }
217 out.print(")");
218 out.println(" ON UPDATE CASCADE ON DELETE RESTRICT;");
219 out.print("CREATE OR REPLACE FUNCTION SYNC"+element.eContainer().name.toUpperCase()+"WITH"+
        element.targetTableName.toUpperCase()+"()");
220 out.println(" RETURNS TRIGGER AS $SYNC"+element.eContainer().name.toUpperCase()+"WITH"+
        element.targetTableName.toUpperCase()+"$");
221 out.println("BEGIN");
222 out.println("DECLARE COUNTER INTEGER;");
223
224 out.println("BEGIN");
225 out.println("IF(TG_OP = 'INSERT') THEN");
226 out.print("SELECT INTO COUNTER COUNT(*) FROM "+element.targetTableName.toUpperCase());
227 out.print(" WHERE ");
228 out.print(element.distinctColumns.last().name.toUpperCase()+"="+"NEW."+element.
        distinctColumns.last().name.toUpperCase());
229 out.println(";");
230 out.println("IF (COUNTER=0) THEN");
231 out.print("INSERT INTO "+element.targetTableName.toUpperCase()+" (");
232 for(attribute in element.columns){
233 if(element.columns.last().equals(attribute)){
234 out.print(attribute.name.toUpperCase());
235 }else{
236 out.print(attribute.name.toUpperCase()+",");
237 }
238 }
239 out.print(")");
240 out.print("VALUES");
241 out.print("(");
242 for(attribute in element.columns){
243 if(element.columns.last().equals(attribute)){
244 out.print("NEW."+attribute.name.toUpperCase());
245 }else{
246 out.print("NEW."+attribute.name.toUpperCase()+",");
247 }
248 }
249 out.println(");");
250 out.println("END IF;");
```

```
251 out.println("RETURN NEW;");
252 out.println("END IF;");
253 out.println();
254 out.println("IF(TG_OP = 'UPDATE') THEN");
255 out.print("IF");
256 out.print("(");
257 
258 for(attribute in element.columns){
259 if(element.columns.last().equals(attribute)){
260 if(not attribute.name.toUpperCase().equals(distinct.toUpperCase())){
261 out.print("OLD."+attribute.name.toUpperCase()+"!=");
262 out.print("NEW."+attribute.name.toUpperCase());
263 }
264 }else{
265 if(not attribute.name.toUpperCase().equals(distinct.toUpperCase())){
266 if(element.columns.last().name.toUpperCase().equals(distinct.toUpperCase())){
267 out.print("OLD."+attribute.name.toUpperCase()+"!=");
268 out.print("NEW."+attribute.name.toUpperCase());
269 }else{
270 out.print("OLD."+attribute.name.toUpperCase()+"!=");
271 out.print("NEW."+attribute.name.toUpperCase()+" OR ");
272 }}}}
273 out.print(")");
274 out.println("THEN");
275 out.print("UPDATE "+element.targetTableName.toUpperCase()+" SET ");
276 for(attribute in element.columns){
277 if(element.columns.last().equals(attribute)){
278 if(not attribute.name.toUpperCase().equals(distinct.toUpperCase())){
279 out.print(attribute.name.toUpperCase()+"=");
280 out.print("NEW."+attribute.name.toUpperCase());
281 }
282 }else{
283 if(not attribute.name.toUpperCase().equals(distinct.toUpperCase())){
284 if(element.columns.last().name.toUpperCase().equals(distinct.toUpperCase())){
285 out.print(attribute.name.toUpperCase()+"=");
286 out.print("NEW."+attribute.name.toUpperCase());
287 }else{
288 out.print(attribute.name.toUpperCase()+"=");
289 out.print("NEW."+attribute.name.toUpperCase()+",");
290 }}}}
291 out.print(" WHERE ");
292 out.print(element.distinctColumns.last().name.toUpperCase()+"=");
293 out.println("OLD."+element.distinctColumns.last().name.toUpperCase()+";");
294 out.println("RETURN NEW;");
295 out.println("ELSE");
296 out.println("RETURN OLD;");
297 out.println("END IF;");
```

```
298 out.println("END IF;");
299 out.println("END;");
300 out.println("END;");
301 out.print("$SYNC"+element.eContainer().name.toUpperCase()+"WITH"+element.targetTableName.
        toUpperCase()+"$");
302 out.println(" LANGUAGE plpgsql;");
303 out.println();
304 out.println();
305 out.print("DROP TRIGGER IF EXISTS SYNC"+element.eContainer().name.toUpperCase()+"WITH"+
        element.targetTableName.toUpperCase());
306 out.println(" ON "+element.eContainer().name.toUpperCase()+";");
307 out.println();
308 out.println("CREATE TRIGGER SYNC"+element.eContainer().name.toUpperCase()+"WITH"+element.
        targetTableName.toUpperCase());
309 out.println("BEFORE INSERT OR UPDATE ON "+element.eContainer().name.toUpperCase());
310 out.println("FOR EACH ROW");
311 out.println("EXECUTE PROCEDURE SYNC"+element.eContainer().name.toUpperCase()+"WITH"+element
        .targetTableName.toUpperCase()+"();");
312 out.println("--AFTER TRANSITIONAL PERIOD RUN THIS SCRIPT");
313 out.print("--ALTER TABLE ");
314 out.print(element.eContainer().name.toUpperCase());
315 for(attribute in element.columns){
316 if(element.columns.last().equals(attribute)){
317 out.print(" DROP ");
318 out.print(attribute.name.toUpperCase());
319 }else{
320 out.print(" DROP ");
321 out.print(attribute.name.toUpperCase()+",");
322 }}
323 out.println(";");
324 }
325 out.println("--END OF SPLIT TABLE PATTERN SCRIPT");
326 }
```

Listing B.11 – Regras de transformação M2T referente à refatoração *Split Table*

APÊNDICE C – CÓDIGOS EOL

Este apêndice contém as operações que suportam as regras EVL e EGL definidas no apêndice D e B, respectivamente.

```
1  operation getChildrensforERelation(relation:ERelation):Collection{
2  var relations: Set;
3        for(eRelation in ERelation.allInstances()){
4         for(const in eRelation.constraints->select(c | c.isKindOf(ForeignKey))){
5         if(relation.equals(const.foreignRelation)){
6         relations.add(eRelation);
7         }
8         }
9        }
10       return relations;
11
12 }
```

Listing C.1 – Operação que retorna as tabelas filhas de uma relação.

```
1
2  operation getPrimaryKey(ERelation:ERelation):PrimaryKey{
3       for(constraint in ERelation.constraints){
4       if(constraint.isKindOf(PrimaryKey)){
5        return constraint;
6       }
7       }
8  }
```

Listing C.2 – Operação que retorna as chaves primárias de uma relação.

```
1
2  operation getPrimaryKeyAttributes(ERelation:ERelation):Collection{
3       for(constraint in ERelation.constraints){
4       if(constraint.isKindOf(PrimaryKey)){
5       return constraint.attributes;
6       }
7       }
8  }
```

Listing C.3 – Operação que retorna os atributos da chave primária de uma relação.

```
1
2  operation getForeignKeys(ERelation:ERelation):Collection{
3        var constraints:Collection;
4       for(constraint in ERelation.constraints){
```

```
5       if(constraint.isKindOf(ForeignKey)){
6       constraints.add(constraint);
7        return constraints;
8
9       }
10      }
11  }
```

Listing C.4 – Operação que retorna as chaves estrangeiras de uma relação.

```
1
2   operation getForeignKeyAttributes(ERelation:ERelation):Collection{
3       for(constraint in ERelation.constraints){
4       if(constraint.isKindOf(ForeignKey)){
5       return constraint.attributes;
6       }
7       }
8   }
```

Listing C.5 – Operação que retorna os atributos das chaves estrangeiras de uma relação.

APÊNDICE D – CÓDIGOS EVL

Este apêndice contém as regras EVL referentes às definições de semântica estática (cf. Seção 4.5).

```
1
2  constraint MustHaveColumnToDrop{
3
4        check: self.columnsToDrop.isDefined()
5
6        message: self.eClass().name+' must have a column to drop.'
7  }
8
9  }
```

Listing D.1 – Validações EVL para o construtor *Drop Column*

```
1
2  context CalculatedColumn{
3  constraint HasColumns{
4        guard:self.satisfies('HasName')
5        check{
6        var flag:Boolean;
7         if(self.columns->forAll(c|c.eContainer() = self.eContainer()) and self.columns.size
              >1){
8         flag:=true;
9         }else if(self.columns.size>0){
10        flag:=true;
11        }else{
12        flag:=false;
13        }
14       return flag;
15       }
16       message:"(RFCACL01) You must define at least one columns"
17 }
18
19  constraint HasName{
20       check: self.name.isDefined()
21        message: '(RFCACL02) '+self.eClass().name+' must have a name'
22  }
23
24 constraint MustBeNumber {
25       check{
26       if(self.dataType = BaseType#INTEGER or self.dataType = BaseType#NUMERIC){
27        return true;
28       } else{
```

```
29        return false;
30       }
31       }
32      message: '(RFCACL03) The data type must be a number'
33 }
34
35 constraint ColumnsMustBeNumberSameRelation{
36          guard: self.satisfies('HasColumns')
37  check{
38  var flag:Boolean;
39  if(self.columns->forAll(c|c.eContainer() = self.eContainer())){
40   for(c in self.columns){
41     if((c.dataType = BaseType#INTEGER) or (c.dataType = BaseType#NUMERIC)){
42            flag:=true;
43            }
44            else{
45            flag:=false;
46            }
47   }
48 }else{
49  flag:=true;
50 }
51  return flag;
52  }
53  message:'(RFCACL04) The columns from the Relation on '+self.eClass().name+' must be of
        type number.'
54 }
55
56 constraint ColumnMustBeNumberRelatedRelation{
57   guard:self.satisfies('relatedEContainer')
58    check{
59    var flag:Boolean;
60    if(not self.columns->forAll(c|c.eContainer() = self.eContainer())){
61      if( (self.columns.last().dataType = BaseType#INTEGER) or (self.columns.last().dataType
           = BaseType#NUMERIC)){
62            flag:=true;
63            }
64            else{
65            flag:=false;
66            }
67            }else{
68            flag:=true;
69            }
70       return flag;
71    }
72    message:'(RFCACL05) The column from the Source Relation on '+self.eClass().name+' must
        be of type number.'
```

```
}

constraint relatedEContainer{
   check{
   var flag:Boolean;
   if(self.columns.size=1 and not self.columns->forAll(c|c.eContainer() = self.eContainer()
       )){
      for(relation in Relation.allInstances()){
   if(relation.constraints->select(c | c.isKindOf(ForeignKey)).size()>0){
         for(c in relation.constraints->select(f | f.isKindOf(ForeignKey))){
        if(self.eContainer().equals(c.foreignRelation)){
            if(c.sourceRelation.attributes->exists(a| a.equals(self.columns.last()))){
            flag:=true;
            }
            else{
            flag:=false;
            }}}}
       }
       }else{
       flag:=true;
       }
       return flag;
   }
      message:'(RFCACL06) You must define a column from the Source Relation on '+self.
          eClass().name+'.'

}

constraint defineColumns{
         check:self.columns.notEmpty()
          message: '(RFCACL07) Columns must be set for '+self.eClass().name+'.'
}
}
```

Listing D.2 – Validações EVL para o construtor *Calculated Pattern*

```

   context ReplaceKey{
       constraint HasKey{
         check: self.key.isDefined()
         message:'(RFRPKY01) You must define a key to Replace on '+self.eContainer().name+'
              Refactoring Implementation.'
       }
       constraint HasNewPrimaryKeyAttributes{
         check: self.newPrimaryKeyAttributes.size()>0
         message:'(RFRPKY02) You must set the Primary Key Attributes to Replace on '+self.
             eContainer().name+' Refactoring Implementation.'
       }
```

```
11 }
```

Listing D.3 – Validações EVL para o construtor *Replace Key*

```
1  context MergeTable{
2
3   constraint NotSameRelation{
4              check: not self.source.equals(self.target)
5              message: '(RFMRTB01) Cannot merge the same Relation.'
6   }
7   constraint hasParent{
8           check{
9           var bol:Boolean;
10                 for(relation in getChildrensforERelation(self.target)){
11                       if(relation.equals(self.source)){
12                        bol = true;
13                       }else{
14                       bol = false;
15                       }
16                 }
17                 return bol;
18          }
19          message: '(RFMRTB02) The source relation must be a children of target.'
20  }
21 }
```

Listing D.4 – Validações EVL para o construtor *Merge Table*

```
1
2  context MoveColumn{
3   constraint MustHaveColumnToMove{
4        check: self.columnToMove.isDefined()
5          message: '(RFMVCL01) '+self.eClass().name+' must have a column to move.'
6        }
7   constraint NotSameRelation{
8              check: not self.source.equals(self.target)
9             message: '(RFMVCL02) Cannot move to the same Relation.'
10  }
11  }
```

Listing D.5 – Validações EVL para o construtor *Move Column*

```
1     context RenameColumn{
2        constraint HasNewName{
3          check: self.renameTo.isDefined()
4          message:'(RFRNCL01) You must set a new Column name for '+self.eClass().name+'
               Refactoring Implementation.'
5        }
6        constraint HasColumnToRename{
```

```
7         check: self.columnToRename.isDefined()
8         message:'(RFRNCL02) You must set the Column to Rename for '+self.eClass().name+'
              Refactoring Implementation.'
9       }
10  }
```

Listing D.6 – Validações EVL para o construtor *Rename Column*

```
1       context RenameTable{
2      constraint HasNewName{
3         check: self.renameTo.isDefined()
4         message:'(RFRNTB01) You must set a new name for '+self.eContainer().name+'.'
5       }
6    }
```

Listing D.7 – Validações EVL para o construtor *Rename Table*

```
1    context RenameView{
2       constraint HasNewName{
3          check: self.renameTo.isDefined()
4           message:'(RFRNTB02) You must set a new name for '+self.eContainer().name+'.'
5       }
6    }
```

Listing D.8 – Validações EVL para o construtor *Rename View*

```
1   context ReplaceColumn{
2       constraint HasNewColumn{
3         check: self.replaceTo.isDefined()
4         message:'(RFRPCL01) You must set a new Column name for '+self.eClass().name+'
              Refactoring Implementation.'
5       }
6
7       constraint HasColumnToReplace{
8          guard:self.satisfies('HasNewColumn')
9         check: self.columnToReplace.isDefined()
10        message:'(RFRPCL02) You must set the column to Replace for '+self.eClass().name+'
              Refactoring Implementation.'
11      }
12  critique CritiqueDataType{
13        guard: self.satisfies('HasColumnToReplace')
14       check{
15             if( not self.replaceTo.dataType.equals(self.columnToReplace.dataType)){
16              return false;
17             }else{
18             return true;
19             }
20        }
```

```
21        message: '(RFRPCL03) The data type of related columns are different. It can lead to
              data lost.'
22      }
23  }
```

Listing D.9 – Validações EVL para o construtor *Replace Column*

```
1     context ExtractAssociativeTable{
2              constraint HasPrimaryKey{
3                   check{
4                       if(self.eContainer().constraints->select(p | p.isKindOf(PrimaryKey))
                            .size()>0){
5                        return true;
6                       } else{
7                        return false;
8                       }
9                   }
10                  message:"(RFEAT02) You must set a Primary Key for relation "+self.
                        eContainer().name+"."
11             }
12             constraint HasName{
13                  check{
14                      if(self.tableName.isDefined()){
15                       return true;
16                      } else{
17                       return false;
18                      }
19                    }
20                  message:"(RFEAT01) You must set a name for the new Associative Table on
                        "+self.eContainer().name+"."
21               }
22             constraint HasForeignKey{
23                  check{
24                      if(self.foreignKey.isDefined()){
25                       return true;
26                      } else{
27                       return false;
28                      }
29                  }
30                  message:"(RFEAT03) You must set a Foreign Key on "+self.eClass().name+".
                        "
31             }
32              constraint hasParent{
33                guard:self.satisfies('HasForeignKey')
34         check{
35         var bol:Boolean;
36                for(relation in getChildrensforERelation(self.foreignKey.foreignRelation))
                       {
```

```
37                      if(relation.equals(self.eContainer())){
38                       bol = true;
39                      }else{
40                      bol = false;
41                      }
42                }
43                return bol;
44          }
45           message: '(RFEAT04) The source relation must be a children of target relation.'
46  }
47      }
```

Listing D.10 – Validações EVL para o construtor *Extract Associative Table*

```
1
2  context SplitTable{
3
4  constraint HasDistictColumn{
5
6   check: self.distinctColumns.isDefined()
7
8   message:'An Distinct Column Must be Defined for '+self.eClass().name
9
10 }
11 constraint HasName{
12    check: self.targetTableName.isDefined()
13
14         message: 'You have to define a target table name on '+self.eClass().name+'
               refactoring element.'
15 }
16 constraint HasColumn{
17
18  check: self.columns.isDefined()
19
20  message:'At least one column bust be Defined for '+self.eClass().name
21
22 }}
```

Listing D.11 – Validações EVL para o construtor *Split Table*