Automação da validação de *scripts* de banco de dados com base em padrões de nomenclaturas

Luiz Guilherme Rodrigues, Evaldo de Oliveira da Silva

Curso de Bacharelado em Sistemas de Informação – Centro de Ensino Superior de Juiz de Fora (CESJF) – Campus Academia
360100-000 – Juiz de Fora – MG– Brasil

{luizrodriguess.dev, evaldo.oliveira}@gmail.com

Abstract.*. This article is a study about the importance of database object names to design a maintainable code. It also investigates the relevance of this issue for organizations, analyzes the patterns they use for naming their databases objects, the feasibility to create a set of algorithms to automatically validate patterns names of database objects and words, making it a part of continuous integration process.*

Resumo: O artigo tem como objetivo estudar a importância da validação de nomenclaturas para objetos de bancos de dados relacionais (nome dado às tabelas, restrições e colunas), levantar a relevância deste tema para as organizações, analisar o processo atual empregado pelas mesmas no que tange a padronização de banco de dados, para que então seja possível confeccionar uma ferramenta conjunta de algoritmos a serem utilizados para automatizar o processo de validação das nomenclaturas dos objetos, de forma a estudar a possibilidade de que a validação, aliada de um vocabulário controlado de siglas, seja automatizada e incluída no processo de entrega contínua das organizações.

1. Introdução

De acordo com Delfmann, Herwig e Lis (2009), a modelagem conceitual de banco de dados é um procedimento estabelecido dentro de um processo de engenharia do conhecimento na construção de sistemas de informação. O conhecimento sobre os termos que estão presentes no modelo conceitual deve ser considerado como uma prévia para a criação de esquemas conceituais. Com isso, torna-se possível oferecer os significados das entidades modeladas de acordo com sintaxes e formatos pré-estabelecidos.

A prática de utilização dos vocabulários controlados pode ser uma atividade que facilita a leitura de modelo de dados, permitindo os nomes dos objetos representados estejam relacionados com terminologias que possam ser reutilizadas para nomear outros elementos e modelos de dados distintos. Garantir esta prática na modelagem de dados é algo desafiador para equipes de Tecnologia da Informação (TI), especialmente quando os modelos são desenvolvidos por profissionais diferentes.

Estudos apontam que os modelos conceituais podem variar muito, particularmente a respeito dos nomes dos elementos modelados. Delfmann, Herwig e Lis (2009) abordam o uso de padrões de nomenclaturas, os quais combinam sinônimos e estruturas de frases com base na linguística, a fim de orientar os projetistas de banco de dados na definição dos significados dos elementos modelados a partir de um repositório compartilhado de vocabulários, a fim de representar de forma unificada o conhecimento empresarial.

Abordagens mais simples são adotadas e visam somente a utilização dos padrões de nomenclatura para organizar os nomes dos objetos de um banco de dados, determinando as abreviaturas, a lista de vocabulário de prefixos que irão descrevê-los (BRITISH COLUMBIA, 2017). É importante ressaltar que estes objetos podem ser: tabelas, índices, visões (*views*), chaves primárias, chaves estrangeiras, ou ainda quaisquer outros que devam ter os seus nomes e propriedades criados seguindo nomenclaturas definidas pela própria equipe de TI. Por exemplo, uma tabela que armazena os clientes em débito, poderia ser criada com o nome "TB_CLIE_DEBT", onde TB seria a designação de tabela, e os termos CLIE e DEBT seriam termos existentes em um vocabulário controlado a fim de conceituar os termos cliente e débito, respectivamente. Com a utilização de padrões de nomenclaturas é possível gerir os termos utilizados nas estruturas de dados, facilitando a leitura e recuperação dos objetos com base nas sintaxes de padronização.

Desta forma, é recomendada a criação de documentos que descrevam como os elementos em um modelo de banco de dados devam ter seus nomes estruturados e, além disso, as equipes de TI devem manter profissionais que estabeleçam a validação de qualquer modelo de dados que será criado em um SGBD (Sistema Gerenciador de Banco de Dados). No entanto, dependendo do tamanho e complexidade dos modelos de dados, a atividade de validação pode ser prejudicada e, consequentemente, gerar problemas futuros de interpretação e recuperação de dados em uma estrutura com termos redundantes e com ausência de padrões.

Neste contexto, este artigo apresenta como proposta a definição de um padrão de nomenclaturas para atribuição dos nomes dos objetos em um banco de dados. Além disso, aborda uma solução de software para ser utilizada como validador de *scripts* para criação de modelos físicos de banco de dados.

É importante destacar que este trabalho é resultado dos estudos realizados pelo Grupo de Estudos sobre Gestão de Conhecimento em Bases de Dados Corporativas (GECON), no qual participam alunos dos cursos de Engenharia de Software e Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora (CESJF),

O restante do artigo segue organizado em seções a serem definidas. A Seção 2 descreve os conceitos que definem demais concepções sobre padrões de nomenclaturas e as tecnologias utilizadas para o desenvolvimento da proposta deste trabalho. A Seção 3 descreve a proposta de um padrão de nomenclaturas com base nas experiências profissionais vividas pelos autores de trabalho e também fundamentado na literatura apresentada na Seção 2. A Seção 4 aborda o protótipo desenvolvido como proposta de validação de *scripts* de banco de dados. Finalmente, a Seção 5 apresenta as considerações finais e trabalhos futuros.

2. Referencial Teórico

Nesta seção serão descritos os conceitos que norteiam este trabalho, e que fundamentam as técnicas, procedimentos e implementação da solução proposta.

2.1. Geração de *scripts* para criação de banco de dados

Muitos procedimentos são utilizados para criação de *scripts* em sistemas de informação. É possível encontrar uma vasta literatura técnica que descreve mecanismos de geração e documentos destes a partir de modelos de banco de dados.

A modelagem de dados trata-se de uma das etapas do desenvolvimento de uma aplicação e descreve a forma na qual os dados serão armazenados, como funcionará, a lógica e auxilia na confecção da aplicação.

No contexto de projeto de banco de dados é possível definir três tipos de modelos que podem ser criados: o modelo conceitual, que é responsável por representar a forma na qual o usuário visualiza as entidades e/ou elementos do mundo real desconsiderando características físicas de um SGBD específico. Já o modelo lógico define a arquitetura na qual será criado o banco de dados, normalmente são criados os bancos de dados relacionais. Também neste modelo é feita a normalização geralmente até a 3ª forma a qual é elaborada para tornar o armazenamento dos dados mais eficiente, eliminando as redundâncias e inconsistências. Por fim, o modelo Físico que especifica qual SGBD onde os dados serão armazenados e é construído baseando-se no modelo lógico (ELMASRI; NAVATHE, 2006).

Os *scripts* são arquivos adicionais que contêm instruções para execução em servidores de banco de dados e servidores de aplicação, ou ainda para executar um conjunto de instruções pré-estabelecidas visando automatizar uma sequência de tarefas (MSDN, 2017; POLIZZI, 2008).

Polizzi (2008) propõe que um sistema de geração de *scripts* deve incluir um dicionário de dados adaptado para descrever dados da estrutura de uma tabela de banco de dados. Um SGBD deve executar um gerador de *script*, que inclui uma pluralidade de modelos definidos pelo usuário. Polizzi também descreve que a ativação do gerador de *script* deve utilizar rotinas internas no SGBD para extrair dados da estrutura da tabela, a fim de gerar *scripts* para consulta de dados.

Gillespie e Powers (1998) propõe um gerador de linguagem para definição de banco de dados que pode ler a entrada do documento de modelo de banco de dados existente em um formato de arquivo estruturado, e utiliza essa informação para criar *scripts* de acordo com a definição de banco de dados de um SGBD específico.

A geração de *scripts* de banco de dados também pode ser realizada diretamente a partir de modelos de banco de dados. De forma semelhante à proposta de Gillespie e Powers, o DBDesigner é uma aplicação de uso livre e disponível na internet, e que possibilita projetistas de banco de dados criar os modelos e gerar *scripts* para um banco de dados específico (DBDESIGNER, 2017).

O CA ERwin Data Modeler, normalmente referido apenas como ERWin, é uma ferramenta de software utilizada para a modelagem de banco de dados, sendo considerada uma ferramenta CASE (*Computer-Aided Software Engineering*). Através do ERWin, o desenvolvedor de um sistema de informação pode especificar os dados envolvidos, as suas relações e os requisitos de análise. A ferramenta permite, depois, a criação das bases de dados, bem como o processo inverso utilizando o conceito de engenharia reversa (CA TECHNOLOGIES, 2017).

A Figura 1 apresenta um exemplo de *script* para criação de banco de dados gerado pela ferramenta ERWin.

```
createtable EMPLOYEE
(
      employee_first_namevarchar(20),
      employee_addressvarchar(20),
      employee_phoneinteger,
       employee_address_2 varchar(20),
      employee_numbervarchar(20),
      soc_sec_numberinteger
      hire_datedatetime,
      salaryinteger,
      emailvarchar(20),
      store_numberinteger,
       supervisor varchar(20)
)
go
altertable EMPLOYEE
addforeignkey (store_number) references STORE (store_number)
on delete NO ACTION
go
```

Figura 1. Exemplo de *script* para criação de banco de dados (CA TECHNOLOGIES, 2017).

2.2. Padrões de Nomenclaturas

O objetivo de um padrão de nomenclaturas é prover informações para formalizar a nomenclatura de objetos de banco de dados, bem como apresentar regras para sua utilização. Evitando assim o hábito de existir diferentes nomenclaturas dentro de uma mesma aplicação (CELEPAR, 2009). De acordo com a metodologia de desenvolvimento de software do CELEPAR (Companhia de Informática do Paraná) um documento que descrito um padrão de nomenclaturas deve apresenta os objetos de banco de dados com três itens, tais como: sintaxe, regras e exemplo, visando facilitar o entendimento do desenvolvedor.

A norma ISO / IEC 11179-5: 2005 fornece instruções para nomear e identificar os seguintes itens administrados: conceito de elemento de dados. A norma também estabelece que os elementos de dados a serem armazenados devem seguir uma estruturação e identificação. A identificação é definida de forma restrita para abranger apenas os meios para estabelecer uma identificação única desses itens administrados dentro de um registro. Além disso, a norma descreve também que deve existir uma padronização que inclui princípios e regras pelas quais as convenções de nomenclatura podem ser desenvolvidas (ou chamadas de *namingconventions*).

O Datasus (Departamento de Informática do SUS) também possui um padrão de nomenclaturas utilizado como metodologia da administração de dados. O padrão proposto pelo Datasus segue a norma ISO / IEC 11179-5:2005. A regras implementadas no padrão do Datasus se encontram resumidas abaixo, e descrevem a padronização dos seguintes objetos de exemplo (DATASUS, 2017):

- [NOME DA TABELA]: o nome da tabela não ter conter o prefixo, a menos que seja uma tabela com prefixo RL, TL_ ou AU_; não deve conter o caractere separador _.
- [NOME DA COLUNA]: não deve conter o caractere separador _, sendo que este caractere deverá conter quando for utilizada mais de uma coluna.

- [NOME FK]: Restrição de integridade que determina que uma coluna ou um conjunto de colunas que possuem valores em outras tabelas. Relativa a uma referência ou a um relacionamento. O padrão estabelecido para chave estrangeira é visto da seguinte forma:
    - 1) Existe um relacionamento entre as duas tabelas:
      FK_+[NOME DA TABELA PAI]+_+ [NOME DA TABELA FILHO]
    - 2) Existe mais de um relacionamento entre as duas tabelas:
      FK_+[NOME DA TABELA PAI]+_+ [NOME DA TABELA FILHO]+[NOME FK] , neste caso o nome da FK deve ser significativo para o negócio ao qual pertence
    - 3-Formação quando existir relacionamento com uma chave candidata da tabela pai (UK)
    - FK_+[NOME DA TABELA PAI]+[NOME SIGNIFICATIVO DO CAMPO NO PAI]+_ +[NOME DA TABELA FILHO]+[NOME SIGNIFICATIVO DO CAMPO NO FILHO]
- [NOME PK]: Identifica de forma única uma linha de uma tabela. O padrão estabelecido para chave primária é visto da seguinte forma:
    - Tabela: TB_MUNICIPIO
        - Primary Key: PK_MUNICIPIO
    - Tabela: TB_TIPO_ENTIDADE
        - Primary Key: PK_TIPOENTIDADE
    - Tabela: RL_USUARIO_UF
        - Primary Key: PK_RLUSUARIO_UF

2.3. Ferramentas de software para automação de padrões de nomenclaturas

Pereira et.al. (2013) discute a importância da verificação dos modelos de dados com os padrões definidos nas organizações, para assegurar que todas estruturas de banco de dados estejam com os padrões adotados, por meio de um trabalho de identificação e processos de validação de modelo de dados com as normas definidas.

O foco do trabalho proposto por Pereira et. (2013) é propor uma solução de software onde é possível realizar o cadastro de padrões e palavras reservadas, que juntos formarão *templates*, que serão utilizados pelos administradores de banco de dados no auxílio da validação do padrão de nomenclatura dos modelos, apontando no script as falhas quando encontrarem, ficando visíveis os pontos a serem atacados, evitando retrabalhos futuros.

Schober et.al. (2011) propõe uma ferramenta para realizar a verificação de convenções de nomes (ou *naming conventions*) utilizados para construção de modelos de ontologias. Este trabalho apresenta também, como implementação de ferramenta de software para automação da proposta, a criação de um *plug-in* no software Protégé, uma ferramenta para modelagem de ontologia. O *plug-in* chamado de *OntoCheck*, permite realizar teste nos modelos de ontologias para a geração de documentos em OWL (*Ontology Web Language*). Em particular, o *OntoCheck* ajuda na validação de modelos de ontologias com base em convenções de nomes e metadados configurados previamente pelas equipes de modelagem.

Checkstyle é uma ferramenta de desenvolvimento para ajudar os programadores a escrever código Java com base em um padrão de codificação. Este software automatiza o processo de verificação do código Java para poupar desenvolvedores da tarefa de verificação de códigos. Isso o torna ideal para projetos que desejam impor um padrão de nomenclaturas (CHECKSTYLE, 2017).

Buytaert (2014) propõe a ferramenta *PL/Scope* compilada para coletar dados sobre identificadores em código fonte PL/SQL no tempo de compilação da unidade de programa. Esta ferramenta também disponibiliza as visualizações do dicionário de dados, a partir de dados coletados que incluem informações sobre tipos de identificadores, usos (declaração, definição, referência, chamada, atribuição) e a localização de cada uso no código-fonte.

A próxima seção apresenta uma proposta de padrão de nomenclatura para ser utilizado como fundamentação para a criação da ferramenta proposta neste trabalho.

3. Proposta de um padrão de nomenclaturas para modelos de banco de dados

Com base na experiência vivida em organizações distintas, foi constatado um processo maduro estabelecido com relação ao versionamento do banco de dados. Em situações reais, para um novo sistema, um banco de dados inicial é criado. Ao fim da demanda inicial, quando o software já possuí alguma funcionalidade, o mesmo começa a ser distribuído em ambientes internos de forma a ser testado e validado junto a equipes de diferentes áreas (equipe de qualidade, engenharia de requisitos, usuários, etc.). Junto a esta primeira entrega (que normalmente faz parte de um processo incremental de entregas) é versionada a aplicação, o *script* responsável pela geração do banco de dados no momento em que a

mesma será implantada. Tal *script* pode se originar da junção dos comandos SQLs usados pelos desenvolvedores na modelagem do banco ou, mais comumente, pela engenharia reversa aplicada ao banco de dados para que seja gerado o script do mesmo.

A partir deste ponto, *scripts* de banco de dados acompanham a aplicação ao longo de todo o seu ciclo de vida (considerando que a aplicação morre ao ser declarado o fim do desenvolvimento e continuidade de suas versões, bem como, fim da equipe de sustentação responsável pela mesma). A cada demanda implementada, sempre que necessário modificar ou incrementar o banco de dados, novos arquivos de script são gerados e/ou alterados.

Com base na premissa de que arquivos no formato .SQL trafegam em repositórios de código, será definida uma ferramenta para validar se scripts seguem boas práticas e padrões empiricamente conhecidos, descritos nas seções anteriores deste artigo. Tal validação atualmente é feita de forma manual nas organizações e seguindo padrões estabelecidos de forma análoga para cada organização.

A partir desta discussão, esta seção irá demonstrar como deve estar estruturado um documento que detalha padrões de nomenclaturas para definição dos nomes de objetos em banco de dados, com base na fundamentação teórica vista na Seção 2.2. Além disso, o trabalho desenvolvido nesta seção é resultado da utilização de padrões de nomenclaturas na própria aplicação proposta neste trabalho e também em outros trabalhos de conclusão de curso no Centro de Ensino Superior de Juiz de Fora (CESJF).

Porém, visando resumir a estrutura da documentação apresentada, esta seção irá apresentar apenas uma proposta de padrão de nomenclaturas para os seguintes objetos de banco de dados: tabelas, atributos, chaves primárias e chaves estrangeiras. O padrão será descrito a partir da Seção 3.1.

3.1. Tabelas

As tabelas são utilizadas como objetos representativos de entidades únicas no banco de dados, definidos da seguinte forma: <prefixo>_<sigla_aplicação>_<finalidade>, onde:

<prefixo> : Vide identificador abaixo.
<sigla_aplicação> : Sigla da aplicação ou schema com 4 caracteres
<finalidade> : Finalidade a que se propõe a tabela com no máximo 22 caracteres.

As regras, a serem seguidas para nomear as tabelas, são:

- O nome fornecido à tabela deverá ter no máximo 30 caracteres, com base na seguinte estruturação:
  - T_ - Identificador obrigatório da tabela.
  - T_<sigla_aplicação>_<finalidade>

Este formato deve ser utilizado para nomes de tabelas representativas de entidades únicas, ou seja, tabelas não derivadas de entidades que representem relacionamentos “n:m” (muitos-para-muitos), como “t_sist_anexo”. Caso o

nome da tabela ultrapasse o limite de caracteres estabelecido, deve-se utilizar um mnemônico que seja o mais sugestivo possível.

- T_<sigla_aplicação>_<tabela-pai>_<tabela-filho>

Este formato deve ser utilizado para nomes de tabelas derivadas de entidades que representem relacionamentos "n:m" (muitos-para-muitos), como "t_sist_requisicao_anexo" que é fruto do relacionamento das tabelas "t_sist_requisicao" caracteres estabelecidos, deve-se utilizar um mnemônico somente na tabela-filho, mantendo o nome completo da tabela-pai sempre que possível.

- Utilizar palavras em minúsculo e no singular.
- Utilizar a sigla resumida da aplicação ou schema em 4 caracteres.
- Utilizar palavras separadas por hífen ("_").
- Não utilizar caracteres especiais (ç ~ ^ # ! @ $ % ¨ & * ( ) [ ] { } ? | , etc).
- Não utilizar espaços em branco.
- Preferencialmente utilizar no máximo 3 palavras, separadas por hífens ("_").
- O nome das tabelas deve retratar dentro do possível a entidade representada.

As tabelas cujo objetivo é o armazenamento de dados temporários provenientes de operações de "spool", a partir de descarga de relatórios, deverão obrigatoriamente armazenar de forma temporária registros de erros ocorridos na aplicação Sistemaço (sigla "sist"). Estes registros correspondem a informações temporárias.

3.2. Convenção dos nomes de colunas

As convenções dos nomes de colunas podem ser feitas por meio do seguinte formato: <prefixo>_<nome_coluna>, onde:

<prefixo> :Vide tabela de prefixos a seguir.

<nome_coluna> :Nome da coluna, vide regras.

Nomes complementados com o sufixo "Temp", seguindo o seguinte formato: "T_<sigla_aplicação>_Temp". Por exemplo, a tabela "t_sist_erro_temp".

As regras a serem seguidas para nomear os nomes de colunas, são:

- Iniciar palavras com minúsculo e com prefixo.
- Utilizar palavras no singular.
- Utilizar palavras separadas com hífen ("_").
- Não utilizar caracteres especiais (ç ~ ^ # ! @ $ % ¨ & * ( ) [ ] { } ? | , etc).
- Não utilizar espaços em branco.
- Preferencialmente utilizar no máximo 3 palavras após o prefixo bem como as palavras existentes na tabela de nomenclatura de colunas.
- O tamanho do nome das colunas não deve ultrapassar 30 caracteres.
- Não repetir o significado do prefixo no nome da coluna. Ex.: Não definir o nome da coluna como "dt_data_cadastro", e sim, como "dt_cadastro".
- Os nomes das colunas que iniciam com "id" (chaves primárias) devem ser sempre concatenados com o nome da tabela. Ex.: A chave primária da tabela "Empregado" é definida como "id_empregado".

- Evitar repetir o nome da tabela no nome da coluna, exceto para as colunas que são identificadoras (primary key) da tabela.
- Em casos onde a concatenação para compor a chave primária se justifique, em termos de performance de acesso às tabelas, opta-se pela concatenação.

A Tabela 1 apresenta uma lista de prefixos que podem ser usados para padronizar nomes de colunas de tabelas.

Tabela 1. Prefixos utilizados para nomear colunas em tabelas de banco de dados. Fonte: Do Autor.

Prefixo	Tipo	Conteúdo	Exemplo	Narrativa
id_	Numeric	Identificação ou Identificador	id_empregado	Identificador único da tabela (chave primária); pode ser um o obejto "sequence"
cd_	numeric char varchar	Código	cd_evento	Código do Evento. Utilizado quando este for conhecido/visível pela aplicação.
dt_	Datetime	Datas	dt_nascimento	Data de nascimento
nu_	Numeric Varchar	Números ou Seqüência numérica iniciada com zero	no_cpf no_telefone	Números utilizados para cálculos.
ds_	char varchar	Descrição	ds_celula	Descrição da célula.
fl_	numeric char	Flag	fl_estado_civil	Flag
seq_	numeric	Seqüências	seq_cadastro	Utilizado para armazenamento de sequências que não são primary keys e deverão ser utilizadas somente quando necessário
nm_	varchar	Nomes	nm_candidato	Utilizado para armazenamento de nomes.
vl_	numeric	Valores	vl_salario	Utilizado para armazenamento de valores
qt_	numeric	Quantidades	qt_ligacoes	Utilizado para armazenamento de quantidades.
hr_	VARCHAR	Horas	hr_inicio	Hora de início

sg_	numeric VARCHAR	Siglas	sg_unidade	Sigla da unidade medida
pc_	numeric	Percentual	pc_aumento	Percentagem de aumento
dc_	numeric	Duração	dc_atendimento	Duração em tempo
tp_	char numeric	tipo	tp_campanha	Utilizado para armazenamento de tipos, ou seja, identificadores que retratam um conjunto de características conhecidas; normalmente possuem check constraints associadas

3.3. Padrões de nomes restrições para chave primária e chave estrangeira

As restrições (*constraints*) deverão ser definidas em um script isolado (separado do script de criação da tabela respectiva), com exceção das restrições nulas (chaves estrangeira nulas) que deverão ser definidas no mesmo bloco de criação de tabelas.

As *constraints* deverão ser agrupadas no bloco de criação na seguinte ordem:

PK - Primary Keys (chave primária)
UK - Unique Keys (chave única)
FK - Foreign Keys (chave estrangeira)
CK - Check (chave de validação)

O formato é definido da seguinte forma: <prefixo>_<sigla_aplicação>_<nome_constraint>, onde:

<prefixo> :Vide relação de identificadores abaixo.
<sigla_aplicação> : Sigla da aplicação ou schema com 4 caracteres.
<nome_constraint> : Nome da constraint, vide regras a seguir.

As regras para definição das restrições são vistas abaixo:

- O nome fornecido à constraint deverá ter no máximo 23 caracteres, com base na seguinte estruturação:

ID_ - Identificador obrigatório do tipo da constraint que pode ser:
PK - Constraint do tipo Chave Primária.
FK - Constraint do tipo Chave Estrangeira.

Para a criação de chave primária (ou *primary key*) estabelece-se o seguinte formato: PK_<sigla_aplicação>_<nome_tabela>, onde:

A chave primária da tabela “t_sist_empregado” deve ser definida como

"pk_sist_empregado".

Para a criação de chave estrangeira (*foreign key*) estabelece-se o seguinte formato FK_<sigla_aplicação>_<nome_tabela-pai>_<nome_tabela-filho><nn>

Em se tratando de uma única chave estrangeira da tabela "t_sist_avaliacao", que tem "t_sist_empregado" como a tabela-pai, a mesma seria definida como "fk_sist_empregado_avaliacao" e o valor de "nn" (seqüência numérica identificadora das foreign keys) seria nulo.

Em caso de se ter mais de um relacionamento entre estas tabelas, o valor de "nn" irá variar de acordo com a quantidade de relacionamentos:

fk_sist_empregado_avaliacao01
fk_sist_empregado_avaliacao02

Nas próximas seções será descrita a ferramenta confeccionada para o estudo da automatização da validação de scripts para banco de dados. Tal ferramenta, por meio de um conjunto de algoritmos, irá apontar inconsistências, segundo regras estabelecidas, para um script piloto criado especificamente para provar a viabilidade de automatizar o processo.

4. Automação da validação de scripts de banco de dados com base em padrões de nomenclaturas

4.1. Visão geral da solução proposta

O foco da ferramenta construída é a validação de arquivos de *script*, sendo uma ferramenta utilizada como prova de conceito dos assuntos tratados até o momento. A solução construída é um protótipo que utiliza a linguagem Java como plataforma de desenvolvimento, e nesta seção serão discutidos os aspectos de desenvolvimento para a automação da validação de *scripts* de banco de dados com base no padrão de nomenclatura proposto na seção 3.

4.2. Aspectos de implementação da automação de validação de scripts de banco de dados

Uma *view* baseada em *Servelet's* foi implementada com o propósito de processar os algoritmos do protótipo proposto, a fim de demonstrar o resultado para o usuário.

A ferramenta processa o arquivo de *script* em diversos blocos, a partir desta separação, cada bloco é tratado individualmente. Dentro destes blocos, são identificados os objetos de banco (colunas, tabelas, comentários e restrições). Por sua vez, os objetos de banco são validados individualmente.

Cada tipo de objeto de banco contém sua própria forma de validação e carrega consigo o conjunto de regras que o tornam válido. A validação das palavras é feita por meio de um vocabulário controlado que se encontra armazenado na base de dados utilizada para auxiliar na validação, juntamente com os prefixos para nomear os objetos de banco de dados. Esta base de dados foi construída como forma de parametrizar prefixos e vocabulários utilizados para formar os nomes dos objetos de banco de dados.

O protótipo acessa o banco a fim de verificar se o prefixo usado é valido para um determinado objeto e se as palavras que compõem o nome do objeto estão previamente definidas no vocabulário controlado. A Figura 2 apresenta um exemplo de blocos que são processados para validação de *scripts*.

```
ALTER TABLE one
ADD two_id INTEGER,
ADD CONSTRAINT FK_Teste FOREIGN KEY(two_id) REFERENCES two(id);

-- -----------------------------------------------------
-- Table `padrao_documento`.`T_SPDC_OBJT`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `padrao_documento`.`T_SPDC_OBJT` ;

SHOW WARNINGS;

CREATE TABLE IF NOT EXISTS `padrao_documento`.`T_SPDC_OBJT` (
  `ID_OBJT` INT NOT NULL COMMENT 'Identificador único do objeto inserido no sistema de padronizao de documentos.',
  `ID_TIPO_OBJT` INT NOT NULL,
  `ID_PRFX` INT NOT NULL COMMENT 'Identificador único do prefixo que compoe o nome do objeto',
  `NM_OBJT` VARCHAR(50) NOT NULL COMMENT 'Nome do objeto de padronizacao de documentos.',
  `DC_OBJT` VARCHAR(1000) NULL COMMENT 'Descricao a respeito do objeto.',
  PRIMARY KEY (`ID_OBJT`),
  CONSTRAINT `fk_tipo_objt_01`
    FOREIGN KEY (`ID_TIPO_OBJT`)
    REFERENCES `padrao_documento`.`T_TIPO_OBJT` (`ID_TIPO_OBJT`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_prfx_objt_02`
    FOREIGN KEY (`ID_PRFX`)
    REFERENCES `padrao_documento`.`T_SPDC_PRFX` (`ID_PRFX`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION);
ENGINE = InnoDB;

SHOW WARNINGS;
CREATE INDEX `IX_SPDC_OBJT01` ON `padrao_documento`.`T_SPDC_OBJT` (`ID_TIPO_OBJT` ASC);

SHOW WARNINGS;
CREATE INDEX `FK_PRFX_OBJT_01` ON `padrao_documento`.`T_SPDC_OBJT` (`ID_PRFX` ASC);
```

Figura 2. Trechos de blocos que são validados pelo protótipo proposto. Fonte: Do Autor.

4.3. Limitações das validações

A respeito da ferramenta em sua forma atual, por ter como objetivo neste momento apenas a prova de conceito, são conhecidas as seguintes limitações quanto a seu funcionamento, que poderão pontualmente ser evoluídas em trabalhos futuros:

- Comentários não devem conter “,” (vírgula).
- Chaves primárias e estrangeiras devem ser definidas na operação ALTER TABLE.
- A criação da tabela deve respeitar o formado [schema].[nome da tabela].
- O separador de palavras no nome do objeto de banco foi definido como “_” (*underscore*).
- Comentários com palavras reservadas podem causar comportamento inesperado na ferramenta.
- O comentário dentro do create table é delimitado por “’” (aspas simples).
- O script deve estar em caixa alta.

Desta forma, foi delimitado como escopo para o protótipo os seguintes itens:

- Validação dos nomes das tabelas no momento de sua criação
- Validação dos nomes identificados para as chaves primária e chaves estrangeiras.

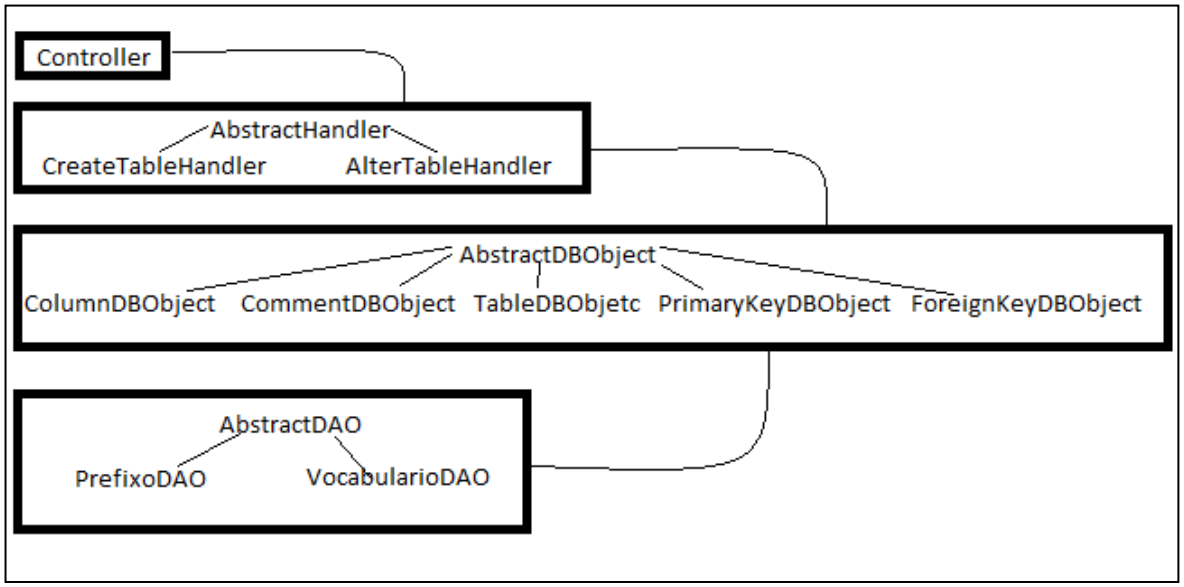
- Validação dos nomes identificados para colunas das tabelas.
- Validação dos comentários, se os mesmos possuem conteúdo.

O sistema não valida em sua forma atual, por exemplo, comentários em um Alter Table.

4.3 Arquitetura e funcionamento da ferramenta

Apesar do teor experimental, a ferramenta foi construída com padrões de desenvolvimento de alto nível de forma a garantir a manutenibilidade do software bem como a escalabilidade do projeto.

A Figura 3 apresenta um esquema do núcleo principal da aplicação seguido de explanação sobre as classes de forma a elucidar simultaneamente o funcionamento do código fonte e a metodologia adotada. Em seguida, são feitas considerações a respeito da implementação.



Dentro do pacote controller, existe atualmente uma única classe. Esta classe é responsável por receber o arquivo de script e implementa o padrão Singleton, pois não se faz nescessário que a mesma possua mais de uma instância. O controller é quem varre o script em busca dos blocos contendo os comandos. Atualmente o sistema trata apenas os comandos <Alter Table> e <Create Table>, por meio dos métodos privados implementados para cada um destes blocos de comandos:

```
private void findCreateTable(String dbScript);
private void findAlterTable(String dbScript);
```

Cada um destes métodos implementa uma lógica própria para procurar por palavras reservadas da linguagem SQL responsáveis por iniciar os blocos de comandos que são de sua responsabilidade. Encontrado um bloco, um objeto do tipo Handler é instanciado, passando-se para o construtor do Handler o bloco identificado. O Handler por sua vez é armazenado em uma Lista.

Um exemplo a respeito do método responsável por encontrar blocos de comandos do tipo Alter Table é apresentada, conforme visto na Figura 4. O método recebe uma String contendo todo o script. A partir desta string ele busca sempre a posição da próxima String “ALTER TABLE” e junto dela a posição do caractere “;” que indica o fim do comando. Delimitadas as posições de início e fim do comando dentro da String, um objeto do tipo AlterTableHandler é instanciado com o texto correspondente ao comando. A partir desta string, ele busca sempre a posição da próxima String “ALTER TABLE” e junto dela a posição do caractere “;” que indica o fim do comando.

```
private void findAlterTable(String dbScript){
    int lastIndexedIndex = 0;
    while (lastIndexedIndex != -1){
        int lastDbCommantIndex = dbScript.indexOf("ALTER TABLE", lastIndexedIndex);
        int lastSemicolom = dbScript.indexOf(";", lastDbCommantIndex);
        if (lastDbCommantIndex == -1){
            break;
        }
        AlterTableHandler alterHandler = new AlterTableHandler(
                dbScript.substring(lastDbCommantIndex, lastSemicolom));
        handlers.add(alterHandler);
        lastIndexedIndex = lastSemicolom;
    }
}
```

Figura 4. Método findAlterTable. Fonte: Do Autor.

Delimitadas as posições de início e fim do comando dentro da String, um objeto do tipo AlterTableHandler é instanciado com o texto correspondente ao comando. O método recebe uma String contendo todo o script. A partir desta string ele busca sempre a posição da próxima String “ALTER TABLE” e junto dela a posição do caractere “;” que indica o fim do comando. Delimitadas as posições de início e fim do comando dentro da String, um objeto do tipo AlterTableHandler é instanciado com o texto correspondente ao comando.

A operação se repete, iniciando-se sempre do ponto onde o último comando indexado acaba (para evitar um loop infinito) até que o resultado da busca pela String de palavras reservadas que iniciam o comando seja -1 (o que indica que a String não foi mais encontrada).

O pacote Handler é composto das classes responsáveis por tratar os comandos encontrados. Cada comando a ser tratado possui sua própria classe (AlterTableHandler.java por exemplo). Esta classe possui uma String que armazena o comando SQL responsável pela operação e também possui a lógica responsável por identificar os objetos de banco (colunas, tabelas, chaves primárias e chaves estrangeiras) passíveis de validação.

As classes do pacote Handler entendem de uma abstrata (AbstractHandler) que contém métodos comuns a todos os Handlers. Além disto, implementam uma interface que as obriga a escrever o método process(), que deve ser escrito contendo a lógica mencionada por instanciar os objetos que representam os objetos de banco de dados e retornar todos os objetos inválidos. Um handler possui um conjunto de métodos privados chamados dentro do método process() responsáveis por encontrar cada tipo de objeto de banco e os instanciar.

O exemplo da implementação do método privado, chamado pelo método process() que foi escrito para a classe CreateTableHandler, pode ser visto na Figura 5. Este método é responsável por instanciar objetos do tipo ColumnDatabaseObject para cada coluna encontrada em um bloco contendo um Create Table.

```
private void findColumns(){
    int startIndex = textBlock.indexOf("(");
    int endIndex = textBlock.indexOf(")");
    String[] attributes = textBlock.substring(startIndex,endIndex).split(",");
    for(String attribute : attributes){
        String s;
        attribute = attribute.replace("(", "");
        attribute = attribute.trim();
        s=removeUnededChars(attribute.split(" ")[0]);
        dbObjects.add(new ColumnDatabaseObject(s));
    }
}
```

Figura 5. Exemplo do método findcolumns() usado para processar a identificação de colunas de uma tabela. Fonte: DoAutor.

O método findcolumns() localiza os índices de textos a serem localizados, responsáveis por delimitar as colunas, em seguida, cria um Array de Strings contendo o código de cada coluna. Para cada coluna encontrada, ele implementa uma lógica que identifica o nome da coluna, remove caracteres não pertinentes à validação por meio de um método definido na superclasse e, finalmente, instância um objeto do tipo ColumnDatabaseObject contendo o nome do objeto a ser validado.

As classes do tipo Database Object são responsáveis por guardar o nome do objeto e são forçadas pela interface a implementarem a regra de validação para o tipo de objeto que representam. As classes Database Object entendem de uma classe abstrata que contém lógicas genéricas a serem usadas para identificar o prefixo de um nome e as palavras que compõem um nome, validar o prefixo com base no tipo de objeto e validar se cada palavra que compõe o nome existe no vocabulário controlado. A classe abstrata também possuí uma String que guarda a última palavra inválida encontrada no objeto, validar. Caso seja a primeira palavra encontra ela é identificada como o prefixo. A Figura 6 apresenta o método de validação de objetos.

```
@Override
public Boolean isValid() {
    String[] words = getObjectName().split(getSeparator());
    int i =0;
    for (String word : words){
        if (i==0){
           if (!validatePrefix(word)){
                return false;
            }
        }else{
            if (!validateWord(word)){
                return false;
            }
        }
        i++;
    }
    return true;
}
```

Figura 6. Exemplo do método isValid() para validar nomes. Fonte: DoAutor.

O método que valida cada palavra, em sua implementação genérica, busca na tabela de siglas do vocabulário controlado se a palavra em questão está cadastrada. Toda conexão com banco de dados e métodos que acessam banco está implementadas em classes do tipo Data Access Object (ou DAO), evitando qualquer acoplamento entre as entidades e classes ou funções referentes à conexão e busca no banco de dados. A Figura 7 apresenta o método validateWord que permite a validação de vocabulários na base de dados. O método validateWord recorre à classe VocabulárioDAO para responder se a palavra está cadastrada no vocabulário controlado.

```
protected Boolean validateWord(String word) {
    VocabularioDAO vocabularioDAO = new VocabularioDAO();
    Boolean isAllowed = vocabularioDAO.IsAllowedWord(word);
    if (!isAllowed){
        wrongWord = word;
    }
    return isAllowed;
}
```

Figura 7. Exemplo do método validateWord () para validar nomes inseridos no banco de dados. Fonte: DoAutor.

Cada classe do tipo Database Object pode, por sua vez, sobrescrever as regras definidas na superclasse mencionada, flexibilizando assim a implementação e atendendo às exigências específicas para a validação de cada tipo de objeto. Um exemplo, seria a classe ColumnDatabaseObject que sobrescreve o método para a validação de prefixo, para que o mesmo seja validado apenas como uma palavra se o prefixo estiver desabilitado para colunas (Figura 8).

```
@Override
protected boolean validatePrefix(String word) {
    return validateWord(word);
}
```

Figura 8. Exemplo do método validatePrefix(). Fonte: DoAutor.

O método validatePrefix() sobrescreve a validação de prefixo, para que o mesmo seja usado apenas como mais uma palavra que compõe o nome da coluna.

Após todos os blocos de código SQL estarem mapeados dentro de classes do tipo Handler e cada classe Handler mapear sua lista de objetos do tipo Database Object, uma reação em cadeia é disparada pelo controller, executando-se em cada Handler um método responsável por devolver todos os objetos inválidos dentre os objetos mapeados por cada Handler. Estes objetos são identificados por um método definido na superclasse AbstractHandler que percorre a lista de objetos, executando a validação de cada um e adicionando as instâncias inválidas a uma lista de retorno, conforme apresentado na Figura 9.

```
protected List<AbstractDatabaseObject> findInvalidObjects(){
    List<AbstractDatabaseObject> invalidObjects = new ArrayList<>();
    for(DatabaseObject dbObject : dbObjects){
        if (!dbObject.isValid()){
            invalidObjects.add((AbstractDatabaseObject)dbObject);
        }
    }
    return invalidObjects;
}
```

Figura 9. Método para lista os objetos validados. Fonte: DoAutor.

5. Considerações Finais

A existência de padrões para nomenclatura de banco de dados é de suma importância para sistemas, por conferir facilidade para o entendimento do banco de dados do domínio a ser representado, por parte dos desenvolvedores. Verificar se o nome de uma tabela não é grande demais, se possui nomes coerentes e se segue um padrão de escrita são pontos que facilitam a manutenção do banco de dados e faz parte de um processo maior que diz respeito à qualidade, manutenibilidade e tempo de vida de uma solução.

As organizações já entendem a necessidade deste processo e estabelecem padrões a serem seguidos pelos desenvolvedores. Scripts de criação de banco são, inclusive, tema dos *Code Reviews* feitos por grande parte das organizações que implementam soluções de software tanto quanto atividade fim ou atividade meio.

Foi possível considerar por meio do estudo e da implementação da ferramenta que automatizar o processo de validação de scripts em relação à nomenclatura de objetos de banco é viável. A ferramenta foi capaz de apontar com precisão os nomes incoerentes com às regras definidas. O desenvolvimento da ferramenta também possibilitou dimensionar qual o grau de dificuldade e os entraves de automatizar o processo a nível de mercado, entregando uma ferramenta que seja consistente, capaz de analisar scripts seguindo as diversas convecções para comandos SQL existentes e parametrizável para atender às necessidades e padrões estabelecidos por cada empresa.

A ferramenta foi desenvolvida de forma a permitir a viabilidade de sua manutenção e a garantir sua escalabilidade. Em certo grau de maturidade, a implementação desta ferramenta abre possibilidade, fazer com que o processo de validação dos scripts de criação e alteração de banco de dados, mapeado nos repositórios da organização que em suma, é validado manualmente, seja parte do processo de integração contínua, tal como é feito por ferramentas para código fonte, uma vez estabelecidos os padrões esperados.

Como trabalho futuro, espera-se criar uma interface para permitir que os dados armazenados para serem utilizados como vocabulários e prefixos do padrão de nomenclaturas sejam mantidos por meio de uma aplicação.

Referências

BRITISH COLUMBIA. **Database design (physical) modeling standards andguidelines** (2017). Disponível em https://www2.gov.bc.ca/assets/gov/british-columbians-our-governments/services-policies-for-government/information-technology/standards/economy-

sector/database_design_physical_modelling_standards_and_guidelines.pdf. Acesso em 07 de novembro de 2017.

BUYTAERT, Nick. **Validating naming conventions in Oracle, 2014.**. Disponível em: https://apexplained.wordpress.com/2014/04/19/validating-naming-conventions-in-oracle/. Acesso em 09 de novembro de 2017.

CELEPAR. **Padrões para Nomenclatura de Banco de Dados. 2009**. Disponível em: http://www.documentador.pr.gov.br/documentador/. Acesso em 30 de Outubro de 2017.

CA TECHNOLOGIES. **CA Erwin Data Modeler**. Disponível em: http://erwin.com/bookshelf/9.7.00/Bookshelf_Files/PDF/Editing%20Forward%20Engineering%20Templates.pdf. . Acesso em: 03 novembro 2017.

CHECKSTYLE. **Overview**. Disponível em : http://checkstyle.sourceforge.net/config.html. Acesso em 10 de novembro de 2017.

DATASUS.**MAD – Metodologia de Administração de Dados**. Disponível em http://datasus.saude.gov.br/estrutura-mad/norma-mad-menu. Acesso em 31 de outubro de 2017.

DBDESIGNER. **DBDesigner Tool**. Disponível em http://fabforce.eu/dbdesigner4/. Acesso em 02 de Novembro de 2017

DELFMANN, Patrick; HERWIG, Sebastian; LIS, **Lukasz. Unifiedenterpriseknowledgerepresentationwith conceptual models-Capturingcorporatelanguage in namingconventions**. ICIS 2009 proceedings, p. 45, 2009.

ELMASRI, R.; NAVATHE. **Sistemas de Banco de Dados: Fundamentos e Aplicações**. Pearson Education, 2006.

GILLESPIE, Horace L.; POWERS, Margaret M. **Databasedefinitionlanguagegenerator**. U.S. Patent n. 5,732,262, 24 mar. 1998.

ISO/IEC 11179-5:2005. **Information technology -- Metadata registries (MDR)**. Disponível em : https://www.iso.org/standard/35347.html. Acesso em 08 de novembro de 2017.

MSDN. **Visão geral dos scripts de banco de dados**. Disponível em https://msdn.microsoft.com/pt-br/library/aa833429(v=vs.90).aspx. Acesso em 09 de novembro de 2017.

PEREIRA, Henrique F. et al. **IMPLEMENTAÇÃO DE UM VALIDADOR DE PADRÃO DE NOMENCLATURA DE SCRIPTS SQL PARA BANCO DE DADOS RELACIONAIS**. e-RAC, v. 3, n. 1, 2013.

POLIZZI, Nicholas P. **Script generator for automating system administrationoperations**. U.S. Patent n. 7,403,934, 22 jul. 2008.

SCHOBER, Daniel et al. **OntoCheck: verifying ontology naming conventions and metadata completeness in Protégé 4**. Journal of biomedical semantics, v. 3, n. 2, p. S4, 2012.