UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

ITELVINA SILVA DE OLIVEIRA

TESTE BASEADO EM DEFEITOS PARA AMBIENTES DE DATA WAREHOUSE

DISSERTAÇÃO

Curitiba

2015

ITELVINA SILVA DE OLIVEIRA

TESTE BASEADO EM DEFEITOS PARA AMBIENTES DE DATA WAREHOUSE

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do título de Mestre em Computação Aplicada.

Área de concentração: Engenharia de Sistemas Computacionais

Orientadora: Profa. Dra. Maria Claudia Figueiredo Pereira Emer
Co-orientador: Prof. Dr. Adolfo Gustavo Serra Seca Neto

Curitiba

2015

Dados Internacionais de Catalogação na Publicação

O48t 2015

Oliveira, Itelvina Silva de
Teste baseado em defeitos para ambientes de data whare-house / Itelvina Silva de Oliveira.-- 2015.
148 p. : il.; 30 cm

Texto em português, com resumo em inglês
Bibliografia: p. 135-139

1. Armazenamento de dados - Testes - Defeitos. 2. Data warehouse - Programa de computador. 3. Sistemas de suporte de decisão. 4. Informática - Dissertações. I. Emer, Maria Cláudia Figueiredo Pereira, orient. II. Seca Neto, Adolfo Gustavo Serra, coorient. III. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada. IV. Título.

CDD: Ed. 22 -- 621.39

Biblioteca Central da UTFPR, Câmpus Curitiba

ATA DE DEFESA DE DISSERTAÇÃO DE MESTRADO Nº 27

Aos 13 dias do mês de agosto de 2015 realizou-se na sala C-301 a sessão pública de Defesa da Dissertação de Mestrado intitulada "Teste Baseado em Defeitos para Ambientes de Data Warehouse", apresentada pela aluna **Itelvina Silva de Oliveira** como requisito parcial para a obtenção do título de Mestre em Computação Aplicada, na área de concentração "Engenharia de Sistemas Computacionais", linha de pesquisa "Engenharia de Software".

Constituição da Banca Examinadora:

Profª. Drª. Maria Cláudia F. Pereira Emer, UTFPR - CT (Presidente) _____

Profª Drª Sílvia Regina Vergílio, UFPR _____

Prof. Dr. Laudelino Cordeiro Bastos, UTFPR - CT _____

Profª. Drª. Nádia Puchalski Kozievitch, UTFPR - CT _____

Em conformidade com os regulamentos do Programa de Pós-Graduação em Computação aplicada e da Universidade Tecnológica Federal do Paraná, o trabalho apresentado foi considerado _____ (aprovado/reprovado) pela banca examinadora. No caso de aprovação, a mesma está condicionada ao cumprimento integral das exigências da banca examinadora, registradas no verso desta ata, da entrega da versão final da dissertação em conformidade com as normas da UTFPR e da entrega da documentação necessária à elaboração do diploma, em até _____ dias desta data.

Ciente (assinatura do aluno): _____

(para uso da coordenação)

A Coordenação do PPGCA/UTFPR declara que foram cumpridos todos os requisitos exigidos pelo programa para a obtenção do título de Mestre.

Curitiba PR, _____/_____/_____ _____

"A Ata de Defesa original está arquivada na Secretaria do PPGCA".

Dedico este trabalho aos meus queridos pais Altemir e Maria Antônia (in memorian) e ao meu querido esposo Ícaro.

AGRADECIMENTOS

A Deus, por me dá forças e ajudar a superar todas as dificuldades.

Ao meu esposo Ícaro, pelo amor, apoio, compreensão e força nesta jornada.

Aos meus pais, Altemir e Maria Antônia (*in memorian*) pelo amor, educação, ensinamento e incentivo constante aos meus estudos e busca pelo conhecimento.

Aos meus orientadores, Profa. Maria Claudia e Prof. Adolfo pela ajuda, amizade, ensinamentos e pela oportunidade.

Aos colegas do grupo de estudos em Engenharia de Software pelas discussões no início deste trabalho e à colaboração do bolsista de iniciação científica Gustavo Bandeira.

Aos professores do PPGCA, em especial Profa. Maria Claudia, Prof. Gustavo Lugo, Prof. Laudelino, Prof. Adolfo e Profa. Nádia pelo incentivo e colaboração com sugestões e críticas construtivas para a melhoria deste trabalho.

Ao Tribunal de Contas do Estado do Paraná, pelo apoio financeiro e colaboração em propiciar a realização desta pesquisa fornecendo ajuda e infraestrutura para realização dos experimentos, em especial ao Ernesto, à equipe do DIE, Daniel Montanher, William e Reginaldo Bitello.

“Todos os nossos sonhos podem se realizar, se tivermos a coragem de persegui-los.”

Walt Disney

RESUMO

OLIVEIRA, Itelvina Silva de. Teste Baseado em Defeitos para Ambientes de Data Warehouse. 148 f. Dissertação- Programa de Pós-Graduação em Computação Aplicada-PPGCA, Universidade Tecnológica Federal do Paraná - UTFPR. Curitiba, 2015.

As organizações necessitam gerenciar informações para obter a melhoria contínua dos seus processos de negócios e agregar conhecimento que ofereça suporte ao processo decisório. Estas informações, muitas vezes, são disponibilizadas por ambientes de *Data Warehouse* (DW), nos quais os dados são manipulados e transformados. A qualidade dos dados nesses ambientes é essencial para a correta tomada de decisão, tornando-se imprescindível a aplicação de testes. O objetivo deste trabalho é elaborar e validar a aplicação de uma abordagem de teste para DW com o emprego de critérios da técnica de teste baseado em defeitos. A aplicação da abordagem possibilitou testar três fases de desenvolvimento do DW, nas quais estão as Fontes de Dados, processo ETL (*Extraction, Transformation and Load*) e dados do DW. O critério de teste Análise de Mutantes foi aplicado ao processo ETL por meio de operadores de mutação SQL e a Análise de Instâncias de Dados Alternativas foi aplicada nas fontes de dados e nos dados do DW por meio de classes de defeito nos dados. Essas classes foram geradas por meio da análise e associação dos problemas de qualidade de dados nas fases de desenvolvimento do DW. Os resultados obtidos em estudos de caso permitiram a validação da aplicabilidade e eficácia da técnica de teste baseado em defeitos para ambientes de DW, possibilitando assim revelar quais defeitos podem ocorrer na geração do DW que poderiam prejudicar a qualidade dos dados armazenados nesses ambientes.

Palavras-chave: *Data Warehouse*, Teste Baseado em Defeitos, Análise de Mutantes, Análise de Instâncias de Dados Alternativas, Qualidade de Dados.

ABSTRACT

OLIVEIRA, Itelvina Silva de. Fault-Based Testing for Data Warehouse Environments. 148 f. Dissertação - Programa de Pós-Graduação em Computação Aplicada - PPGCA, Universidade Tecnológica Federal do Paraná - UTFPR. Curitiba, 2015.

Organizations need to manage information for a continuous improvement of its business processes and aggregate knowledge that help in the decision-making process. This information often is provided by Data Warehouse environments (DW), in which data are handled and processed. The quality of data in these environments is essential to make correct decisions, becoming it necessary the application of tests. The objective of this work is to develop and validate the implementation of a testing approach for DW using criteria of Fault-based Testing techniques. The application of the approach enabled tests in three phases of development of the DW, which are the data sources, ETL and DW data. The test criteria Mutation Analysis was applied to the ETL process (Extraction, Transformation and Load) through SQL mutation operators and the Alternative Data Instances Analysis was applied to the data sources and DW data through fault classes on the data. These classes were generated by analyzing and associating of data quality problems in the DW development stages. The results obtained through the case studies allowed assessment of the applicability and effectiveness of testing technique fault for DW environments, thus enabling to reveal faults, which may occur in the generation of DW that could harm the quality of the data stored in these environments.

Keywords: Data Warehouse, Fault-based Testing, Mutation Analysis, Instances Analysis of Alternatives Data, Data Quality.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

AIDA	Análise de Instâncias de Dados Alternativas
BD	Base de Dados
BDT	Base de Dados de Teste
BDP	Base de Dados de Produção
BI	*Business Intelligence*
DM	*Data Marts*
DMA	*Data Multiplication Algorithm*
DSA	*Data Staging Area*
DSG	*Dataset Generator*
DW	*Data Warehouse*
ETL	*Extraction, Transformation and Load*
MSG	*Mutant Schemata Generation*
NIST	*National Institute of Standards and Technology*
ODS	*Operational Data Store*
OLAP	*On-line Analytical Processing*
UI	Aplicação de Interface de Usuário
VV&T	Validação, Verificação e Teste de Software

SUMÁRIO

1. INTRODUÇÃO

1.1 CONTEXTO

As organizações necessitam consolidar os dados gerados por diversas fontes em seu ambiente de negócios e disponibilizar informações consistentes a fim de facilitar a tomada de decisão por seus usuários de negócios. As informações geradas por meio de sistemas transacionais ganham suporte ao processamento analítico por meio de tecnologias de *Data Warehouse* (DW), que oferecem às organizações uma maneira flexível e eficiente de obter as informações necessárias aos seus processos decisórios (AMARAL, 2003).

A geração de um *Data Warehouse* possui complexidade que exige critérios no processo de manipulação dos dados, pois erros em suas fases de desenvolvimento são refletidos nos dados, ocasionando análises errôneas que podem acarretar em estratégias erradas de tomada de decisão, danos financeiros e perdas de oportunidades de negócio para a organização (ELGAMAL; ELBASTAWISSY; GALAL-EDEEN, 2013). Nesse contexto, o teste surge como uma atividade crítica que requer um estudo aprofundado para o sucesso de projetos de *Data Warehouse*, pois os usuários precisam confiar na qualidade dos dados que acessam.

Segundo Delamaro, Maldonado e Jino (2007), teste de software é uma atividade dinâmica, cujo objetivo consiste em executar um programa ou modelo utilizando entradas específicas e verificar se seu comportamento está de acordo com o esperado; e quando a execução apresenta resultados não especificados significa que um erro ou defeito foi identificado.

O principal objetivo do teste de software é revelar a presença de defeitos no produto de software, e conforme define a Regra 10 de Myers, quanto mais cedo for descoberto e corrigido um defeito, menor será o seu custo para o projeto (MYERS, 2004). Portanto, o teste de software é uma atividade fundamental para aumentar o nível de confiança na correção de um software, que por si só não garante a qualidade do software, mas tem a função de ajudar a medir a qualidade do software.

Segundo Barbosa et al. (2000), técnicas de teste foram propostas na literatura, entre elas se podem citar as técnicas funcionais, as estruturais e as baseadas em defeito. A técnica de teste baseada em defeitos utiliza informações sobre os defeitos típicos do processo de desenvolvimento de software para derivar os requisitos de teste (DELAMARO et al., 2007). Um dos critérios dessa técnica é a análise de mutantes que considera dois pressupostos: a Hipótese do Programador Competente e o Efeito do Acoplamento (DEMILLO; LIPTON; SAYWARD, 1978).

O DW é uma coleção de dados orientados por assunto, integrados, não voláteis, variáveis com o tempo que servem para dar suporte ao processo de tomada de decisão tendo como foco a organização como um todo, englobando várias áreas por assunto, e consequentemente criando *Data Marts* que se relacionam entre si (INMON, 2012).

Dentre os passos de desenvolvimento do DW tem-se o ETL (*Extraction, Transformation and Load*), no qual as informações são extraídas do ambiente de negócio, por meio de sistemas legados e aplicações internas e as mesmas são carregadas em um repositório temporário (*Staging Area* - fonte de dados transiente) para fazer a limpeza e transformação desses dados, para que os mesmos sejam carregados e armazenados no *Data Warehouse*, possibilitando posteriormente a geração de *Data Marts*. O ETL e as ferramentas de limpeza de dados consomem um terço do orçamento em um projeto de *Data Warehouse*, e 80% do tempo de desenvolvimento de um *Data Warehouse* consiste no processo de ETL (KIMBALL; ROSS, 2013).

Segundo ElGamal (2015), a qualidade de dados em *Data Warehouses* tem sido um assunto em destaque entre pesquisadores, analistas, projetistas, desenvolvedores, testadores e usuários do *Data Warehouse*, que reconhecem que apenas a finalização do desenvolvimento de um *Data Warehouse* não implica em um projeto de DW com êxito, mas que entregar um produto com alta qualidade é o fator principal de sucesso, destacando que a qualidade de um *Data Warehouse* pode ser visualizada a partir das seguintes perspectivas: Qualidade das Fontes dos Dados, Qualidade dos Dados armazenados no DW, Qualidade de *Front-End* (saídas dos dados para visualização do usuário), Qualidade do Projeto do *Data Warehouse*, Qualidade do Processo de *Data Warehousing*, Qualidade da Entrega do Sistema.

Problemas de qualidade de dados podem ocorrer em diversas fases do desenvolvimento de um sistema de informações como o *Data Warehouse*, conforme descrito em Singh e Singh (2010). Desse modo, o presente trabalho aborda a atividade de teste nas fases de desenvolvimento de um *Data Warehouse*, envolvendo os aspectos

de Qualidade das Fontes de Dados e de Qualidade dos Dados armazenados no DW, assim como, as consultas SQL do processo ETL.

Em aplicações de bases de dados existem vários trabalhos na literatura que investigam a atividade de teste por meio de técnicas de teste diversas, porém a aplicação de teste baseado em defeitos em ambientes de *Data Warehouse* é uma nova área de pesquisa explorada, com o propósito de reduzir custos de desenvolvimento/manutenção e aumentar a confiabilidade nos dados contidos no *Data Warehouse,* e consequentemente, nas decisões tomadas com base nesses dados que serão projetadas em relatórios gerenciais.

Portanto, o presente trabalho justifica-se pela utilização de técnica de teste em um ambiente de *Data Warehouse* para prevenir defeitos, podendo auxiliar na descoberta de problemas no desenvolvimento desses ambientes, e possivelmente, reduzir custos com correções e contribuir com a qualidade das informações extraídas desses ambientes.

1.2 OBJETIVOS

O objeto de estudo deste trabalho é auxiliar na integridade e confiabilidade dos dados a serem extraídos de sistemas de bases transacionais e armazenados em um ambiente de *Data Warehouse*. O objetivo geral deste trabalho é desenvolver uma abordagem de teste em ambiente de *Data Warehouse* por meio da Técnica de Teste Baseado em Defeitos, no intuito de melhorar a integridade e a confiabilidade dos dados extraídos desses ambientes. Os objetivos específicos do presente trabalho são:

- Selecionar problemas de qualidade de dados, classes de defeito e operadores de mutação para aplicar nas fases de desenvolvimento de um *Data Warehouse*, por meio dos estudos de trabalhos da literatura*;*
- Propor uma abordagem de teste que envolva as principais fases de desenvolvimento de um ambiente de *Data Warehouse*;
- Implementar ferramenta para validar a proposta;
- Validar a abordagem para verificar a aplicabilidade e eficácia da proposta em um ambiente de *Data Warehouse,* por meio de execução da abordagem em estudos de caso;
- Avaliar os resultados obtidos com a realização dos estudos de caso.

1.3 ORGANIZAÇÃO DO TRABALHO

O trabalho está estruturado da seguinte forma: no Capítulo 2 são descritos os conceitos básicos relacionados aos assuntos do trabalho. No Capítulo 3 são descritos os trabalhos relacionados a Testes em *Data Warehouse*, assim como, teste envolvendo o critério de teste Análise de Mutantes a contextos SQL e em outros ambientes. Conceitos relacionados à Qualidade de Dados também são abordados, assim como problemas relacionados à Qualidade de Dados em ambientes de *Data Warehouses*. No Capítulo 4 é descrita a proposta da abordagem de teste em *Data Warehouse*, apresentando de que forma executou-se o processo de teste neste ambiente. No Capítulo 5, são apresentados os estudos de caso realizados aplicando a abordagem de teste baseado em defeitos em *Data Warehouse*, assim como os resultados e análise dos mesmos. No Capítulo 6 encontram-se as Conclusões e nos Apêndices A e B são descritas as ferramentas de Qualidade de Dados e Teste de Mutação SQL utilizadas nos estudos de caso.

2. CONCEITOS BÁSICOS

Neste capítulo os conceitos, técnicas e os critérios de teste de software são descritos, focando no critério de teste Análise de Mutantes que faz parte da Técnica de Teste Baseado em Defeitos. Em seguida, é apresentada a terminologia relativa aos ambientes de *Data Warehouses*, assim como os desafios para realização de testes nesses ambientes e os conceitos relacionados à Qualidade de Dados.

2.1 TESTE DE SOFTWARE

Segundo Delamaro, Maldonado e Jino (2007) a construção de um software não é uma tarefa simples. Pelo contrário pode se tornar bastante complexa, dependendo das características e dimensões do sistema a ser criado. Por isso, está sujeita a diversos tipos de problemas, a maioria deles causados por erro humano, que acabam resultando na obtenção de um produto diferente daquele que se esperava.

De acordo com Myers (2004), o principal objetivo do teste de software é revelar a presença de defeitos no produto de software e, portanto, um teste bem-sucedido é aquele que tem uma elevada probabilidade de revelar um erro ainda não descoberto por meio de bons casos de teste. Segundo Barbosa et al. (2000) tem-se observado que a própria atividade de projeto de casos de teste é bastante efetiva em evidenciar a presença de defeitos de software.

O teste é parte de um amplo processo de validação, verificação e teste (VV&T) do desenvolvimento do software. Verificação e Validação não representam a mesma coisa, embora sejam frequentemente discutidas. De acordo com Delamaro, Maldonado e Jino (2007) as atividades de verificação e validação podem ser divididas em estáticas e dinâmicas, no qual as estáticas são aquelas que não necessitam da execução ou da existência de um programa ou modelo executável para serem conduzidas enquanto que as dinâmicas refletem o oposto, baseando-se na execução de um programa ou modelo.

Para Sommerville (2007), o objetivo final das atividades de verificação e validação é estabelecer a confiança de que o software atende ao seu propósito, e isso significa que o sistema deve ser bom o suficiente para seu intuito, expressando resumidamente a diferença entre validação e verificação:

- Validação: Estamos construindo o produto certo?
- Verificação: Estamos construindo o produto da maneira certa?

Em Delamaro, Maldonado e Jino (2007) verificam-se ainda os conceitos de defeito, falha, erro e engano, que por muitas vezes acabam sendo também confundidos. Um defeito (*fault*) representa um passo, processo ou definição de dados incorretos e engano (*mistake*) corresponde a uma ação humana que produz um defeito. A existência de um defeito pode ocasionar a ocorrência de um erro (*error*) durante a execução de um programa. Esse erro pode resultar em um estado inconsistente ou inesperado chamado de falha (*failure*), que é observada quando o resultado produzido pela execução é diferente do resultado esperado.

O teste de software não deve ser uma atividade realizada de forma isolada. O teste está altamente integrado ao processo de desenvolvimento de software, fazendo parte do seu ciclo de vida de desenvolvimento. Nesse contexto, destaca-se o modelo V (CRAIG; JASKIEL, 2002), no qual a estrutura apresenta uma abordagem para o processo de teste integrado ao processo de desenvolvimento.

Em Craig e Jaskiel (2002) apresenta-se o modelo V dividido em fases ou níveis, no qual para cada atividade de desenvolvimento do software há uma atividade de preparação e execução de testes, conforme mostra a Figura 2.1. Assim para a atividade de análise de requisitos há a elaboração de testes de aceitação, para a atividade de projeto do sistema ou especificação funcional há a elaboração de testes de sistema; para a atividade de projeto técnico funcional ou especificação técnica há a elaboração de testes de integração e para a atividade de especificação de componentes ou construção/codificação do software há a elaboração de testes unitários.

As fases de teste no modelo V são descritas a seguir:

- Teste Unitário: os testes são realizados em cada unidade de execução (programa e seus componentes), examinando a estrutura de dados, identificando erros de lógica e implementação;
- Teste de Integração: os testes são realizados na integração dos módulos do software, identificando erros de interface entre os mesmos;
- Teste de Sistema: os testes são realizados tendo como base os critérios de validação estabelecidos durante a análise dos requisitos. Permite garantir que ao final do desenvolvimento, o software atenderá à todas as exigências funcionais, comportamentais e de desempenho;

- Teste de Aceitação (ou Homologação): verifica a satisfação do cliente e do usuário em relação à solução como um todo, testando os principais requisitos do sistema, identificando os erros de requisitos funcionais e de desempenho;

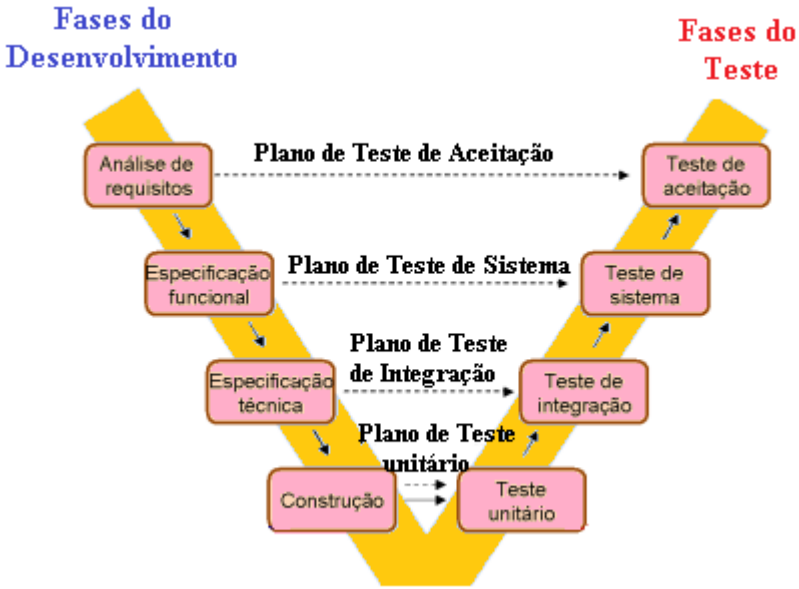


Figura 2.1 Estrutura do Modelo em V - Adaptado de Craig e Jaskiel (2002)

Segundo Barbosa et al. (2000), as técnicas e critérios de teste constituem um mecanismo para projetar bons casos de teste que possibilitem revelar a maioria dos defeitos com o mínimo de esforço e tempo, auxiliando na garantia da qualidade do teste.

As principais técnicas de teste de software são descritas a seguir:

i) Teste Funcional (*Black-box*): é uma técnica utilizada para projetar casos de teste considerando o programa ou sistema como uma caixa-preta. No teste funcional os requisitos de teste são obtidos de especificação funcional, no qual são fornecidas entradas e avaliadas as saídas geradas para verificar se estão em conformidade com os objetivos especificados. Nessa técnica, os detalhes de implementação não são considerados e o software é avaliado segundo o ponto de vista do usuário (FABBRI; VINCENZI; MALDONADO, 2007).

O teste funcional pode ser utilizado em qualquer contexto (procedimental ou orientado a objetos) e em qualquer fase de teste sem a necessidade de modificação. Exemplos de critérios de teste funcional são: Particionamento em Classe de Equivalência, Análise do Valor Limite, Grafo de Causa-Efeito e Teste Baseado em Estado.

ii) Teste Estrutural (*White-box*): também conhecido como Teste Caixa-Branca, estabelece os requisitos de teste com base em uma dada implementação, requerendo a execução de partes ou de componentes elementares do programa. Os caminhos lógicos do software são testados, fornecendo casos de teste que põem à prova tanto conjuntos específicos de condições e/ou laços bem como pares de definições e uso de variáveis (BARBOSA et al., 2007). Os critérios de teste da técnica de teste estrutural mais conhecidos são: Complexidade Ciclomática, Fluxo de Controle e Fluxo de Dados.

iii) Teste Baseado em Defeitos: essa técnica utiliza certos tipos de erros para derivar requisitos de teste. Segundo Jia e Harman (2011), o critério de teste baseado em defeitos mais utilizado é a Análise de Mutantes ou Teste de Mutação apresentado inicialmente em DeMillo, Lipton e Sayward (1978).

Existe outro critério baseado na técnica de teste baseado em defeitos conhecido como Semeadura de Defeitos (BUDD, 1981). No contexto deste trabalho o critério Análise de Mutantes é importante, pois o mesmo será aplicado em uma fase de desenvolvimento do DW.

2.1.1 Análise de Mutantes

A Análise de Mutantes ou Teste de Mutação é baseado em dois princípios fundamentais, definidos em DeMillo, Lipton e Sayward (1978): Hipótese do Programador Competente e Efeito de Acoplamento.

O princípio da Hipótese do Programador Competente estabelece que programadores experientes escrevem códigos-fontes corretos ou próximos do correto. Sendo assim, considera-se que os defeitos que ocorrem em códigos de programas são decorrentes de pequenos enganos (alterações sintáticas) causados pelos programadores. Se não houver erros sintáticos, esses pequenos enganos podem alterar semanticamente o programa causando um resultado não desejado. O outro princípio, denominado Efeito de Acoplamento, considera que erros complexos estão associados a erros simples, assim sendo, espera-se que conjuntos de casos de teste capazes de revelar erros simples são também capazes de revelar erros complexos (BARBOSA et al., 2000).

Segundo Bashir e Nadeem (2012), um processo de teste de mutação deverá iniciar com a geração de mutantes. Um mutante é uma cópia do programa original contendo uma permutação (defeito) que é introduzida por meio de um operador de mutação. Os operadores de mutação determinam o tipo de alteração sintática que deve ser feita para a

geração dos mutantes. Após a geração dos mutantes, os casos de teste são executados com todos os programas mutantes e as saídas dos mutantes são comparadas com as saídas do programa original.

O objetivo do teste de mutação é obter casos de teste que apresentem como resultados apenas mutantes “Mortos”, ou seja, quando o resultado do teste do programa mutante for diferente do resultado do programa original. Se no resultado da execução dos casos de teste, o comportamento do programa original for igual ao do programa mutante, o mutante é considerado “Vivo”. No caso de ocorrer um mutante “Vivo” existem duas situações: ou o mutante é equivalente ao programa original ou o caso de teste não foi capaz de revelar a alteração realizada no programa mutante (QUEIROZ, 2013).

Um mutante é considerado “equivalente” quando o resultado da sua execução for semanticamente igual ao resultado do programa original, e avaliar se um programa é equivalente ou não, é um problema indecidível (BUDD, 1981). Nestas situações, é necessário que uma pessoa avalie o motivo pelo qual este mutante permaneceu vivo (BASHIR; NADEEM, 2012).

A aplicação de desvios sintáticos em um programa gera um conjunto de mutantes (programas similares) por meio de operadores de mutação. Um operador de mutação contém um conjunto de regras que definem como o programa em teste será alterado, e em geral, os mesmos são projetados tendo por base a experiência no uso de dada linguagem, bem como os enganos mais comuns cometidos durante a sua utilização (DELAMARO et al., 2007). Alguns exemplos de operadores de mutação apresentados em Agrawal et al. (1989) para linguagem C são:

- SSDL (*Statement Deletion*): eliminação de comandos, retirando um comando de cada vez do programa;
- ORRN (*Relational Operator Replacement*): troca de operador relacional, substituindo um operador relacional por todos os outros;
- VSRR (*Scalar Variable Reference Replacement*): troca de referência a uma variável escalar, ou seja, cada referência à uma variável não estruturada é substituída pela referência à todas as outras variáveis escalares do programa, uma de cada vez;
- STRI (*Trap on IF Condition*): “armadilha” em condição IF, e tem o objetivo de auxiliar na análise da cobertura de desvios;

Estudos empíricos mostram que os mutantes gerados fornecem uma boa indicação da capacidade de detecção de defeitos de um conjunto de testes (ANDREWS; BRIAND; LABICHE, 2005).

Por fim, na Análise de Mutantes é gerada uma medida de cobertura do teste baseado em defeitos chamada escore de mutação, definido em DeMillo, Lipton e Sayward (1978) que é a razão entre o número de mutantes gerados e o número de mutantes mortos (exceto aqueles considerados equivalentes). Segundo Cabeça, Jino e Leitão-Junior (2009), este escore, que varia entre 0 e 1, indica a qualidade dos casos de teste selecionados para exercitar o programa, calculando-se o escore de mutação *EscMut (P,T)* da seguinte forma (notar que M , E):

$$EscMut\ (P,T) = \frac{K}{M - E} \tag{2.1}$$

Onde:

P: representa o programa em teste;

T: representa o conjunto de casos de teste;

K: representa a quantidade de mutantes mortos;

M: representa o total de mutantes gerados;

E: representa o total de mutantes equivalentes;

Após o cálculo do escore de mutação, fica a critério do testador a decisão sobre a continuidade ou não dos testes. Se o valor do escore de mutação for igual a 1 ou bem próximo de 1, de acordo com a avaliação do testador, os testes poderão ser concluídos e considera-se que os casos de teste possuem uma boa medida de qualidade para testar o programa original, caso contrário, se existirem mais mutantes considerados vivos os testes deverão ser continuados.

Na Figura 2.2 é mostrado o processo genérico de análise de mutantes, que funciona da seguinte forma: dado um programa original P como entrada, pequenas alterações no código de P (programa em teste) são inseridas através de operadores de mutação resultando na geração de programas mutantes P’ ligeiramente diferentes do original. Posteriormente verifica-se se tais modificações são detectadas pelo conjunto de casos de teste T, cuja adequação deseja-se avaliar. Quanto maior o número de alterações identificadas, maior é a qualidade do conjunto de testes T. Essa identificação é feita comparando o resultado da execução do programa original P usando T, com o resultado da execução de cada mutante P’ também usando o conjunto de testes T. Se o resultado

de P’ for diferente do resultado do programa original P diz-se que o mutante está morto, ou seja, a alteração feita em P’ foi identificada por T. Se os resultados de P e P’ forem iguais, o mutante é considerado vivo. Nesta situação, avalia-se se o programa mutante P’ é equivalente a P ou se o conjunto de casos de teste T que não foi capaz de detectar o defeito associado à alteração (mutação) feita em P’. Quando isso ocorre, é necessária uma intervenção humana para analisar e identificar o motivo da sobrevivência de um mutante (QUEIROZ, 2013).

Segundo Delamaro et al. (2007) , sucintamente o processo da Análise de Mutantes divide-se em:

- Geração dos mutantes;
- Execução do programa em teste;
- Execução dos mutantes;
- Análise dos mutantes.

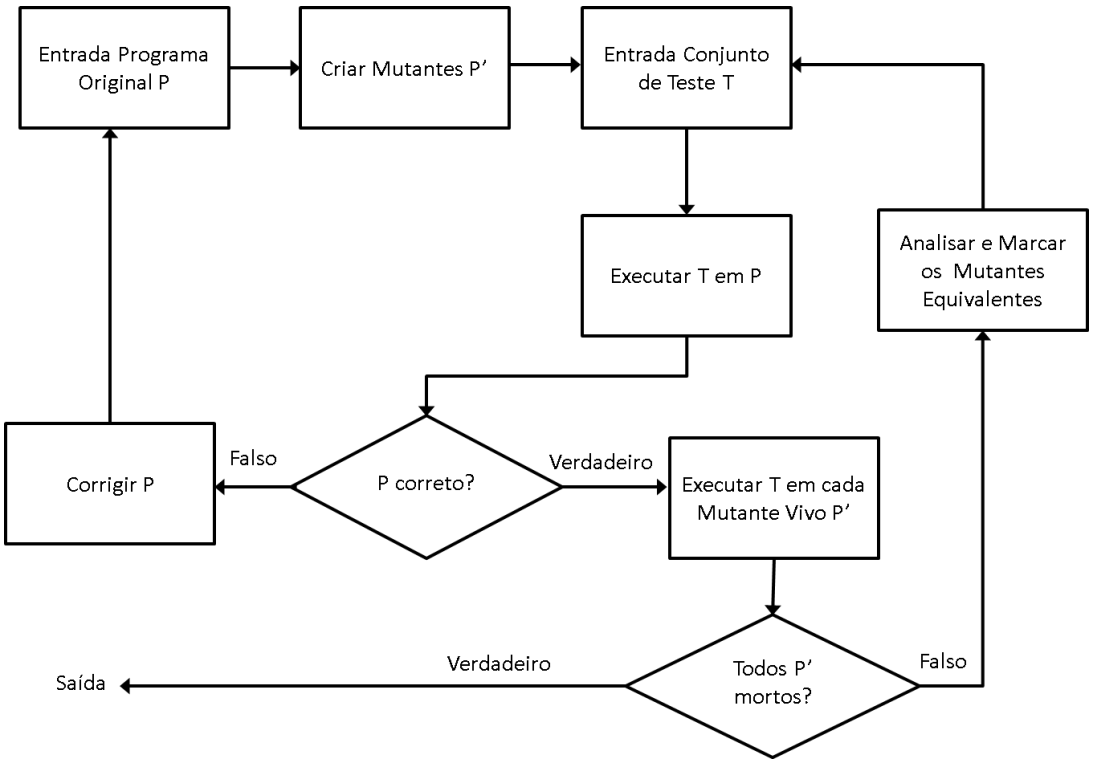


Figura 2.2 Processo Genérico de Análise de Mutação - Extraído de Jia e Harman (2011)

2.1.2 Aplicação do Critério Análise de Mutantes

As primeiras pesquisas realizadas com Análise de Mutantes ocorreram em linguagens de programação como Fortran (ACREE et al., 1979; BUDD et al., 1978; LIPTON; SAYWARD, 1978; KING; OFFUTT, 1991); Ada (BOWSER, 1988; OFFUTT; VOAS; PAYN, 1996) e C (AGRAWAL et al., 1989; JIA; HARMAN, 2008; DELAMARO; MALDONADO, 1996; DELAMARO; MALDONADO, 1999; DELAMARO et al., 2001).

Entre o final da década de 90 e início da década de 2000, foram realizados alguns trabalhos com software Orientado a Objetos e em linguagem Java (KIM; CLARK; MCDERMID, 1999; KIM; CLARK; MCDERMID, 2000; MA; KWON; OFFUTT, 2002; ALEXANDER et al., 2002; MA; OFFUTT; KWON, 2005; BIEMAN; GHOSH; ALEXANDER, 2001; BRADBURY; CORDY; DINGEL, 2006). A partir do ano de 2005 surgiram as primeiras pesquisas com utilização do critério de teste Análise de Mutantes em aplicações de Banco de Dados (CHAN; CHEUNG; TSE, 2005; TUYA; SUAREZ-CABAL; DE LA RIVA, 2006; TUYA; SUAREZ-CABAL; DE LA RIVA, 2007; SHAHRIAR; ZULKERNINE, 2008; ZHOU; FRANKL, 2009; CABEÇA; JINO; LEITÃO-JUNIOR, 2009). O critério começou a ser aplicado também em interfaces, especificações e modelos, como descrito na pesquisa de Jia e Harman (2011).

Na Figura 2.3 (JIA; HARMAN, 2011) é apresentada a quantidade de teste de mutação aplicado tanto em linguagens de programação quanto em especificações até o ano de 2008, sendo também observado que houve mais trabalhos de teste de mutação em linguagens de programação que em especificação. Fortran representa uma grande parte do trabalho sobre teste de mutação, pois foi uma das primeiras linguagens de programação a ser utilizada. Já na Figura 2.4 (JIA; HARMAN, 2011) é apresentada a porcentagem de publicações por área e linguagens de programação que aplicaram análise de mutação até o ano de 2008, e notadamente, mais de 50 % desses trabalhos foram aplicados a Java, Fortran e C.

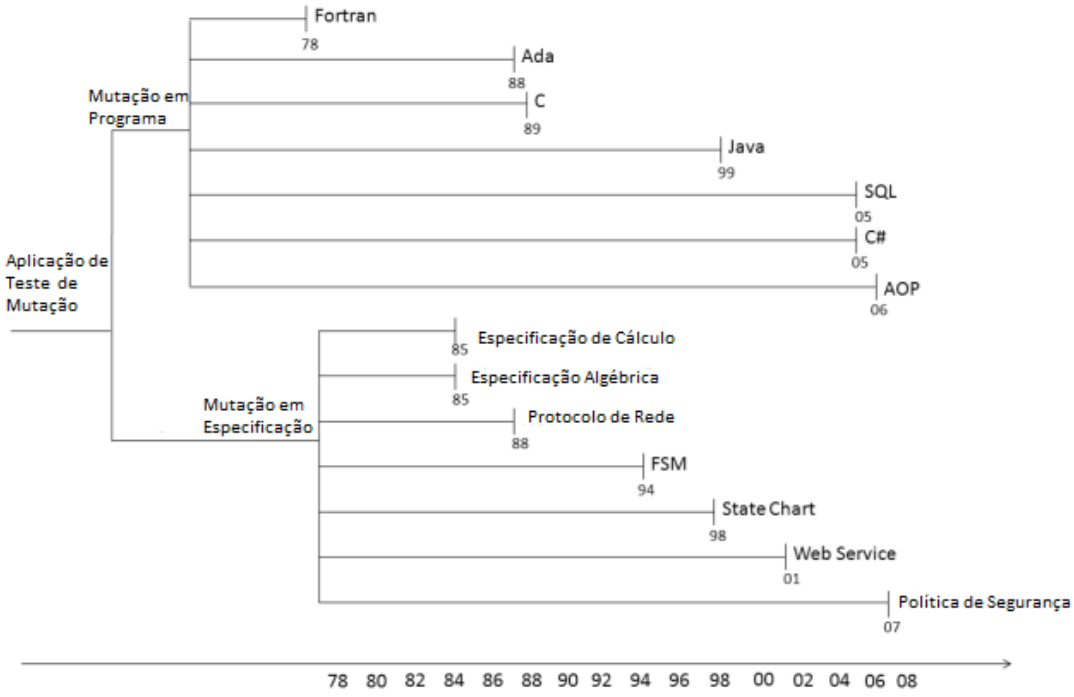


Figura 2.3 Publicações das Áreas de Aplicação dos Testes de Mutação – Extraído de Jia e Harman (2011)

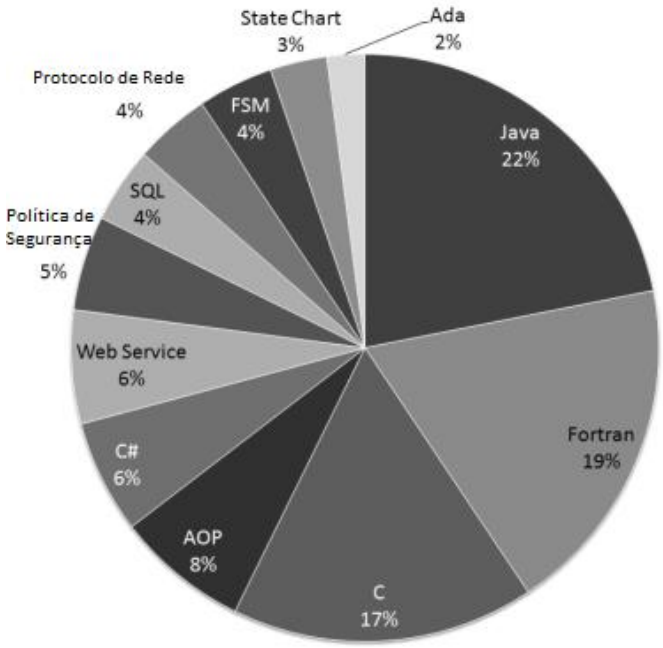


Figura 2.4 Porcentagem de publicações das áreas de aplicação de Testes de Mutação- Extraído de Jia e Harman (2011)

2.2 *DATA WAREHOUSE*

Segundo Inmon (2012), *Data Warehouse* é uma coleção de dados orientados por assunto, integrados, não voláteis, variáveis com o tempo, para dar suporte ao processo

de tomada de decisão. Tem como foco a organização como um todo, englobando várias áreas por assunto, e consequentemente criando *Data Marts* que se relacionam entre si.

Para Kimball e Ross (2013), *Data Warehousing* é um conjunto de ferramentas e técnicas de projeto, que quando aplicadas às necessidades específicas dos usuários e aos bancos de dados específicos permite que seja planejado e construído um *Data Warehouse*.

Para Laudon e Laudon (2013), com uma visão mais administrativa, *Data Warehouse* é um banco de dados com ferramentas de consultas e relatórios, que armazena dados atuais e históricos extraídos de vários sistemas operacionais e consolidados para fins de análise e relatórios administrativos.

Em Kimball e Ross (2013) são destacados os seguintes objetivos para um *Data Warehouse* (DW):

- O *Data Warehouse* deve fornecer acesso aos dados corporativos ou organizacionais;
- Os dados do *Data Warehouse* devem ser consistentes;
- Os dados do *Data Warehouse* podem ser separados e combinados usando-se qualquer medição possível do negócio (*slice and dice*);
- O *Data Warehouse* não consiste apenas em dados, mas também em um conjunto de ferramentas para consultar, analisar e apresentar informações;
- O *Data Warehouse* é o local em que são publicados dados confiáveis;
- A qualidade dos dados no *Data Warehouse* impulsiona a reengenharia de negócios, ou seja, o *Data Warehouse* não pode aprimorar dados de baixa qualidade;

De acordo com Inmon (2012) os seguintes requisitos básicos para o *Data Warehouse* devem ser destacados:

- Deve ser organizado por assuntos, ou seja, armazenar informações sobre temas específicos importantes para o negócio;
- Deve possuir capacidade de integração, ou seja, garantir a consistência e uniformidade para as informações por meio da padronização das mesmas;
- Deve ser variante no tempo, ou seja, o *Data Warehouse* deve referir-se a informação em algum momento específico, significando que não é atualizável;
- Deve ser flexível o suficiente para atender às exigências de mudança rapidamente;

- Os dados devem ser não voláteis e devem ser carregados apenas uma vez e depois apenas consultados.
- Os dados devem existir em vários níveis de granularidade.

Segundo ElGamal (2015), a arquitetura de um *Data Warehouse* difere de um projeto para outro com base nos requisitos de negócios específicos, e o componente base que está presente em todos os projetos de *Data Warehouse* é a fonte de dados, e qualquer outro componente pode ser incluído ou excluído dependendo da necessidade do negócio.

Para ElGamal, ElBastawissy e Galal-Edeen (2011) a arquitetura global de um DW consiste de componentes inter-relacionados, que precisam ser testados para verificar a sua eficiência independentemente.

A Figura 2.5 (ElGamal, 2015) apresenta a arquitetura mais genérica e completa para um DW, conhecida como "*Kim-mon Architecture*" que representa uma combinação das arquiteturas de *Ralph Kimball* e *Bill Inmon's*, cujos componentes são:

- Fontes de Dados - são representadas por todos os sistemas legados, bases de dados transacionais, planilhas eletrônicas, dados de formulários, páginas da internet, e é de onde são extraídas as informações para a DSA (*Data Staging Area*);
- DSA (*Data Staging Area*) - representa uma área de armazenamento intermediário dos dados, no qual é realizado o processo ETL (*Extraction, Transformation and Load*) a fim de tratar e transformar dados em informação para serem enviadas para o ODS (*Operational Data Store*);
- ODS (*Operational Data Store*) - ODS é um repositório, que armazena apenas as informações correntes, antes de serem carregadas para DW. ODS é implementado quando existe a necessidade de analisar informações do dia-a-dia. Os dados são atualizados e ficam armazenados para consultas em um período de tempo curto, geralmente de 1 a 3 meses. Após esse período esses dados são armazenados em um DW. Em alguns projetos de DW esse componente é excluído, conforme descrito em ElGamal (2015);
- DW (*Data Warehouse)* - representa uma grande base de dados capaz de integrar dados históricos, cuja finalidade é armazenar informações que permitam mostrar indicadores e apresentar a evolução de valores ao longo de um período de tempo;

- DM (*Data Marts)* - representam um subconjunto de dados do DW, direcionados geralmente a uma área específica de um processo de negócio;
- UI (Aplicações de Interface de Usuário) – são representadas por ferramentas do tipo OLAP (*On-line Analytical Processing*) e permitem ao usuário analisar as informações e resultados obtidos das consultas ao DW. Ex: relatórios OLAP, Ferramentas de Suporte à Decisão e Ferramentas de Análise.

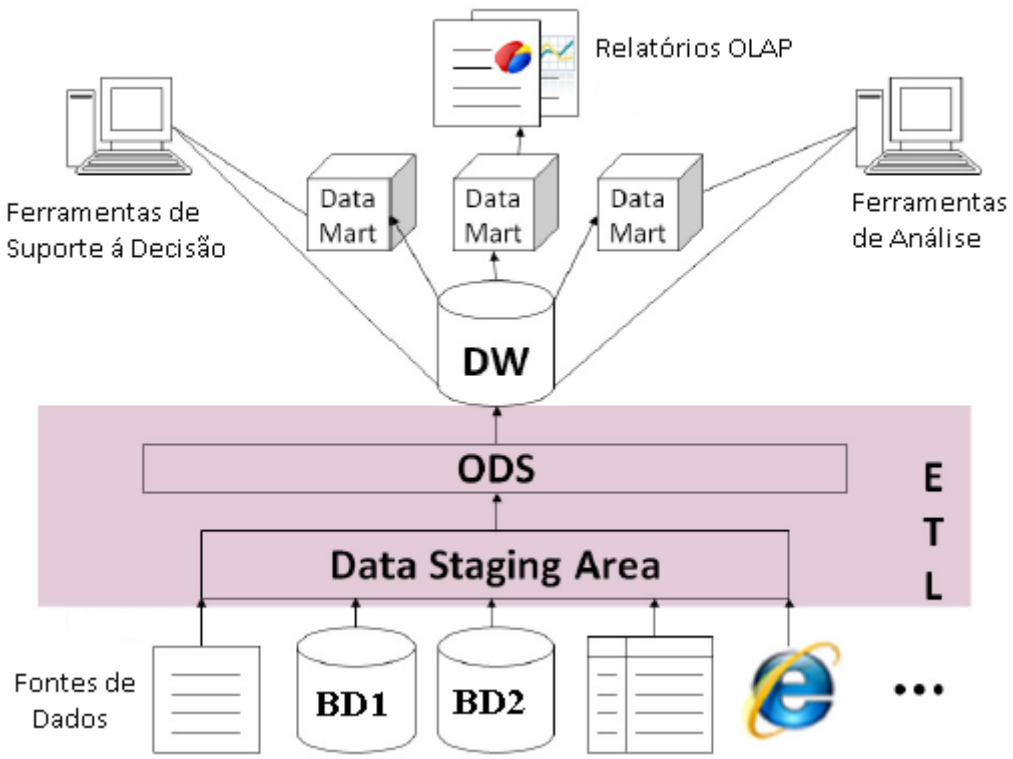


Figura 2.5 Arquitetura Genérica do DW – Extraído de ElGamal (2015)

Nos trabalhos de ElGamal, ElBastawissy e Galal-Edeen (2013) , Golfarelli e Rizzi (2009) são apresentados os desafios para testar sistemas DW, dentre eles:

- DW possuem consultas *ad-hoc*, então é quase impossível realizar testes antes do sistema DW ser desenvolvido, porque você não sabe quais as consultas possíveis a serem feitas no sistema;
- Testes de sistemas convencionais são geralmente focados no código enquanto testes em DW são centralizados nos dados;
- DW sempre lida com grande volume de dados;
- Um projeto de DW não termina ao concluir o seu desenvolvimento, pois sempre existirá exigência de processo de tomada de decisão para as mudanças em curso;

- Volume de dados de teste em DW é consideravelmente grande comparado com qualquer outro processo de teste;
- Para realização de testes em sistemas convencionais, os cenários de casos de teste são limitados, enquanto no DW, os casos de teste são ilimitados justamente devido ao fato de que os DWs devem permitir todas as possibilidades de consulta e exibição de dados;
- Teste em DW consiste em diferentes tipos de testes, pois depende da fase de aplicação do teste e qual o componente a ser testado. Por exemplo, o teste de carga inicial de dados é diferente do teste de carga de dados incremental.

2.2.1 Processo ETL

O processo ETL (*Extraction, Transformation and Load*) é o processo mais crítico e demorado na construção de um *Data Warehouse*. ETL e as ferramentas de limpeza de dados consomem um terço do orçamento em um projeto de *Data Warehouse*, e 80% do tempo de desenvolvimento de um *Data Warehouse* consiste no processo de ETL (INMON, 2012).

A Figura 2.6 mostra o processo simplificado do ETL (TURBAN et al., 2010) , no qual as informações são extraídas do ambiente de negócio, tais como sistemas legados e aplicações internas e as mesmas são carregadas em um repositório temporário (*Staging Area* - fonte de dados transiente) para fazer a limpeza e transformação desses dados para que os mesmos sejam posteriormente carregados e armazenados no *Data Warehouse* ou no ODS, conforme explicado anteriormente na Figura 2.5, possibilitando posteriormente a geração de *Data Marts*.

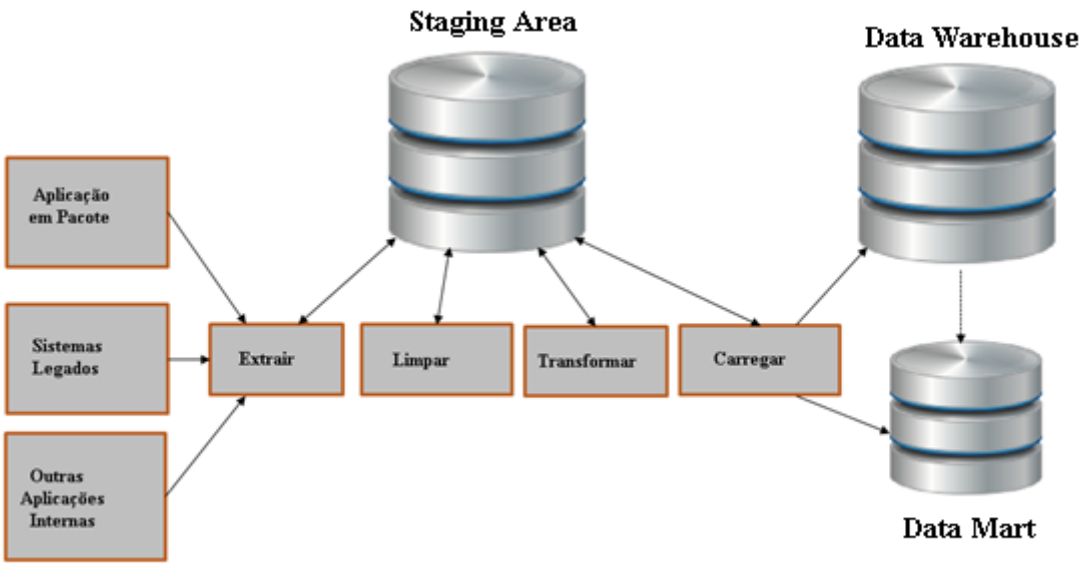


Figura 2.6 Processo ETL de um *Data Warehouse* - Adaptado de Turban et al. (2010)

Segundo Kimball e Ross (2013) os principais requisitos para o processo ETL são:

- Ter conhecimento das necessidades de negócio, ou seja, verificar nos indicadores de negócio quando existem mudanças que geralmente estão associadas a uma entrada do DW;
- Obter conhecimento da conformidade (Contexto Legal);
- Saber qual nível de qualidade dos dados deseja-se obter no DW;
- Verificar os requisitos de segurança, ou seja, quais os requisitos e limitações de acesso para o DW;
- Entender o nível de Integração dos Dados, pois tem-se várias bases de dados que replicam as mesmas informações;
- Qual a Latência dos Dados, ou seja, o tempo necessário do dado ser modificado e cadastrado no DW e estar disponível para o usuário do negócio no DW;
- Estabelecer e entender como é feito o Arquivamento dos Dados e *Lineage*, pois em algum momento tem-se que abdicar de algumas informações no DW, pois o espaço para a informação é limitado, e então arquivam-se essas informações em outra mídia que tem que estar disponível, mas que possa tratar da linhagem e evolução dos dados;
- Definir o que tem que ser entregue nas interfaces de entrega de BI *(Business Intelligence)*;

- Antes de desenvolver um processo de ETL verificar as cargas que já são feitas, fazendo reuso do que já foi implementado anteriormente.

2.3 QUALIDADE DE DADOS

A Qualidade de Dados define-se como a atividade que detecta e corrige anomalias nos dados (BATINI; SCANNAPIECO, 1998). Uma definição para não ocorrer qualidade de dados refere-se aos dados que estão faltando, ou estão incorretos ou inválidos em algum contexto. Uma definição mais ampla descreve que a qualidade de dados é alcançada quando a organização utiliza dados que são abrangentes, compreensíveis, consistentes, relevantes e oportunos (SINGH; SINGH, 2010). A falta de qualidade de dados tem consequências graves em longo prazo para a eficácia e eficiência dos negócios nas organizações. O relatório de qualidade de dados do *Data Warehouse Institute* estimou que os problemas de qualidade de dados custaram para as empresas mais de 600 bilhões de dólares por ano, conclusões que foram baseadas em entrevistas com especialistas, clientes e dados de pesquisa a partir de 647 respondentes (BATINI; SCANNAPIECO, 1998).

2.3.1 Tipos de Dados

Os dados representam objetos do mundo real com capacidade de armazenamento, recuperação e elaboração por meio de um processo de software, podendo se comunicar por uma rede (SIDI et al., 2012). Pesquisadores forneceram classificações diferentes para os dados em diferentes áreas. De acordo com Batini e Scannapieco (1998), três tipos de dados são descritos a seguir na área de Qualidade de Dados (*Data Quality* - DQ):

Estruturado - quando cada elemento de dados tem uma estrutura fixa associada. Exemplo: Tabelas relacionais são os tipos mais populares de dados estruturados, dados estatísticos;

Semi-estruturado – quando os dados possuem uma estrutura com algum grau de flexibilidade. Algumas características comuns são:

- Os dados podem conter campos não conhecidos em tempo de *design*. Exemplo: um arquivo XML que não tem um arquivo de esquema XML associado;

- O mesmo tipo de dados pode ser representado em várias formas. Exemplo: uma data pode ser representada por um campo ou por vários campos, mesmo dentro de um único conjunto de dados;
- Entre os campos conhecidos em tempo de *design*, muitos campos não terão valores;

Não estruturado - quando os dados são expressos em linguagem natural e não possui estrutura ou nenhum domínio específico. Exemplo: texto no corpo de um e-mail.

2.3.2 Dimensões da Qualidade de Dados

A qualidade dos dados é um conceito multifacetado, cuja definição permite diferentes dimensões. Uma das dimensões de qualidade, por exemplo, a precisão, pode ser facilmente detectada em alguns casos (por exemplo, erros ortográficos), mas são mais difíceis em outros casos (por exemplo, nos quais são fornecidos os valores admissíveis, mas não corretos).

Segundo Batini e Scannapieco (1998) existem quatro categorias de dados que são importantes para a avaliação da qualidade de dados:

Intrínseca: características intrínsecas dos dados, independentes da sua aplicação, ou seja, não é analisada a qual aplicação o dado está relacionado;

Acessibilidade: aspectos relativos ao acesso e à segurança dos dados;

Contextual: características que ao contrário da intrínseca, são dependentes do contexto de utilização dos dados;

Representação: características derivadas da forma como a informação é apresentada.

Para essas categorias são associadas algumas dimensões da qualidade dos dados, conforme pode ser visto na Tabela 2.1, na qual são apresentadas as dimensões da qualidade dos dados e suas definições.

Tabela 2.1 Dimensões da Qualidade dos Dados (BATINI; SCANNAPIECO, 1998)

Categoria	**Dimensão**	**Definição**
Intrínseca	Credibilidade (*Beleivability*)	Quanto a informação é considerada como verdadeira e verossímil
	Acurácia (*Accuracy*)	Quanto a informação é correta e confiável
	Objetividade (*Objectivity*)	Quanto a informação é parcial
continua na próxima página		

Tabela 2.1 Dimensões da Qualidade dos Dados (BATINI; SCANNAPIECO, 1998)- (continuação)

Categoria	Dimensão	Definição
	Reputação (*Reputation*)	Quanto a informação é considerada verdadeira em termos de sua fonte ou conteúdo
Acessibilidade	Acessibilidade (*Accessibility*)	Quanto a informação está disponível, ou fácil e rapidamente recuperável
	Segurança no Acesso (*Access Security*)	Quanto o acesso à informação é restrito apropriadamente para manter sua segurança
Contextual	Relevância (*Relevancy*)	Quanto a informação é aplicável e útil para a tarefa a ser realizada
	Valor agregado (*Value-Added*)	Quanto a informação é benéfica e proporciona vantagens por seu uso
	Temporalidade/Oportunidade (*Timeliness*)	Quanto a informação está suficientemente atualizada para a tarefa a ser realizada
	Integridade/perfeição (*Completeness*)	Quanto a informação não está extraviada e é suficiente para a tarefa em amplitude e profundidade
	Quantidade de Informação Apropriada (*Appropriate amount*)	Quanto o volume da informação é apropriado para a tarefa a ser executada
Representação	Interpretabilidade (*Interpretabiity*)	Quanto a informação está em linguagem apropriada, símbolos e unidades, e as definições são claras
	Facilidade de entendimento (*Ease of understanding*)	Quanto a informação é facilmente compreendida
	Representação concisa (*Concise representation*)	Quanto a informação está compactamente representada
	Representação consistente (*Consistent representation*)	Quanto a informação é apresentada em um mesmo formato
	Facilidade de manipulação (*Ease of manipulation/operation*)	Quanto a informação é fácil de ser manipulada e aplicada em diferentes tarefas

2.4 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados conceitos importantes que serão úteis para a compreensão dos assuntos abordados nesse trabalho, como conceitos básicos sobre a atividade de teste de software, critérios e técnicas de teste de software, tendo como principal foco a técnica de Teste Baseado em Defeitos e o critério de teste Análise de Mutantes e sua aplicabilidade. Foram apresentados também conceitos relacionados à *Data Warehouse* e Qualidade de Dados.

No próximo capítulo será apresentado o estado da arte relacionado a esses assuntos e a importância dos mesmos para essa pesquisa.

3. TRABALHOS RELACIONADOS

Neste capítulo são apresentados os trabalhos na literatura que tratam de testes em DW (GOLFARELLI; RIZZI, 2009; GOLFARELLI; RIZZI, 2011; MUNAWAR; SALIM; IBRAHIM, 2011; MEKTEROVIC; BRKIC; BARANOVIC, 2011; ELGAMAL; ELBASTAWISSY; GALAL-EDEEN, 2013; KULDEEP, 2013; ELGAMAL, 2015). Além desses trabalhos, são abordados trabalhos que fazem uso de análise de mutantes em consultas SQL fora do ambiente de DW (CHAN; CHEUNG; TSE, 2005; TUYA; SUAREZ-CABAL; DE LA RIVA, 2006; TUYA; SUAREZ-CABAL; DE LA RIVA, 2007; CABEÇA; JINO; LEITÃO-JUNIOR, 2009) e também trabalhos relacionados a Análise de Instâncias de Dados Alternativas (EMER, 2007; NAZAR, 2007).

Existem ainda trabalhos que tratam os problemas de qualidade de dados com relação ao DW (RAHM; DO, 2000; SINGH; SINGH, 2010; BARATEIRO; GALHARDAS, 2005).

3.1 TESTES EM *DATA WAREHOUSE*

Em Singh e Singh (2008) foi desenvolvido um protótipo para geração de dados de teste para *Data Warehouse* baseado em uma arquitetura de geração de dados (DSG- *DataSet Generator*) e um algoritmo de multiplicação de dados (DMA- *Data Multiplication Algorithm*) que utiliza algoritmos genéticos para geração de dados sintéticos preservando a integridade, tendências, equilíbrio correto e simetria de dados em comparação aos dados reais. A arquitetura é baseada em duas fases. Na primeira fase a tabela é gerada pelos fatos relevantes a partir de fontes múltiplas e na segunda fase a base de dados é sinteticamente ampliada usando o DMA. Foi necessário um conjunto de dados iniciais para gerar os casos de teste. Os autores destacam que o extrator de dados usado no algoritmo possui um escopo limitado, pois extrai dados apenas de projetos .mdb, e esforços estavam sendo feitos para que pudesse também extrair dados de bases de dados *Oracle* e SQL. Concluíram que os dados gerados são autênticos e imunes às legalidades e éticas de negócio, podendo ser utilizados para teste de DW.

Em Golfarelli e Rizzi (2009) foi apresentada uma metodologia para construção de um *framework* que define os componentes do DW que devem ser testados e como deve ser realizada a atividade de teste no DW. Os seguintes componentes do DW são testados: Esquema Conceitual, Esquema Lógico, Procedures ETL (Extração, Transformação e Carga), Banco de Dados e *Front-End*. Para serem capazes de testar esses componentes, os autores abordaram sete tipos de testes: Teste Funcional, Teste de Usabilidade, Teste de Desempenho, Teste de Stress, Teste de Recuperação, Teste de Segurança e Teste de Regressão, relacionando quais tipos de testes obtiveram uma cobertura melhor em cada componente, conforme apresentado na Tabela 3.1. Os autores ressaltam em seu trabalho que testes em DW são amplamente baseados em dados, portanto um teste bem-sucedido deve contar com situações reais, mas deve também incluir dados simulados para reproduzir o maior número de erros comuns que podem ser encontrados no processo ETL. Neste artigo, não são apresentados resultados de experimentos em situações reais, pois os autores apenas fizeram uma avaliação empírica da abordagem.

Tabela 3.1 O quê versus Como testar (GOLFARELLI; RIZZI, 2009)

Tipo de Teste	**Esquema Conceitual**	**Esquema Lógico**	**Processo ETL**	**Base de Dados**	***Front-End***
Funcional	X	X	X		X
Usabilidade	X	X			X
Desempenho		X	X	X	X
Stress			X	X	X
Recuperação			X	X	
Segurança			X	X	X
Regressão	X	X	X	X	X
	Análise e *Design*		**Implementação**		

Segundo ElGamal, ElBastawissy e Galal-Edeen (2013), os testes em DW representam um estágio crítico no desenvolvimento do DW. Desta forma, os autores apresentaram uma matriz de teste em DW categorizada pelas seguintes questões: “O que”, “Onde” e “Quando”. A questão “O que” avalia se o esquema, dados e operações estão sendo testados entre as fases do DW. A questão “Onde” evidencia em quais fases do DW devem ser aplicados os testes. A questão “Quando” avalia se o teste será utilizado antes ou depois da entrega do DW. Depois disso, os autores analisaram dez abordagens da literatura e determinaram quais questões foram resolvidas por cada abordagem. Dessa forma, as necessidades do ambiente de teste de DW foram

determinadas e posteriormente, ElGamal (2015) apresentou uma proposta de *framework* de teste visando atender as necessidades do ambiente de teste em DW.

No trabalho de Mekterovic, Brkic e Baranovic (2011) foi estabelecido um procedimento genérico para integração do teste de determinados aspectos do procedimento ETL. Nesta abordagem, os procedimentos ETL são tratados como uma caixa-preta e são testados comparando as suas respectivas entradas e saídas, ou seja, o conjunto de dados. O procedimento realiza o teste no DW por meio dos dados consolidados no *Staging Area* e dos dados da *Relational Source*. O procedimento proposto é genérico e pode ser implementado em qualquer DW empregando um modelo dimensional e tendo um banco de dados relacional como uma fonte.

Em Golfarelli e Rizzi (2011) foi discutida uma abordagem abrangente para teste em DW, mostrando como uma série de atividades de teste específicas foram classificadas dentro de uma metodologia baseada em um protótipo sobre as fases do DW, como esquema multidimensional, procedimentos ETL, esquema físico e *front-end.* Os autores executaram um estudo de caso em uma empresa para avaliar a sua metodologia e citaram que os resultados mostraram uma favorável adaptação e expansão da metodologia de teste aplicada ao *Data Warehouse*. Os autores descreveram que o projeto do DW levou um tempo de nove meses para ser implementado e apresentaram os seguintes resultados para o esforço do teste executado em cada fase de desenvolvimento: 10% para os testes no esquema multidimensional, 53% para testes no ETL, 32% para testes *Front-End* e 5% para testes no esquema físico.

Em Munawar, Salim e Ibrahim (2011) foi apresentada uma abordagem para integração da qualidade de dados, qualidade de negócios e qualidade da informação na análise de requisitos e na fase do projeto conceitual do *Data Warehouse*. Esta abordagem estabeleceu bases para garantir a qualidade dos aspectos que devem ser devidamente incorporados aos vários níveis do processo de desenvolvimento de DW assim como cobrir todas as camadas de desenvolvimento do DW. Os autores utilizaram um questionário com os usuários para avaliar a sua abordagem, entretanto os autores não apresentaram quais foram os resultados obtidos e nem a quantidade de pessoas entrevistadas.

No trabalho de Kuldeep (2013) foi apresentada uma abordagem de Teste Baseado em Modelo para *Data Warehouses* e como esta abordagem pode ajudar a enfrentar os desafios em testes de *Data Warehouse*. A abordagem gera um modelo de teste sobre o sistema (SUT- *System Under Test*) usando um mapeamento de dados em UML baseado

no mapeamento do fluxo de dados em nível de atributo e tabela. O mapeamento em nível de tabela irá conter a relação entre a fonte e as tabelas do DW. O mapeamento em nível de atributo irá conter a relação entre a fonte, os atributos e as transformações. O autor utilizou técnicas de particionamento em classes de equivalência para geração de dados de teste e os modelos UML obtidos pelo SUT para gerar os casos de teste. Apresentou como vantagens em utilizar a abordagem proposta, o desenvolvimento do gerador de dados de teste que incluiu todas as categorias de dados válidos e inválidos, o tempo limitado para geração de casos de teste e o grande número de combinações dos casos de teste, principalmente, a facilidade em mudar um caso de teste, e também a geração e execução do mesmo de forma automatizada a partir da mudança do modelo de teste. Entretanto, o autor não apresentou resultados experimentais, e levantou a necessidade da abordagem ser validada e verificada para saber se o modelo reduz o esforço dos testes em DW e apresenta uma melhor cobertura de testes.

3.2 ANÁLISE DE MUTANTES EM DIFERENTES CONTEXTOS

Na literatura são encontrados diversos trabalhos que utilizam a Técnica de Teste Baseado em Defeitos por meio do critério Análise de Mutação, e alguns deles são descritos a seguir.

Na pesquisa de Derezinska (2003) foram investigados quais operadores de mutação OO podem ser aplicados na especificação de classes em UML ou no código escrito em uma linguagem OO (especificamente C++). São identificados os seguintes operadores de mutação OO para o código e a especificação: Herança, Associação, Objeto, Membro e Acesso. No artigo é utilizado como exemplo, a especificação de um carro e no diagrama de classes em UML foram aplicados os operadores de mutação para as características de Herança, Associação e Acesso antes da geração do código. Os grupos de operadores de mutação Objeto e Membro foram aplicados diretamente no código. Para o conjunto de casos de teste selecionados, foram satisfeitos os critérios de cobertura, com cobertura de 83% para as funções e 85% para as linhas de código. O trabalho desenvolvido necessita de experimentos adicionais para avaliação dos operadores OO.

Em Ma, Offutt e Kwon (2005) foi apresentado um método para reduzir o custo da execução do teste de mutação em programas OO através de duas metodologias: *Mutant*

Schemata Generation (MSG) e tradução de *bytecode*. Para reduzir o custo computacional do teste de mutação foram desenvolvidas algumas abordagens utilizando as estratégias: *do fewer*, *do smarter* e *do faster*. A estratégia *do fewer* tenta executar menos programas mutantes sem perder eficácia. A estratégia *do smarter* distribui o custo computacional através da execução em várias máquinas. A estratégia *do faster* concentra-se em formas de gerar e executar mutantes tão rápido quanto possível. Porém essas estratégias foram desenvolvidas para aplicação em linguagem de programação tradicional, e não são todas aplicadas a linguagens OO. Portanto, foi verificada apenas a aplicabilidade do método *do faster* para teste de mutação inter-classe OO, na tentativa de reduzir o tempo de compilação. A abordagem dos autores também não apresentou uma avaliação experimental da eficácia do método, necessitando de um refinamento para filtrar quais tipos de defeitos podem ser encontrados nos testes que satisfaçam a mutação Orientada a Objetos.

No trabalho de Bashir e Nadeem (2012) foi apresentado um *survey* de teste de mutação Orientado a Objetos, no qual foram discutidos os problemas do teste de mutação nesse contexto. A partir de questões levantadas desses problemas foram avaliadas as soluções propostas na literatura. Essas questões foram referentes ao custo efetivo, detecção de mutantes equivalentes, características da orientação a objeto, nível de teste, mutação do estado do objeto e operadores de mutação potenciais. Os autores concluíram que o teste de mutação seletiva reduzia o custo computacional e que experimentos em grande escala ajudavam a identificar operadores de mutação potenciais para evitar a geração de mutantes equivalentes. Os autores destacaram que embora existissem técnicas propostas para discutir os problemas, nenhuma delas forneceu uma solução prática para resolvê-los.

Em Papadakis e Le Traon (2012) foi proposto um novo método de localização de defeitos por meio de mutação. O artigo analisa a precisão do método utilizando critérios de seleção de teste como mutação, *branch* e bloco. O experimento mostra que a abordagem de mutação é bastante eficaz na identificação de programas defeituosos desconhecidos. Além disso, os resultados experimentais revelaram que os conjuntos de testes baseados em mutação foram significativamente mais eficazes no apoio da localização de defeitos do que blocos ou *branch*. De acordo com os autores, a abordagem possui um custo computacional alto necessitando de uma alternativa para localização de defeitos baseado em mutantes.

Em Camargo e Vergilio (2013) foram propostas classes de defeitos genéricas por meio da metodologia de identificação de defeitos ODC (*Orthogonal Defect Classification*) para aplicação em programas *MapReduce*, resultando na classificação de sete classes de defeitos para esses programas através de estudos empíricos. Os autores ressaltam que as classes de defeitos foram obtidas de forma empírica e que, portanto, existem ameaças referentes à validade dos resultados, sendo necessária a validação dessa classificação em trabalhos futuros.

3.3 ANÁLISE DE MUTANTES SQL

Em Chan, Cheung e Tse (2005) é proposta a integração de instruções SQL com o modelo de dados conceitual de uma aplicação para realizar o teste baseado em defeitos. Foi definido um conjunto de operadores de mutação baseado nos padrões e tipos de restrições presentes no modelo entidade-relacionamento, conforme pode ser visto na Tabela 3.2. Os operadores de mutação são semânticos e foram gerados para comandos SQL do tipo SELECT. No artigo, nenhuma ferramenta foi construída com a finalidade de automatizar a geração, execução e avaliação dos mutantes, não sendo possível devido à isso, apresentar resultados com relação à capacidade de detecção de defeitos da proposta de pesquisa.

Tabela 3.2 Operadores de Mutação de Substituição (CHAN; CHEUNG; TSE, 2005)

Operadores de Mutação Semântico	Sigla	Descrição
Substituição de Restrições de Participação	PTCR	Alterna os requisitos de participação de tipos de entidade na relação.
Substituição de Restrições de Cardinalidade	CDCR	Substitui as cardinalidades dos tipos de entidade na relação.
Identificação/Substituição de entidades do tipo fraca	IWKR	Substitui uma expressão do tipo de identificação por uma expressão do tipo de entidade fraca, ou vice versa.
Substituição de Atributos	ATTR	Substitui uma expressão de atributo(s) por uma expressão de outro(s) atributo(s) de um tipo compatível.
continua na próxima página		

Tabela 3.2 Operadores de Mutação de Substituição (CHAN; CHEUNG; TSE, 2005) - (continuação)

Operadores Semânticos de Mutação	Sigla	Descrição
Substituição de Generalização / Especialização de Integralidade	GSCR	Substitui uma expressão em uma superclasse parcial por uma expressão em uma subclasse na forma de negação da superclasse.
Substituição de Generalização / Especialização de Disjunção	GSDR	Substitui uma expressão de tipo de entidade irmão por outra expressão do tipo de entidade irmão na mesma superclasse.
Substituição de integralidade do tipo união	UTCR	Substitui um tipo de entidade por uma subclasse e / ou superclasses da subclasse, de tal forma que estas superclasses tenha a mesma restrição de tipo união.

No trabalho de Leitão-Junior, Vilela e Jino (2005) são investigadas as falhas decorrentes de manipulação de código SQL com o objetivo de avaliar os resultados obtidos e compreender a relação entre falhas e defeitos revelados. A análise do mapeamento de dados indicou que: i) existe um mapeamento de muitos-para-muitos entre falhas e defeitos; ii) dimensões de falha são dependentes do tipo de defeito, do comando com defeito, e do próprio banco de dados; e iii) o conhecimento da manipulação dos defeitos é crucial para programação e teste em aplicações e banco de dados. O artigo não apresentou resultados em aplicações de bases de dados reais, apenas a análise do mapeamento entre falhas e defeitos. A seguir, a estrutura de comandos SELECT, INSERT, UPDATE e DELETE (Tabelas 3.3 e 3.4), e os tipos de defeitos mapeados (Tabela 3.5) são apresentados respectivamente.

Tabela 3.3 Estrutura do Comando SELECT (LEITÃO-JUNIOR; VILELA; JINO, 2005)

Item Estrutural	Descrição do Item Estrutural	Exemplo de Comando
[s1]	Uma lista ordenada de expressões usadas para computar os valores de atributos retornados	select salary, count(*), sum(bonus) from empl where (salary + bonus) > 1050 group by salary having count(*) > 1 order by salary desc
[s2]	Uma lista de nomes de tabelas usadas como fontes de dados	
[s3]	Um predicado usado para seleção de tuplas	
[s4]	Uma lista de expressões usadas para agrupamento de dados	
[s5]	Um predicado usado para seleção de grupo de dados	
[s6]	Uma lista ordenada de argumentos usados para ordenação de dados	

Tabela 3.4 Estrutura dos Comandos INSERT, UPDATE e DELETE (LEITÃO-JUNIOR; VILELA; JINO, 2005)

Comando	Item Estrutural	Descrição do Item Estrutural	Exemplo de Comando
insert (1)	[i1]	Um nome de tabela	insert into empl (emplno, name, salary, bonus) values (1234 , 'ana' , 1060 , 35)
	[i2]	Uma lista ordenada de atributos	
	[i3]	Uma lista ordenada de valores dos atributos	

continua na próxima página

Tabela 3.4 Estrutura dos Comandos INSERT, UPDATE e DELETE (LEITÃO-JUNIOR; VILELA; JINO, 2005) – (continuação)

Comando	Item Estrutural	Descrição do Item Estrutural	Exemplo de Comando
insert (2)	[i1]	Um nome de tabela	insert into empl (emplno, name, salary, bonus) select custno, name, salary, 0 from customer
	[i2]	Uma lista ordenada de atributos	
	[i4]	Uma *subquery*	
delete	[d1]	Um nome de tabela	delete from empl where (salary + bonus) > 1050
	[d2]	Um predicado usado para selecionar tuplas	
update	[u1]	Um nome de tabela	update empl set salary = salary * 1.01, bonus = bonus * 1.10 where (salary + bonus) > 1050
	[u2]	Uma lista de valores de atributos setados	
	[u3]	Um predicado usado para seleção de tupla	

Tabela 3.5 Lista de Tipos de Defeitos para os comandos INSERT, UPDATE e DELETE (LEITÃO-JUNIOR; VILELA; JINO, 2005)

ID do Defeito	Descrição
[s1]-1 até [s1]-4	Falta expressão; Expressão indevidamente presente; Ordem errada de expressões; Expressão errada.
[s2]-1 até [s2]-3	Nome de tabela faltando; Nome de tabela presente indevidamente; Nome de tabela errada.
[s3]-1 até [s3]-3	Falta de predicado; Predicado indevidamente presente; Predicado errado.
[s4]-1 até [s4]-5	Falta de lista de expressões; Lista de expressões indevidamente presentes; Falta de expressões; Expressão indevidamente presente; Expressão errada.

continua na próxima página

Tabela 3.5 Lista de Tipos de Defeitos para os comandos INSERT, UPDATE e DELETE (LEITÃO-JUNIOR; VILELA; JINO, 2005)- (continuação)

ID do Defeito	Descrição
[s5]-1 até [s5]-3	Falta de predicado; Predicado indevidamente presente; Predicado errado.
[s6]-1 até [s6]-4	Falta lista de argumentos ordenados; Lista de argumentos presente indevidamente; Ordem errada de argumentos; Argumento errado.
[i1]-1	Nome de tabela incorreto.
[i2]-1 até [i2]-2	Atributo faltando; Atributo presente indevidamente.
[i3]-1 até [i3]-2	Valor errado; Ordem errada de valores.
[i4]-1	Subconsulta com defeito.
[d1]-1	Nome de tabela incorreto.
[d2]-1 até [d2]-3	Falta predicado; Predicado presente indevidamente; Predicado errado.
[u1]-1	Nome de tabela incorreto.
[u2]-1 até [u2]-4	Falta de atribuição; valores setados indevidamente; Expressão errada está na direita; Atributo errado está à esquerda.
[u3]-1 até [u2]-4	Falta de predicado; Predicado presente indevidamente; Predicado errado.

Em Tuya, Suarez-Cabal e De la Riva (2006) foi desenvolvida uma ferramenta chamada *SQLMutation* para gerar automaticamente mutantes de consultas de banco de dados SQL. Os operadores de mutação foram divididos em quatro categorias, conforme exibido na Tabela 3.6 e descritos detalhadamente em Tuya, Suarez-Cabal e De la Riva (2007). A ferramenta foi avaliada por meio de um exercício desenvolvido por sete alunos que introduziram defeitos em um conjunto de consultas SQL de um mesmo esquema de banco de dados e executaram esse conjunto de instruções em uma base de teste criada pelos mesmos. A maioria dos defeitos inseridos pertence à categoria de operadores de mutação SC (cláusula SELECT) e NL (Nulos), que apresentaram os menores escores de mutação, com 59,6% e 78,4% respectivamente, demonstrando que os mutantes gerados a partir dos operadores desta categoria foram mais difíceis de serem mortos. No artigo não foram executados experimentos com aplicações reais, e o conjunto de casos de teste foi insuficiente para uma melhor avaliação da ferramenta.

Tabela 3.6 Operadores de Mutação SQL (TUYA; SUAREZ-CABAL; DE LA RIVA, 2006)

Categoria	**Sigla**	**Descrição**
Cláusula SQL	SC	Executam alterações nas principais cláusulas SQL: SELECT, JOIN, *sub-queries*, GROUP BY, UNION, ORDER BY e funções de agregação.
Operadores de Substituição	OR	Executam as alterações em operadores de modificação de expressões e operadores específicos de condições como LIKE e BETWEEN.
Valores Nulos	NL	Executam as mutações em valores nulos, garantindo a existência de casos de teste para detectar valores nulos tanto nas condições quanto na saída das consultas.
Substituição de Identificadores	IR	Executam substituições em colunas, constantes e parâmetros de consulta que estão presentes tanto na consulta quanto nas tabelas utilizadas pela consulta.

Em Tuya, Suarez-Cabal e De la Riva (2007) foram realizados outros experimentos utilizando a categoria de operadores propostos inicialmente em Tuya, Suarez-Cabal e De la Riva (2006). Este trabalho utilizou uma aplicação comercial em um ambiente de teste chamada *NIST SQL Conformance Test* desenvolvida pela NIST (*National Institute of Standards and Technology*) com o objetivo de avaliar os produtos (módulos) nos quais são realizadas consultas no referido ambiente. Os operadores de mutação foram aplicados nesses módulos representados por pequenos programas com instruções SQL. Neste trabalho os autores geraram outros operadores de mutação para cobrir diferentes características SQL. Experimentos adicionais foram realizados para investigar a redução do custo computacional de execução dos mutantes através de Mutação Seletiva, ou através da priorização de conjunto de casos de teste que matam mutantes de forma mais rápida. No artigo os autores destacaram que devido os experimentos não terem utilizados dados reais, impossibilitou a avaliação comparativa da aplicação desses operadores em aplicações reais e levantou incertezas com relação à cobertura de defeitos representados pelos mesmos.

A Tabela 3.7 apresenta os operadores de mutação SQL que foram aplicados na ferramenta NIST SQL.

Tabela 3.7 Operadores de Mutação SQL (TUYA; SUAREZ-CABAL; DE LA RIVA, 2007)

Categoria	**Tipo**	**Descrição**
SC	SEL (cláusula SELECT)	Alterna entre as palavras chaves SELECT e SELECT DISTINCT. Se ocorrer um SELECT altera para SELECT DISTINCT e vice-versa.
	JOI (cláusula Join)	Substitui cada palavra chave (INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN, CROSS JOIN) por outra da classe JOIN.
	SUB (predicados de subconsultas)	Cada ocorrência de uma palavra chave em um predicado de qualquer tipo é substituído por outra palavra chave do mesmo tipo, exceto a substituição de ANY por SOME.
	GRU(Agrupamentos)	Remove a expressão GROUP BY .
	AGR (Funções de Agregação)	Substitui cada função de agregação (MAX, MIN, AVG, AVG (DISTINCT), SUM, SUM (DISTINCT), COUNT, COUNT (DISTINCT)) por outra.
	UNI (Concatenação de Consultas)	Substitui cada ocorrência de uma palavra chave (UNION e UNION ALL) por outra ou remove as consultas de união.
	ORD (Ordenação do resultado)	Para cada ocorrência de uma palavra chave da expressão ORDER BY substitui as palavras chaves ASC e DESC ou as remove da expressão ORDER BY.
OR	ROR (Substituição de Operador Relacional)	cada ocorrência de um operador relacional {=, <>, <, <=, >,>=} é substituída por outro.
	LCR (Operador de Conexão Lógica)	Cada ocorrência de um operador lógico (AND, OR) é substituído por outro.
	UOI (Operador Unário de Inserção)	Cada expressão aritmética (e) ou referência a um número (e) é substituído por - e, e + 1 e e – 1.

continua na próxima página

Tabela 3.7 Operadores de Mutação SQL (TUYA; SUAREZ-CABAL; DE LA RIVA , 2007)- (continuação)

Categoria	Tipo	Descrição
	ABS (Inserção de Valor Absoluto)	Cada expressão aritmética (e) ou referência a um número (e) é substituído por ABS (e) e - ABS (e).
	AOR (Substituição de Operador Aritmético)	Cada operador aritmético (=, -, *, /, %) é substituído por outro.
	BTW (Predicado BETWEEN)	Cada condição na forma a *BETWEEN* x *AND* y é substituído por a>x *AND* a<=y, e também por a>=x *AND* a<y.
	LIKE (Predicado LIKE)	Executa alterações na expressão a LIKE s exercitando o comportamento dos wildcards {%,_}, no qual o símbolo por cento significa que é para qualquer sequência de caracteres e sublinhado significa para um caractere individual (removendo, substituindo, adicionando opções ao {%,_}.
NL	NLF (Checa predicados nulos)	Cada ocorrência dos predicados IS NULL ou IS NOT NULL é substituída por outra.
	NLS (Nulo em lista de seleção)	Altera cada item na lista de seleção (nomes ou expressões de colunas), gerando mutantes que serão mortos quando esse valor for NULL.
	NLI (Inclusão de Nulos)	Força um valor verdadeiro de uma condição quando existe um valor nulo
	NLO (Outros Nulos)	Cada atributo a na condição C é substituído por NOT C OR a IS NULL, a IS NULL, a IS NOT NULL.
IR	IRC (Substituição de Coluna)	Cada referência à coluna é substituída por outra referência de coluna, constante ou parâmetro que estão presentes na consulta e são de tipos compatíveis.

continua na próxima página

Tabela 3.7 Operadores de Mutação SQL (TUYA; SUAREZ-CABAL; DE LA RIVA , 2007)- (continuação)

Categoria	Tipo	Descrição
	IRT (Substituição de Constantes)	Cada constante é substituída por outra constante, coluna ou parâmetro que estão presentes na consulta e são de tipos compatíveis.
	IRP (Substituição de Parâmetros)	Cada parâmetro de consulta é substituído por outra referência de coluna, constante ou parâmetro que estão presentes na consulta e são de tipos compatíveis.
	IRH (Substituição de colunas ocultas)	Cada referência a um atributo de coluna é substituído por outro tipo compatível definido na tabela, não sendo aplicável para os outros operadores IR, como o IRC, IRT ou IRP.

No trabalho de Shahriar e Zulkernine (2008) foi apresentada uma abordagem para revelar as vulnerabilidades da injeção SQL por meio de testes de mutação. Os autores propuseram nove operadores de mutação, apresentados na Tabela 3.8, que injetam SQL no código-fonte e geram mutantes que podem ser mortos somente com os dados de teste contendo ataques de injeção SQL. Desta forma, eles alcançaram um conjunto de dados de teste capaz de revelar as vulnerabilidades da injeção SQL. Os autores desenvolveram uma ferramenta chamada MUSIC (*MUtation-based SQL Injection vulnerabilities Checking*) para gerar automaticamente os mutantes para aplicações escritas em *Java Server Pages* (JSP) e revelar as vulnerabilidades da injeção SQL. De acordo com os autores, a ferramenta não gera mutantes para *stored procedures* e funciona bem para consultas SQL simples, porém para operadores complexos como UNION e JOIN o método não gera mutante. O trabalho também não analisa as mensagens de erro de banco de dados geradas pelas aplicações. Além disso, a ferramenta não funciona para aplicações distribuídas que executam diferentes operações de banco de dados ao mesmo tempo. Outra limitação é que a ferramenta trabalha apenas com código-fonte desenvolvido em JSP e necessita reforçar os operadores para tratar de outros ataques baseados na Web ligados à injeção SQL (por exemplo, *cross site scripting*).

Em Derezinska (2009) foram avaliados os operadores de mutação para consultas SQL em itens de desempenho e usabilidade. Esses operadores foram aplicados em uma base de dados real de uma empresa de seguros. A autora avaliou a habilidade de

detecção de mutantes por caso de teste e por tempo de execução dos mesmos. Os mutantes foram gerados pela ferramenta *SQLMutation* (TUYA; SUAREZ-CABAL; DE LA RIVA, 2006) totalizando 1.159 mutantes. Para matar os mutantes foi utilizada a comparação de *checksums* calculados por algoritmo *hash* de criptografia SHA-1 sobre os dados obtidos na consulta do *hash* gerado pelo SQL original, o SQL mutante e o tempo médio de execução das consultas no banco de dados. O trabalho teve um custo computacional alto para analisar uma grande quantidade de mutantes.

Tabela 3.8 Operadores de Mutação para Injeção SQL (SHAHRIAR, ZULKERNINE; 2008)

Categoria	**Operador**	**Descrição**
WC (Condições Where)	RMWH	Remove as condições e palavras-chaves do WHERE.
	NEGC	Nega cada expressão unitária dentro das condições WHERE.
	FADP	Adiciona parênteses nas condições WHERE e precede “FALSE AND” depois da palavra-chave WHERE.
	UNPR	Altera os parênteses das expressões de condições WHERE.
AMC (Chamadas do método API em banco de dados)	MQFT	Seta múltiplos indicadores de execução da consulta como TRUE.
	OVCR	Sobrepõem as opções de COMMIT e ROLLBACK.
	SMRZ	Seta o número máximo de registros retornados por um *resultset* para infinito.
	SQDZ	Seta o *delay* de execução da consulta para infinito.
	OVEP	Sobrepõem os *flags* de processamento de caracteres de escape.

Em Cabeça, Jino e Leitão-Junior (2009) foi apresentada uma abordagem que teve como objetivo alcançar efetividade nos testes pela seleção de bases de dados reveladoras de defeitos. Utilizaram o critério análise de mutantes em comandos SQL e discutiram dois cenários de aplicação para as técnicas de mutação forte e fraca. Foi

desenvolvida uma ferramenta para auxiliar na automatização do processo de teste e nos experimentos realizados utilizaram aplicações reais, sendo que os defeitos reais e dados reais, segundo os autores, foram conduzidos para: (i) avaliar a aplicabilidade da abordagem; e (ii) comparar bases de dados de entrada quanto à habilidade para detectar defeitos. Os operadores foram divididos em cinco categorias de acordo com a funcionalidade do comando SQL, conforme apresentado na Tabela 3.9. Alguns operadores de Cabeça, Jino e Leitão-Junior (2009) foram utilizados nesse trabalho para o teste nas consultas SQL do ETL.

Tabela 3.9 Operadores de Mutação SQL (CABEÇA; JINO; LEITÃO-JUNIOR, 2009)

Categoria do Operador de Mutação	**Operador**	**Descrição**
Operadores de SQL	tOpMt	Troca de Operador Matemático.
	tOpCp	Troca de Operador de Comparação.
	tOpCj	Troca de Operador Conjuntivo.
	tOpLg	Troca de Operador Lógico.
	iNot	Inserção de Operador de Negação.
	rNot	Retirada de Operador de Negação.
Operadores de Mutação de Miscelânea	tPoAt	Troca de Posição de Atributo.
	rAtr	Retirada de Atributo.
	iAtr	Inserção de Atributo.
	tAt	Troca de Atributo.
	tPoVr	Troca de Posição de Valor.
	tVr	Troca de Valor.
	tTpVar	Troca de Tipo de Variável.
	tNmTb	Troca de Nome de Tabela.
	tNmRole	Troca de Nome de ROLE.
	iRole	Inserção de ROLE.
	rRole	Retirada de ROLE.
	tNmCursor	Troca de Nome de Cursor.
	tFuAg	Troca de Função de Agregação.
	tInSec	Troca de Intersecção.
	tJoin	Troca de Join.
Operadores de Mutação para Fluxo de Dados	tBlCmEstRep	Troca de Bloco de Comandos nas estruturas de condição e repetição.
	rCmBlRep	Retirada de Comando do Bloco de Repetição/Condição.
	iCmBlRep	Inserção de Comando do Bloco de Repetição/Condição.

continua na próxima página

Tabela 3.9 Operadores de Mutação SQL (CABEÇA; JINO; LEITÃO-JUNIOR, 2009)- (continuação)

Categoria do Operador de Mutação	**Operador**	**Descrição**
	tPosLeave	Troca de Posição do Leave no Bloco de Comandos.
	rLeave	Retirada de LEAVE.
	iLeave	Inserção de LEAVE.
Operadores de Mutação para Controle de Transações	iCM	Inserção de COMMIT.
	rCM	Retirada de COMMIT.
	iRb	Inserção de ROLLBACK.
	rRb	Retirada de ROLLBACK.
	tCmRb	Troca de COMMIT por ROLLBACK.
	tRbCm	Troca de ROLLBACK por COMMIT.
	tNmSP	Troca de Nome do SAVEPOINT.
	tPerm	Troca de Permissão.
	tPriv	Troca de Privilégio.
	tGrRe	Troca de GRANT por REVOKE.
	tReGr	Troca de REVOKE por GRANT.
	tNmUsr	Troca de Nome de Usuário.
Operadores de Mutação para Funções, Procedimentos e *Triggers*	tNm	Troca de Nome da Função, Procedimento, VIEW ou TRIGGER.
	tPoReFu	Troca Posição de Retorno da Função.
	rReFu	Retirada de Retorno da Função.
	tPaPro	Troca de Parâmetros da PROCEDURE (*in* por *out*).
	tEv	Troca de Evento na TRIGGER.

No trabalho de Zhou e Frankl (2011) foi apresentada uma ferramenta chamada *Java Database Application Mutation Analyser* (JDAMA) que estendeu o trabalho desenvolvido por Tuya, Suarez-Cabal e De la Riva (2007) integrando a Análise de Mutantes SQL para análise e modificação do *bytecode* da aplicação. Essa modificação compara os resultados das consultas executadas pela aplicação em teste com os resultados dos mutantes dessas consultas. Desta forma, JDAMA pode ser usada tanto para teste de mutação em aplicações de banco de dados em Java quanto para avaliar as técnicas de teste para aplicações em banco de dados.

Em McCormick, Frakes e Anguswamy (2012) foi apresentado um estudo relatando as capacidades de detecção de defeitos no conjunto de casos de teste de aplicações de banco de dados do mundo real e em um conjunto de teste do fornecedor

SQL (NIST SQL) com base nos escores de mutação. Quanto maior o escore de mutação, mais bem sucedido o conjunto de testes será em detectar defeitos. A ferramenta *SQLMutation* foi utilizada para gerar os mutantes de consulta SQL a partir de esquemas de amostragem obtidos a partir de três fabricantes de banco de dados - *MySQL, SQL Server* e *Oracle*. Quatro operadores *SQLMutation* foram aplicados tanto no mundo real quanto nos conjuntos de teste de conformidade do fornecedor NIST SQL- Cláusula SQL (SC), Substituição do Operador (OR), NULL (NL) e Substituição do Identificador (RI). Os dois operadores de mutação, SC e NL geraram escores de mutação significativamente menores em conjuntos de teste do mundo real do que no conjunto de teste do NIST SQL. O operador RI gerou escores de mutação significativamente mais altos em conjuntos de teste do mundo real do que no conjunto de teste do NIST SQL. O operador OR produziu o mesmo escore de mutação, tanto no mundo real quanto nos conjuntos de teste do NIST SQL.

3.4 PERTURBAÇÃO NOS DADOS

No trabalho de Offutt e Xu (2004), os autores estenderam o conceito de mutações para geração de mutações nos valores dos dados ao invés de apenas realizá-la em programas. A perturbação nos dados utiliza uma entrada de dados de um determinado sistema para gerar um conjunto de outras entradas utilizando operadores de perturbação que modificam o dado original. Essa abordagem foi muito utilizada para testar a interação entre componentes *Web*, tais como *Web Services*, realizando a alteração em documentos *XML* (*eXtensible Markup Language*) e mensagens de requisição *SOAP* (acrônimo para *Simple Object Access Protocol*). Dois tipos de perturbações foram introduzidas: perturbação de valores de dados e de interação.

Perturbação de valores de dados modifica os valores em mensagens *SOAP* de acordo com as regras definidas sobre os tipos dos valores com base em testes de valor limite. A perturbação de interação altera mensagens na comunicação *RPC* usando *SOAP*, que são mensagens com valores para os argumentos de funções de procedimento remoto, e mensagens de comunicação de dados, que são mensagens para a transferência de dados. Um modelo formal para documentos *XML*, denominado *RTG* (*Regular Tree Grammar*), é definido para aplicar a perturbação de dados. Esse modelo representa um *XML Schema* e deriva documentos *XML* baseados nesse esquema. As definições do

esquema, representadas no modelo, são adotadas para realizar as alterações nas mensagens *XML* (OFFUTT; XU, 2004).

O trabalho de Emer (2007) gera uma abordagem genérica de teste baseado em defeitos denominada AIDA (Análise de Instâncias de Dados Alternativas) para aplicação em esquemas de dados de diferentes contextos. Se o esquema estiver incorreto, ou seja, se alguma definição referente aos dados contiver um defeito em relação à especificação dos dados, dados inválidos podem ser aceitos para o processamento ou ainda, dados válidos podem não ser aceitos, também gerando resultados inesperados no processamento da aplicação de software. A autora cita que uma limitação do trabalho foi não ter sido realizada uma comparação dos resultados obtidos da aplicação da AIDA com resultados obtidos pelo teste tradicional da aplicação; por exemplo, com o teste funcional, para verificar se os mesmos defeitos são detectados ou não. O trabalho de Nazar (2007) implementa a ferramenta *XTool* que executou a abordagem de teste criada no trabalho de Emer (2007), testando esquemas de dados a partir de classes de defeitos previamente definidas.

3.5 PROBLEMAS RELACIONADOS À QUALIDADE DE DADOS EM *DATA WAREHOUSE*

Problemas relacionados à qualidade de dados no que tange à fase de limpeza de dados em fontes de dados simples e na integração de múltiplas fontes são classificados por Rahm e Do (2000), fornecendo uma visão geral das principais estratégias de solução para os mesmos. A fase de Limpeza de Dados faz-se necessária principalmente quando existe integração de diversas fontes de dados que devem ser tratadas em conjunto com as transformações de dados relacionados ao esquema. Em ambientes de *Data Warehouses*, por exemplo, a limpeza de dados representa grande parte do processo de ETL. O trabalho de Rahm e Do (2000) classificam os problemas de qualidade de dados relacionados à fase de limpeza dos dados em nível de esquema e instância de dados, tanto para fontes de dados simples quanto para múltiplas fontes.

No trabalho de Barateiro e Gualhardas (2005) apresentou-se uma taxonomia para os problemas de qualidade de dados dividindo-se os problemas de dados em nível de esquema e em instância de dados. Os problemas de qualidade de dados em nível de esquema foram ainda divididos pelos autores em: i) problemas suportados pelo banco de dados, ou seja, que o próprio banco detecta como, por exemplo, as restrições de

integridade; ii) problemas não suportados pelo banco de dados, como por exemplo, categoria de dados errada, atribuir um valor errado ao *range* de uma categoria de cidade e seus respectivos Estados. Os problemas de qualidade de dados em nível de instância de dados também foram divididos pelos autores em problemas que ocorrem em registro único ou problemas que ocorrem em múltiplos registros. Para os problemas que ocorrem em registro único cita-se, por exemplo, a falta de dados corretos para um campo definido como *Not Null*, e para os problemas envolvendo múltiplas instâncias de dados, descreve-se como exemplo, a ocorrência de registros duplicados e a não padronização nos dados. Esta taxonomia dos problemas de qualidade dos dados é genérica, podendo ser aplicada tanto para bases de dados convencionais quanto para *Data Warehouses*.

No trabalho de Singh e Singh (2010) foram descritos os diversos problemas e desafios existentes ao gerar um *Data Warehouse*, visto que todas as fases de desenvolvimento do *Data Warehouse* são responsáveis pela qualidade de dados do mesmo. Esses problemas foram classificados para as seguintes fases: Fontes de Dados, Integração de Dados e *Data Profiling*, *Data Staging* e ETL, Modelagem do *Data Warehouse* e Projeto do Esquema. Os autores identificaram os motivos pelos quais ocorrem esses problemas através de revisão da literatura e entrevistas com profissionais da área destacando que compreender as principais dimensões de qualidade de dados é o primeiro passo para garantir a qualidade dos mesmos. Definiram seis dimensões-chave como critérios de qualidade de dados: Completude (Abrangência), Consistência, Validade, Conformidade, Acurácia (Precisão) e Integridade.

Para Singh e Singh (2010), a principal causa de insucesso em projetos de *Data Warehouse* e BI (*Business Intelligence*) é obter dados errôneos ou de qualidade pobre. Consequentemente, os dados que são armazenados em um DW provém de várias fontes de dados, inclusive de tipos diferentes, e devido a isso, existem muitos fatores que contribuem para problemas de qualidade de dados nas fontes de dados.

Em sua pesquisa, Singh e Singh (2010) destacaram também que a fase de *Staging* e ETL é considerada o estágio mais crucial do *Data Warehouse* na qual reside a responsabilidade máxima dos esforços para qualidade de dados, significando um local privilegiado para a validação da qualidade de dados a partir da fonte, auditoria e rastreamento dos problemas em dados.

A Tabela 3.10 apresenta as causas de problemas de qualidade de dados em *Data Staging* e processo ETL levantadas pelos autores nessa pesquisa e a Tabela 3.11

apresenta as causas de problemas de qualidade de dados na fase de Fontes de Dados no desenvolvimento de um DW.

Tabela 3.10 Problemas de Qualidade dos Dados associados ao Data Staging e ETL

Número	Causas de Problemas de Qualidade de Dados em *Data Staging* e Fase ETL
1	A arquitetura projetada para o *Data Warehouse* afeta a qualidade de dados (*Arquitetura Staging* e *Não Staging*).
2	Tipo de *Staging Area*, relacional ou não relacional afetam a qualidade de dados.
3	Diferentes regras de negócios nas fontes de dados geram os problemas de qualidade de dados.
4	Falta de regras de negócios corrente contribuem para o problema de qualidade de dados.
5	A incapacidade de agendar extrações por hora, intervalo ou evento causam problemas de qualidade de dados.
6	Falta de captura para mudanças em arquivos fontes causa problema de qualidade de dados.
7	Falta de atualização periódica do armazenamento de dados integrados (*Data Staging Area*) causa degradação da qualidade de dados.
8	Truncar os dados na *Staging Area* causa problemas de qualidade de dados, porque não podemos obter os dados de volta para conciliá-los.
9	Desativar as restrições de integridade dos dados nas tabelas do *Data Staging* causa dados e relacionamentos errados para serem extraídos, e portanto, causam problema de qualidade de dados.
10	Eliminação dos dados do *Data Warehouse* causa problemas na qualidade dos dados.
11	Ferramentas ETL codificadas manualmente usadas para o *DW* causam a falta de geração de armazenamento de metadados lógicos simples.
12	Falta de um repositório centralizado de metadados causa má qualidade de dados.
13	Falta de reflexão nas regras estabelecidas para limpeza dos dados em metadados causa dados pobres de qualidade.
14	Preparação imprópria de mapeamento lógico dos dados causa problemas de qualidade dos dados.
15	Má interpretação ou implementação errada das estratégias de dimensões SCD (*Slowly Change Dimensions*) na fase ETL causa grandes problemas de qualidade de dados.
16	Interpretação ou uso inconsistente de códigos, símbolos e formatos causam problemas de qualidade de dados.
17	Extração indevida de dados para campos obrigatórios causa problemas de qualidade de dados.

continua na próxima página

Tabela 3.10 Problemas de Qualidade dos Dados associados ao Data Staging e ETL- (continuação)

Número	Causas de Problemas de Qualidade de Dados em Data Staging e Fase ETL
18	A falta de bom funcionamento da lógica de extração para cada sistema de origem (cargas históricas e incrementais) causa problemas de qualidade de dados.
19	Valores Nulos não tratados no processo ETL causam problemas de qualidade dos dados.
20	Falta de geração de fluxo de dados e documentação da linhagem dos dados por meio do processo ETL causam problemas de qualidade dos dados.
21	Falta de disponibilidade de instalação automatizada de testes de unidade em ferramentas ETL causam problemas de qualidade de dados.
22	Falta de relatório de erros, validação e atualização dos metadados no processo ETL causam problemas de qualidade de dados.
23	Manuseio inadequado de estratégias de reexecução durante o processo ETL causa problemas de qualidade dos dados.
24	Manuseio inadequado de colunas de auditoria, tais como data de criação, data de processamento e data de atualização em ETL causa problemas de qualidade de dados.
25	Inadequado processo ETL para estratégias de atualização (*insert/update/ delete*) leva a problemas na qualidade dos dados.
26	Tipo de estratégia de carga escolhida (Bulk, carga em *batch* ou carga simples) causa problemas de qualidade dos dados.
27	Falta de consideração de regras de negócios pela transformação lógica causa problemas de qualidade dos dados.
28	Convenções de nomenclaturas não padronizadas nos processos do ETL (*Jobs, Sessions, Workflows*) causam problemas de qualidade dos dados.
29	Análise de impacto errada de respostas às mudanças no processo ETL causa problemas na qualidade dos dados.
30	Perda de dados durante o processo ETL (registros rejeitados) causa problemas de qualidade dos dados.
31	Pobres conversões de sistemas, migrações, reengenharia ou consolidação contribuem para problemas na qualidade dos dados.
32	Incapacidade de reiniciar o processo ETL a partir de *checkpoints* sem perda de dados causam problemas de qualidade dos dados.
33	Falta do fornecimento de perfis internos ou integração de perfis de dados de terceiros e ferramentas de limpeza de dados causa problemas de qualidade de dados.
34	Falta de geração automática de regras para ferramentas ETL para construir mapeamentos que detectam e corrigem defeitos dos dados causa problemas de qualidade de dados.

continua na próxima página

Tabela 3.10 Problemas de Qualidade dos Dados associados ao Data Staging e ETL- (continuação)

Número	Causas de Problemas de Qualidade de Dados em Data Staging e Fase ETL
35	Incapacidade de integrar as tarefas de limpeza em fluxos de trabalhos visuais e diagramas causa problemas de qualidade de dados.
36	Incapacidade de possibilitar ferramentas de perfil, limpeza e ETL para a troca de dados e metadados causa problemas de qualidade de dados.

Tabela 3.11 Problemas de Qualidade dos Dados associados às Fontes de Dados

Número	Problemas de Qualidade de Dados em Fontes de Dados
1	Inadequada seleção de fontes de dados candidatas causa problemas de qualidade de dados.
2	Com o tempo, as fontes de dados aumentam, e as chances de obter dados corretos diminuem.
3	A falta de conhecimento adequado da interdependência entre as fontes de dados causa problemas de qualidade de dados.
4	Incapacidade de lidar com dados antigos contribui para problema de qualidade de dados.
5	Variação de "*timeliness*" das fontes de dados causa problema de qualidade de dados.
6	A falta de rotinas de validação em fontes de dados causa problemas de qualidade de dados.
7	Mudanças inesperadas nos sistemas de origem causam problemas de qualidade de dados.
8	Múltiplas fontes de dados geram heterogeneidade semântica que leva a problemas de qualidade de dados.
9	A complexidade de um DW aumenta geometricamente com o espaço de tempo entre os dados a serem armazenados no mesmo e levam a problemas de qualidade de dados.
10	Uso descontrolado de aplicações e bases de dados como fontes de dados para o DW nas organizações e levam a problemas de qualidade de dados.
11	Uso de diferentes formatos de representação em fontes de dados causa problema de qualidade de dados.
12	Erros de Medida causam problemas de qualidade de dados.
13	Descumprimento de dados com as normas em fontes de dados causa problema de qualidade de dados.
14	O insucesso da atualização das fontes de dados em tempo hábil causa problemas de qualidade de dados.

continua na próxima página

Tabela 3.11 Problemas de Qualidade dos Dados associados às Fontes de Dados - (continuação)

Número	Problemas de Qualidade de Dados em Fontes de Dados
15	Falhas na atualização de cópia de dados levam a problemas de qualidade de dados.
16	Presença de registros duplicados em uma ou múltiplas fontes causa problemas de qualidade de dados, podendo ter várias fontes de dados para a mesma informação ou vários registros iguais da mesma entidade.
17	Chaves aproximadas ou substitutas usadas nas fontes de dados causam problemas de qualidade de dados.
18	Informações contraditórias presentes em fontes de dados causam problemas de qualidade de dados.
19	Diferentes formatos de codificação (ASCII, EBCDIC) causam problemas de qualidade de dados.
20	Testes de qualidade de dados insuficientes em fontes de dados individuais levam a uma pobre qualidade nos dados.
21	Falta de domínio do negócio, políticas e planejamento dos dados da empresa contribuem para problemas de qualidade de dados.
22	Colunas que possuem valores de dados incorretos.
23	Formato de dados incorreto/inconsistente (por exemplo, o nome de uma pessoa é armazenado em uma tabela no formato "Nome Sobrenome" e em outra tabela está no formato "Sobrenome, Nome") causa problema de qualidade de dados.
24	Campos da tabela projetados para permitir formas livres.
25	Colunas faltando (por exemplo, precisa-se do nome do meio de uma pessoa., mas não existe coluna para armazenar essa informação).
26	Valores em falta nas fontes de dados causam problemas de qualidade de dados.
27	Erros ortográficos nos dados causam problemas de qualidade de dados.
28	Colunas adicionais causam problemas de qualidade de dados.
29	Múltiplas fontes de dados para o mesmo dado causam problemas de qualidade de dados.
30	Várias estratégias de chaves para o mesmo tipo de entidade, por exemplo, uma tabela que armazena dados de Cliente, usando o CPF como chave, em outra usa *ClientId* como chave, e em outra usa uma chave *surrogate* (substituta) causa problemas de qualidade de dados.
31	Uso indevido de caracteres especiais causa problema de qualidade de dados.
32	Tipos de dados diferentes para as mesmas colunas.
33	Variação de valores padrão usados para a falta de dados.
34	Várias representações do dado em fontes de dados (o dia da semana Terça-feira armazenado como 'T', ou como 'Ter', ou como o número 2, ou ainda como 'Terça').

continua na próxima página

Tabela 3.11 Problemas de Qualidade dos Dados associados às Fontes de Dados- (continuação)

Número	Problemas de Qualidade de Dados em Fontes de Dados
35	Falta de validação no nível de registro em fontes de dados causa problemas de qualidade de dados.
36	Os valores dos dados são diferentes de sua descrição e regras de negócio de campo.
37	Relações inadequadas entre os dados das tabelas.
38	Relacionamento de dados não existente entre dados membros.
39	Não especificação do caractere NULL em fontes de dados de arquivos texto.
40	Delimitador que vem como um caractere em algum campo do arquivo pode representar diferentes significados de dados em relação ao arquivo atual.
41	Número errado de delimitadores nas fontes de dados.
42	Presença de "*outliers*".
43	Dados órfãos ou pendentes.
44	Incompatibilidade entre os dados e metadados.
45	Importantes entidades, atributos e relacionamentos estão escondidos e flutuando em campos texto.
46	Uso inconsistente de caracteres especiais em várias fontes de dados.
47	Campos com múltiplos propósitos presentes em fontes de dados.
48	Erros de entrada de dados deliberativos contribuem para problemas de qualidade de dados.
49	Entrada pobre de dados causa problemas de qualidade de dados.
50	Esquema da tabela indevidamente projetado.
51	Diferentes regras de negócios de várias fontes de dados criam problemas de qualidade de dados.
52	Plausibilidade insuficiente dos dados e nas condições *checks* em aplicações de sistemas.

3.6 CONSIDERAÇÕES FINAIS

Os trabalhos que irão contribuir de forma significativa para o projeto estão relacionados à análise de mutação aplicada a consultas SQL, perturbação nos dados e problemas de qualidade de dados em *Data Warehouse*. Neste capítulo foram apresentadas as principais pesquisas realizadas sobre teste em ambientes de *Data Warehouse*, análise de mutantes SQL e aplicação da análise de mutação em outras áreas. Também foram apresentados trabalhos que tratam da técnica de teste de perturbação nos dados e geração de instâncias de dados alternativas.

Alguns dos trabalhos abordados neste capítulo discutiram a aplicabilidade da técnica de teste baseado em defeitos por meio do critério Análise de Mutantes em diferentes contextos, porém em nenhum dos contextos já pesquisados, havia sido abordado o cenário de *Data Warehouse*. Geralmente os contextos de aplicação da técnica de teste baseado em defeitos são direcionados às linguagens de programação e nesse trabalho está sendo abordado em contexto bem diferente dos já apresentados. O processo de teste ocorre por meio da aplicação de Perturbação nos Dados para os dados das Fontes de Dados e dados do DW e do critério de teste Análise de Mutação para o processo ETL do DW. O trabalho de Emer (2007) contribuiu para que fosse elaborado o processo de teste nos dados das fontes de dados e dados do DW devido a técnica AIDA (Análise de Instâncias de Dados Alternativas).

Os trabalhos que tratam sobre Análise de Mutantes SQL são de suma importância para o desenvolvimento dessa pesquisa, visto que apresentam a aplicação do critério Análise de Mutantes em aplicações de banco de dados que é um contexto semelhante ao da aplicação neste trabalho. O *Data Warehouse* também é considerado uma base de dados, só que uma grande base de dados multidimensional com dados centralizados, na qual a principal diferença em relação aos bancos de dados transacionais é o seu processo de desenvolvimento diferente, que apresentam fases diferentes para o desenvolvimento e possui uma complexidade maior em relação aos bancos de dados comuns. Os operadores de mutação SQL descritos no trabalho de Tuya, Suarez-Cabal e De la Riva (2007) e de Cabeça, Jino e Leitão-Junior (2009) foram utilizados nos experimentos que envolve a fase de teste no ETL do DW.

As pesquisas que abordam os problemas de qualidade de dados foram de fundamental importância para este trabalho, pois a partir da taxonomia dos problemas de qualidade de dados identificados em Barateiro e Gualhardas (2005) e do levantamento dos problemas de qualidade de dados para cada fase do desenvolvimento de um DW em Singh e Singh (2010) foram geradas classes de defeito com regras pré-definidas a fim de testar as fases de desenvolvimento do DW, avaliando a qualidade dos dados nos dados das Fontes de Dados e no DW e também as consultas do processo ETL no DW.

4. ABORDAGEM DE TESTE BASEADO EM DEFEITOS PARA *DATA WAREHOUSE*

Este capítulo apresenta a abordagem de teste baseada em defeitos para DW proposta neste trabalho. Para isso, a classificação de defeitos que podem ser revelados nas fontes de dados, nos dados do DW e também durante a fase de extração, transformação e carga de dados é apresentada. Além disso, o processo de teste, contemplando as três fases de desenvolvimento do DW: Fase de Fontes de Dados, ETL e Dados do DW é proposto.

As classes de defeito apresentadas nas seções seguintes descrevem os principais erros que podem ocorrer ao ser projetado um DW desde a fase inicial, que é a extração de dados das fontes de dados, passando pelo processo ETL, até a fase final, que é representada pelos dados armazenados no DW. Essas classes de defeito estão associadas a defeitos nos dados, das fontes de dados e do DW, e em instruções SQL, da fase de ETL.

4.1 CLASSIFICAÇÃO DE DEFEITOS NOS DADOS

As classes de defeito nos dados foram definidas de acordo com alguns problemas de qualidade de dados para fontes de dados de um DW descritos em Singh e Singh (2010) e, também, com base nos problemas de qualidade de dados descritos em Barateiro e Galhardas (2005). Algumas classes de defeito nos dados, descritas por Emer (2007), também foram relacionadas aos problemas de qualidade de dados em fonte de dados.

Esses problemas de qualidade nos dados foram convertidos para classes de defeito nos dados e posteriormente associados às causas de alguns problemas, de qualidade de dados nas fontes de dados, descritos em Singh e Singh (2010). Essas classes de defeito contêm regras para a geração de instâncias de dados alternativas que são os dados contendo alguma alteração (perturbação nos dados) definida nas classes de defeito.

A seguir são descritas as classes de defeito por meio da identificação (nome da Classe), representação do erro e problema de qualidade de dados associado.

1. Falta de Dados Corretos (FDC): são colunas que foram definidas como *Not Null* no banco de dados, sendo necessário preencher a coluna com um valor, permitindo assim a inserção de um valor incorreto.

-Representação do Erro: a obrigatoriedade do preenchimento de um número de documento de identificação de uma pessoa, por exemplo, o CPF que não foi preenchido no formulário de papel, levando o usuário da aplicação a preencher um código qualquer, como exemplo, 9999999 para que o sistema não apresente um erro.

-Problema(s) de Qualidade de Dados Associado(s):

i) Os valores dos dados são diferentes de sua descrição e regras de negócio de campo;

ii) Campos da tabela projetados para permitir formas livres;

iii) Colunas que possuem valores de dados incorretos;

iv) Várias estratégias de chaves para o mesmo tipo de entidade.

2. Dados Contraditórios (DC): são registros que se referem à mesma entidade, mas que tem algo de contraditório na sua informação.

-Representação do Erro: suponha que a base de dados possua os seguintes registros de determinado funcionário:

Funcionario1 (Nome='Joao Sales', Endereço='Av. Comendador Franco, 223, Uberaba, Curitiba, PR', DataNascimento='12/11/1955')

Funcionario2 (Nome='J. Sales', Endereço='Av. Comendador Franco, 223, Centro, Curitiba, PR', DataNascimento='12/11/1935')

No exemplo dado, o nome do **Funcionario2** aparece abreviado e o ano da data de nascimento também está diferente do **Funcionario1**, sendo que provavelmente trata-se da mesma pessoa;

-Problema(s) de Qualidade de Dados Associado(s):

i) Informações contraditórias presentes em fonte de dados causam problemas de qualidade de dados;

3. Codificação Incorreta (CI): ocorre quando as bases de dados utilizam diferentes formatos de codificação ou representação para os dados, tornando inválida a sua comparação.

-Representação do Erro: O uso de diferentes formatos de codificação, ASCII e UTF-8.

-Problema(s) de Qualidade de Dados Associado(s):

i) Formato de dados incorreto/inconsistente;

ii) Diferentes formatos de codificação (ASCII, EBCDIC);

iii) Uso de diferentes formatos de representação em fontes de dados.

iv) Várias representações do dado em fontes de dados.

4. Valores Nulos (VN): ocorrência de valores nulos quando especificados como valores não nulos.

-**Representação do Erro:** inserção de nulos.

-**Problema(s) de Qualidade de Dados Associado(s)**:

i) Dados órfãos ou pendentes;

ii) Valores ausentes nas fontes de dados.

5. Caracteres de Espaço em Branco Incorretos (CEBI): o tratamento que deve ser dado aos caracteres de espaço em branco, no conteúdo de atributos ou elementos pode estar especificado incorretamente.

-**Representação do Erro:** Inserção de espaços em branco no campo *varchar*, por exemplo.

-**Problema(s) de Qualidade de Dados Associado(s):**

i) Campos da tabela projetados para permitir formas livres.

6. Fora do Domínio (FD): ocorrência de valores diferentes do domínio ou das normas especificadas.

-**Representação do Erro:** inserção de valores inteiros em um tipo *varchar*.

-**Problema(s) de Qualidade de Dados Associado(s):**

i) Colunas que possuem valores de dados incorretos;

ii) Uso indevido de caracteres especiais;

iii) Tipos de dados diferentes para as mesmas colunas;

iv) Várias representações do dado em fontes de dados;

v) Incompatibilidade entre os dados e metadados;

vi) Descumprimento de dados com as normas em fontes de dados;

vii) Esquema da tabela indevidamente projetado.

7. Tamanho Incorreto (TI): a quantidade de caracteres ou dígitos inseridos são menores ou maiores do que o limite máximo e mínimo do tamanho do campo especificado.

-**Representação do Erro:** um tamanho de um tipo *varchar* especificado para 10 caracteres recebendo mais de 10 caracteres na base de dados.

-**Problema(s) de Qualidade de Dados Associado(s):**

i) Campos da tabela projetados para permitir formas livres;

ii) Descumprimento de dados com as normas em fontes de dados;

iii) Esquema da tabela indevidamente projetado.

8. Transposição de Palavras (TP): dados onde as informações estão trocadas.

-Representação do Erro: um determinado registro aparece com informações pessoais como nome, endereço, data de nascimento, no qual o nome está como "João Silva" e em outro registro o nome aparece como "Silva João".

-Problema(s) de Qualidade de Dados Associado(s):

i) Campos da tabela projetados para permitir formas livres;

ii) Os valores dos dados são diferentes de sua descrição e regras de negócio de campo;

9. Valores Embutidos (VE): são referentes a palavras que se adicionam aos campos, porém não fazem parte do mesmo.

-Representação do Erro: suponha que o campo nome tem o seguinte valor: 'Presidente Barack Obama'. Neste exemplo, a palavra 'Presidente' não deveria estar contida no nome da pessoa, pois o campo diz respeito ao nome de uma pessoa e não ao cargo exercido na sociedade;

-Problema(s) de Qualidade de Dados Associado(s):

i) Campos da tabela projetados para permitir formas livres;

ii) Os valores dos dados são diferentes de sua descrição e regras de negócio de campo.

10. Duplicados (DUP): são registros iguais na base de dados, podem até ter um código sequencial diferente.

-Representação do Erro: a inserção de registros iguais na base de dados gera duplicidade de informação, podem ter códigos sequenciais diferentes também, porém corresponde a mesma entidade.

-Problema(s) de Qualidade de Dados Associado(s):

i) Presença de registros duplicados em uma ou múltiplas fontes causam problemas de qualidade de dados, podendo ter várias fontes de dados para a mesma informação ou vários registros iguais da mesma entidade.

11. Conflitos de Estrutura (CE): quando existem representações de um mesmo objeto em bases de dados distintas.

-Representação do Erro: suponha o campo Endereço. Em um registro de uma base de dados o endereço está em um único campo, porém em outra base de dados o endereço está dividido em vários campos.

-Problema(s) de Qualidade de Dados Associado(s):

i) Relações inadequadas entre os dados das tabelas;

ii) Incompatibilidade entre os dados e metadados.

12. Unidade de Medida Incorreta (UMI): ocorre quando unidades de medidas são representadas de forma diferente, embora forneçam a mesma informação, existindo inconsistências na representação das unidades de medida utilizadas em diferentes valores.

-Representação do Erro: Por exemplo, a distância entre localidades pode ser descrita num valor em quilômetros e outro em milhas; ou o tamanho de um objeto pode ser em centímetros ou polegadas, sendo que estes valores estão sendo atribuídos na mesma coluna do banco de dados;

-Problema(s) de Qualidade de Dados Associado(s):

i) Erros de medida

A Tabela 4.1 apresenta sumariamente a relação identificada nesta pesquisa entre os problemas de qualidade de dados nas fontes de dados associados às classes de defeito nos dados.

Tabela 4.1 Problemas de Qualidade de Dados em Fontes de Dados x Classes de Defeito nos Dados

Número	**Problemas de Qualidade de Dados em Fontes de Dados (Singh; Singh, 2010)**	**Classes de Defeito nos Dados**
1	Presença de registros duplicados em uma ou múltiplas fontes causam problemas de qualidade de dados, podendo ter várias fontes de dados para a mesma informação ou vários registros iguais da mesma entidade.	Duplicados

continua na próxima página

Tabela 4.1 Problemas de Qualidade de Dados em Fontes de Dados x Classes de Defeito nos Dados (continuação)

2	Informações contraditórias presentes em fontes de dados causam problemas de qualidade de dados.	Dados Contraditórios
3	Diferentes formatos de codificação (ASCII, EBCDIC).	Codificação Incorreta
4	Colunas que possuem valores de dados incorretos.	Falta de Dados Corretos, Fora do Domínio
5	Formato de dados incorreto/inconsistente	Codificação Incorreta
6	Erros de medida	Unidade de Medida Incorreta
7	Uso indevido de caracteres especiais	Fora do Domínio
8	Tipos de dados diferentes para as mesmas colunas	Fora do Domínio
9	Várias representações do dado em fontes de dados.	Fora do Domínio, Codificação Incorreta
10	Os valores dos dados são diferentes de sua descrição e regras de negócio de campo.	Falta de Dados Corretos, Transposição de Palavras, Valores Embutidos
11	Dados órfãos ou pendentes.	Valores Nulos
12	Várias estratégias de chaves para o mesmo tipo de entidade.	Falta de Dados Corretos
13	Campos da tabela projetados para permitir formas livres.	Falta de Dados Corretos, Caracteres de espaço em branco incorretos, Tamanho Incorreto, Transposição de Palavras, Valores Embutidos
14	Valores ausentes nas fontes de dados.	Valores Nulos
15	Relações inadequadas entre os dados das tabelas.	Conflitos de Estrutura

continua na próxima página

Tabela 4.1 Problemas de Qualidade de Dados em Fontes de Dados x Classes de Defeito nos Dados (continuação)

16	Descumprimento de dados com as normas em fontes de dados.	Fora do Domínio, Tamanho Incorreto
17	Incompatibilidade entre os dados e metadados.	Conflitos de Estrutura, Fora do Domínio
18	Esquema da tabela indevidamente projetado.	Fora do Domínio, Tamanho Incorreto
19	Uso de diferentes formatos de representação em fontes de dados.	Codificação Incorreta.

4.1.1 Operadores de Perturbação Associados às Classes de Defeito nos Dados

Os operadores de perturbação definidos na Tabela 4.2 caracterizam os defeitos que, em geral, são criados por algum defeito na definição dos dados e estão associados às classes de defeito de qualidade de dados.

Para cada operador de perturbação são fornecidos exemplos de alterações pré-definidas nos dados que podem representar o defeito definido pela classe de defeito associada ao operador. Essas alterações geram instâncias de dados alternativas baseadas nos problemas de qualidade nos dados.

Tabela 4.2 Exemplos de Alterações nos Dados para as Classes de Defeito

Classe de Defeito	**Operador de Perturbação**	**Exemplos de Alterações nos Dados**
Falta de Dados Corretos	**FDC**	- Se o campo for uma data, insere uma data inválida. Ex: 07-3000-12; - Se o campo for *time*, insere uma hora errada /inválida. Ex: 25:00:00 - Se o campo for *datetime* insere uma data/hora inválida. Ex: 07-3000-12 00:00:00 - Se o campo for um número inteiro insere um valor errado. Ex: 999999999 - Se o campo for um número flutuante insere um valor incorreto. Ex: 9999.9999 - Se o campo for uma string, insere conjunto com ao menos 1 caractere especial(@#), letras em maiúsculos e dígitos;

continua na próxima página

Tabela 4.2 Exemplos de Alterações nos Dados para as Classes de Defeito (continuação)

Classe de Defeito	Operador de Perturbação	Exemplos de Alterações nos Dados
Dados Contraditórios	**DC**	O registro na base de dados está sendo inserido com as seguintes alterações: - Se o campo for uma *string*/texto (text, varchar, char, nvarchar) - considerando os tipos de atributo do SQL *Server* - substitui a primeira palavra pela letra inicial seguida de ponto. Ex: o nome Maria Luiza será substituído por M. Luiza. Porém, se este campo estiver nulo ou vazio será inserido um ponto ’.’; - Se o campo for uma data ou data/hora subtrai o equivalente a 30 anos. Porém se este campo estiver nulo ou vazio insere uma data. Ex: 3000-12-07;
Codificação Incorreta	**CI**	Inserção de dados no formato UTF-8 quando os dados estão no formato ISO-8859-1
Valores Nulos	**VN**	Altera o conteúdo para valor nulo (*Null*)
Caracteres de Espaço em Branco Incorretos	**CEBI**	Em um campo *string*, o Nome: “João” tem seu conteúdo alterado para “João ” com espaço inserido no final da *string*, ou o contrário, com espaço inserido no início da *string*.
Fora do Domínio	**FD**	O campo *datetime* (data/hora) contendo o valor '2014-07-11 10:00:00' é substituído por um valor de tipo diferente, como por exemplo, *time* (hora) com valor '11:00:00'.
Tamanho Incorreto	**TI**	Um campo com o valor de tamanho máximo 14, por exemplo, um CNPJ, 38599032000128, será substituído por 385990320001282345, ou seja, irá inserir um valor com tamanho 17.
Transposição de Palavras	**TP**	O campo com o valor "Jose Souza" irá gerar um mutante "Souza Jose".
Valores Embutidos	**VE**	- Inserção da palavra ‘Presidente’ no início de um campo do tipo *string*; - Inserção da palavra ‘Senhor’ no campo Nome Pessoa;

continua na próxima página

Tabela 4.2 Exemplos de Alterações nos Dados para as Classes de Defeito (continuação)

Classe de Defeito	Operador de Perturbação	Exemplos de Alterações nos Dados
Duplicados	**DUP**	Funcionario1(ID=1, Nome='Maria Simoes', Endereço='Av. Comendador Franco, 223, Uberaba,Curitiba,PR',DataNascimento='12/11/1955') Funcionario2(ID=2, Nome='Maria Simoes', Endereço='Av. Comendador Franco, 223, Uberaba, Curitiba, PR', DataNascimento='12/11/1955')
Conflitos de Estrutura	**CE**	Inserir endereço completo (nome da rua, número, bairro, CEP, cidade, estado) no campo "Endereço" e em outra tabela tem-se a mesma informação do endereço em uma tabela endereço onde as informações estão também inseridas de forma diferente: Logradouro no campo Logradouro, número no campo número, bairro no campo Bairro, CEP no campo CEP e cidade e estado no campo Estado.
Unidade de Medida Incorreta	**UMI**	Dado uma distância de 417 (km), a mesma é convertida para milhas 259.111787 e depois inserida.

4.2 CLASSIFICAÇÃO DE DEFEITOS NO ETL

Como o processo ETL é composto de instruções SQL executadas nas bases de dados que darão suporte à geração do DW, o presente trabalho propõe classes de defeito ETL baseadas na associação de alguns problemas de qualidade de dados na fase ETL, descritos por Singh e Singh (2010), com alguns operadores de mutação SQL propostos por Tuya, Suarez-Cabal e De la Riva (2007) e Cabeça, Jino e Leitão-Junior (2009).

As classes de defeito a seguir são apresentadas e descritas por meio da identificação (nome da Classe), representação do erro e problema de qualidade de dados associado.

1. Regra de Negócio (RN): relacionadas a procedimentos e verificações que têm a ver com o conteúdo, definindo como devem ser relacionadas entre si as informações, e também com as definições de usuários e suas permissões de acesso. Podem incluir procedimentos automáticos, programados internamente (*triggers*) para verificação e

validação dos dados de acordo com a necessidade do usuário, ou ainda definições de usuários e permissões de acesso, alteração e exclusão das informações.

-Representação do Erro: um erro que pode ocorrer é a troca de um evento em uma *trigger* ou chamada incorreta de uma função ou procedimento; ou então um acesso de permissão errôneo a um usuário.

-Problema(s) de Qualidade de Dados Associado(s):

i) Diferentes regras de negócios nas fontes de dados geram os problemas de qualidade de dados;

ii) Falta de regras de negócios corrente contribuem para o problema de qualidade de dados;

iii) Falta de consideração de regras de negócios pela transformação lógica causam problemas de qualidade dos dados;

iv) A incapacidade de agendar extrações por hora, intervalo ou evento causam problemas de qualidade de dados.

2. Valores Nulos (VN): quando não há tratamento adequado para valores Nulos.

-Representação do Erro: um campo de dados da fonte de origem que está como *Null* deveria ser transformado para "Nenhum" no campo de dados de destino. No entanto, se esta transformação não for implementada, o resultado que sairá nos dados do DW será um campo de dados com valores nulos.

-Problema(s) de Qualidade de Dados Associado(s):

i) Valores Nulos não tratados no processo ETL causam problemas de qualidade dos dados.

3. Perda de Dados (PD): ocorre quando no processo ETL os dados por algum erro não são totalmente capturados, ou então quando ocorre transformação dessa informação, no caso de truncar os dados, ocorrendo perda dos dados.

-Representação do Erro: quando não se leva em consideração determinada condição do dado que não foi tratada no ETL para uma linha específica, e assim, por exemplo, na tabela de vendas será extraído um Relatório de Final de Ano das Vendas que não refletirá a realidade. Outra representação é a questão de não levar em consideração o tamanho do campo na fonte de origem, e na captura dessa informação é inserido um conteúdo com um tamanho menor, por exemplo, pois o campo destino tinha um tamanho menor que o da origem.

-Problema(s) de Qualidade de Dados Associado(s):

i) Truncar os dados na *Staging Area* causam problemas de qualidade de dados, porque não se podem obter os dados de volta para conciliá-los;

ii) Perda de dados durante o processo ETL (registros rejeitados) causam problemas de qualidade dos dados.

4. Codificação Incorreta (CI): ocorre quando há o uso incorreto de códigos, símbolos e formatos.

- Representação do Erro: este erro ocorre quando o código utilizado difere um do outro, campo sexo representado por M ou F e por 0 ou 1.

- Problema(s) de Qualidade de Dados Associado(s):

i) Interpretação ou uso inconsistente de códigos, símbolos e formatos.

5. Atualização Incorreta (AI): podem ocorrer problemas ao inserir, atualizar ou apagar informações do DW, caso não tenham sido escolhidas boas estratégias para isso.

-Representação do Erro: inserir um valor errado ou atualizar uma informação com um valor errôneo.

-Problema(s) de Qualidade de Dados Associado(s):

i) Inadequado processo ETL para estratégias de atualização (*insert/update/delete*) levam a problemas na qualidade dos dados**.**

6. Extração Incorreta (EI): pode ocorrer a seleção e extração de dados incorreta das fontes de dados origem para a fonte de dados destino.

-Representação do Erro: ao selecionar dados de uma tabela fazer a troca de tabela ou de uma função de agregação por exemplo.

-Problema(s) de Qualidade de Dados Associado(s):

i) Extração indevida de dados para campos obrigatórios causam problemas de qualidade de dados;

ii) A falta de bom funcionamento da lógica de extração para cada sistema de origem (cargas históricas e incrementais) causam problemas de qualidade de dados.

7. Auditoria (AUD): ocorre quando há erros no armazenamento das informações nas tabelas de auditoria dos dados.

-Representação do Erro: troca de valores de campos em uma inserção, por exemplo, inserir o valor antigo do campo na tabela de auditoria como se fosse o valor atual e o valor atual como se fosse antigo.

-Problema(s) de Qualidade de Dados Associado(s):

i) Manuseio inadequado de colunas de auditoria, tais como valores antigos e atuais, data de criação, data de processamento e data de atualização em ETL.

A Tabela 4.3 apresenta sumariamente a relação identificada nesta pesquisa entre os problemas de qualidade de dados no ETL e as Classes de Defeito no ETL.

Tabela 4.3 Problemas de Qualidade de Dados no ETL x Classes de Defeito no ETL

Número	**Problemas de Qualidade de Dados em Fase ETL (Singh; Singh, 2010)**	**Classes de Defeito no ETL**
1	Diferentes regras de negócios de várias fontes de dados geram os problemas de qualidade de dados	Regra de Negócio
2	Falta de regras de negócios corrente contribui para o problema de qualidade de dados	Regra de Negócio
3	Truncar os dados na *Staging Area* causa problemas de qualidade de dados, porque não podemos obter os dados de volta para conciliá-los.	Perda de Dados
4	Interpretação ou uso inconsistente de códigos, símbolos e formatos.	Codificação Incorreta
5	Extração indevida de dados para campos obrigatórios causa problemas de qualidade de dados.	Extração Incorreta
6	Valores Nulos não tratados no processo ETL causam problemas de qualidade dos dados	Valores Nulos

continua na próxima página

Tabela 4.3 Problemas de Qualidade de Dados no ETL x Classes de Defeito no ETL (continuação)

Número	Problemas de Qualidade de Dados em Fase ETL (Singh; Singh, 2010)	Classes de Defeito no ETL
7	Manuseio inadequado de colunas de auditoria, tais como valores antigos e atuais, data de criação, data de processamento e data de atualização em ETL	Auditoria
8	Inadequado processo ETL para estratégias de atualização (*insert/update/ delete*) leva a problemas na qualidade dos dados	Atualização Incorreta
9	Falta de consideração de regras de negócios pela transformação lógica causam problemas de qualidade dos dados	Regra de Negócio
10	Perda de dados durante o processo ETL (registros rejeitados) causa problemas de qualidade dos dados.	Perda de Dados
11	A incapacidade de agendar extrações por hora, intervalo ou evento causa problemas de qualidade de dados.	Regras de Negócio
12	A falta de bom funcionamento da lógica de extração para cada sistema de origem (cargas históricas e incrementais) causa problemas de qualidade de dados.	Extração Incorreta

4.2.1 Operadores de Mutação Associados às Classes de Defeito no ETL

Os operadores de mutação apresentados na Tabela 4.4 caracterizam os defeitos que, em geral, são criados por algum defeito no processo do ETL e estão associados às classes de defeito de qualidade de dados. As definições de cada operador de mutação SQL são descritas nas Tabelas 3.7 e 3.9 do Capítulo 3 - Trabalhos Relacionados.

Tabela 4.4 Classes de Defeito ETL x Operadores de Mutação SQL

Classe de Defeito no ETL	**Operadores de Mutação (Cabeça; Jino; Leitão-Junior, 2009; Tuya; Suarez-Cabal; De La Riva, 2007)**
Regra de Negócio (RN)	tNmRole (Troca de Nome de Role), iRole (Inserção de Role), rRole (Retirada de Role), tEv (Troca de Evento), tNm (Troca de Nome)
Extração Incorreta (EI)	tPoAt (Troca de Posição de Atributo), rAtr (Retirada de Atributo), iAtr (Inserção de Atributo), tPoVr (Troca de Posição de Valor), tVr (Troca de Valor), tNmTb (Troca de Nome da Tabela),tFuAg (Troca de função de agregação), JOI (Troca de Join), SEL (SELECT), ABS (Inserção de valores absolutos), UOI (Inserção de Operador Unário), LCR (Operador de Conector Lógico), ROR (Troca de Operador Relacional), IRT/IRC, BTW.
Valores Nulos (VN)	NLF (Null Check Predicates), NLI (Include Nulls), NLO (Other Nulls) e NLS (Null in Select List).
Codificação Incorreta (CI)	tVr (Troca de Valor), tPoVr (Troca de Posição de Valor), iAtr (Inserção de Atributo), rAtr (Retirada de Atributo)
Perda de Dados (PD)	tPoAt (Troca de Posição de Atributo), tVr (Troca de Valor), tPoVr (Troca de Posição de Valor), tNmTb (Troca de Nome da Tabela), rAtr (Retirada de Atributo), tAt (Troca de Atributo)
Atualização Incorreta (AI)	tPoAt (Troca de Posição de Atributo), rAtr (Retirada de Atributo), iAtr (Inserção de Atributo), tPoVr (Troca de Posição de Valor), tVr (Troca de Valor), tAt (Troca de Atributo) e tNmTb (Troca de Nome da Tabela), ROR (Troca de Operador Relacional), LCR(Operador de Conector Lógico)
Auditoria (AUD)	tPoAt (Troca de Posição de Atributo), rAtr (Retirada de Atributo), iAtr (Inserção de Atributo), tPoVr (Troca de Posição de Valor), tVr (Troca de Valor), tAt (Troca de Atributo)

A seguir são apresentados exemplos de defeitos em instruções SQL que podem ocorrer durante a manipulação dos dados no processo ETL. Os operadores de mutação SQL associados aos problemas de qualidade de dados no processo ETL foram aplicados nas instruções SQL originais, sendo apresentada cada instrução SQL original e o seu respectivo mutante. As instruções SQL originais mostradas a seguir foram utilizadas

nos cenários dos experimentos do sistema de vendas e do sistema de empenho descritos no Capítulo 6- Estudos de Caso.

i) Operador: iAtr (Inserção de Atributo)

A instrução a seguir executa uma seleção na tabela *Vendedor*, retornando cinco atributos.

Código Original:

```
SELECT cod_vendedor,nome_vendedor,cod_funcional,data_nasc,salario
 FROM [Vendas].Vendedor;
```

No código mutante, a alteração insere o atributo data_admissao na seleção.

Código Mutante

```
SELECT cod_vendedor, nome_vendedor, cod_funcional, data_nasc,
data_admissao, salario
 FROM [Vendas].Vendedor;
```

ii) Operador: rAtr (Retirada de Atributo)

A instrução a seguir executa uma seleção na tabela *Vendedor*, retornando cinco atributos.

Código Original:

```
 SELECT cod_vendedor,nome_vendedor,cod_funcional,data_nasc,salario
 FROM [Vendas].Vendedor;
```

No código mutante, a alteração retira da seleção o atributo cod_funcional.

Código Mutante:

```
SELECT cod_vendedor,nome_vendedor,data_nasc,salario
FROM [Vendas].Vendedor;
```

iii) Operador: tPoAt (Troca de Posição de Atributo)

A instrução a seguir executa uma seleção na tabela *Vendedor*, retornando cinco atributos.

Código Original:

```
 SELECT cod_vendedor,nome_vendedor,cod_funcional,data_nasc,salario
 FROM [Vendas].Vendedor;
```

No código mutante, a alteração está na ordem de seleção dos atributos.

Código Mutante:

SELECT cod_vendedor,**cod_funcional,nome_vendedor**,data_nasc,salario
FROM [Vendas].Vendedor;

iv) Operador: tPoVr (Troca de Posição de Valor)

A instrução a seguir executa uma inserção na tabela *Vendedor*, inserindo cinco atributos.

Código Original:

INSERT INTO [Vendas].Vendedor (cod_vendedor, nome_vendedor,
cod_funcional, data_nasc,salario)
VALUES (1,'JOSE',1234,'20/05/1980',3500)

No código mutante, houve uma troca entre os valores dos campos cod_funcional e salario sendo inseridos em ordens trocadas.

Código Mutante:

INSERT INTO [Vendas].Vendedor (cod_vendedor, nome_vendedor,
cod_funcional, data_nasc,salario)
VALUES (1,'JOSE',**3500**,'20/05/1980',**1234**)

v) Operador: tVr (Troca de Valor)

Na instrução a seguir é feita uma inserção na tabela *Vendedor*.

Código Original:

INSERT INTO [Vendas].Vendedor
(cod_vendedor,nome_vendedor,cod_funcional,data_nasc,salario)
VALUES (1,'JOSE',1234,'20/05/1980',3500)

No código mutante é feita a troca do valor do campo salario que é um valor do tipo flutuante para NULL.

Código Mutante:

INSERT INTO [Vendas].Vendedor
(cod_vendedor,nome_vendedor,cod_funcional,data_nasc,salario)
VALUES (1,'JOSE',1234,'20/05/1980',**null**)

vi) Operador: iRole (Inserção de Role)

Na instrução a seguir são criadas as regras insere_Produto, calcula_vendas e insere_Departamento sendo atribuído privilégio de seleção e atualização às mesmas.

Código Original:

CREATE ROLE insere_ Produto

CREATE ROLE calcula_Vendas

CREATE ROLE insere_Departamento

GRANT SELECT to insere_Produto, insere_Departamento

GRANT UPDATE to calcula_Vendas

No código mutante gerado o privilégio de atualização também é atribuído à regra insere_Departamento, diferente do código original que somente atribui permissão de atualização à regra calcula_Vendas.

Código Mutante:

CREATE ROLE insere_ Produto

CREATE ROLE calcula_Vendas

CREATE ROLE insere_Departamento

GRANT SELECT to insere_Produto, insere_Departamento

GRANT UPDATE to calcula_Vendas,insere_Departamento

vii) Operador: rRole (Retirada de Role)

Na instrução a seguir são criadas as regras insere_Produto, calcula_vendas sendo atribuído privilégio de seleção às mesmas.

Código Original:

CREATE ROLE insere_Produto

CREATE ROLE calcula_Vendas

GRANT SELECT to insere_Produto,calcula_vendas

O mutante gerado retira a permissão de seleção à regra calcula_Vendas.

Código Mutante:

CREATE ROLE insere_Produto

CREATE ROLE calcula_Vendas

GRANT SELECT to insere_Produto

viii) Operador: tNmRole (Troca de Nome de Role)

Na instrução a seguir são criadas as regras insere_Produto, calcula_venda e insere_Departamento sendo atribuído privilégio de seleção às regras insere_Produto e insere_Departamento no código original.

Código Original:

CREATE ROLE insere_Produto

CREATE ROLE calcula_Vendas

CREATE ROLE insere_Departamento

GRANT SELECT to insere_Produto, insere_Departamento

GRANT UPDATE to insere_Produto,calcula_Vendas

O mutante gerado troca o nome da regra insere_Produto para calcula_Vendas atribuindo permissão de seleção.

Código Mutante:

CREATE ROLE insere_Produto

CREATE ROLE calcula_Vendas

CREATE ROLE insere_Departamento

GRANT SELECT to **calcula_vendas**, insere_Departamento

GRANT UPDATE to insere_Produto,calcula_Vendas

ix) Operador: UOI (Inserção de Operador Unário)

A instrução a seguir executa uma seleção na tabela ammModalidadeLicitacao buscando todas as modalidades de licitação diferentes do valor 9.

Código Original:

SELECT * FROM ammModalidadeLicitacao

WHERE cdModalidadeLicitacao<>9

O mutante gerado adiciona o operador 1 ao código da Modalidade da Licitação (cdModalidadeLicitacao).

Código Mutante:

SELECT * FROM ammModalidadeLicitacao

WHERE **((cdModalidadeLicitacao)+1)**<>9

x) Operador: LCR (Operador de Conector Lógico)

A instrução a seguir utiliza a operação INNER JOIN para combinar (relacionar) todas as linhas da tabela ammAlteracaoOrcamentaria com a tabela pePrestaContas satisfazendo as condições das chaves e também a condição do nrAnoOrcamento (Número ano orçamento) que devem ser a partir do ano de 2005.

Código Original:

SELECT P.* FROM ammAlteracaoOrcamentaria P

INNER JOIN pePrestaContas PC on P.idPessoa=pc.idPessoa AND p.nrAnoOrcamento=year(pc.dtInicio)

WHERE p.nrAnoOrcamento >=2005

O mutante gerado altera o operador lógico AND para OR.

Código Mutante:

SELECT P. * FROM ammAlteracaoOrcamentaria P

INNER JOIN pePrestaContas PC on P.idPessoa=pc.idPessoa **OR** p.nrAnoOrcamento=year(pc.dtInicio)

WHERE p.nrAnoOrcamento >=2005

xi) Operador: ROR (Troca de Operador Relacional)

A instrução a seguir executa uma seleção na tabela ammModalidadeLicitacao buscando todas as modalidades de licitação diferentes do valor 9.

Código Original:

SELECT * FROM ammModalidadeLicitacao

WHERE cdModalidadeLicitacao<>9

O mutante gerado altera o operador relacional <> para =.

Código Mutante:

SELECT * FROM ammModalidadeLicitacao

WHERE cdModalidadeLicitacao=9

xii) Operador: JOI (Operador Troca de Join)

A instrução a seguir utiliza a operação JOIN para combinar (relacionar) todas as linhas da tabela ammAlteracaoOrcamentaria com a tabela pePrestaContas satisfazendo as condições das chaves e também a condição do nrAnoOrcamento (Números Orçamento) que devem ser a partir do ano de 2005.

Código Original:

SELECT P.* FROM ammAlteracaoOrcamentaria P

JOIN pePrestaContas PC on P.idPessoa=pc.idPessoa

AND p.nrAnoOrcamento=year(pc.dtInicio)

WHERE p.nrAnoOrcamento >=2005

O mutante gerado altera o operador de junção JOIN para LEFT JOIN.

Código Mutante:

SELECT P.* FROM ammAlteracaoOrcamentaria P

LEFT JOIN pePrestaContas PC on P.idPessoa=pc.idPessoa

AND p.nrAnoOrcamento=year(pc.dtInicio)

WHERE p.nrAnoOrcamento >=2005

xiii) Operador: IRT (Troca de Constante)

A instrução a seguir seleciona em uma tabela o valor 2, correspondente a O e Obras e Serviços de Engenharia.

Código Original:

SELECT 2, ‘O’, ‘Obras e Serviços de Engenharia’

O mutante gerado altera o valor da constante “Obras e Serviços de Engenharia’ para ‘O’.

Código Mutante:

SELECT 2, ‘O’,’O’

xiv) Operador: SEL (Operador de Seleção)

A instrução a seguir executa uma seleção na tabela ammfuncao buscando todas as informações a partir do ano 2005.

Código Original:

SELECT * FROM ammfuncao

WHERE nrano>=2005

O mutante gerado adiciona o comando DISTINCT para a seleção.

Código Mutante:

SELECT **DISTINCT** * FROM ammfuncao

WHERE nrano>=2005

xv) Operador: ABS (Operador de Inserção de Valor Absoluto)

A instrução a seguir executa uma seleção na tabela ammfuncao buscando todas as informações a partir do ano 2005.

Código Original:

SELECT * FROM ammfuncao

WHERE nrano>=2005

O mutante gerado adiciona o comando ABS para a seleção.

Código Mutante:

```
SELECT * FROM ammfuncao
WHERE ABS(nrano)>=2005
```

vxi) Operador: Troca de Nome da Tabela (tNmTb)

A instrução a seguir utiliza a operação JOIN para combinar (relacionar) todas as linhas da tabela ammAlteracaoOrcamentaria com a tabela pePrestaContas satisfazendo as condições das chaves e também a condição do nrAnoOrcamento (Números Orçamento) que devem ser a partir do ano de 2005.

Código Original:

```
SELECT P.* FROM ammAlteracaoOrcamentaria P
JOIN pePrestaContas PC on P.idPessoa=pc.idPessoa
AND p.nrAnoOrcamento=year(pc.dtInicio)
WHERE p.nrAnoOrcamento >=2005
```

O mutante gerado realizou a troca de nome entre as tabelas ammAlteracaoOrcamentaria e pePrestaContas.

Código Mutante:

```
SELECT P.*
FROM pePrestaContas P
JOIN ammAlteracaoOrcamentaria PC
ON P.idPessoa=pc.idPessoa
AND p.nrAnoOrcamento=year(pc.dtInicio)
WHERE p.nrAnoOrcamento >=2005
```

4.3 PROCESSO DE TESTE

A Figura 4.1 apresenta o processo do teste baseado em defeitos em uma arquitetura de *Data Warehouse*, que está dividido em 3 fases distintas. Estas fases têm por objetivo verificar a qualidade dos dados e das instruções SQL empregadas para gerar o *Data Warehouse*.

Na Fase 1 são geradas instâncias de dados alternativas para verificar a qualidade dos dados das fontes de dados. As consultas SQL utilizadas no processo ETL são testadas por meio da geração de mutantes SQL na Fase 2. Na Fase 3, a qualidade dos dados armazenados no *Data Warehouse* é testada por meio da análise de instâncias de dados alternativas. Estas fases do teste de DW são detalhadas a seguir.

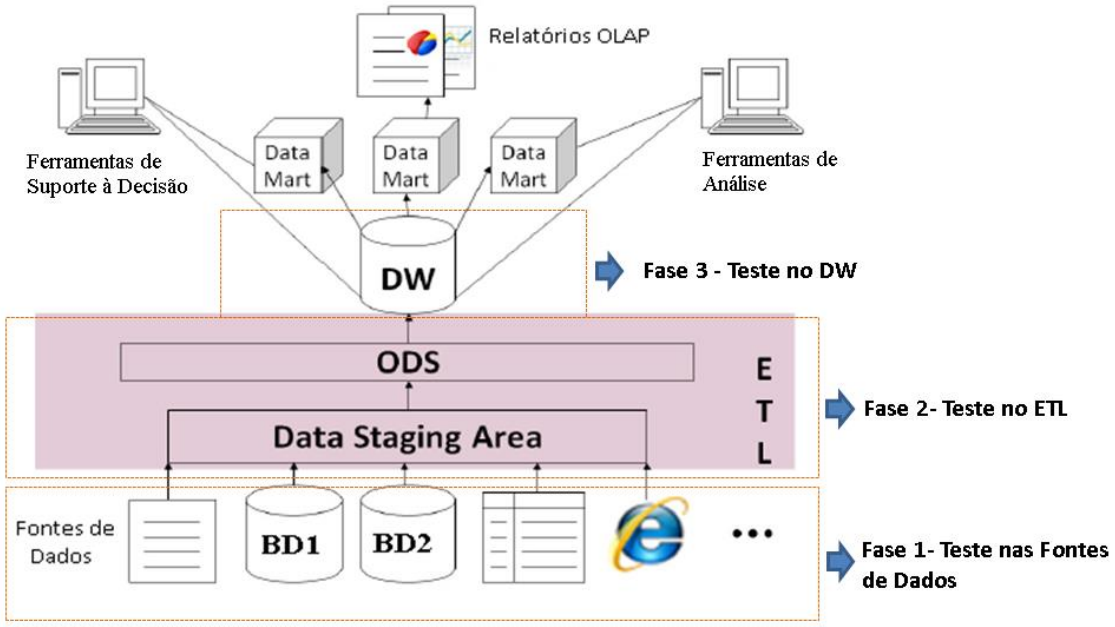


Figura 4.1 Fases do Teste Baseado em Defeitos em uma Arquitetura de *Data Warehouse*- Adaptado de ElGamal (2015)

4.3.1 Fase 1- Teste nas Fontes de Dados

No processo de teste (Fase 1), o esquema do banco de dados e os dados de entrada das fontes de dados são fornecidos como informações de entrada para o teste pelo testador (Figura 4.2).

São geradas instâncias de dados com alterações (chamadas de instâncias de dados alternativas) realizadas de acordo com as classes de defeito pré-definidas na Seção 4.1. Essas instâncias de dados com defeitos são inseridas no banco de dados a fim de verificar se o mesmo irá aceitá-las ou não. O resultado deste processo deverá ser analisado pelo testador, que verifica quais valores aceitos e quais valores rejeitados revelam de fato defeitos nos dados das fontes de dados de acordo com a especificação correta dos dados.

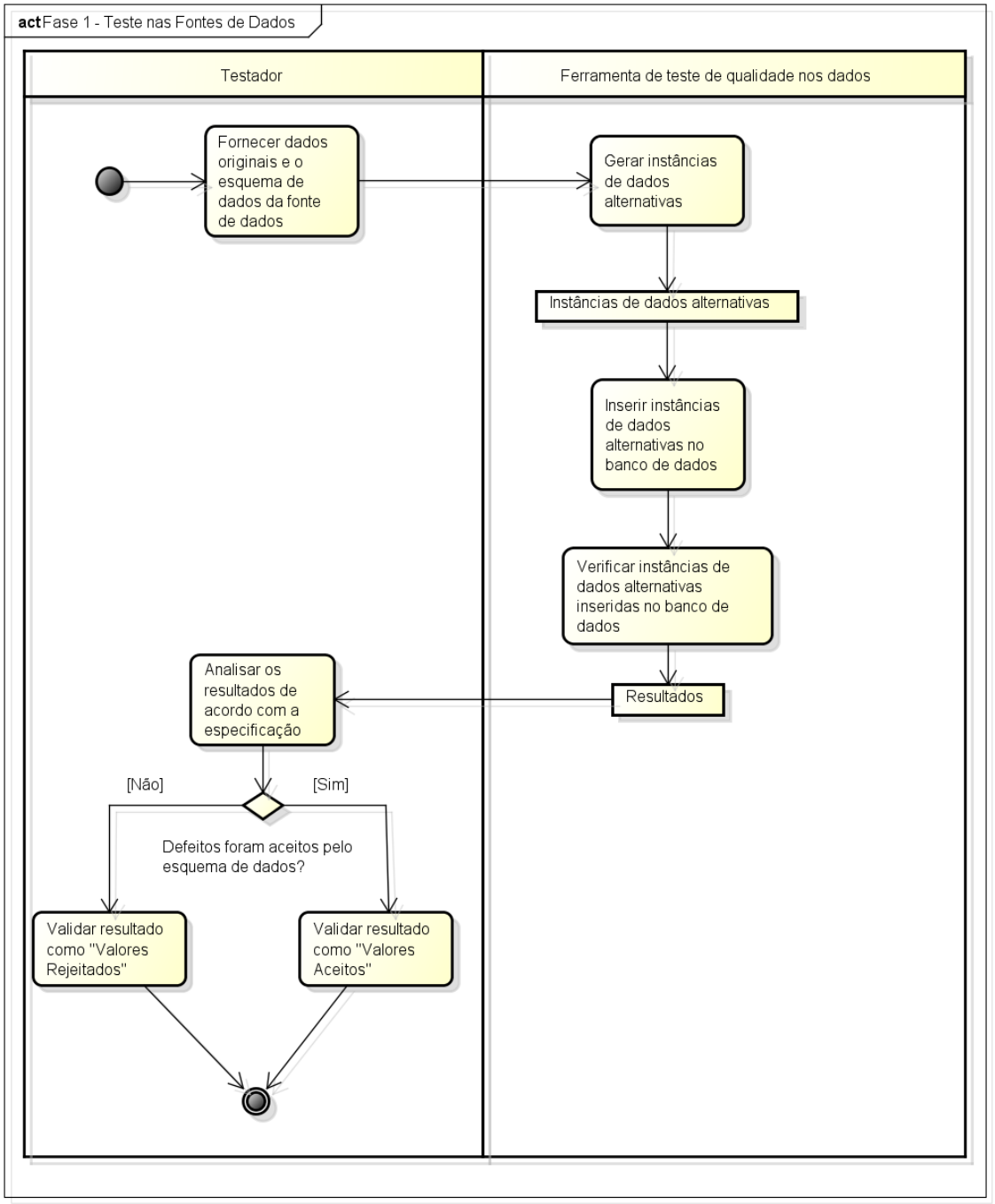


Figura 4.2 Processo de Teste na Fase 1- Teste nas Fontes de Dados

4.3.2 Fase 2 - Teste no ETL

No processo de teste (Fase 2), as consultas utilizadas no processo ETL são fornecidas como informações de entrada para o teste pelo testador (Figura 4.3). São geradas instruções SQL mutantes alteradas de acordo com as classes de defeitos e operadores de mutação SQL definidos em Cabeça, Jino e Leitão-Junior (2009) e Tuya, Suarez-Cabal e De La Riva (2007). Essas instruções SQL mutantes são executadas em

instâncias da base de dados original a fim de verificar o retorno dos dados. No caso de uma instrução SQL mutante retornar dados iguais aos da instrução original, diz-se que o mutante é Vivo. No caso de retornar dados diferentes diz-se que os mutantes gerados são Mortos. O resultado deve ser analisado pelo testador a fim de verificar se a consulta é um mutante vivo ou um mutante equivalente. Em caso de ser um mutante Vivo, o testador deverá decidir se deve ou não continuar os testes. No caso de realizar uma nova "rodada" de testes, outras instâncias de dados deverão ser utilizadas com o objetivo de matar esses mutantes. Ao final, esses mutantes podem permanecer vivos ou serem considerados equivalentes após análise do testador.

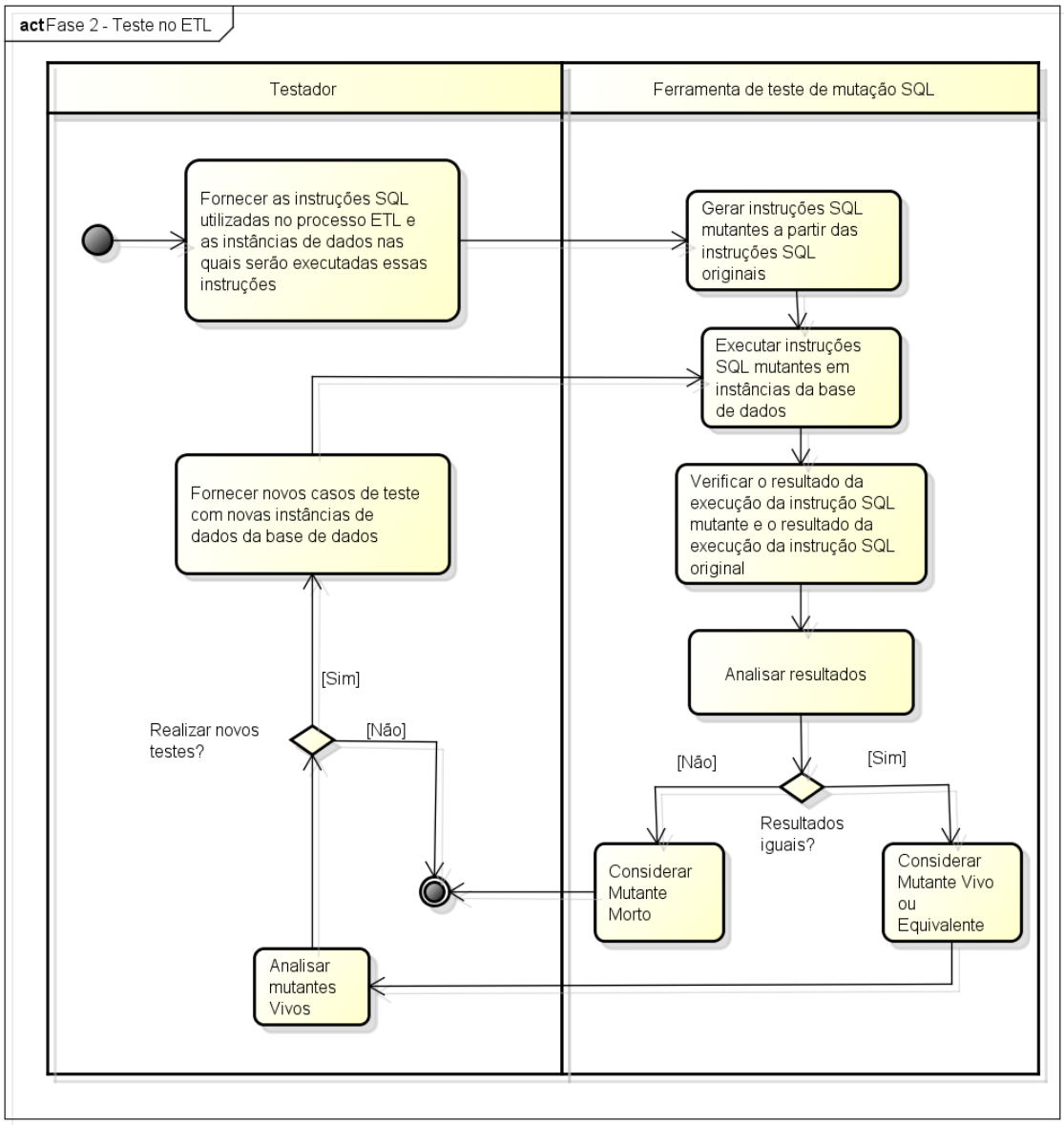


Figura 4.3 Processo de Teste na Fase 2- Teste no ETL

4.3.3 Fase 3 – Teste no DW

No processo de teste (Fase 3), o esquema do banco de dados do DW e os dados de entrada do DW são fornecidos como informações de entrada para o teste pelo testador (Figura 4.4). Nesta fase foram utilizadas as mesmas classes de defeito geradas para a Fase 1, só que agora para a geração de instâncias de dados alternativas para o DW.

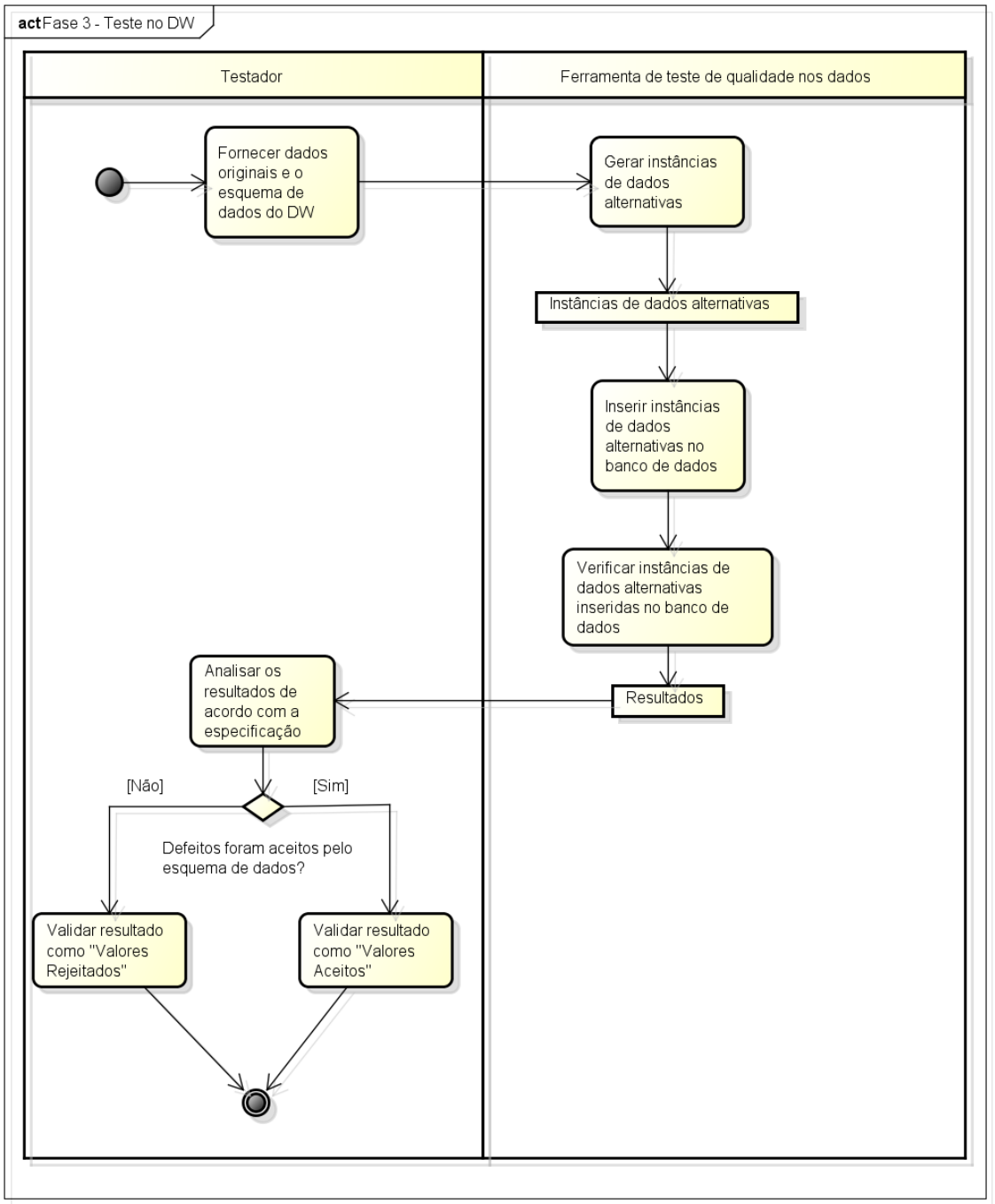


Figura 4.4 Processo de Teste Fase 3 - Teste no DW

São geradas instâncias de dados com alterações (chamadas de instâncias de dados alternativas) realizadas de acordo com as classes de defeitos pré-definidas na seção 4.1. Essas instâncias de dados com defeitos são inseridas no DW a fim de verificar se o mesmo irá aceitá-las ou não. O resultado deste processo deverá ser analisado pelo testador que deve verificar quais valores aceitos e quais valores rejeitados revelam de fato defeitos nos dados do DW de acordo com a especificação correta dos dados.

4.4 CONSIDERAÇÕES FINAIS

Neste capítulo foram definidas classes de defeito para os dados e para o ETL com base nos trabalhos de Barateiro e Galhardas (2005), Singh e Singh (2010), Emer (2007), Tuya, Suarez-Cabal e De La Riva (2007) e Cabeça, Jino e Leitão-Junior (2009).

Os problemas de qualidade de dados na fase de Fonte de Dados foram analisados e relacionados às classes de defeito nos dados. Essas classes de defeito contêm regras para alteração nos dados gerando instâncias de dados alternativas que foram testadas nas fontes de dados. As classes de defeito nos dados também foram aplicadas aos dados do DW.

Os problemas de qualidade de dados na Fase de ETL foram analisados e relacionados às classes de defeito no ETL. Essas classes de defeito foram analisadas e relacionadas a alguns operadores de mutação SQL propostos na literatura por Tuya, Suarez-Cabal e De La Riva (2007) e Cabeça, Jino e Leitão-Junior (2009).

A arquitetura de um DW foi associada a um processo de teste baseado em defeitos em instruções SQL do ETL e em dados da Fonte de Dados e do DW , dividido de acordo com as três fases de desenvolvimento de um DW.

As três fases de teste compreendem um processo abrangente de teste, no qual devem ser testadas as principais fases de desenvolvimento do *Data Warehouse* responsáveis pela integridade e confiabilidade dos dados nesses ambientes: Fontes de Dados, ETL e o próprio DW, que se não forem testadas podem causar problemas de qualidade nos dados, conforme descrito neste trabalho.

No próximo capítulo são abordados os estudos de caso realizados, apresentando os resultados obtidos com a execução do processo de teste baseado em defeitos no contexto de *Data Warehouse* proposto no presente trabalho.

5. ESTUDOS DE CASO

5.1 CONTEXTUALIZAÇÃO

Neste capítulo são apresentados três estudos de caso realizados com o objetivo de investigar a aplicabilidade e eficácia do teste baseado em defeitos no contexto de *Data Warehouse*, considerando a aplicação de mutação SQL na fase de ETL e a análise de instâncias de dados alternativas nas fontes de dados e nos dados do *Data Warehouse*.

Os estudos de caso foram realizados, inicialmente, em um sistema de vendas, contendo dados fictícios, para verificar a aplicabilidade do processo de teste proposto e, posteriormente, em dois sistemas de um órgão público, contendo dados reais, para verificar a eficácia do processo de teste proposto.

Devido à quantidade de mutantes gerados a partir das classes de defeito nos dados ou dos operadores de mutação SQL tornou-se necessário o desenvolvimento de ferramentas para automatizar a geração, execução e a análise de mutantes e instâncias de dados alternativas. Com isso, os estudos de caso, em todo o processo de teste, foram apoiados por uma ferramenta de teste de mutação SQL e também por uma ferramenta para teste da qualidade dos dados.

A ferramenta utilizada para análise da qualidade dos dados teve suas classes de defeito baseadas nos problemas de qualidade de dados descritos em Barateiro e Gualhardas (2005) e a ferramenta desenvolvida para análise dos mutantes SQL teve sua implementação baseada nos operadores de mutação SQL descritos nos trabalhos de Tuya, Suarez-Cabal e De la Riva (2007) e Cabeça, Jino e Leitão-Junior (2009). A ferramenta de qualidade nos dados é descrita no Apêndice A e a ferramenta de Teste de Mutação SQL é descrita no Apêndice B. As seções seguintes apresentam os estudos de caso realizados.

5.2 ESTUDO DE CASO I - SISTEMA DE VENDAS

5.2.1 Descrição

O sistema de Vendas é um sistema gerado com dados fictícios criado inicialmente com a finalidade de verificar a aplicabilidade do processo de teste proposto para *Data Warehouse.*

O sistema utiliza a plataforma Banco de Dados *SQL Server 2012*. Tanto a fonte de dados (tabelas Relacionais) quanto o DW (Tabelas Fatos e Dimensões em modelo Estrela) do sistema de vendas foram desenvolvidos em *SQL Server*.

A seguir é apresentado o modelo relacional de dados do sistema de vendas, com as tabelas Produto, Departamento, Vendedor e Vendas (Figura 5.1) e o modelo dimensional de dados Estrela do *Data Warehouse* referente a este sistema (Figura 5.2) com as tabelas dimensões DIM_VENDEDOR, DIM_PRODUTO, DIM_DEPARTAMENTO, DIM_DATA e a tabela fato FATO_VENDAS.

Para o processo de teste foram utilizadas 4 tabelas da base de dados relacional e 5 tabelas do DW. Todas as tabelas, tanto da fonte de dados como do DW foram duplicadas (somente estrutura de dados) para uma base de dados teste para que o estudo de caso fosse realizado. Após isso, executou-se o processo de teste descrito nas três fases da próxima seção.

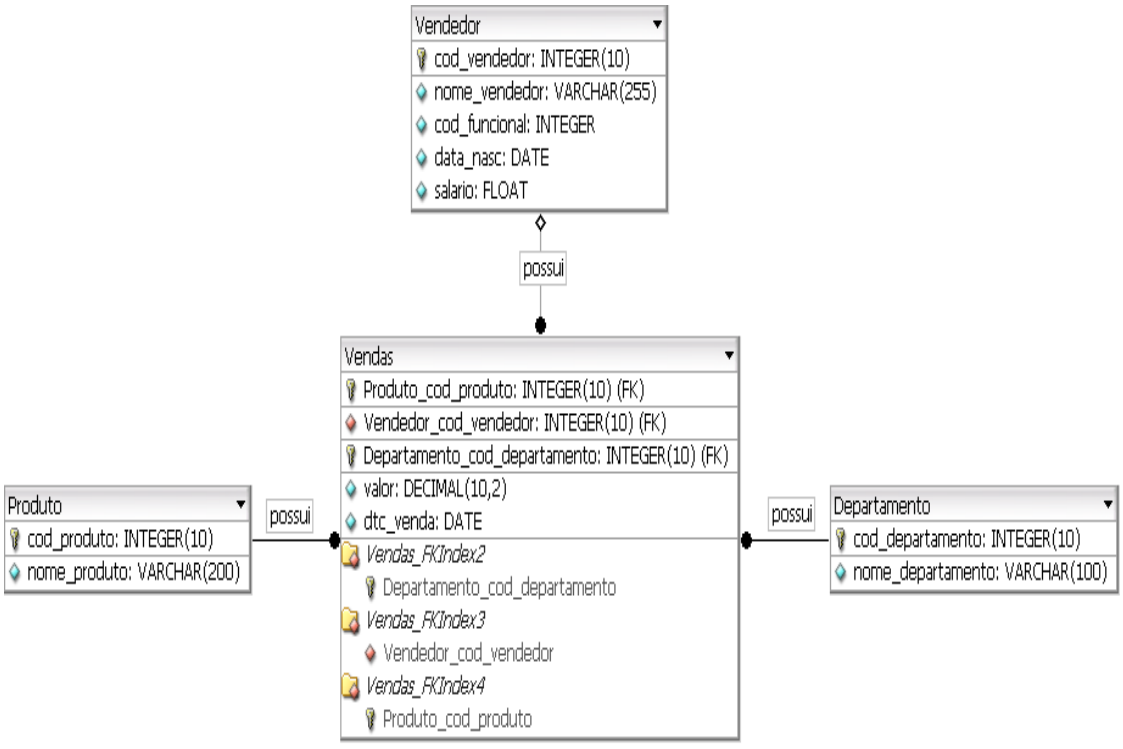


Figura 5.1 Modelo Relacional Sistema de Vendas

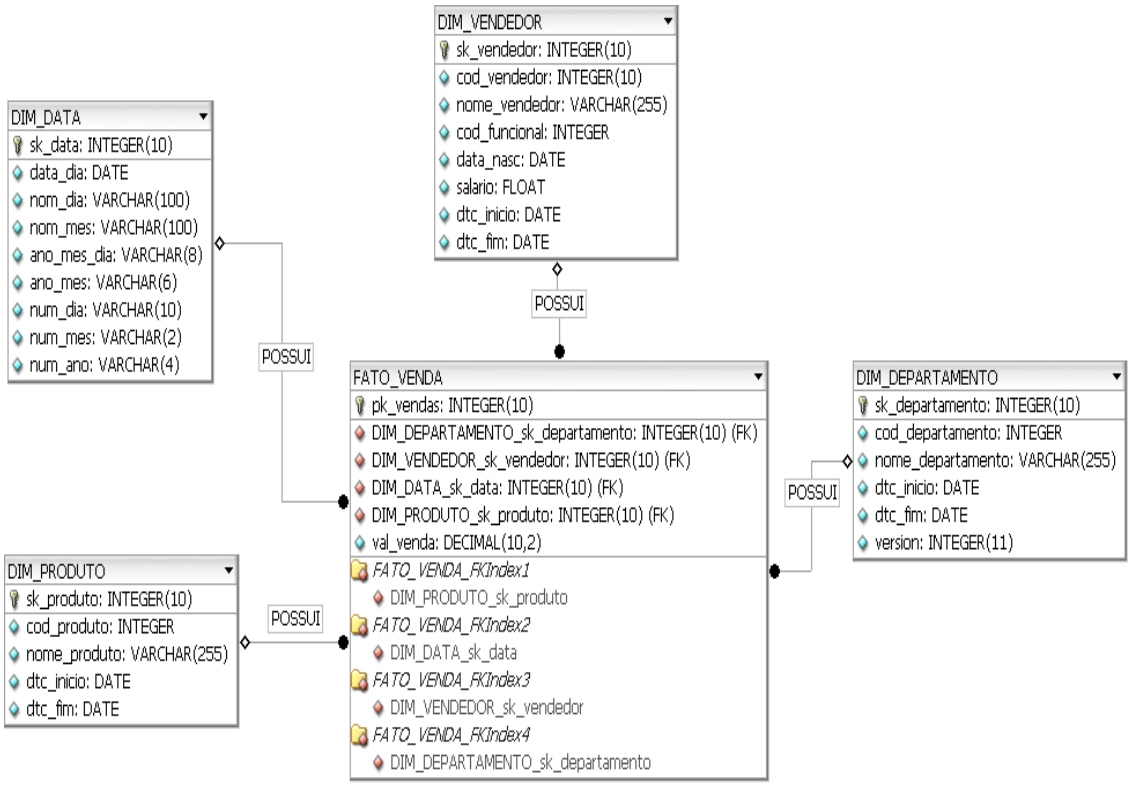


Figura 5.2 Modelo Dimensional Estrela do Sistema de Vendas

5.2.2 Fases do Processo de Teste

Fase 1: Teste na Fonte de Dados

No processo de teste, são fornecidos como entrada o esquema do banco de dados (tabelas relacionais) utilizado pelo sistema de vendas e os dados da fonte de dados.

Ao receber as instâncias de dados, a ferramenta que avalia a qualidade nos dados gera automaticamente instâncias de dados com alterações (chamadas de instâncias de dados alternativas) de acordo com as classes de defeito pré-definidas. Essas instâncias de dados são geradas a partir dos 10 primeiros registros de cada tabela da fonte de dados original, considerando 100% dos atributos de cada tabela quando possível, ou seja, o defeito de cada classe foi aplicado para todos os atributos de determinada tabela quando viável a sua aplicação, pois a aplicação das alterações relacionadas às classes de defeito depende das restrições e definições associadas aos atributos das tabelas.

Os dados com defeitos são inseridos no banco de dados teste que contém uma réplica do esquema de dados da fonte de dados a fim de verificar se o mesmo irá aceitá-los ou não. Esse teste de validação verifica se os dados seguem regras estabelecidas pelo

esquema, caso não siga, e após verificação do testador, significa que existem restrições ausentes ou incorretas no esquema dos dados para a fonte de dados em teste. O resultado desse processo foi analisado a fim de verificar quais valores aceitos e quais valores rejeitados revelaram de fato defeitos no esquema de dados da fonte de dados e, consequentemente, possíveis defeitos nos dados das fontes de dados de acordo com a especificação correta dos dados.

A Tabela 5.1 apresenta exemplos de instâncias de dados alternativas inseridas no banco de dados teste da fonte de dados.

Tabela 5.1 Exemplos dos Defeitos inseridos para as Classes de Defeito nos Dados da Fonte de Dados

Classes de Defeitos nos Dados	**Dado Original**	**Exemplos de Instâncias de Dados Alternativas Geradas**
Caracteres de Espaço em Branco Incorretos	['Ana Luiza'] ['Celular LG']	[' Ana Luiza'] [' Celular LG']
Codificação Incorreta	['Ana Luiza'] ['Celular LG']	['Nan Yhvmn'] ['Pryhyne YT']
Dados Contraditórios	['Ana Luiza'] ['Celular LG']	['A. Luiza'] ['Celular GL']
Duplicado	['Ana Luiza'] ['Celular LG']	['Ana Luiza'] ['Celular LG']
Falta de Dados Corretos	[' 49644705637'] ['861.6745']	['99999999999'] ['999.9999']
Fora do Domínio	['Ana Luiza'] ['Celular LG']	[4299.0) ['2015-07-08']
Nulos	['Ana Luiza'] ['Celular LG']	[NULL] [NULL]
Tamanho Incorreto	['Ana Luiza'] ['Celular LG']	('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa') ('aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa')
Transposição de Palavras	['Ana Luiza'] ['Celular LG']	['Luiza Ana'] ['LG Celular']
Valores Embutidos	['Ana Luiza'] ['Maria Assis']	['Presidente Ana Luiza'] ['Senhora Maria Assis']

Fase 2: Teste no ETL

Nessa fase de teste foram utilizados 10 comandos de manipulação SQL utilizados no processo ETL do DW e que correspondem às regras das classes de defeitos no ETL. Para a execução desta fase foi considerada uma Base de Dados de Teste (BDT) contendo as tabelas relacionais e os dados copiados do banco de dados original sem nenhuma alteração/redução dos dados.

Os operadores de mutação SQL são executados somente se for possível a sua aplicabilidade aos comandos selecionados, pois existem alguns operadores de mutação SQL que não se aplicam a determinadas consultas, como por exemplo, o operador de mutação JOI, que realiza troca de *join* nas consultas e, portanto, somente pode ser aplicado em consultas que possuem *JOIN*.

Os operadores de mutação SQL utilizados nesse estudo de caso foram: SEL (SELECT), ROR (substituição de operador relacional), LCR (substituição de operador lógico), tNmTb (troca de nome de tabela), tNmRole (troca de nome de role), iRole (inserção de role), rRole (retirada de role), tPoAt (troca de posição de atributo), iAtr (inserção de atributo), rAtr (retirada de atributo), tPoVr (troca de posição de valor), tAt (troca de atributo), tVr (troca de valor), ABS (inserção de valor absoluto), UOI (inserção de operador unário), IRC (substituição de coluna), NLI/NLO (nulos na entrada de dados).

Após a execução das consultas mutantes nas instâncias da base de dados, a ferramenta realiza uma análise automática do resultado das consultas avaliando se os mutantes ficaram mortos ou vivos. Para isso, a ferramenta realiza uma comparação entre os resultados gerados pela execução da consulta original e os resultados gerados pela consulta mutante. No caso do resultado da consulta ser diferente do resultado da consulta original considera-se mutante “Morto”. No caso dos resultados dessas consultas serem iguais, considera-se mutante “Vivo” e no caso de mutantes vivos o testador deverá analisar se o mutante é equivalente ou não, caso não seja equivalente, ou seja, o mutante continuou “Vivo”, o testador decide se deve ou não continuar os testes, sempre levando em consideração o escore de mutação alcançado para o teste.

A quantificação dos resultados dos testes é avaliada por meio do escore de mutação gerado para cada caso de teste executado no estudo de caso. Quanto mais próximo de 1 melhor o conjunto de casos de teste utilizado para revelar os defeitos. Os resultados serão apresentados e discutidos posteriormente na seção Resultados.

Fase 3: Teste no DW

Nesta fase do processo de teste proposto, são fornecidos como entrada o esquema do banco de dados do DW utilizado no sistema de vendas e os dados do DW.

Após receber as instâncias de dados, a ferramenta de qualidade de dados gerou as instâncias de dados com as alterações (instâncias de dados alternativas) de acordo com as classes de defeito pré-definidas. Essas instâncias de dados também foram geradas a partir dos 10 primeiros registros de cada tabela do DW, considerando 100% dos atributos de cada tabela, quando possível, conforme explicado anteriormente.

Os dados com defeitos são inseridos no banco de dados teste que contém uma réplica do esquema de dados do DW a fim de verificar se o mesmo irá aceitá-los ou não. Esse teste de validação verifica se os dados seguem regras estabelecidas pelo esquema, caso não siga, e após verificação do testador, significa que existem restrições ausentes ou incorretas no esquema dos dados para a fonte de dados em teste. O resultado desse processo foi analisado a fim de verificar quais valores aceitos e quais valores rejeitados revelaram de fato defeitos no esquema de dados do DW e assim possíveis defeitos nos dados do DW de acordo com a especificação correta dos dados.

5.2.3 Resultados

Os resultados obtidos para cada fase do processo de teste no sistema de vendas são apresentados detalhadamente a seguir.

Fase 1: Teste na Fonte de Dados do Sistema de Vendas

O resultado da aplicação das classes de defeito nos dados da fonte de dados na base de teste do sistema de vendas resultou na Tabela 5.2. Essa tabela apresenta o total de instâncias de dados alternativas geradas para cada classe de defeito, e deste total quantos defeitos foram considerados como aceitos e quantos defeitos foram rejeitados no processo de teste.

Tabela 5.2 Quantidade de Defeitos por Classes de Defeito na Fonte de Dados do Sistema de Vendas

Classes de Defeito nos Dados	Total de Instâncias de Dados Alternativas	Total de Valores Aceitos	Total de Valores Rejeitados
Caracteres de Espaço em Branco Incorretos	70	70	0
Codificação Incorreta	70	70	0
Dados Contraditórios	130	80	50
Duplicados	40	30	10
Falta de Dados Corretos	130	80	50
Valores Embutidos	560	560	0
Fora do Domínio	910	490	420
Nulos	130	0	130
Tamanho Incorreto	120	0	120
Transposição de Palavras	70	70	0

A seguir, a Figura 5.3 apresenta graficamente a porcentagem de defeitos aceitos na fonte de dados do sistema de vendas para cada classe defeito.

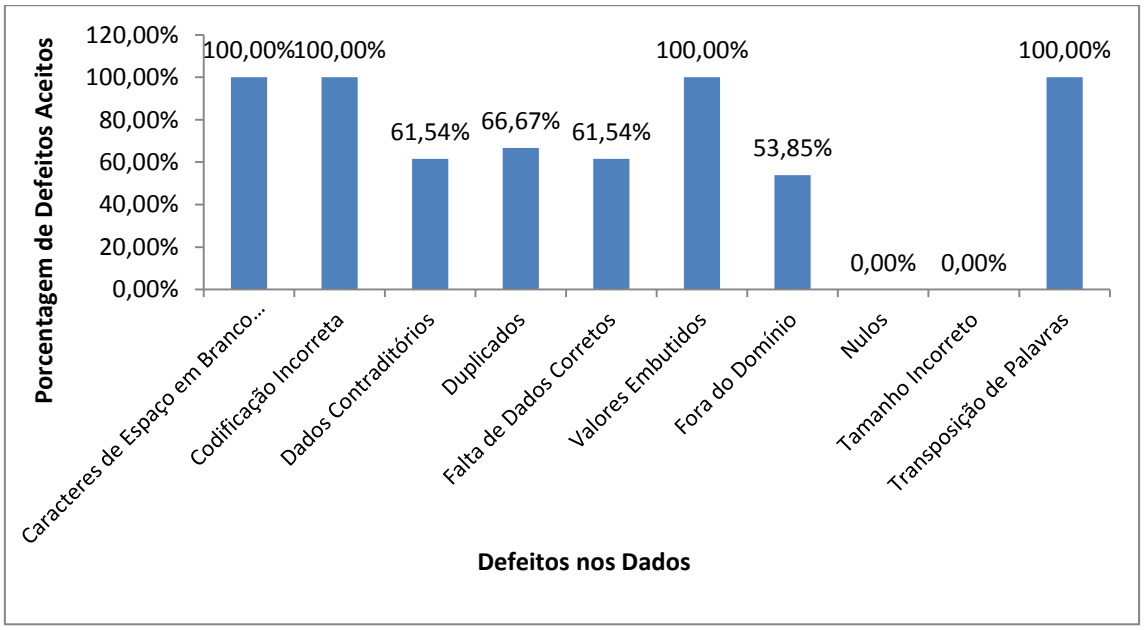


Figura 5.3 Classes de Defeito nos Dados da Fonte de Dados do Sistema de Vendas

De acordo com os resultados, verificou-se que para a fonte de dados do sistema de vendas houve maior incidência das classes de defeito Caracteres de Espaço em Branco Incorretos, Codificação Incorreta, Valores Embutidos e Transposição de Palavras. Com relação ao defeito Caracteres de Espaço em Branco Incorretos, foi possível visualizar a

ausência de uma restrição que impossibilite a inserção de espaços em branco no início do conteúdo de um campo *string*.

Com relação ao defeito Codificação Incorreta, as tabelas não possuem restrições que limitem a inserção de codificações UTF-8 em tabelas com codificação ISO 8859-1. Para a classe de defeito Transposição de Palavras há ausência de restrições que evitem a inserção de textos com a transposição da palavra e por fim para Valores Embutidos, apresentam uma ausência de restrição para evitar a adição de pronomes de tratamento no início dos campos de texto.

Não houve incidência de defeito para a classe de defeito Tamanho Incorreto e Nulos, pois o banco resultou em um *DataError* que não aceita tamanhos além do tamanho determinado no banco, e também com relação a obrigatoriedade de campos não-nulos.

Com a realização do estudo de caso, no qual as classes de defeitos foram aplicadas no esquema de dados e nos dados da fonte de dados, foi possível verificar a aplicabilidade e eficácia da utilização das classes de defeito em revelar os defeitos associados aos problemas de qualidade de dados na fase de Fonte de Dados descrito em Singh e Singh (2010).

Fase 2: Teste no ETL

Nessa fase de teste, foram executados os casos de teste, tendo-se como programa em teste 10 consultas SQL originais e como entrada para os casos de teste as instâncias de dados da BDT. Os resultados obtidos do processo de teste nessa fase são apresentados nas Tabelas 5.3 e 5.4.

A Tabela 5.3 apresenta a quantidade de mutantes mortos, vivos e equivalentes com seu respectivo escore de mutação por consulta SQL considerando os casos de teste utilizados em duas execuções do experimento, enquanto que a Tabela 5.4 apresenta os escores de mutação por operador de mutação, informando também o número de mutantes mortos, vivos e equivalentes para cada operador de mutação e por consulta SQL considerando os casos de teste utilizados nas duas execuções. A segunda execução de testes foi realizada com a finalidade de matar os mutantes que ficaram vivos. A maioria dos mutantes foram mortos na segunda execução por meio da adição de instâncias de dados na base de dados capazes de matar esses mutantes, permanecendo vivos alguns mutantes dos operadores tNmTb, ROR e LCR, sendo estes mais difíceis

para matar. Nessa fase foram gerados um total de 4563 mutantes, distribuídos em 10 classes de operadores de mutação SQL.

Na Figura 5.4 verifica-se que os melhores escores de mutação, com valor igual a 1 foram obtidos para os operadores tNmRole (troca de nome de role), iRole (inserção de role), rRole (retirada de role), tNmTb (troca de Nome de Tabela), tPoAt (troca de posição de atributo), iAtr (inserção de atributo), rAtr (retirada de atributo), tPoVr (troca de posição de valor), tAt (troca de atributo), ABS (inserção de valor absoluto), UOI (inserção de operador unário), IRC (substituição de coluna), NLO (outros nulos), NLI (incluir nulos) e tVr (troca de valor) e os menores escores de mutação foram obtidos para os operadores SEL (SELECT), LCR (substituição de operador lógico) e ROR (substituição de operador relacional).

Na Figura 5.5 verifica-se que os operadores de mutação com maior número de mutantes gerados foi tPoAt (troca de posição de atributo) e tPoVr (troca de posição de valor).

A média do escore de mutação para todo o conjunto de casos de teste foi de 0,88 mostrando que o conjunto de casos de teste executado conseguiu detectar grande parte dos defeitos representados pelos mutantes, representando uma boa medida para a qualidade do conjunto de casos de teste.

O custo para geração dos mutantes é alto, porém foi possível verificar a eficácia do teste de mutação associado aos problemas de qualidade nos dados do processo ETL.

Tabela 5.3 Resultados do Teste de Mutação SQL no Sistema de Vendas

Resultados do Teste de Mutação SQL – Sistema de Vendas								
BDT – Base de Dados de Teste								
Cons. SQL	**Mutantes**							
	1ª Execução				**2ª Execução**			
	Mortos	**Vivos**	**Equivalente**	**Escores Mutação**	**Mortos**	**Vivos**	**Equivalente**	**Escores Mutação**
1	10	0	0	1	10	0	0	1
2	1372	100	0	0,93	1472	0	0	1
3	6	7	0	0,46	6	7	0	0,46
4	6	13	0	0,32	6	13	0	0,32
5	22	3	0	0,88	22	0	3	1

continua na próxima página

Tabela 5.3 Resultados do Teste de Mutação SQL no Sistema de Vendas (continuação)

Resultados do Teste de Mutação SQL – Sistema de Vendas								
BDT – Base de Dados de Teste								
Cons. SQL	Mutantes							
	1ª Execução				2ª Execução			
	Mortos	Vivos	Equivalente	Escores Mutação	Mortos	Vivos	Equivalente	Escores Mutação
6	10	2	0	0,83	10	0	2	1
7	10	2	0	0,83	11	0	1	1
8	70	0	0	1	70	0	0	1
9	1372	100	0	0,93	1472	0	0	1
10	1409	49	0	0,97	1458	0	0	1

Tabela 5.4 Operadores e seus respectivos Escores de Mutação no Sistema de Vendas

Consulta SQL	Operador	Mortos	Vivos	Equivalentes	Escores de Mutação
1	tNmRole	4	0	0	1
	iRole	4	0	0	1
	rRole	2	0	0	1
2	tPoAt	720	0	0	1
	iAtr	2	0	0	1
	rAtr	6	0	0	1
	tPoVr	720	0	0	1
	tAt	24	0	0	1
3	tNmTb	6	2	0	0,75
	ROR	0	5	0	0
4	LCR	0	1	0	0
	ROR	0	10	0	0
	tNmTb	6	2	0	0,75
5	SEL	0	0	1	0
	ABS	2	0	2	1
	UOI	6	0	0	1
	IRC	14	0	0	1
6	SEL	0	0	1	0
	ABS	1	0	1	1
	UOI	3	0	0	1
	IRC	6	0	0	1
7	SEL	0	0	1	0
	NLI	1	0	0	1
	NLO	3	0	0	1
	ROR	7	0	0	1
8	tPoAt	25	0	0	1
	rAtr	6	0	0	1
	tVr	1	0	0	1
	tAt	40	0	0	1
	tPoVr	26	0	0	1
9	tPoAt	720	0	0	1
	iAtr	2	0	0	1
	rAtr	6	0	0	1

continua na próxima página

Tabela 5.4 Operadores e seus respectivos Escores de Mutação (continuação)

Consulta SQL	Operador	Mortos	Vivos	Equivalentes	Escores de Mutação
	tPoVr	720	0	0	1
	tAt	24	0	0	1
10	tPoAt	720	0	0	1
	rAtr	6	0	0	1
	tPoVr	720	0	0	1
	tAt	12	0	0	1

Nos gráficos das Figuras 5.4 e 5.5 são apresentados respectivamente, a Média do Escore de Mutação por Operador de Mutação SQL e a Quantidade de Mutantes por Operador de Mutação SQL.

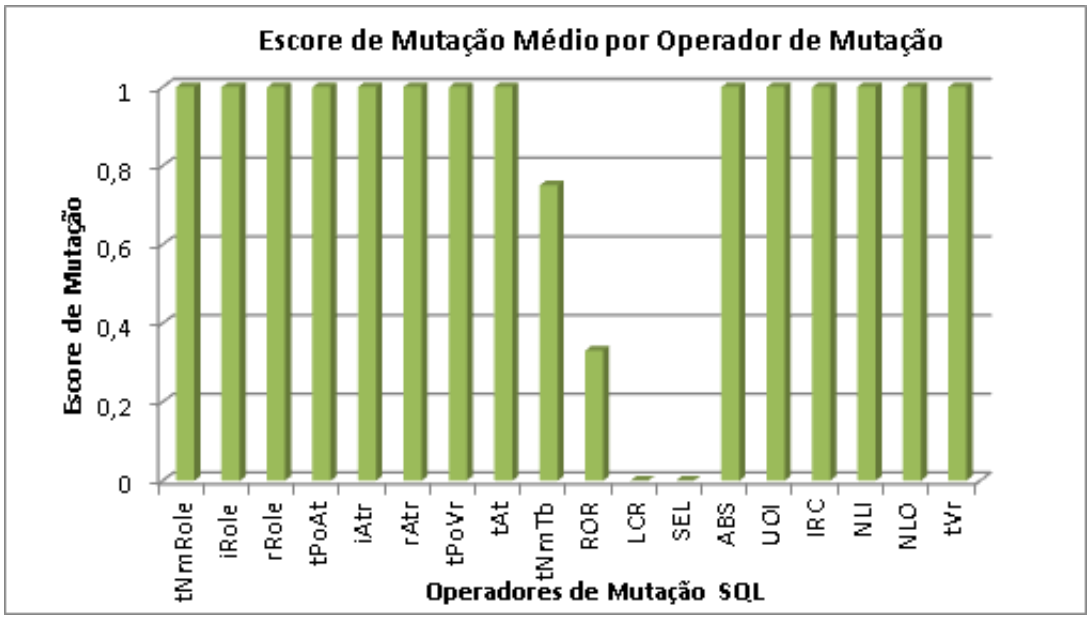


Figura 5.4 Escore de Mutação Médio por Operador de Mutação no Sistema de Vendas

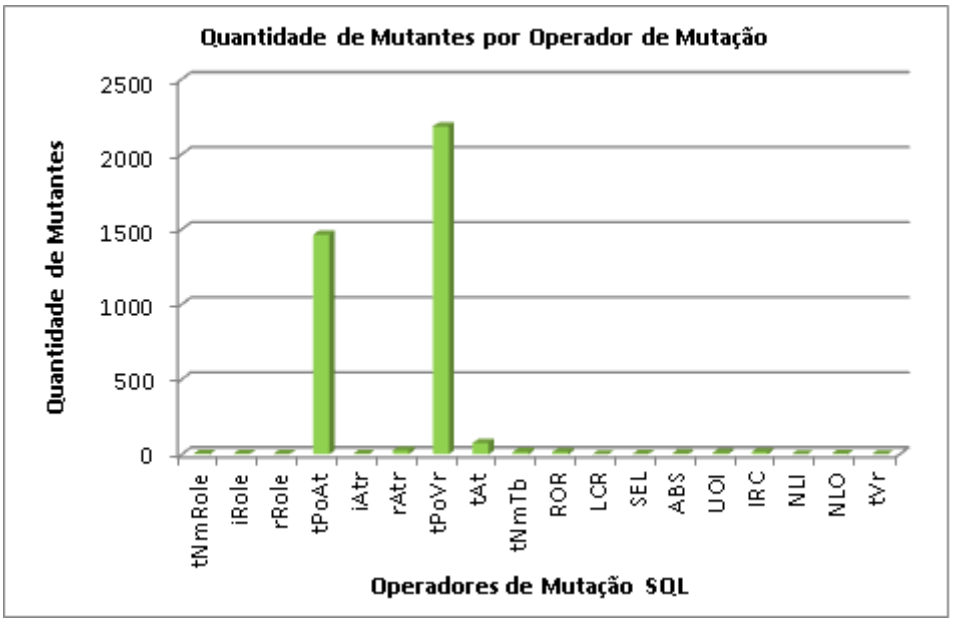


Figura 5.5 Quantidade de Mutantes por Operador de Mutação SQL no Sistema de Vendas

Fase 3: Teste no DW do Sistema de Vendas

O resultado da aplicação das classes de defeito nos dados do DW na base de teste do sistema de vendas resultou na Tabela 5.5.

A Tabela 5.5 apresenta o total de instâncias de dados alternativas geradas para cada classe de defeito, e deste total quantos defeitos foram considerados como aceitos e quantos defeitos foram rejeitados no processo de teste para o DW.

Tabela 5.5 Quantidade de Defeitos por Classes de Defeito no DW do Sistema de Vendas

Classes de Defeito nos Dados	Total de Instâncias de Dados Alternativas	Total de Valores Aceitos	Total de Valores Rejeitados
Caracteres de Espaço em Branco Incorretos	180	147	33
Codificação Incorreta	180	147	33
Dados Contraditórios	410	285	125
Duplicados	40	27	13
Falta de Dados Corretos	410	282	128
Valores Embutidos	1440	1200	240
Fora do Domínio	2870	998	1872
Nulos	410	178	232
Tamanho Incorreto	400	0	400
Transposição de Palavras	180	147	33

A seguir, a Figura 5.6 apresenta graficamente a porcentagem de defeitos aceitos no DW do sistema de vendas para cada classe de defeito.

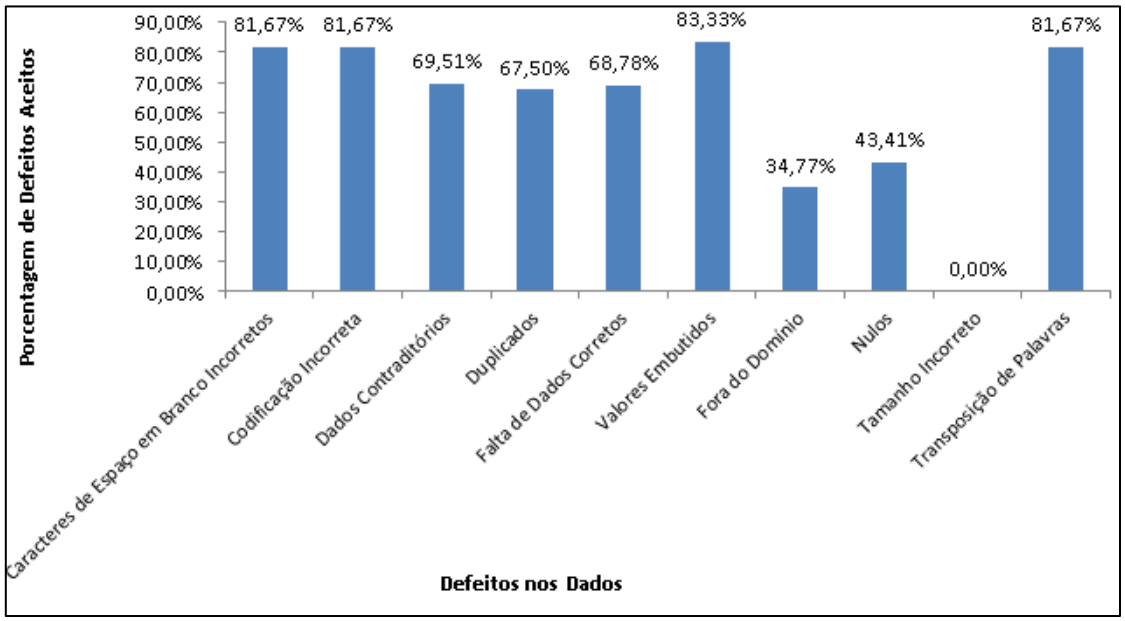


Figura 5.6 Classes de Defeito nos Dados do DW do Sistema de Vendas

De acordo com os resultados, verificou-se que para o DW do sistema de vendas houve maior incidência da classe de defeito Valores Embutidos, pois as tabelas apresentam uma ausência de restrição para evitar a adição de pronomes de tratamento no início dos campos de texto.

Não houve incidência de defeito para a classe de defeito Tamanho Incorreto, pois o banco resultou em um *DataError* que não aceita tamanhos além do tamanho determinado no banco.

Nesta fase de teste verificou-se que alguns dos defeitos presentes na fase inicial de Fontes de Dados permaneceram no DW prejudicando a qualidade dos dados desses ambientes.

Com a realização do estudo de caso, foi possível verificar a aplicabilidade e eficácia da aplicação da técnica de teste baseado em defeitos nas três fases de desenvolvimento do DW, por meio das classes de defeitos nos dados e operadores de mutação SQL aplicados ao ETL.

5.3 ESTUDO DE CASO II - SISTEMA CONTROLE DE TRÂMITES

5.3.1 Descrição

O sistema Controle de Trâmites é responsável pelo controle de fluxo dos processos em um Órgão Público e auxílio à emissão de Atos referente a esses processos. Os assuntos dos processos estão relacionados à Prestação de Contas Municipais (qualquer entidade que receba dinheiro do Estado, Município ou do Governo Federal).

O sistema utiliza a plataforma Banco de Dados *SQL Server 2012* e a ferramenta *Integrations Services do SQL* para o desenvolvimento do processo ETL do DW. Tanto a fonte de dados (tabelas Relacionais) quanto o DW (Tabelas Fatos e Dimensões em modelo Estrela) do sistema Controle de Trâmites foram desenvolvidos em *SQL Server* pela equipe do Órgão Público.

Para o processo de teste foram utilizadas 30 tabelas da base de dados relacional e 27 tabelas do DW, as quais foram copiadas e disponibilizadas em servidor remoto pelo Órgão Público, com acesso a essas bases de dados por meio de acesso remoto.

Todas as tabelas, tanto da fonte de dados como do DW foram duplicadas (somente estrutura de dados) para uma base de dados teste no servidor remoto para que o estudo de caso fosse realizado. Após isso, executou-se o processo de teste descrito nas três fases a seguir.

5.3.2 Fases do Processo de Teste

Fase 1: Teste na Fonte de Dados

No processo de teste, são fornecidos como entrada o esquema do banco de dados (tabelas relacionais) utilizado pelo sistema Controle de Trâmites e os dados das fontes de dados.

Ao receber as instâncias de dados, a ferramenta que avalia a qualidade nos dados gera automaticamente instâncias de dados com alterações (chamadas de instâncias de dados alternativas) de acordo com as classes de defeito pré-definidas. Essas instâncias de dados são geradas a partir dos 10 primeiros registros de cada tabela da fonte de dados original, considerando 100% dos atributos de cada tabela quando possível, ou seja, o defeito de cada classe foi aplicado para todos os atributos de determinada tabela quando viável a sua aplicação, pois a aplicação das alterações relacionadas as classes de defeito depende das restrições e definições associadas aos atributos das tabelas.

Os dados com defeitos são inseridos no banco de dados teste que contém uma réplica do esquema de dados da fonte de dados a fim de verificar se o mesmo irá aceitá-los ou não. Esse teste de validação verifica se os dados seguem regras estabelecidas pelo esquema, caso não siga, e após verificação do testador, significa que existem restrições ausentes ou incorretas no esquema dos dados para a fonte de dados em teste. O resultado desse processo foi analisado a fim de verificar quais valores aceitos e quais valores rejeitados revelaram de fato defeitos no esquema de dados da fonte de dados e, consequentemente, possíveis defeitos nos dados das fontes de dados de acordo com a especificação correta dos dados.

Fase 2: Teste no ETL

Nessa fase de teste foram utilizados 20 comandos de manipulação SQL retirados das *procedures* do processo de Carga dos Dados (*Load*) do ETL do DW. Para a execução desta fase de teste foi considerada uma Base de Dados de Teste (BDT) contendo as tabelas relacionais e os dados copiados do banco de dados original sem nenhuma alteração/redução dos dados.

Os operadores de mutação SQL são executados somente se for possível a sua aplicabilidade aos comandos SQL, pois existem alguns operadores de mutação SQL que não se aplicam a determinadas consultas.

Os operadores de mutação SQL utilizados nesse estudo de caso foram: SEL (SELECT), JOI (troca de *Join*), iNot (inserção de operador de negação), rNot (retirada de operador de negação), ROR (substituição de operador relacional), AOR (substituição de operador aritmético), tNmTb (troca de nome de tabela), tFuAg (troca de função de agregação), LCR (substituição de operador lógico) e BTW (*between predicate*).

Após a execução das consultas mutantes nas instâncias da base de dados, a ferramenta realiza uma análise automática do resultado das consultas avaliando se os mutantes ficaram mortos ou vivos. Para isso, a ferramenta realiza uma comparação entre os resultados gerados pela execução da consulta original e os resultados gerados pela consulta mutante. No caso do resultado da consulta ser diferente do resultado da consulta original considera-se mutante “Morto”. No caso dos resultados dessas consultas serem iguais, considera-se mutante “Vivo” e no caso de mutantes vivos o testador deverá analisar se o mutante é equivalente ou não, caso não seja equivalente, ou seja, o mutante continuou “Vivo”, o testador decide se deve ou não continuar os testes, sempre levando em consideração o escore de mutação alcançado para o teste.

A quantificação dos resultados dos testes é avaliada por meio do escore de mutação gerado para cada caso de teste executado no estudo de caso. Quanto mais próximo de 1 melhor o conjunto de casos de teste utilizado para revelar os defeitos. Os resultados serão apresentados e discutidos posteriormente na seção Resultados.

Fase 3: Teste no DW

Nesta fase do processo de teste proposto, são fornecidos como entrada o esquema do banco de dados do DW utilizado no sistema Controle de Trâmites e os dados do DW.

Após receber as instâncias de dados, a ferramenta de qualidade de dados gerou as instâncias de dados com as alterações (instâncias de dados alternativas), alteradas de acordo com as classes de defeito pré-definidas. Essas instâncias de dados também foram geradas a partir dos 10 primeiros registros de cada tabela do DW, considerando 100% dos atributos de cada tabela, quando possível, conforme explicado anteriormente.

Os dados com defeitos são inseridos no banco de dados teste que contém uma réplica do esquema de dados do DW a fim de verificar se o mesmo irá aceitá-los ou não. Esse teste de validação verifica se os dados seguem regras estabelecidas pelo esquema, caso não siga, e após verificação do testador, significa que existem restrições ausentes ou incorretas no esquema dos dados para a fonte de dados em teste. O resultado desse processo foi analisado a fim de verificar quais valores aceitos e quais valores rejeitados revelaram de fato defeitos no esquema de dados do DW e assim possíveis defeitos nos dados do DW de acordo com a especificação correta dos dados.

5.3.3 Resultados

Os resultados obtidos para cada fase do processo de teste no sistema Controle de Trâmites são apresentados detalhadamente a seguir.

Fase 1: Teste na Fonte de Dados Controle de Trâmites

O resultado da aplicação das classes de defeito nos dados da fonte de dados na base de teste do sistema Controle de Trâmites resultou na Tabela 5.6. Essa tabela apresenta o total de instâncias de dados alternativas geradas para cada classe de defeito, e deste total quantos defeitos foram considerados como aceitos e quantos defeitos foram rejeitados no processo de teste.

Tabela 5.6 Quantidade de Defeitos por Classes de Defeito na Fonte de Dados do Sistema Controle de Trâmites

Classes de Defeito nos Dados	Total de Instâncias de Dados Alternativas	Total de Valores Aceitos	Total de Valores Rejeitados
Caracteres de Espaço em Branco Incorretos	2160	1529	631
Codificação Incorreta	1600	1154	446
Dados Contraditórios	4460	1807	2653
Duplicados	300	197	103
Falta de Dados Corretos	4460	2125	2335
Valores Embutidos	12800	9208	3592
Fora do Domínio	31430	8786	22644
Nulos	4460	1807	2653
Tamanho Incorreto	3970	10	3960
Transposição de Palavras	1600	1154	446

A seguir, a Figura 5.7 apresenta graficamente a porcentagem de defeitos aceitos na fonte de dados do sistema Controle de Trâmites para cada classe de defeito.

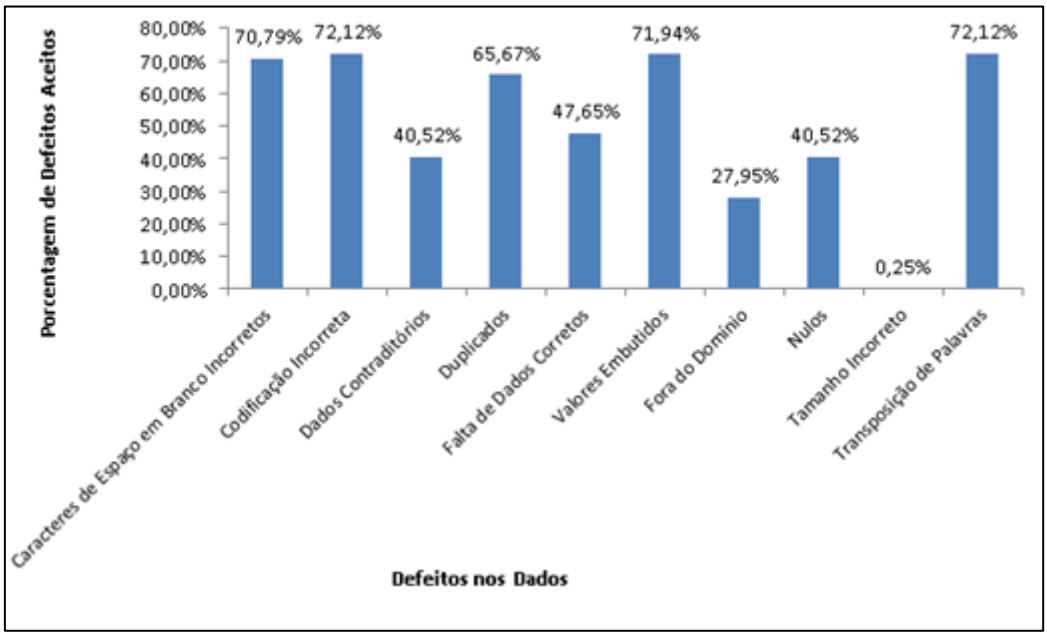


Figura 5.7 Classes de Defeito nos Dados da Fonte de Dados do Sistema Controle de Trâmites

De acordo com os resultados, verificou-se que para a fonte de dados do sistema Controle de Trâmites houve maior incidência da classe de defeito Codificação Incorreta e Transposição de Palavras, pois as tabelas não possuem restrições que limitem a inserção de codificações UTF-8 em tabelas com codificação ISO 8859-1 e restrições que evitem a inserção de textos com a transposição da palavra.

Em segundo lugar, a maior incidência de defeitos foi para as classes de Valores Embutidos, pois há uma ausência de restrição que evite inserir pronomes de tratamento em campos de texto.

Em terceiro lugar a maior incidência de defeitos foi para a classe de defeitos Caracteres de Espaço em Branco Incorretos, no qual foi possível visualizar a ausência de uma restrição que impossibilite a inserção de espaços em branco no início do conteúdo de um campo *string*.

A menor incidência de defeito foi para a classe de defeito Tamanho Incorreto, pois o banco resultou em um *DataError* que não aceita tamanhos além do tamanho determinado no banco, entretanto, alguns casos de Tamanho Incorreto foram aceitos, pois nos campos da base de dados contendo o Número do Documento como CPF e CNPJ foi inserido um valor com tamanho menor que a especificação padrão.

Com a realização do estudo de caso, no qual as classes de defeitos foram aplicadas no esquema de dados e nos dados da fonte de dados, foi possível verificar a eficácia da utilização das classes de defeito em revelar os defeitos associados aos problemas de qualidade de dados na fase de Fonte de Dados descrito em Singh e Singh (2010).

Fase 2: Teste no ETL

Nessa fase de teste, foram executados os casos de teste, tendo-se como programa em teste 20 consultas SQL originais e como entrada para os casos de teste as instâncias de dados da BDT. Os resultados obtidos do processo de teste nessa fase são apresentados nas Tabelas 5.7 e 5.8.

A Tabela 5.7 apresenta a quantidade de mutantes mortos e vivos com seu respectivo escore de mutação por consulta SQL considerando os casos de teste utilizados em três execuções do experimento, enquanto que a Tabela 5.8 apresenta os escores de mutação por operador de mutação, informando também o número de mutantes mortos e vivos para cada operador de mutação e por consulta SQL considerando os casos de teste utilizados nas três execuções. A segunda e terceira execução de teste foram realizadas com a finalidade de matar os mutantes que ficaram vivos. A maioria dos mutantes foram mortos na segunda execução por meio da adição de registros na base de dados capazes de matar esses mutantes, permanecendo vivos na terceira execução alguns mutantes dos operadores JOI e ROR, sendo estes mais difíceis para matar. Para o operador de mutação SEL que gera o mutante SELECT DISTINCT foi adicionada uma linha idêntica para matar esse mutante. Nessa fase de teste foi

gerado um total de 1336 mutantes, distribuídos em 10 classes de operadores de mutação SQL.

Na Figura 5.8 verifica-se que os melhores escores de mutação, ao final das três execuções de casos de teste, com valor igual a 1, foram obtidos para os operadores SEL (SELECT), iNot (inserção de operador de negação), rNot (inserção/remoção de operador de negação), tNmTb (troca de nome de tabela), tFuAg (troca de função de agregação), LCR (substituição de operador lógico), AOR (substituição de operador aritmético) e BTW (*between predicate*) e os menores escores de mutação foram obtidos para os operadores JOI (troca de *Join*) e ROR (substituição de operador relacional).

Na Figura 5.9 verifica-se que os operadores de mutação com maior número de mutantes gerados foi ROR (substituição de operador relacional) e tNmTb (troca de nome de tabela).

A média do escore de mutação para todo o conjunto de casos de testes foi de 0,98 mostrando que o conjunto de casos de teste executado conseguiu detectar grande parte dos defeitos representados pelos mutantes, representando uma boa medida para a qualidade do conjunto de casos de teste. Dos mutantes que ficaram vivos, não houve identificação de mutantes equivalentes.

O custo para geração dos mutantes é alto, porém foi possível verificar a eficácia do teste de mutação associados aos problemas de qualidade nos dados do processo ETL.

Tabela 5.7 Resultados do Teste de Mutação SQL em Controle de Trâmites

Resultados do Teste de Mutação SQL de Controle de Trâmites									
BDT – Base de Dados de Teste									
Cons. SQL	**Mutantes**								
	1ª Execução			**2ª Execução**			**3ª Execução**		
	Mortos	**Vivos**	**Escore de Mutação**	**Mortos**	**Vivos**	**Escore de Mutação**	**Mortos**	**Vivos**	**Escore de Mutação**
1	76	5	0,94	80	1	0,99	80	1	0,99
2	76	5	0,94	80	1	0,99	80	1	0,99
3	78	3	0,96	81	0	1	81	0	1
4	76	5	0,94	81	0	1	81	0	1
5	79	10	0,89	89	0	1	89	0	1
6	77	4	0,95	81	0	1	81	0	1
7	70	5	0,93	75	0	1	75	0	1
8	70	5	0,93	75	0	1	75	0	1
9	70	5	0,93	75	0	1	75	0	1
10	73	2	0,97	74	1	0,99	74	1	0,99

continua na próxima página

Tabela 5. 7 Resultados do Teste de Mutação SQL em Controle de Trâmites (continuação)

Resultados do Teste de Mutação SQL de Controle de Trâmites									
BDT – Base de Dados de Teste									
Cons. SQL	Mutantes								
	1ª Execução			2ª Execução			3ª Execução		
	Mortos	Vivos	Escore de Mutação	Mortos	Vivos	Escore de Mutação	Mortos	Vivos	Escore de Mutação
11	4	2	0,67	4	2	0,67	4	2	0,67
12	60	3	0,95	62	1	0,98	62	1	0,98
13	47	3	0,94	50	0	1	50	0	1
14	50	1	0,98	50	1	0,98	50	1	0,98
15	51	1	0,98	51	1	0,98	51	1	0,98
16	109	6	0,95	115	0	1	115	0	1
17	80	2	0,98	81	1	0,99	82	0	1
18	49	4	0,92	52	1	0,98	53	0	1
19	32	3	0,91	34	1	0,97	35	0	1
20	35	0	1	35	0	1	35	0	1

Tabela 5.8 Operadores e seus respectivos Escores de Mutação no Estudo de Caso Controle de Trâmites

Consulta SQL	Operador	Mortos	Vivos	Escores de Mutação
1	SEL	1	0	1
	JOI	3	1	0,75
	iNOT	1	0	1
	ROR	35	0	1
	tFuAg	6	0	1
	LCR	5	0	1
	tNmTb	29	0	1
2	SEL	1	0	1
	JOI	3	1	0,75
	iNOT	1	0	1
	LCR	5	0	1
	ROR	35	0	1
	tFuAg	6	0	1
	tNmTb	29	0	1
3	SEL	1	0	1
	JOI	4	0	1
	iNOT	1	0	0
	ROR	35	0	1
	LCR	5	0	1
	tFuAg	6	0	1
	tNmTb	29	0	1
4	SEL	1	0	1
	JOI	4	0	1
	iNOT	1	0	1
	ROR	35	0	1
	LCR	5	0	1
	tFuAg	6	0	1
	tNmTb	29	0	1

continua na próxima página

Tabela 5.8 Operadores e seus respectivos Escores de Mutação no Estudo de Caso Controle de Trâmites (continuação)

Consulta SQL	Operador	Mortos	Vivos	Escores de Mutação
5	SEL	1	0	1
	JOI	4	0	1
	iNOT	2	0	1
	ROR	40	0	1
	LCR	7	0	1
	tFuAg	6	0	1
	tNmTb	29	0	1
6	SEL	1	0	1
	JOI	4	0	1
	iNOT	1	0	1
	ROR	35	0	1
	LCR	5	0	1
	tFuAg	6	0	1
	tNmTb	29	0	1
7	SEL	1	0	1
	JOI	4	0	1
	iNOT	1	0	1
	ROR	30	0	1
	LCR	4	0	1
	tFuAg	6	0	1
	tNmTb	29	0	1
8	SEL	1	0	1
	JOI	4	0	1
	iNOT	1	0	1
	ROR	30	0	1
	LCR	4	0	1
	tFuAg	6	0	1
	tNmTb	29	0	1
9	SEL	1	0	1
	JOI	4	0	1
	iNOT	1	0	1
	ROR	30	0	1
	LCR	4	0	1
	tFuAg	6	0	1
	tNmTb	29	0	1
10	SEL	1	0	1
	JOI	3	1	0,75
	iNOT	1	0	1
	ROR	30	0	1
	LCR	4	0	1
	tFuAg	6	0	1
	tNmTb	29	0	1
11	SEL	1	0	1
	JOI	2	2	0,5
	LCR	1	0	1
12	SEL	1	0	1
	JOI	3	1	0,75
	iNOT	1	0	1

continua na próxima página

Tabela 5.8 Operadores e seus respectivos Escores de Mutação no Estudo de Caso Controle de Trâmites (continuação)

Consulta SQL	Operador	Mortos	Vivos	Escores de Mutação
	ROR	25	0	1
	LCR	3	0	1
	tNmTb	29	0	1
13	SEL	1	0	1
	BTW	2	0	1
	rNOT	2	0	1
	iNOT	2	0	1
	ROR	10	0	1
	LCR	4	0	1
	tNmTb	29	0	1
14	SEL	1	0	1
	JOI	4	0	1
	iNOT	1	0	1
	ROR	14	1	0,93
	LCR	1	0	1
	tNmTb	29	0	1
15	SEL	1	0	1
	JOI	4	0	1
	iNOT	1	0	1
	ROR	14	1	0,93
	LCR	2	0	1
	tNmTb	29	0	1
16	SEL	1	0	1
	JOI	4	0	1
	iNOT	5	0	1
	rNOT	1	0	1
	AOR	24	0	1
	ROR	40	0	1
	LCR	11	0	1
	tNmTb	29	0	1
17	SEL	1	0	1
	JOI	4	0	1
	iNOT	3	0	1
	rNOT	1	0	1
	ROR	30	0	1
	tFuAg	6	0	1
	LCR	8	0	1
	tNmTb	29	0	1
18	SEL	1	0	1
	iNOT	1	0	1
	ROR	20	0	1
	LCR	2	0	1
	tNmTb	29	0	1
19	SEL	1	0	1
	ROR	4	0	1
	TNT	29	0	1
20	SEL	1	0	1
	ROR	5	0	1
	tNmTb	29	0	1

Nos gráficos das Figuras 5.8 e 5.9 são apresentados respectivamente, a Média do Escore de Mutação por Operador de Mutação SQL e a Quantidade de Mutantes por Operador de Mutação SQL.

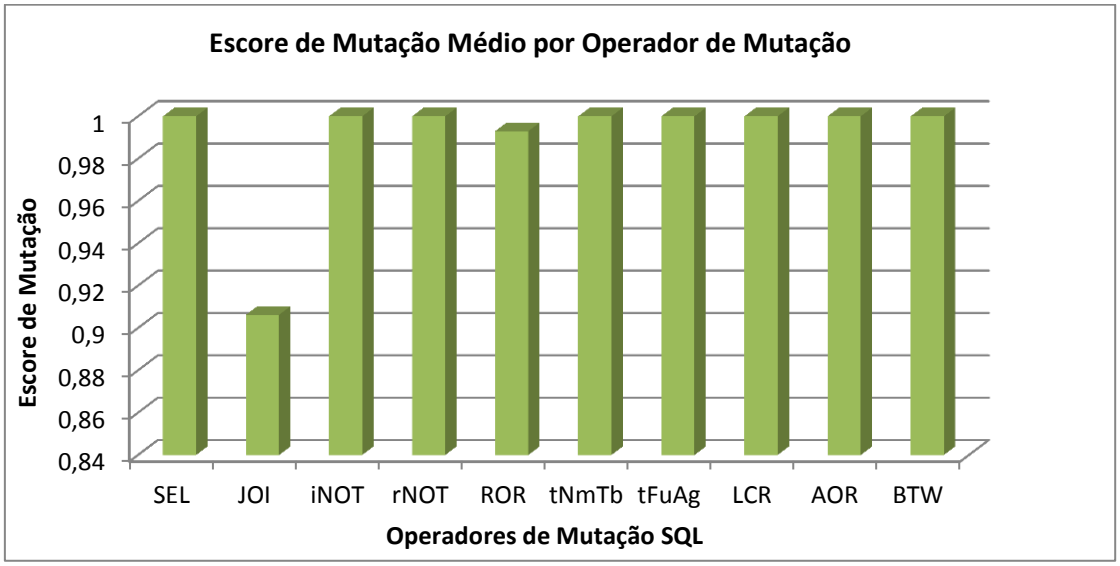


Figura 5.8 Escore de Mutação Médio por Operador de Mutação no Sistema Controle de Trâmites

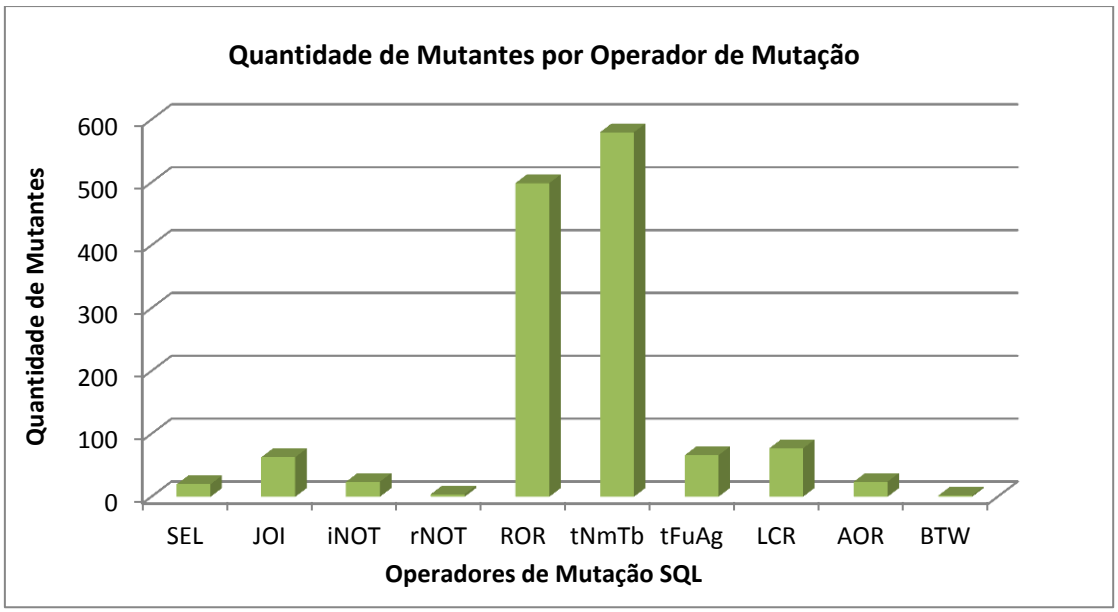


Figura 5.9 Quantidade de Mutantes por Operador de Mutação SQL no Sistema Controle de Trâmites

Fase 3: Teste no DW de Controle de Trâmites

O resultado da aplicação das classes de defeito nos dados do DW na base de teste do sistema Controle de Trâmites resultou na Tabela 5.9.

A Tabela 5.9 apresenta o total de instâncias de dados alternativas geradas para cada classe de defeito, e deste total quantos defeitos foram considerados como aceitos e quantos defeitos foram rejeitados no processo de teste para o DW.

Tabela 5.9 Quantidade de Defeitos por Classes de Defeito no DW do Sistema Controle de Trâmites

Classes de Defeito nos Dados	**Total de Instâncias de Dados Alternativas**	**Total de Valores Aceitos**	**Total de Valores Rejeitados**
Caracteres de Espaço em Branco Incorretos	1321	475	846
Codificação Incorreta	925	344	581
Dados Contraditórios	4900	785	4115
Duplicados	232	70	162
Falta de Dados Corretos	4900	788	4112
Valores Embutidos	7400	2754	4646
Fora do Domínio	34300	3844	30456
Nulos	4900	778	4122
Tamanho Incorreto	4460	60	4400
Transposição de Palavras	925	346	579

A seguir, a Figura 5.10 apresenta graficamente a porcentagem de defeitos aceitos no DW do sistema Controle de Trâmites para cada classe defeito.

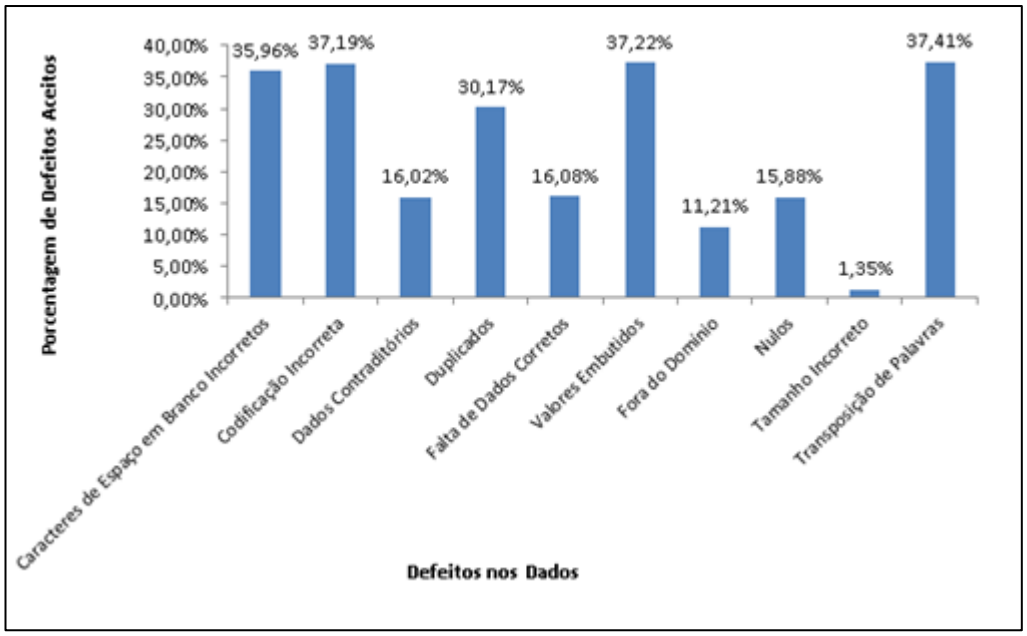


Figura 5.10 Classes de Defeito nos Dados do DW do Sistema Controle de Trâmites

De acordo com os resultados, verificou-se que para o DW do sistema Controle de Trâmites houve maior incidência da classe de defeito Transposição de Palavras, pois as tabelas não possuem restrições que evitem a inserção da mesma *string* com palavras desordenadas.

Em segundo lugar a maior incidência de defeitos foi para a classe de Valores Embutidos, pois as tabelas apresentam uma ausência de restrição para evitar a adição de pronomes de tratamento no início dos campos de texto.

Em terceiro lugar a maior incidência de defeitos foi para a classe de defeitos Codificação Incorreta, na qual foi possível inserir codificações de texto diferentes do padrão.

A menor incidência de defeito foi para a classe de defeito Tamanho Incorreto, pois o banco resultou em um *DataError* que não aceita tamanhos além do tamanho determinado no banco, entretanto alguns casos de Tamanho Incorreto foram aceitos, no campo definido como bit com tamanho 1 foi aceito um valor flutuante de tamanho 2.

Com a realização do estudo de caso, no qual as classes de defeitos foram aplicadas no esquema de dados e nos dados do DW, foi possível verificar que alguns dos defeitos permaneceram no DW prejudicando a qualidade dos dados desses ambientes.

5.4 ESTUDO DE CASO III - SISTEMA DE EMPENHO

5.4.1 Descrição

O Sistema de Empenho é responsável pelo controle de informações de acompanhamentos dos empenhos emitidos no exercício financeiro de trabalho, bem como dos empenhos inscritos em Restos a Pagar. Os empenhos de Restos a Pagar surgem em decorrência da extinção de entidades integrantes do orçamento do Município. Os empenhos incorporados por cisão, fusão ou extinção de entidades do Município também devem ser declarados e identificados pela sua origem.

O sistema utiliza a plataforma Banco de Dados *SQL Server 2012* e a ferramenta *Integration Services* do SQL para o desenvolvimento do processo ETL do DW. Tanto as fontes de dados originais (tabelas Relacionais) quanto o DW (Tabelas Fatos e Dimensões em modelo Estrela) do sistema de Empenho foram desenvolvidos em *SQL Server* pela equipe de um Órgão Público.

Nesse estudo de caso foram utilizadas as bases de dados relacionais e base de dados do DW, disponibilizadas em máquina local no Órgão Público. Todas as tabelas, tanto das fontes de dados como do DW, assim como no Controle de Trâmites, foram copiadas para uma base de dados de teste para que fosse possível a execução do estudo de caso.

O sistema de Empenho utiliza tabelas de 4 bases de dados relacionais, BD1 (44 tabelas), BD2 (28 tabelas), BD3 (1 tabela), BD4 (1 tabela) e a base do DW com 35 tabelas, totalizando 109 tabelas utilizadas no estudo de caso.

Todas as tabelas, tanto da fonte de dados como do DW foram duplicadas (somente estrutura de dados) para uma base de dados teste para que o estudo de caso fosse realizado. Após isso, foi executado o processo de teste descrito nas três fases a seguir.

5.4.2 Fases do Processo de Teste

Fase 1: Teste nas Fontes de Dados

No processo de teste proposto, são fornecidos como entrada o esquema do banco de dados (tabelas relacionais) utilizado pelo sistema de Empenho e os dados de entrada das fontes de dados. Nesse estudo de caso para execução do processo de teste, são dados como entrada para execução do processo uma base de dados por vez.

Ao receber as instâncias de dados, a ferramenta que avalia a qualidade nos dados gera automaticamente instâncias de dados com alterações (chamadas de instâncias de dados alternativas) realizadas de acordo com as classes de defeito pré-definidas. Essas instâncias de dados são geradas a partir dos 10 primeiros registros de cada tabela da fonte de dados original, considerando 100% dos atributos de cada tabela, quando possível. Os dados com defeitos são inseridos no banco de dados teste, que contém uma réplica do esquema de dados da fonte de dados, a fim de verificar se o mesmo irá aceitá-los ou não. Esse teste de validação verifica se os dados seguem regras estabelecidas pelo esquema, caso não siga, e após verificação do testador, significa que existem restrições ausentes ou incorretas no esquema dos dados para a fonte de dados em teste. O resultado desse processo foi analisado a fim de verificar quais valores aceitos e quais valores rejeitados revelaram de fato defeitos no esquema de dados da fonte de dados e, assim, possíveis defeitos nos dados das fontes de dados de acordo com a especificação correta dos dados.

Fase 2: Teste no ETL

Nesse estudo de caso foram utilizados 34 comandos de manipulação SQL retirados do processo ETL do DW do sistema de Empenho.

Para a execução desta fase foram considerados os dados das 4 Bases de Dados de Produção (BDP). Os operadores de mutação SQL são aplicados somente se for possível a sua aplicação aos comandos selecionados.

Os operadores de mutação SQL utilizados nesse estudo de caso foram: SEL (SELECT), ABS (inserção de valor absoluto), UOI (inserção de operador unário), IRD (substituição de identificador), ROR (substituição de operador relacional), JOI (troca de Join), IRC (substituição de coluna), LCR (substituição de operador lógico), NLS (Nulos em Lista de Seleção), IRT (substituição de constante), BTW (predicado *between*), SUB (predicados de subconsulta).

Após a execução dos mutantes gerados sobre as instâncias de dados, a ferramenta realiza uma análise automática do resultado das consultas avaliando se os mutantes ficaram mortos ou vivos. Os mutantes vivos foram analisados pelo testador a fim de verificar se os mesmos eram equivalentes ou foi decidido pelo mesmo executar novos casos de teste a fim de matar os mutantes vivos. O escore de mutação foi gerado para cada caso de teste executado no estudo de caso.

Fase 3: Teste no DW

No processo de teste, são fornecidos como entrada o esquema do banco de dados do DW utilizado no sistema de Empenho e os dados do DW.

A ferramenta de qualidade de dados gerou automaticamente instâncias de dados alternativas de acordo com as classes de defeito pré-definidas. Essas instâncias de dados foram geradas a partir dos 10 primeiros registros de cada tabela do DW, considerando 100% dos atributos de cada tabela, quando possível.

Os dados com defeitos são inseridos no banco de dados teste, que contém uma cópia do esquema de dados do DW, a fim de verificar se o mesmo irá aceitá-los ou não. O resultado desse processo foi analisado a fim de verificar quais valores aceitos e quais valores rejeitados revelaram de fato defeitos no esquema de dados do DW e, assim, possíveis defeitos nos dados do DW de acordo com a especificação correta dos dados.

5.4.3 Resultados

Os resultados obtidos para cada fase do processo de teste no sistema de Empenho são apresentados detalhadamente a seguir.

Fase 1: Teste nas Fontes de Dados do Sistema de Empenho

Os resultados da aplicação das classes de defeito nos dados da fonte de dados do sistema de Empenho resultaram nas tabelas 5.10, 5.11, 5.12 e 5.13, pois foram utilizadas as quatro bases de dados das fontes de dados no estudo de caso.

As Tabelas 5.10, 5.11, 5.12 e 5.13 apresentam o total de instâncias de dados alternativas geradas para cada classe de defeito, quais defeitos foram aceitos e quais defeitos foram rejeitados no processo de teste para as bases de dados 1,2,3 e 4 respectivamente.

Tabela 5.10 Quantidade de Defeitos por Classes de Defeito para a Base de Dados 1 do Sistema de Empenho

BD1- Base de Dados 1			
Classes de Defeito nos Dados	**Total de Instâncias de Dados Alternativas**	**Total de Valores Aceitos**	**Total de Valores Rejeitados**
Caracteres de Espaço em Branco Incorretos	751	190	561
Codificação Incorreta	719	70	649
Dados Contraditórios	2596	307	2289

continua na próxima página

Tabela 5.10 Quantidade de Defeitos por Classes de Defeito para a Base de Dados 1 do Sistema de Empenho (continuação)

BD1- Base de Dados 1			
Classes de Defeito nos Dados	**Total de Instâncias de Dados Alternativas**	**Total de Valores Aceitos**	**Total de Valores Rejeitados**
Duplicados	395	20	375
Falta de Dados Corretos	2596	377	2219
Valores Embutidos	5752	910	4842
Fora do Domínio	18282	1060	17222
Nulos	2596	90	2506
Tamanho Incorreto	2466	10	2456
Transposição de Palavras	719	70	649

Tabela 5.11 Quantidade de Defeitos por Classes de Defeito para a Base de Dados 2 do Sistema de Empenho

BD2- Base de Dados 2			
Classes de Defeito nos Dados	**Total de Instâncias de Dados Alternativas**	**Total de Valores Aceitos**	**Total de Valores Rejeitados**
Caracteres de Espaço em Branco Incorretos	2030	372	1658
Codificação Incorreta	1522	146	1376
Dados Contraditórios	4020	557	3463
Duplicados	280	46	234
Falta de Dados Corretos	4020	544	3476
Valores Embutidos	12176	1803	10373
Fora do Domínio	28360	2537	25823
Nulos	4020	416	3604
Tamanho Incorreto	3790	10	3780
Transposição de Palavras	1522	141	1381

Tabela 5.12 Quantidade de Defeitos por Classes de Defeito para a Base de Dados 3 do Sistema de Empenho

BD3- Base de Dados 3			
Classes de Defeito nos Dados	**Total de Instâncias de Dados Alternativas**	**Total de Valores Aceitos**	**Total de Valores Rejeitados**
Caracteres de Espaço em Branco Incorretos	100	100	0
Codificação Incorreta	100	100	0
Dados Contraditórios	100	98	2
Duplicados	10	10	0

continua na próxima página

Tabela 5.12 Quantidade de Defeitos por Classes de Defeito para a Base de Dados 3 do Sistema de Empenho (continuação)

BD3- Base de Dados 3			
Classes de Defeito nos Dados	**Total de Instâncias de Dados Alternativas**	**Total de Valores Aceitos**	**Total de Valores Rejeitados**
Falta de Dados Corretos	100	100	0
Valores Embutidos	800	725	75
Fora do Domínio	700	372	328
Nulos	100	90	10
Tamanho Incorreto	100	0	100
Transposição de Palavras	100	100	0

Tabela 5.13 Quantidade de Defeitos por Classes de Defeito para a Base de Dados 4 do Sistema de Empenho

BD4- Base de Dados 4			
Classes de Defeito nos Dados	**Total de Instâncias de Dados Alternativas**	**Total de Valores Aceitos**	**Total de Valores Rejeitados**
Caracteres de Espaço em Branco Incorretos	150	150	0
Codificação Incorreta	150	150	0
Dados Contraditórios	220	203	17
Duplicados	10	10	0
Falta de Dados Corretos	220	210	10
Valores Embutidos	1200	1115	85
Fora do Domínio	1540	785	755
Nulos	220	70	150
Tamanho Incorreto	220	0	220
Transposição de Palavras	150	150	0

A seguir, a Figura 5.11 apresenta graficamente a porcentagem de defeitos aceitos nas fontes de dados do sistema de Empenho para cada classe defeito.

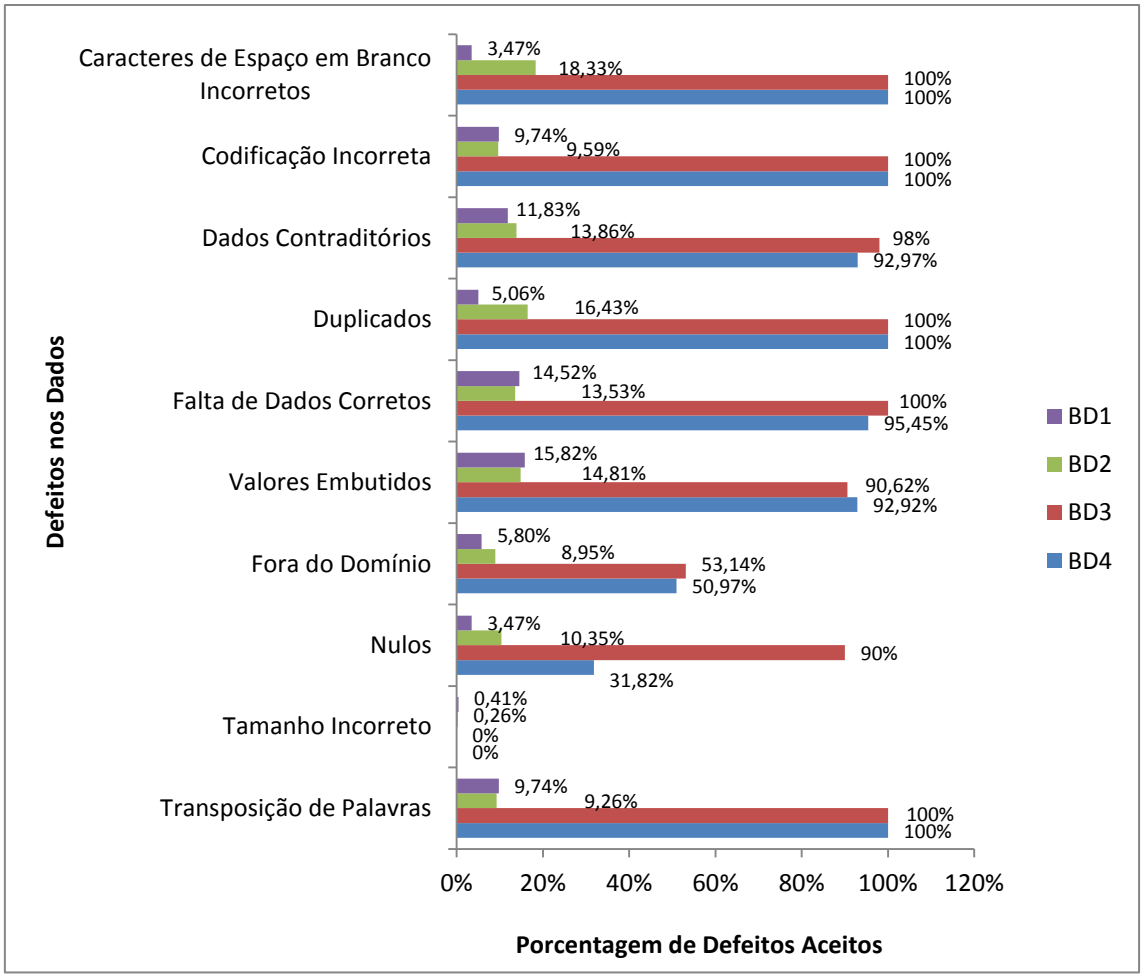


Figura 5.11 Classes de Defeitos nas Fontes de Dados do Sistema de Empenho

De acordo com os resultados, verificou-se que para a BD1 do sistema de Empenho houve maior incidência da classe de defeito de Valores Embutidos e a menor incidência foi para classe de defeito Tamanho Incorreto, pois as tabelas da BD1 não apresentam uma restrição para impedir a inserção de pronomes de tratamento nos campos *string* e a classe de defeito Tamanho Incorreto apresentou um erro *DataError* do banco de dados, impedindo a inserção de dados com tamanho acima do determinado no campo *string*.

Para a BD2 houve maior incidência da classe de defeito Caracteres de Espaço em Branco Incorretos e a menor incidência foi para classe de defeito Tamanho Incorreto, pois as tabelas da BD2 não apresentam uma restrição para impedir a inserção de espaços em branco no início dos campos *string* de cada tabela e a classe de defeito Tamanho Incorreto recebeu um erro *DataError* do banco de dados, impedindo a inserção de dados com tamanho acima do determinado no campo *string*.

Para a BD3 e BD4 verificou-se que não houve incidência da classe de defeito Tamanho Incorreto, porém houve total incidência das classes de defeito Caracteres de Espaço em Branco Incorretos, Codificação Incorreta, Duplicados, Falta de Dados Corretos e Transposição de Palavras, pois as estruturas de dados das tabelas contidas nas BD3 e BD4 possuem em sua maioria campos de tipo *string* com quase nenhum campo com *Not Null* (a maioria aceitando Nulo) e nenhum campo *unique*, ou seja, não havia nenhuma restrição para as classes de defeito permitindo, desta forma, uma grande quantidade de mutantes inseridos das classes de defeito Caracteres de Espaço em Branco Incorretos, Codificação Incorreta, Duplicados, Falta de Dados Corretos e Transposição de Palavras. Enquanto que a classe de defeito Tamanho Incorreto apresentou o erro *DataError* do banco de dados, impedindo a inserção de dados com tamanho acima do determinado nos campos de tipos *string*, flutuante e inteiro.

Com a realização desse estudo de caso, no qual as classes de defeito foram aplicadas no esquema de dados e nos dados da fonte de dados, foi possível verificar a eficácia da utilização das classes de defeito em revelar os defeitos associados aos problemas de qualidade de dados na fase de Fonte de Dados descrito em Singh e Singh (2010).

Fase 2: Teste no ETL

Nessa fase de teste, foram executados os casos de teste, tendo como programa em teste 34 consultas SQL originais e como entrada para os casos de teste as instâncias de dados da base de dados de produção. Os resultados obtidos do processo de teste nessa fase são apresentados nas Tabelas 5.14 e 5.15.

A Tabela 5.14 apresenta a quantidade de mutantes mortos, vivos e equivalentes com seu respectivo escore de mutação por consulta SQL considerando os casos de teste utilizados por execução (1ª, 2ª e 3ª), enquanto que a Tabela 5.15 apresenta os escores de mutação por operador de mutação, informando também o número de mutantes mortos, vivos e equivalentes para cada operador de mutação e por consulta SQL considerando os casos de teste utilizados nas três execuções. A segunda e terceira execução de testes foram realizadas com a finalidade de matar os mutantes que ficaram vivos. A maioria dos mutantes foram mortos na segunda execução por meio da adição de registros na base de dados capazes de matar esses mutantes, permanecendo vivos na terceira execução alguns mutantes dos operadores JOI e apenas um operador SEL. Os mutantes dos operadores JOI são mais difíceis de matar, e seriam necessárias mais execuções de

casos de teste para verificar se conseguiria matar todos. Neste experimento, a maioria dos operadores considerados mutantes equivalentes foram da categoria SEL para o mutante SELECT DISTINCT e ABS. Nesse estudo de caso foi gerado um total de 873 mutantes, distribuídos em 12 classes de operadores de mutação SQL.

Na Figura 5.12 verifica-se que os melhores escores de mutação, com valor igual a 1 foram obtidos para os operadores LCR (substituição de operador lógico), IRT (substituição de constante), BTW (*between predicate*), SUB (predicados de subconsulta), ABS (inserção de valor absoluto), UOI (inserção de operador unário), IRD (substituição de identificador), ROR (substituição de operador relacional), IRC (substituição de coluna), NLS (Nulos em Lista de Seleção) e os menores escores de mutação foram obtidos para os operadores SEL (SELECT) e JOI (troca de JOIN).

Na Figura 5.13 verifica-se que os operadores de mutação com maior número de mutantes gerados foram o IRD (substituição de identificador) e o ROR (substituição de operador relacional).

A média do escore de mutação para todo o conjunto de casos de testes foi de 0,85 mostrando que os testes realizados nas três execuções conseguiu detectar mais defeitos representados pelos mutantes, apresentando uma boa medida para a qualidade do conjunto de casos de teste utilizado no processo de teste.

Tabela 5.14 Resultados do Teste de Mutação SQL no Sistema de Empenho

Resultados do Teste de Mutação SQL do Sistema de Empenho												
BDT – Base de Dados de Teste												
Cons. SQL	**Legenda: MT- Mortos / VV- Vivos / EM- Escore de Mutação / EQ- Equivalente**											
	Mutantes											
	1ª Execução				**2ª Execução**				**3ª Execução**			
	MT	**VV**	**EQ**	**EM**	**MT**	**VV**	**EQ**	**EM**	**MT**	**VV**	**EQ**	**EM**
1	0	1	0	0	0	0	1	0	0	0	1	0
2	0	1	0	0	0	0	1	0	0	0	1	0
3	13	2	0	0,87	13	0	2	1	13	0	2	1
4	13	2	0	0,87	13	0	2	1	13	0	2	1
5	11	2	0	0,85	11	0	2	1	11	0	2	1
6	11	2	0	0,85	11	0	2	1	11	0	2	1
7	13	2	0	0,87	13	0	2	1	13	0	2	1
8	61	4	0	0,94	62	3	0	0,95	62	0	3	1
9	87	11	0	0,89	90	7	1	0,93	90	3	5	0,97
10	68	12	0	0,85	70	10	0	0,88	72	3	5	0,96
11	13	2	0	0,87	13	0	2	1	13	0	2	1

continua na próxima página

Tabela 5.14 Resultados do Teste de Mutação SQL – Sistema de Empenho (continuação)

Resultados do Teste de Mutação SQL do Sistema de Empenho												
BDT – Base de Dados de Teste												
Cons. SQL	Legenda: MT- Mortos / VV- Vivos / EM- Escore de Mutação / EQ- Equivalente											
	Mutantes											
	1ª Execução				2ª Execução				3ª Execução			
	MT	VV	EQ	EM	MT	VV	EQ	EM	MT	VV	EQ	EM
12	13	2	0	0,87	13	0	2	1	13	0	2	1
13	16	2	0	0,89	16	0	2	1	16	0	2	1
14	13	2	0	0,87	13	0	2	1	13	0	2	1
15	21	3	0	0,88	22	2	0	0,92	22	0	2	1
16	17	3	0	0,85	18	1	1	0,95	18	0	2	1
17	15	4	0	0,79	17	2	0	0,89	17	0	2	1
18	6	2	0	0,75	6	0	2	1	6	0	2	1
19	19	4	0	0,83	20	0	3	1	20	0	3	1
20	0	1	0	0	0	0	1	0	0	0	1	0
21	0	1	0	0	0	0	1	0	0	0	1	0
22	6	1	0	0,86	7	0	0	1	7	0	0	1
23	18	0	0	1	18	0	0	1	18	0	0	1
24	0	1	0	0	0	0	1	0	0	0	1	0
25	11	2	0	0,85	11	0	2	1	11	0	2	1
26	4	2	0	0,67	4	0	2	1	4	0	2	1
27	2	1	0	0,67	2	0	1	1	2	0	1	1
28	2	1	0	0,67	2	0	1	1	2	0	1	1
29	3	0	0	1	3	0	0	1	3	0	0	1
30	12	1	0	0,92	12	0	1	1	12	0	1	1
31	93	0	0	1	93	0	0	1	93	0	0	1
32	44	4	0	0,92	44	0	4	0,98	44	0	4	1
33	113	4	0	0,97	113	2	2	0,98	113	2	2	0,98
34	64	6	0	0,91	64	1	5	0,98	64	1	5	0,98

O custo para geração dos mutantes é alto, porém foi possível verificar a eficácia do teste de mutação associado aos problemas de qualidade nos dados do processo ETL.

Tabela 5.15 Escores de Mutação por Operadores no Sistema de Empenho

Consulta SQL	Operador	Quantidade de Mutantes	Mortos	Vivos	Equivalentes	Escores Mutação
1	SEL	1	0	0	1	0
2	SEL	1	0	0	1	0
3	SEL	1	0	0	1	0
	ABS	1	0	0	1	0
	UOI	3	3	0	0	1
	IRD	2	2	0	0	1
	ROR	7	7	0	0	1
4	SEL	1	0	0	1	0
	ABS	2	1	0	1	1
	UOI	3	3	0	0	1

continua na próxima página

Tabela 5.15 Escores de Mutação por Operadores no Sistema de Empenho (continuação)

Consulta SQL	Operador	Quantidade de Mutantes	Mortos	Vivos	Equivalentes	Escores Mutação
	IRD	2	2	0	0	1
	ROR	7	7	0	0	1
5	SEL	1	0	0	1	0
	ABS	2	1	0	1	1
	UOI	3	3	0	0	1
	ROR	7	7	0	0	1
6	SEL	1	0	0	1	0
	ABS	2	1	0	1	1
	UOI	3	3	0	0	1
	ROR	7	7	0	0	1
7	SEL	1	0	0	1	0
	ABS	2	1	0	1	1
	UOI	3	3	0	0	1
	IRD	2	2	0	0	1
	ROR	7	7	0	0	1
8	SEL	1	0	0	1	0
	JOI	4	3	0	1	1
	ABS	8	7	0	1	1
	UOI	12	12	0	0	1
	IRC	5	5	0	0	1
	IRD	9	9	0	0	1
	ROR	21	21	0	0	1
	LCR	5	5	0	0	1
9	SEL	1	0	0	1	0
	JOI	4	1	3	0	0,25
	ABS	8	4	0	4	1
	UOI	12	12	0	0	1
	IRC	5	5	0	0	1
	IRD	42	42	0	0	1
	ROR	21	21	0	0	1
	LCR	5	5	0	0	1
10	SEL	1	0	0	1	0
	JOI	4	1	3	0	0,25
	ABS	8	4	0	4	1
	UOI	12	12	0	0	1
	IRC	5	5	0	0	1
	IRD	24	24	0	0	1
	ROR	21	21	0	0	1
	LCR	5	5	0	0	1
11	SEL	1	0	0	1	0
	ABS	2	1	0	1	1
	UOI	3	3	0	0	1
	IRD	2	2	0	0	1
	ROR	7	7	0	0	1
12	SEL	1	0	0	1	0
	ABS	2	1	0	1	0
	UOI	3	3	0	0	1
	IRD	2	2	0	0	1

continua na próxima página

Tabela 5.15 Escores de Mutação por Operadores no Sistema de Empenho (continuação)

Consulta SQL	Operador	Quantidade de Mutantes	Mortos	Vivos	Equivalentes	Escores Mutação
	ROR	7	7	0	0	1
13	SEL	1	0	0	1	0
	ABS	2	1	0	1	1
	UOI	3	3	0	0	1
	IRD	5	5	0	0	1
	ROR	7	7	0	0	1
14	SEL	1	0	0	1	0
	ABS	2	1	0	1	1
	UOI	3	3	0	0	1
	IRD	2	2	0	0	1
	ROR	7	7	0	0	1
15	SEL	1	0	0	1	0
	ABS	2	1	0	1	1
	UOI	3	3	0	0	1
	IRD	11	11	0	0	1
	ROR	7	7	0	0	1
16	SEL	1	0	0	1	0
	ABS	2	1	0	1	1
	UOI	3	3	0	0	1
	IRD	7	7	0	0	1
	ROR	7	7	0	0	1
17	SEL	1	0	0	1	0
	ABS	2	1	0	1	1
	UOI	3	3	0	0	1
	IRD	6	6	0	0	1
	ROR	7	7	0	0	1
18	SEL	1	0	0	1	0
	ABS	2	1	0	1	1
	UOI	6	6	0	0	1
	IRC	2	2	0	0	1
19	SEL	1	0	0	1	0
	SUB	1	1	0	0	1
	ABS	4	2	0	2	1
	UOI	6	6	0	0	1
	NLS	1	1	0	0	1
	IRC	5	5	0	0	1
	IRT	5	5	0	0	1
20	SEL	1	0	0	1	0
21	SEL	1	0	0	1	0
22	SEL	1	1	0	0	1
	ABS	2	2	0	0	1
	UOI	3	3	0	0	1
	IRD	1	1	0	0	1
23	SEL	1	1	0	0	1
	ABS	2	2	0	0	1
	UOI	3	3	0	0	1
	IRC	2	2	0	0	1
	IRD	2	2	0	0	1

continua na próxima página

Tabela 5.15 Escores de Mutação por Operadores no Sistema de Empenho (continuação)

Consulta SQL	Operador	Quantidade de Mutantes	Mortos	Vivos	Equivalentes	Escores Mutação
	BTW	4	4	0	0	1
	IRT	4	4	0	0	1
24	SEL	1	0	0	1	0
25	SEL	1	0	0	1	0
	ABS	2	1	0	1	1
	UOI	3	3	0	0	1
	ROR	7	7	0	0	1
26	SEL	1	0	0	1	0
	ABS	2	1	0	1	1
	UOI	3	3	0	0	1
27	SEL	1	0	0	1	0
	IRT	2	2	0	0	1
28	SEL	1	0	1	0	0
	IRT	2	2	0	0	1
29	SEL	1	1	0	0	1
	IRT	2	2	0	0	1
30	SEL	1	0	0	1	0
	ABS	2	2	0	0	1
	UOI	3	3	0	0	1
	ROR	7	7	0	0	1
31	SEL	1	1	0	0	1
	JOI	8	8	0	0	1
	ABS	10	10	0	0	1
	UOI	15	15	0	0	1
	IRC	8	8	0	0	1
	IRD	30	30	0	0	1
	ROR	21	21	0	0	1
32	SEL	1	0	0	1	0
	ABS	6	4	0	2	1
	UOI	9	9	0	0	1
	IRD	19	19	0	0	1
	JOI	4	3	0	1	1
	IRC	2	2	0	0	1
	ROR	7	7	0	0	1
33	SEL	1	0	0	1	0
	ABS	14	13	0	1	1
	UOI	21	21	0	0	1
	IRD	43	43	0	0	1
	JOI	8	6	0	2	1
	IRC	16	16	0	0	1
	ROR	14	14	0	0	1
34	SEL	1	0	0	1	0
	ABS	8	4	0	4	1
	UOI	12	12	0	0	1
	IRC	6	6	0	0	1
	IRD	32	32	0	0	1
	JOI	4	3	1	0	0,75
	ROR	7	7	0	0	1

Nos gráficos das Figuras 5.12 e 5.13 são apresentadas respectivamente, a Média do Escore de Mutação por Operador de Mutação SQL e a Quantidade de Mutantes por Operador de Mutação SQL.

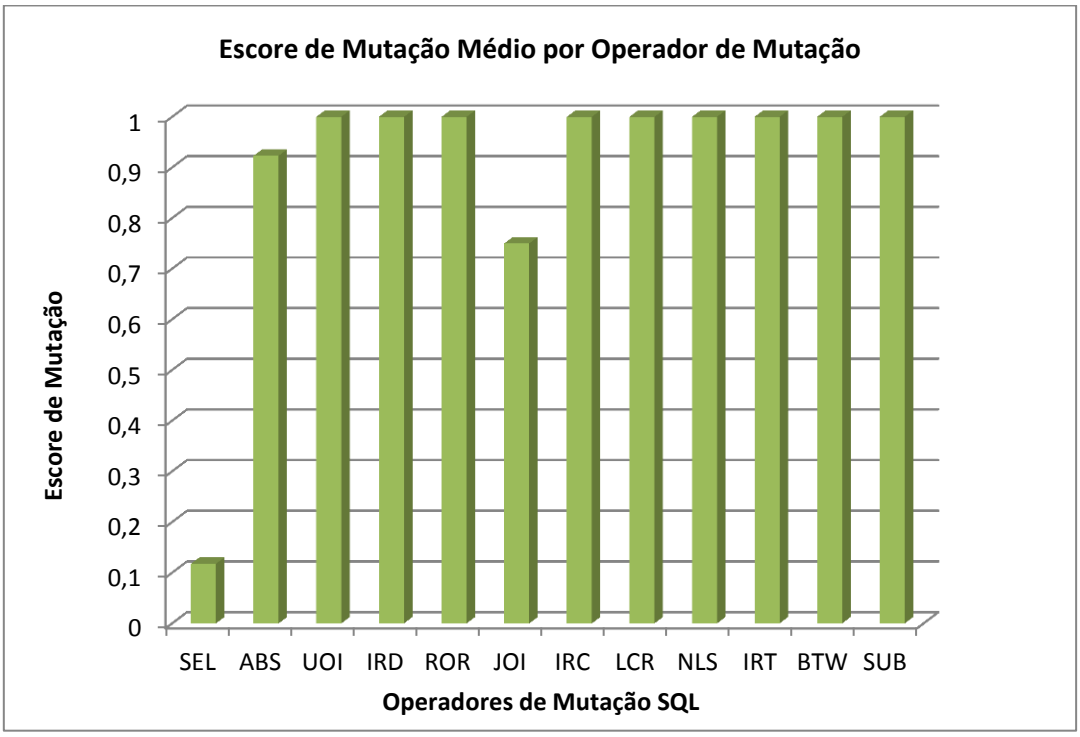


Figura 5.12 Escore de Mutação Médio por Operador de Mutação no Sistema de Empenho

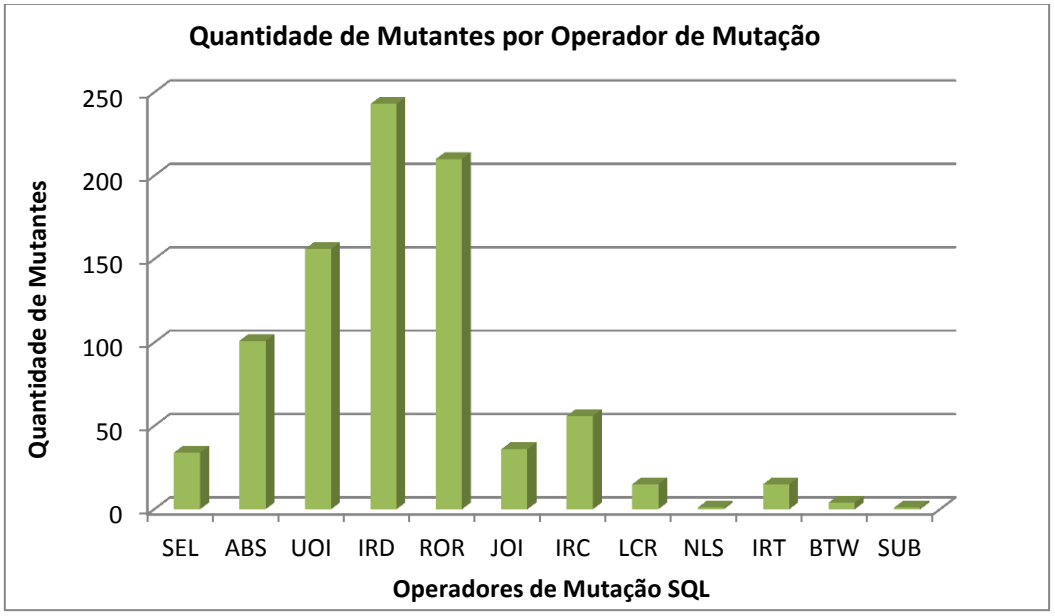


Figura 5.13 Quantidade de Mutantes por Operador de Mutação SQL no Sistema de Empenho

Fase 3: Teste no DW do Sistema de Empenho

O resultado da aplicação das classes de defeito nos dados do DW na base de teste do sistema de Empenho resultou na Tabela 5.16.

A Tabela 5.16 apresenta o total de instâncias de dados alternativas geradas para cada classe de defeito, e deste total quantos defeitos foram considerados como aceitos e quantos defeitos foram rejeitados no processo de teste para o DW.

Tabela 5.16 Quantidade de Defeitos por Classes de Defeito no DW do Sistema de Empenho

Base de Dados do DW			
Classes de Defeito nos Dados	**Total de Instância de Dados Alternativos**	**Total de Valores Aceitos**	**Total de Valores Rejeitados**
Caracteres de Espaço em Branco Incorretos	1983	1305	678
Codificação Incorreta	1770	1077	693
Dados Contraditórios	4198	2014	2184
Duplicados	295	175	120
Falta de Dados Corretos	4198	2013	22185
Valores Embutidos	14160	8703	5457
Fora do Domínio	29386	9278	20108
Nulos	4198	1404	2794
Tamanho Incorreto	3998	3	3995
Transposição de Palavras	1770	1089	681

A seguir, a Figura 5.14 apresenta graficamente a porcentagem de defeitos aceitos no DW do sistema de Empenho para cada classe defeito.

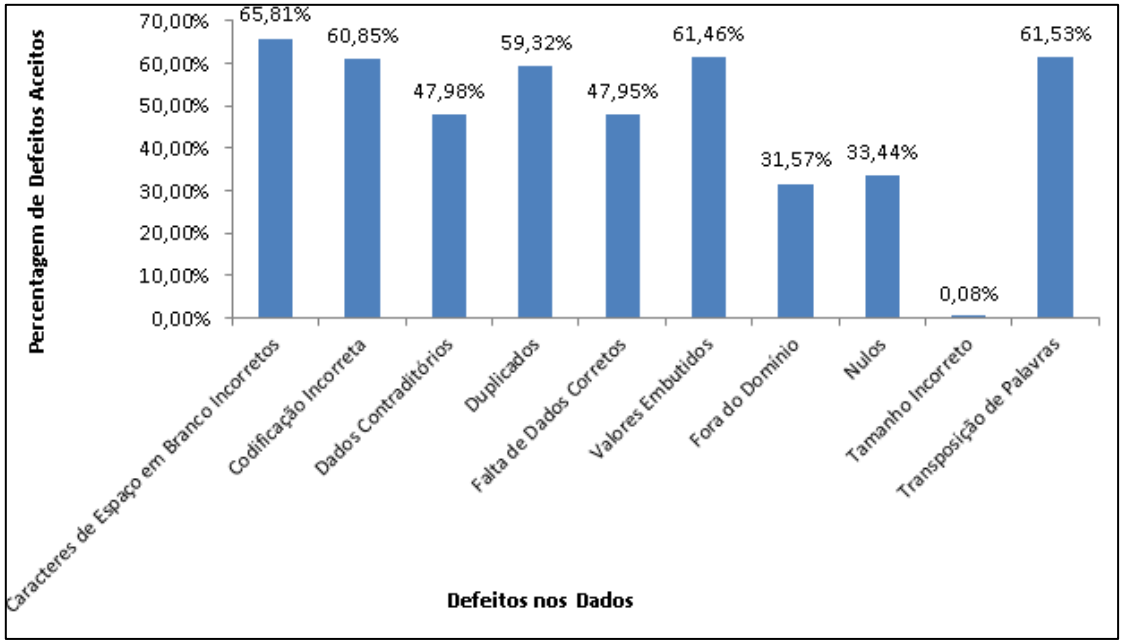


Figura 5.14 Classes de Defeito nos Dados do DW do Sistema de Empenho

De acordo com os resultados, verificou-se que para o DW do sistema de Empenho houve maior incidência da classe de defeito Caracteres de Espaço em Branco Incorretos, pois as tabelas contidas na base de dados não possuem uma restrição que impeça a inserção de espaços em branco no início do valor no campo de tipo *string*.

A menor incidência de defeito foi para classe de defeito Tamanho Incorreto, isto porque a base de dados impede a inserção de dados de tamanho maior do que o determinado na tabela, para cada inserção apresentou a categoria de erro *DataError*.

Com a realização do estudo de caso, no qual as classes de defeitos foram aplicadas no esquema de dados e nos dados do DW, foi possível verificar que alguns dos defeitos permaneceram no DW em menor proporção, indicando mesmo assim possíveis problemas de qualidade nos dados desses ambientes.

5.5 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados três estudos de caso com o objetivo de apresentar a aplicabilidade e a eficácia da técnica de teste baseado em defeitos aplicado nas três fases de desenvolvimento do *Data Warehouse*. O processo de teste em *Data Warehouse* foi validado por meio de estudos de caso com dados fictícios e reais, nos quais foram revelados por meio da análise de instâncias de dados alternativas os problemas de qualidade de dados que podem prejudicar os dados na fase de Fonte de

Dados e dados no DW. Para o ETL aplicaram-se operadores de mutação SQL associados às classes de defeitos no ETL a fim de que essas consultas fossem testadas e validadas por meio do critério Análise de Mutantes SQL. Ferramentas foram desenvolvidas para automatizar o processo de teste e aumentar o seu desempenho, visto que tanto o número de mutantes SQL gerados quanto o número de instâncias de dados alternativas geradas é muito elevado, e seria praticamente inviável a análise de todas as informações manualmente.

Na ferramenta utilizada para análise da qualidade dos dados foram implementadas as classes de defeitos nos dados baseadas nos problemas de qualidade de dados descritos em Barateiro e Gualhardas (2005) e Singh e Singh (2010) e a ferramenta desenvolvida para análise dos mutantes SQL teve sua implementação baseada nos operadores de mutação SQL descritos nos trabalhos de Tuya, Suarez-Cabal e De La Riva (2007) e Cabeça, Jino e Leitão-Junior (2009).

A Tabela 5.17 apresenta um resumo quantitativo de análise de instâncias de dados alternativas geradas para as Fases I e III e o número de mutantes gerados para a Fase II de cada estudo de caso. As Fases I e III correspondem aos dados de resultados de teste de qualidade nos dados nas fontes de dados e no DW e a Fase III corresponde aos dados de resultados de teste de mutação SQL nas consultas ETL do DW.

Tabela 5.17 Resumo Quantitativo do Resultado das Fases de Teste 1, 2 e 3

Estudo de Caso	**Fase de Teste (FT)**	**N° de Mutantes Gerados/ N° de Instâncias de Dados Alternativas Geradas**	**Mutantes Vivos/Valores Aceitos**	**Mutantes Mortos/ Valores Rejeitados**
I	**FT1**	2230	1450	780
	FT2	4563	4537	20
	FT3	6520	3411	3109
II	**FT1**	67240	27777	39463
	FT2	1330	8	1322
	FT3	64263	10244	54019
III	**FT1**	104902	14214	90688
	FT2	870	9	796
	FT3	65956	27061	53983

Conforme verificado na Tabela 5.17, foram revelados defeitos em todas as fases de teste do processo de desenvolvimento do DW. Na Fase de Teste 1 dos estudos de caso foram encontrados defeitos aceitos pelo esquema de dados que prejudicam a qualidade de dados das fontes de dados, pois os dados originais das fontes de dados possivelmente já possuem erros que serão passados para a fase seguinte de desenvolvimento do DW que é o processo ETL, na qual os dados são integrados e manipulados.

Na Fase de Teste 2 dos estudos de caso, por meio da análise de mutação SQL foram testadas as consultas SQL do processo ETL. Os mutantes gerados, executados e analisados nos casos de teste possibilitaram revelar a maioria dos defeitos, porém alguns mutantes ainda ficaram vivos no teste de algumas consultas, e para matá-los houve necessidade de testar mais vezes com novos conjuntos de casos de teste. Com as execuções dos casos de teste nem todos os mutantes foram mortos, alguns ainda permaneceram vivos, porém os escores de mutação para os três estudos de caso realizados apresentaram boas medidas, não havendo a necessidade de realizar mais testes. Houve nos testes desta fase em alguns dos estudos de caso, a ocorrência de mutantes equivalentes, cujo resultado da execução foi igual ao resultado dos programas em teste, que neste caso, são as consultas SQL originais.

Na Fase de Teste 3 que também é semelhante ao processo de teste da Fase 1, foi testada a qualidade de dados nos dados do DW por meio da validação do esquema de dados e inserção de defeitos com as instâncias de dados alternativas, sendo possível também revelar defeitos que foram aceitos devido à ocorrência de restrições ausentes ou incorreta no esquema de dados. A proporção de defeitos inseridos nesta fase de teste apresentou-se menor que a proporção de defeitos inseridos na Fase de Teste 1.

Os defeitos apareceram em todas as fases de teste devido ao fato de que o processo de teste tem como foco realizar a aplicação da técnica de teste baseado em defeitos com a finalidade de revelar os defeitos e não de corrigi-los. Então se aplicado o processo de teste na Fase 1 de um DW e apresentado os defeitos que possui o esquema de dados e os dados dessa fase, e se corrigidos estes defeitos antes da Fase de Teste 2, provavelmente o número de defeitos encontrados poderia ser reduzido na continuidade do desenvolvimento do DW.

6. CONCLUSÕES E TRABALHOS FUTUROS

6.1 SÍNTESE DO TRABALHO

O presente trabalho explorou o uso da técnica de teste baseado em defeitos aplicada desde a fase inicial de desenvolvimento do *Data Warehouse* representada pelas Fontes de Dados, ao processo ETL do *Data Warehouse,* que é considerado o processo mais crítico de desenvolvimento desses ambientes, e o próprio *Data Warehouse*.

Neste trabalho foi proposto um processo de teste para DW, no qual foram identificadas três etapas de teste, nas fontes de dados, no ETL e nos dados do DW. Nessas etapas foram investigados os operadores de mutação SQL, as instâncias de dados alternativas para as fontes de dados e dados do DW, além dos problemas de qualidade nos dados. A partir do estudo sobre os problemas relacionados com a qualidade de dados em DW encontrados na literatura, diversos problemas que ocorrem no desenvolvimento desses ambientes em todas as suas fases de desenvolvimento foram identificados, sendo explorados os problemas apresentados nas Fontes de Dados e na fase ETL. Como o processo ETL envolve instruções SQL para manipulação, limpeza e transformação dos dados, executados principalmente na *Staging Area*, esses problemas de qualidade de dados foram analisados e relacionados aos operadores de mutação SQL existentes na literatura.

Duas ferramentas foram desenvolvidas para auxiliar na execução do processo de teste proposto. Uma ferramenta que implementa os operadores de mutação SQL com a finalidade de aplicação no teste das consultas SQL do ETL e uma ferramenta que implementa as classes de defeito nos dados por meio da análise de instâncias de dados alternativas com o intuito de testar a qualidade dos dados das fontes de dados e dados do DW.

Nos estudos de caso, os operadores de mutação SQL foram empregados em instruções aplicadas na fase ETL de um DW para verificar se a aplicabilidade da abordagem é capaz de revelar defeitos nesta fase. As alterações nos dados também foram geradas para verificar se a aplicabilidade da análise de instâncias de dados alternativas nas fontes de dados e nos próprios dados do DW revelam defeitos nestas fases.

Os resultados obtidos com a realização dos estudos de caso foram promissores em relação a aplicabilidade e eficácia da técnica de teste baseado em defeitos no contexto de DW, pois foram capazes de revelar defeitos nas três fases de desenvolvimento do DW.

Nos estudos de caso foi observado que os defeitos testados nas fontes de dados apresentaram maior proporção de defeitos aceitos em relação ao DW, visto que a permanência dos mesmos ocorreu devido às restrições ausentes ou incorretas no esquema de dados presentes nas fontes de dados. Alguns defeitos levaram à descoberta de outros defeitos, confirmando a hipótese do Efeito do Acoplamento descrito na técnica de teste baseado em defeitos.

São contribuições deste trabalho:

- A abordagem de teste incluindo a fase de desenvolvimento inicial do DW que são as Fontes de Dados, passando pelo processo ETL considerado a fase mais trabalhosa e crítica do DW e a fase final, que são os dados do DW. Os testes realizados desde as fases iniciais de desenvolvimento do DW diminuem os riscos da geração de dados errôneos e evita altos custos com correção e manutenção nesses ambientes;
- A identificação e classificação de defeitos associados aos problemas de qualidade dos dados no ETL que posteriormente foram relacionados aos operadores de mutação SQL associados aos problemas de qualidade de dados na Fase de ETL, compreendendo a fase de extração, limpeza, transformação e carga dos dados para um *DW*;
- A identificação e classificação de defeitos nos dados associados aos problemas de qualidade encontrados na fase de Fontes de Dados e a aplicação destes também nos dados do DW;
- A verificação da aplicabilidade do critério de teste análise de mutantes em uma abordagem de teste no contexto de ambientes de DW para o processo ETL e o uso da análise de instâncias de dados alternativas para a abordagem de teste nas fontes de dados e nos dados do DW, visto que ainda não existiam classes de defeitos para *Data Warehouse*;
- O desenvolvimento de duas ferramentas: uma para teste de mutação SQL e outra para teste de Qualidade de Dados. A primeira ferramenta implementa a geração, execução e análise dos mutantes SQL. A segunda ferramenta implementa a

geração, execução e análise de instâncias de dados alternativas, ambas contribuem na automatização do processo e simplificam o trabalho do testador;

- A execução da abordagem em estudos de caso com o objetivo de validação da abordagem de teste baseado em defeitos com o emprego das ferramentas;
- A investigação dos problemas que ocorrem no desenvolvimento de um *Data Warehouse*, dando foco na qualidade de dados desses ambientes, que é considerado o fator de sucesso para projetos de DW.

6.2 TRABALHOS FUTUROS

Alguns estudos e melhorias podem ser realizados na abordagem de teste para DW proposta neste trabalho, dentre eles destacam-se:

- O estudo e aplicabilidade das técnicas de redução de bases de dados e também das técnicas de geração de dados de testes para os ambientes de DW, visto que principalmente as técnicas de redução de dados de teste tornam-se necessárias nos contextos de DW devido ao grande volume de informações nesses ambientes, sendo esta uma limitação deste trabalho, visto que não foi investigado nem explorado o uso de nenhuma técnica de redução nos dados;
- A investigação da viabilidade da aplicação da técnica de teste baseado em defeitos em outras fases de desenvolvimento do DW, visto que no trabalho de Singh e Singh (2010) foram citados também problemas de qualidade de dados existentes em outras fases de desenvolvimento do DW, como as fases de *Data Profiling* e *Design* do Esquema, que também tem sua importância no desenvolvimento do DW.
- Melhorias nas funcionalidades da ferramenta de teste de mutação SQL, pois a mesma está permitindo conexão apenas com o banco de dados SQL Server, e também melhorias na análise de resultados, possibilitando a geração de relatórios para o testador, pois a mesma apresenta os resultados apenas em tela.
- Melhorias na ferramenta de teste de qualidade nos dados, pois a mesma não possui interface gráfica, o que dificulta a interação com um usuário comum. Também deverá ser aperfeiçoada para ser aplicada em outros bancos de dados, como *MySQL*, *PostgreSQL*, visto que na sua implementação são utilizados alguns comandos e tipos de atributos específicos do SQL Server. As evoluções

nas ferramentas seriam interessantes para disseminação do conhecimento da abordagem da técnica de teste baseado em defeito, pouco conhecida no meio acadêmico e industrial.

- A abrangência da abordagem nos contextos de *Data Warehouse* para dados espaciais também é interessante, assim como a melhoria do processo de teste realizado na fase inicial de fontes de dados que se limitou aos esquemas de bancos de dados relacionais.
- A geração, execução e análise de novas classes de defeito para as fases de desenvolvimento estudadas ou para fases não exploradas no contexto de desenvolvimento do DW neste trabalho.
- A comparação dos conjuntos de casos de teste em relação ao critério Análise de Mutação SQL, com a utilização de bases de dados distintas (Desenvolvimento, Testes e Produção), aplicando redução de dados e geração de dados de testes para analisar a eficácia dos testes no ETL executados nesses contextos.

REFERÊNCIAS

ACREE, A.; BUDD, T.; DEMILLO, R.; LIPTON, R., SAYWARD, F. Mutation Analysis. Technical Report GIT-ICS-79/08, Georgia Inst. of Technology, 1979.

AGRAWAL, H.; DEMILLO, R.A.; HATHAWAY, R.; HSU, W.; KRAUSER, E. W.; MARTIN, R. J.; MATHUR, A. P.; SPAFFORD, E. Design of mutants operators for the C programming language. Technical Report SERC-TR41-P, Software Engineering Research Center, Purdue University, West Lafayette, EUA, 1989.

ALEXANDER, R.; BIEMAN, J.; GHOSH, S.; JI, B. Mutation of Java objects. Proc. 13th International Symp. Software Reliability Engineering, p. 341-351, 2002.

AMARAL, G. C. M. AQUAWARE: Um ambiente de suporte à qualidade de dados em Data Warehouse. 2003. 164 f. Dissertação (Mestrado em Informática)- Universidade Federal do Rio de Janeiro- UFRJ, Instituto de Matemática – IM, Núcleo de Computação Eletrônica - NCE, 2003.

ANDREWS, J. H.; BRIAND, L. C.; LABICHE, Y. Is mutation an appropriate tool for testing experiments? In Proceedings of the 27th international conference on Software engineering, p. 402-411. ACM, 2005.

BARBOSA, E. F.; MALDONADO, J. C.; VINCENZI, A. M. R.; DELAMARO, M. E.; SOUZA, S., JINO, M. . Introdução ao teste de software. Minicurso apresentado no XIV Simpósio Brasileiro de Engenharia de Software, 2000.

BARATEIRO, J.; GALHARDAS, H. A survey of data quality tools. Datenbank-Spektrum, v. 14, n. 48, p. 15-21, 2005.

BARBOSA, E. F.; CHAIM, M.L.; VINCENZI, A.M.R.; DELAMARO, M.E.; JINO,M.; MALDONADO, J.C. Teste Estrutural. In: Introdução ao Teste de Software. Rio de Janeiro: Elsevier, 2007, Cap. 4, p 47-76.

BASHIR, M.; NADEEM, A. Object oriented mutation testing: A survey. International Conference on Emerging Technologies, p. 1-6, Islamabad, 2012.

BATINI, C.; SCANNAPIECO, M. Data Quality: Concepts, Methodologies and Techniques. Springer, New York, 1a Ed., 1998.

BIEMAN, J.; GHOSH, S.; ALEXANDER, R. A technique for mutation of Java objects. IEEE 16th Annual International Conference on Automated Software Engineering, p. 337-340, 2001.

BOWSER, J. Reference manual for ADA Mutant Operators. Technical Report GIT-SERC-88/02, Georgia Inst. of Technology, 1988.

BRADBURY, J.; CORDY, J.; DINGEL, J. ExMAn: A generic and customizable framework for experimental mutation analysis. Proc. Second Workshop Mutation Analysis, p. 57-62, 2006.

BUDD, T.; DEMILLO, R.; LIPTON, R.; SAYWARD, F. The design of a prototype mutation system for program testing. Proc. Am. Fed. of Information Processing Soc. National Computer Conference, v. 74, p. 623-627, 1978.

BUDD, T. A. Mutation analysis: Ideas, examples, problems and prospects. Computer Program Testing, v. 8, p. 129-l48, 1981.

CABEÇA, A.; JINO, M.; LEITAO-JUNIOR, P. Mutation analysis for SQL database applications. 4th International Conference on Software Engineering Advances, p. 146-151, 2009.

CAMARGO, L.C.; VERGILIO, S. Classificação de defeitos para programas MapReduce: Resultados de um estudo empírico. 7th Brazilian Workshop on Systematic and Automated Software Testing, SAST, 2013.

CHAN, W.; CHEUNG, S.; TSE, T. Fault-based testing of database application programs with conceptual data model. Proc. Fifth International Conference Quality Software, p. 187-196, 2005.

CRAIG, R. D.; JASKIEL, S. P. Systematic Software Testing. Artech House, 2002.

DELAMARO, M.; MALDONADO, J. Proteum - A tool for the assessment of test adequacy for C programs. Proc. Conf. Performability in Computing Systems, p. 79-95, 1996.

DELAMARO, M.; MALDONADO, J. Interface mutation: Assessing testing quality at interprocedural level. 19th International Conference Chilean Computer Science Soc., p. 78-86, 1999.

DELAMARO, M.; MALDONADO, J.; PASQUINI, A.; MATHUR, A. Interface mutation test adequacy criterion: An empirical evaluation. Empirical Software Eng., p. 111-142, 2001.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. Conceitos Básicos. In: Introdução ao Teste de Software. Rio de Janeiro: Elsevier, 2007 , Cap. 1, p. 1-7.

DELAMARO, M. E.; BARBOSA, E. F.; VINCENZI, A.M.R.; MALDONADO, J.C. Teste de Mutação. In: Introdução ao Teste de Software. Rio de Janeiro: Elsevier, 2007, Cap. 5, p. 77-118.

DEMILLO, R.; LIPTON, R.; SAYWARD, F. Hints on test data selection: Help for the practicing programmer. Computer, p. 34-41, 1978.

DEREZINSKA, A. Object-oriented mutation to assess the quality of tests. 29th EUROMICRO Conference New Waves in System Architecture, IEEE, p. 417, 2003.

DEREZINSKA, A. An experimental case study to applying mutation analysis for SQL queries. International Multiconference on Computer Science and Information Technology, p. 559-566, 2009.

ELGAMAL, N.; ELBASTAWISSY, A.; GALAL-EDEEN, G. Towards a Data Warehouse testing framework. 9th International Conference on ICT and Knowledge Engineering, 2011.

ELGAMAL, N.; ELBASTAWISSY, A.; GALAL-EDEEN. Data Warehouse testing. Proceedings of the Joint EDBT/ICDT, 2013.

ELGAMAL, N. Data Warehouse Testing. 2015. 279 f. Tese (Doctoral in Information Systems) - Faculty of Computers and Information, Cairo University, 2015.

EMER, M. C. F. P. Abordagem de teste baseada em defeitos para esquemas de dados. 2007. 140 f. Tese (Doutorado em Engenharia Elétrica/ Engenharia de Computação) – Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas, 2007.

FABBRI, S.C.P.F.; VINCENZI, A.M.R.; MALDONADO, J.C. Teste Funcional. In: Introdução ao Teste de Software. Rio de Janeiro: Elsevier, 2007, Cap. 2, p. 9-25.

GOLFARELLI, M.; RIZZI, S. A comprehensive approach to Data Warehouse testing. Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP, p. 17-24, 2009.

GOLFARELLI, M.; RIZZI, S. Data Warehouse testing: A prototype-based methodology. Information and Software Technology, v. 53, p. 1183-1198, 2011.

INMON, W. H. Building the Data Warehouse. Wiley Publishing, 4th Edition, 2012.

JIA, Y.; HARMAN, M. MILU: A customizable, runtime-optimized higher order mutation testing tool for the full C language. Proc. Third Testing: Academic and Industrial Conf. Practice and Research Techniques, p. 94-98, 2008.

JIA, Y.; HARMAN, M. An analysis and survey of the development of mutation testing. IEEE Transactions on Software Engineering, v. 37, n. 5, p. 649-678, 2011.

KIM, S.; CLARK, J.; MCDERMID, J. Assessing test set adequacy for object oriented programs using class mutation. Proc. Third Symp. Software Technology, 1999.

KIM, S.; CLARK, J.; MCDERMID, J. Class mutation: Mutation testing for object-oriented programs. Proc. Net. Object Days Conf. Object-Oriented Software Systems, 2000.

KIMBALL, R.; ROSS, M. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling. John Wiley & Sons, 3th Edition, 2013.

KING, K.; OFFUTT, A. A Fortran language system for mutation-based software testing. Software: Practice and Experience, p. 685-718, 1991.

KULDEEP, D. Model based testing of Data Warehouse. IJCSI International Journal of Computer Science Issues, v. 10, p. 330-336, 2013.

LAUDON, K. C.; LAUDON, J. P. Sistemas de Informação Gerenciais. Prentice Hall, 2013.

LEITÃO-JUNIOR, P. S.; VILELA, P. R. S.; JINO, M.. Mapping faults to failures in SQL manipulation commands. 3rd ACS/IEEE International Conference on Computer Systems and Applications, 2005.

LIPTON, R.; SAYWARD, F. The status of research on program mutation. Proc. Workshop Software Testing and Test Documentation, 1978.

MA, Y. S.; KWON, Y.R.; OFFUTT, A. Inter-class mutation operators for Java. 13th International Symp. Software Reliability Eng., 2002.

MA, Y.S.; OFFUTT, A.; KWON, Y.R. MuJava: An automated class mutation system. Software Testing, Verification and Reliability, v. 15, n. 2, p. 97-133, 2005.

MCCORMICK, D.; FRAKES, W.; ANGUSWAMY, R. A comparison of database fault detection capabilities using mutation testing. International Symposium on Empirical Software Engineering and Measurement (ESEM), p. 323-326, 2012.

MEKTEROVIC, I.; BRKIC, L.; BARANOVIC, M. A generic procedure for integration testing of ETL procedures. Automatika, v. 52, n. 2 , p. 169-178, 2011.

MUNAWAR, M.; SALIM, N.; IBRAHIM, R. Towards data quality into the Data Warehouse development. IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing (DASC), p. 1199-1206, 2011.

MYERS, G. J. The Art of Software Testing. John Wiley & Sons, 2nd Ed., 2004.

NAZAR, I. F. X-Tool: uma ferramenta de teste de esquemas para estrutura de dados. 2007. 62 f. Dissertação (Mestrado em Informática) – Programa de Pós-Graduação em Informática, Universidade Federal do Paraná, 2007.

OFFUTT, A.; VOAS, J.; PAYN, J. Mutation Operators for Ada. Technical Report ISSE-TR-96-09, George Mason University, 1996.

OFFUTT, J.; XU, W. Generating test cases for web services using data perturbation. ACM SIGSOFT Software Engineering Notes, v. 29, n. 5, p. 1-10, 2004.

PAPADAKIS, M.; LE TRAON, Y. Using mutants to locate "unknown" faults. IEEE Fifth International Conference on Software Testing, Verification and Validation, p. 691-700, 2012.

QUEIROZ, L. Um benchmark para avaliação de técnicas de busca no contexto de análise de mutantes SQL. 2013. 171 f. Dissertação (Mestrado em Computação) –

Programa em Pós-Graduação do Instituto de Informática, Universidade Federal de Goiás, 2013.

RAHM, E.; DO, H. H. Data Cleaning: Problems and current approaches. IEEE Data Eng. Bull, v. 23, n. 4, p. 3-13, 2000.

SHAHRIAR, H.; ZULKERNINE, M.. MUSIC: Mutation-based SQL injection vulnerability checking. Eighth International Conference Quality Software, p. 77-86, 2008.

SIDI, F.; SHARIAT PANAHY, P.; AFFENDEY, L.; JABAR, M.; IBRAHIM, H.; MUSTAPHA, A. Data Quality: A survey of data quality dimensions. International Conference on Information Retrieval & Knowledge Management (CAMP), p. 300-304, 2012.

SINGH, J.; SINGH, K. Designing a customized test data generator for effective testing of a large database. International Conference on Advanced Computer Theory and Engineering, IEEE, p. 84-88, 2008.

SINGH, R.; SINGH, K. A descriptive classification of causes of data quality problems in Data Warehousing. IJCSI International Journal of Computer Science Issues, v. 7, n. 3, p. 41-50, 2010.

SOMMERVILLE, I. Engenharia de Software. São Paulo: Pearson Addison Wesley, 8ª Edição, 2007.

TURBAN, E.; SHARDA, R.; DELEN, D.; KING, D. Business Intelligence: A Managerial Approach. 2. Ed., Prentice Hall, 2010.

TUYA, J.; SUAREZ-CABAL, M.; DE LA RIVA, C. SQLMutation: A tool to generate mutants of SQL database queries. Second Workshop Mutation Analysis, p. 1, 2006.

TUYA, J.; SUAREZ-CABAL, M.; DE LA RIVA, C. Mutating database queries. Information and Software Technology, v. 49, n. 4, p. 398-417, 2007.

ZHOU, C.; FRANKL, P. Mutation testing for Java database applications. Second International Conference Software Testing Verification and Validation, p. 396-405, 2009.

ZHOU, C.; FRANKL, P. JDAMA: Java Database Application Mutation Analyzer. Software Testing, Verification and Reliability, v. 21, n. 3, p. 241–263, 2011.

APÊNDICE A - DESCRIÇÃO DA FERRAMENTA DE TESTE DE QUALIDADE DOS DADOS

A ferramenta gerada para suporte à geração e execução de instâncias de dados alternativas foi desenvolvida em *Python* que é uma linguagem *Open Source*.

A ferramenta não possui interface gráfica, e, portanto é utilizado o *Prompt de Comando* para execução da mesma. O script denominado Mutation.py contém o código que apresenta toda a lógica de criação das instâncias de dados alternativas que gera os resultados da aplicação das classes de defeitos relacionados aos problemas de qualidade de dados pré-definidos.

Para execução do script Mutation.py deve-se entrar no diretório que está salvo o script e executá-lo através do comando ..\Python27\python.exe Mutation.py, conforme mostra a Figura A.1.

A ferramenta para avaliação de qualidade dos dados após sua execução apresenta como resultado quatro tipos de arquivos com extensão .csv descritos a seguir:

- **1º Arquivo: DefeitosTabela.csv**- Arquivo que apresenta a quantidade de Valores Aceitos, Valores Rejeitados e o Percentual de Defeitos Rejeitados por tabela do banco de dados em teste. Gera apenas um arquivo deste tipo, conforme apresentado na Figura A.4;
- **2º Arquivo: Mutantes_NomeTabela.csv**- Arquivo que apresenta as instâncias de dados alternativas por tabela inseridos da base de dados em teste, ou seja, gera um arquivo chamado Mutantes concatenado com o nome da tabela testada. A quantidade de arquivos gerados varia conforme a quantidade de tabelas em teste, conforme apresentado na Figura A.5;
- **3º Arquivo: Log_NomeTabela.csv-** Arquivo com logs de erros por tabela da base de dados apresentando os erros que surgiram ao tentar inserir o defeito em determinada tabela do banco de dados. Gera um arquivo chamado LogMutation concatenado com o nome da tabela testada. A quantidade de arquivos gerados varia conforme a quantidade de tabelas em teste, conforme apresentado na Figura A.6;
- **4º Arquivo: PercentualDefeitosAceitos.csv-** Arquivo que apresenta o percentual médio de defeito com Valores Aceitos por Classe de Defeito para todas as tabelas da base de dados em teste. Gera apenas um arquivo deste tipo, conforme apresentado na Figura A.7.

As configurações de acesso ao banco de dados a ser testado são informadas por meio do script Mutation.py, onde são informados: servidor de conexão, nome do banco de dados, usuário e senha (se houver). Devem ser informados os parâmetros do banco de origem e do banco de teste. Os nomes das tabelas do banco de dados de origem também são informados por meio do script.

Após digitar o comando de execução o usuário deverá pressionar a tecla “Enter”. A Figura A.2 apresenta a execução da ferramenta, na qual são apresentadas todas as tabelas que foram testadas na ferramenta. Após finalizar este processo, a execução do script finaliza conforme apresentado na Figura A.3.

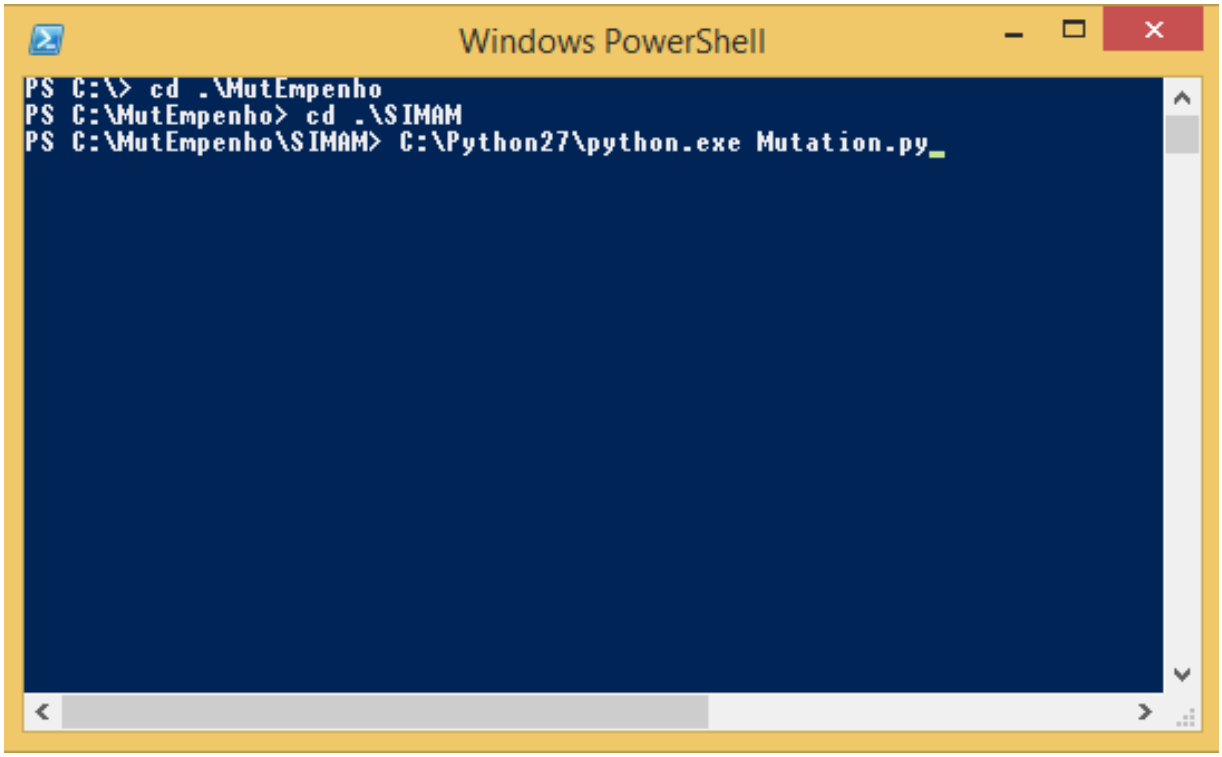


Figura A.1 Comando para Execução do script Mutation.py

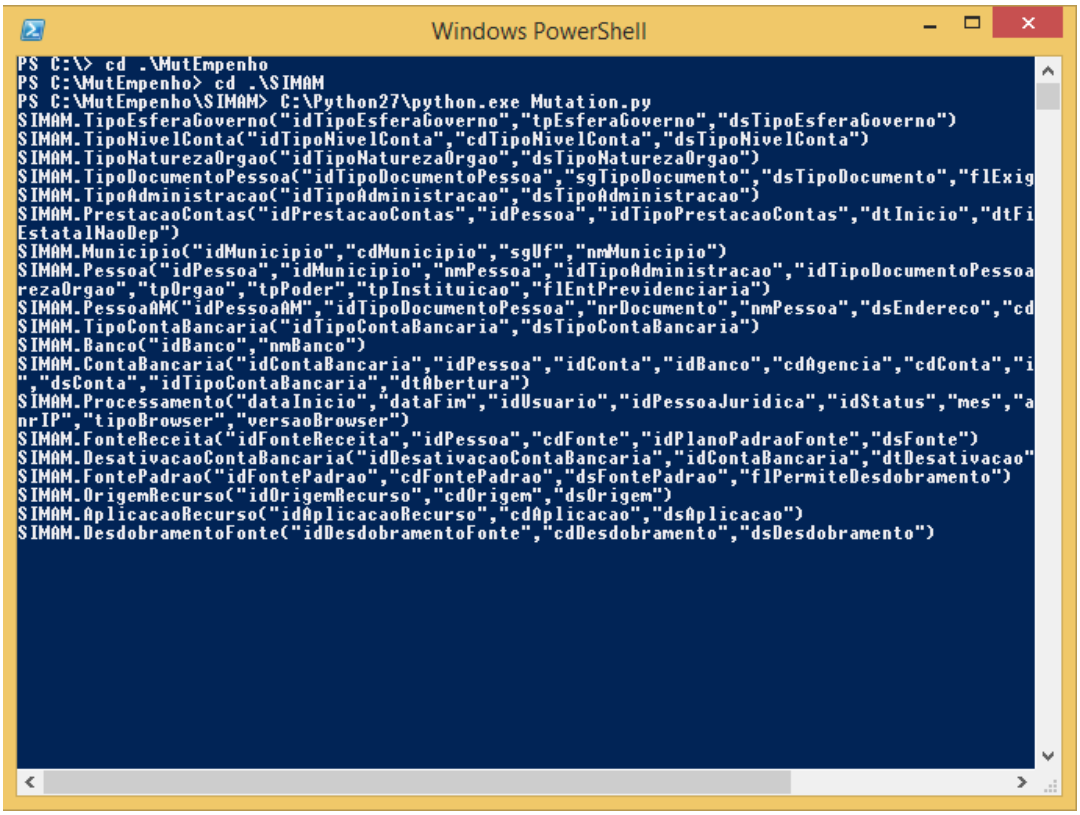


Figura A.2 Execução da ferramenta de Qualidade de Dados

Figura A.3 Fim do Processo de Execução da ferramenta de Qualidade de Dados

O foco da ferramenta são os resultados gerados. A seguir são apresentados exemplos dos 4 tipos de arquivos gerados pela ferramenta.

DefeitosTabela.csv- gera um arquivo CSV com as seguintes informações: Tabela (Nome da Tabela testada), Operador (classe de defeito aplicada), Total de Instâncias (Instâncias de Dados Alternativos gerados), Quantidade de Valores Aceitos, Quantidade de Valores Rejeitados e o Percentual de Defeitos Rejeitados por tabela do banco de dados em teste conforme apresentado na Figura A.4.

	A	B	C	D	E	F
1	Tabela	Classe de defeito	Total de instâncias geradas	Valores Aceitos	Valores Rejeitados	(%) Defeitos Rejeitados
2	despesa.dimCalendario	Nulos	570.0	0.0	570.0	100.00%
3	despesa.dimCalendario	Dados Contraditorios	570.0	0.0	570.0	100.00%
4	despesa.dimCalendario	Fora do Dominio	3990.0	0.0	3990.0	100.00%
5	despesa.dimCalendario	Duplicado	10.0	0.0	10.0	100.00%
6	despesa.dimCalendario	Caracteres de Espaco em Branco Incorretos	280.0	0.0	280.0	100.00%
7	despesa.dimCalendario	Valores Embutidos	2240.0	0.0	2240.0	100.00%
8	despesa.dimCalendario	Codificacao incorreta	280.0	0.0	280.0	100.00%
9	despesa.dimCalendario	Transposicao de palavras	280.0	0.0	280.0	100.00%
10	despesa.dimCalendario	Tamanho Incorreto	500.0	0.0	500.0	100.00%
11	despesa.dimCalendario	Falta de Dados Corretos	570.0	0.0	570.0	100.00%
12	despesa.dimCalendarioAno	Nulos	40.0	10.0	30.0	75.00%
13	despesa.dimCalendarioAno	Dados Contraditorios	40.0	10.0	30.0	75.00%
14	despesa.dimCalendarioAno	Fora do Dominio	280.0	20.0	260.0	92.86%
15	despesa.dimCalendarioAno	Duplicado	10.0	0.0	10.0	100.00%
16	despesa.dimCalendarioAno	Caracteres de Espaco em Branco Incorretos	10.0	10.0	0.0	0.00%
17	despesa.dimCalendarioAno	Valores Embutidos	80.0	80.0	0.0	0.00%
18	despesa.dimCalendarioAno	Codificacao incorreta	10.0	0.0	10.0	100.00%

Figura A.4 Arquivo DefeitosTabela.csv

Mutantes_despesa.DimTipoDespesa.csv- gera um arquivo CSV com as seguintes informações: tabela (Nome da tabela testada), neste exemplo, o nome da tabela é despesa.DimTipoDespesa, Classe Defeito (classe de defeito aplicada), dado inicial (dado original da tabela, sem alterações), dados gerados (dado com defeito) conforme apresentado na Figura A.5.

	A	B	C	D	E	F
1	Tabela	Classe Defeito	dado inicial	dados gerados		
2	despesa.DimTipoDespesa	Fora do Dominio	[-3, '3', 'NAO SE APLICA']	(9187, '3', 'N\xc3O SE APLICA')		
3	despesa.DimTipoDespesa	Fora do Dominio	[-3, '3', 'NAO SE APLICA']	(2820.0, '3', 'N\xc3O SE APLICA')		
4	despesa.DimTipoDespesa	Fora do Dominio	[-3, '3', 'NAO SE APLICA']	(3908, '3', 'N\xc3O SE APLICA')		
5	despesa.DimTipoDespesa	Fora do Dominio	[-3, '3', "NAO SE APLICA']	(6752, '3', 'N\xc3O SE APLICA')		
6	despesa.DimTipoDespesa	Fora do Dominio	[-3, '3', 'NAO SE APLICA']	(-3, 6848, 'N\xc3O SE APLICA')		
7	despesa.DimTipoDespesa	Fora do Dominio	[-2, '2', 'NAO INFORMADO']	(2060, '2', 'N\xc3O INFORMADO')		
8	despesa.DimTipoDespesa	Fora do Dominio	[-2, '2', 'NAO INFORMADO']	(7733.0, '2', 'N\xc3O INFORMADO')		
9	despesa.DimTipoDespesa	Fora do Dominio	[-2, '2', 'NAO INFORMADO']	(5830, '2', 'N\xc3O INFORMADO')		
10	despesa.DimTipoDespesa	Fora do Dominio	[-2, '2', 'NAO INFORMADO']	(7095, '2', 'N\xc3O INFORMADO')		
11	despesa.DimTipoDespesa	Fora do Dominio	[-2, '2', "NAO INFORMADO']	(-2, 2681, 'N\xc3O INFORMADO')		
12	despesa.DimTipoDespesa	Fora do Dominio	[3, 'E', 'Outros Encargos']	(5290, 'E', 'Outros Encargos')		
13	despesa.DimTipoDespesa	Fora do Dominio	[3, 'E', 'Outros Encargos']	(3945.0, 'E', 'Outros Encargos')		
14	despesa.DimTipoDespesa	Fora do Dominio	[3, 'E', 'Outros Encargos']	(3528, 'E', 'Outros Encargos')		
15	despesa.DimTipoDespesa	Fora do Dominio	[3, 'E', 'Outros Encargos']	(9385, 'E', 'Outros Encargos')		
16	despesa.DimTipoDespesa	Fora do Dominio	[3, 'E', 'Outros Encargos']	(3, 8064, 'Outros Encargos')		
17						

Figura A.5 Arquivo Mutantes_despesa.DimTipoDespesa.csv

Log_despesa.DimTipoDespesa.csv- gera um arquivo CSV com as seguintes informações: tabela (Nome da tabela testada), neste exemplo, o nome da tabela é despesa.DimTipoDespesa, classe defeito (classe de defeito aplicada), error (tipo do erro ocorrido), descrição_error (descrição do erro ocorrido) conforme apresentado na Figura A.6.

	A	B	C	D
1	tabela	classe defeito	error	descricao_error
2	despesa.DimTipoDespesa	Fora do Dominio	DataError	('22018', "[22018] [Microsoft][ODBC SQL Server Driver][SQL Server]Conversion failed when converting the varchar value 'aaaaaaaaaa' to data type smallint
3	despesa.DimTipoDespesa	Fora do Dominio	DataError	('22018', "[22018] [Microsoft][ODBC SQL Server Driver][SQL Server]Conversion failed when converting the varchar value '3000-07-12 00:00:00' to data type
4	despesa.DimTipoDespesa	Fora do Dominio	DataError	('22018', "[22018] [Microsoft][ODBC SQL Server Driver][SQL Server]Conversion failed when converting the varchar value 'aaaaaaaaaa' to data type smallint
5	despesa.DimTipoDespesa	Fora do Dominio	DataError	('22003', '[22003] [Microsoft][ODBC SQL Server Driver][SQL Server]Arithmetic overflow error converting numeric to data type varchar. (8115) (SQLExecDire
6	despesa.DimTipoDespesa	Fora do Dominio	DataError	('22001', '[22001] [Microsoft][ODBC SQL Server Driver][SQL Server]String or binary data would be truncated. (8152) (SQLExecDirectW)
7	despesa.DimTipoDespesa	Fora do Dominio	IntegrityError	('23000', "[23000] [Microsoft][ODBC SQL Server Driver][SQL Server]Violation of PRIMARY KEY constraint 'DimTipoDespesa_PK'. Cannot insert duplicate key
8	despesa.DimTipoDespesa	Fora do Dominio	DataError	('22001', '[22001] [Microsoft][ODBC SQL Server Driver][SQL Server]String or binary data would be truncated. (8152) (SQLExecDirectW)
9	despesa.DimTipoDespesa	Fora do Dominio	IntegrityError	('23000', "[23000] [Microsoft][ODBC SQL Server Driver][SQL Server]Violation of PRIMARY KEY constraint 'DimTipoDespesa_PK'. Cannot insert duplicate key
10	despesa.DimTipoDespesa	Fora do Dominio	IntegrityError	('23000', "[23000] [Microsoft][ODBC SQL Server Driver][SQL Server]Violation of PRIMARY KEY constraint 'DimTipoDespesa_PK'. Cannot insert duplicate key
11	despesa.DimTipoDespesa	Fora do Dominio	IntegrityError	('23000', "[23000] [Microsoft][ODBC SQL Server Driver][SQL Server]Violation of PRIMARY KEY constraint 'DimTipoDespesa_PK'. Cannot insert duplicate key
12	despesa.DimTipoDespesa	Fora do Dominio	IntegrityError	('23000', "[23000] [Microsoft][ODBC SQL Server Driver][SQL Server]Violation of PRIMARY KEY constraint 'DimTipoDespesa_PK'. Cannot insert duplicate key
13	despesa.DimTipoDespesa	Fora do Dominio	IntegrityError	('23000', "[23000] [Microsoft][ODBC SQL Server Driver][SQL Server]Violation of PRIMARY KEY constraint 'DimTipoDespesa_PK'. Cannot insert duplicate key
14	despesa.DimTipoDespesa	Fora do Dominio	IntegrityError	('23000', "[23000] [Microsoft][ODBC SQL Server Driver][SQL Server]Violation of PRIMARY KEY constraint 'DimTipoDespesa_PK'. Cannot insert duplicate key
15	despesa.DimTipoDespesa	Fora do Dominio	IntegrityError	('23000', "[23000] [Microsoft][ODBC SQL Server Driver][SQL Server]Violation of PRIMARY KEY constraint 'DimTipoDespesa_PK'. Cannot insert duplicate key
16	despesa.DimTipoDespesa	nulos	IntegrityError	('23000', "[23000] [Microsoft][ODBC SQL Server Driver][SQL Server]Cannot insert the value NULL into column 'skTipoDespesa', table 'NIPTI.despesa.DimTip
17	despesa.DimTipoDespesa	nulos	IntegrityError	('23000', "[23000] [Microsoft][ODBC SQL Server Driver][SQL Server]Cannot insert the value NULL into column 'sgTipoDespesa', table 'NIPTI.despesa.DimTip
18	despesa.DimTipoDespesa	nulos	IntegrityError	('23000', "[23000] [Microsoft][ODBC SQL Server Driver][SQL Server]Cannot insert the value NULL into column 'dsTipoDespesa', table 'NIPTI.despesa.DimTip

Figura A.6 Arquivo Log_despesa.DimTipoDespesa.csv

PercentualDefeitosAceitos.csv - gera um arquivo CSV com as seguintes informações: Classe de Defeito, Percentual de Defeitos (percentual médio de defeitos com Valores Aceitos por Classe de Defeito) para a base de dados em teste, conforme apresentado na Figura A.7.

	A	B
1	Classe de Defeito	Percentual de Defeitos
2	Nulos	1.52%
3	Dados Contraditorios	4.11%
4	Fora do Dominio	3.58%
5	Duplicado	26.47%
6	Caracteres de Espaco em Branco Incorretos	10.17%
7	Falta de Dados Corretos	3.45%
8	Codificacao incorreta	6.78%
9	Transposicao de palavras	6.78%
10	Tamanho Incorreto	0.00%
11	Valores Embutidos	10.17%
12		

Figura A.7 Arquivo PercentualDefeitosAceitos.csv

APÊNDICE B - DESCRIÇÃO DA FERRAMENTA DE TESTE DE MUTAÇÃO SQL

A ferramenta gerada para suporte a geração de consultas mutantes foi desenvolvida em JavaSE 8.0. Devem ser instalados os pacotes de instalação do Java para execução da mesma.

A ferramenta possui interfaces gráficas simples e funciona na plataforma Desktop. O arquivo denominado Operadores.java é o arquivo principal, contendo o código apresentando toda a lógica de criação das consultas SQL mutantes assim como os resultados da aplicação dos operadores de mutação SQL aplicados nas consultas originais.

A seguir são apresentadas as interfaces da ferramenta.

A Figura B.1- Tela de Conexão com o Banco de Dados é a tela inicial do sistema, na qual são solicitadas as informações de conexão com o banco de dados que será utilizado no teste.

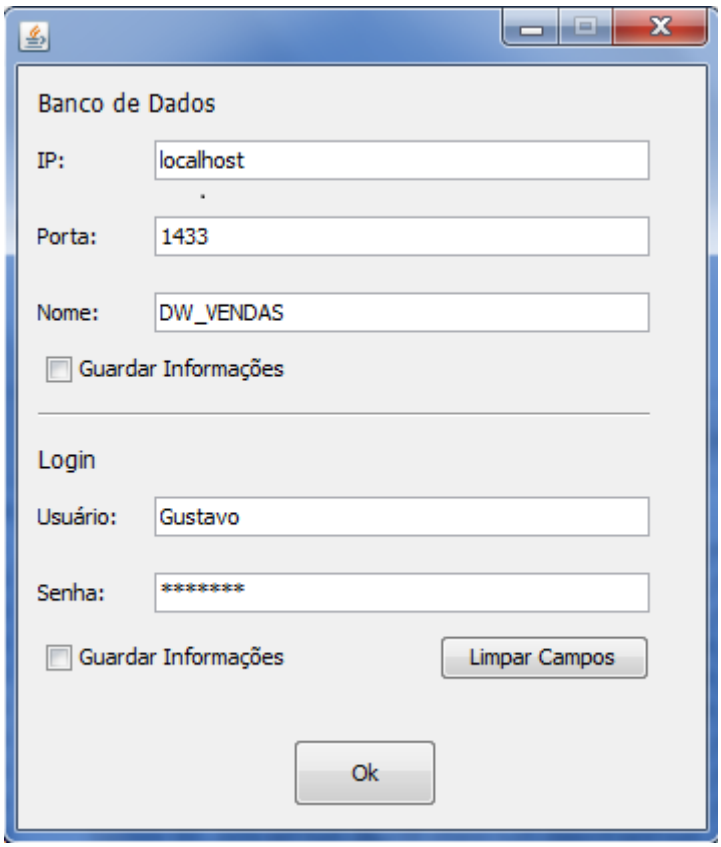


Figura B.1 Tela de Conexão com o Banco de Dados

Após clicar em <OK> na Figura B.1, será apresentada a Figura B.2 - Geração dos Mutantes. A consulta SQL original poderá ser digitada no campo texto ou poderá ser carregada através de arquivo com extensão sql ou txt clicando no botão <Carregar>.

Após inserir a consulta SQL, o usuário deverá clicar no botão <Gerar Mutantes> para que sejam geradas as consultas mutantes referentes à mesma.

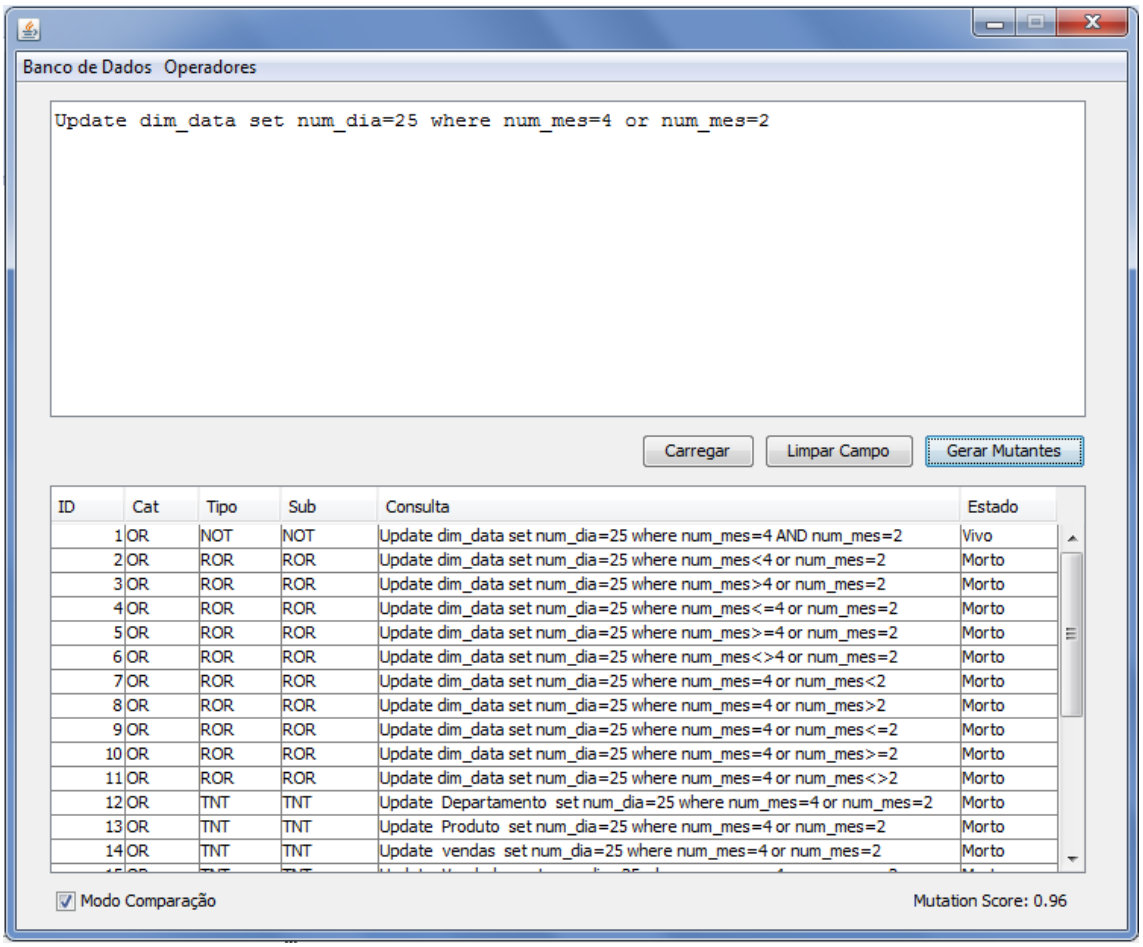


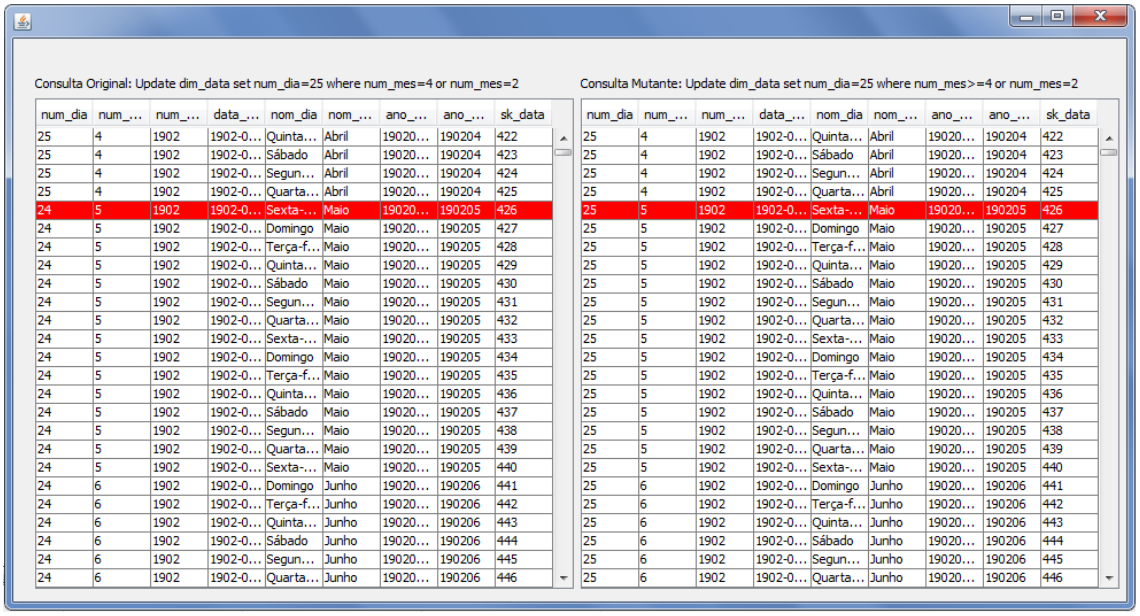
Figura B.2 Geração dos Mutantes

A ferramenta irá gerar as consultas mutantes e executá-las no banco de dados afim de que seja apresentado o resultado da mesma, se Mutante Vivo ou Morto.

Ao final da tela, verifica-se o *Mutation Score* (Escore de Mutação) gerado para o programa em teste (Consulta SQL e Instâncias de Dados) conforme pode ser visto na Figura B.2.

Ao selecionar <Modo Comparação>, uma nova tela será mostrada, na qual serão apresentadas do lado esquerdo a Consulta SQL Original e o resultado dessa consulta por meio das instâncias de dados; e do lado direito a Consulta Mutante e o resultado dessa consulta por meio das instâncias de dados, conforme pode ser visto na Figura B.3.

A linha destacada em vermelho apresenta as diferenças encontradas comparando o resultado da consulta original com o resultado da consulta mutante. Ao ser encontrada uma diferença em uma instância o mutante será considerado Morto. No caso de retornar Vivo, o resultado das duas tabelas terão que ser exatamente iguais, e nenhuma linha será destacada.



num_dia	num_...	num_...	data_...	nom_dia	nom_...	ano_...	ano_...	sk_data
25	4	1902	1902-0...	Quinta...	Abril	19020...	190204	422
25	4	1902	1902-0...	Sábado	Abril	19020...	190204	423
25	4	1902	1902-0...	Segun...	Abril	19020...	190204	424
25	4	1902	1902-0...	Quarta...	Abril	19020...	190204	425
24	5	1902	1902-0...	Sexta-...	Maio	19020...	190205	426
24	5	1902	1902-0...	Domingo	Maio	19020...	190205	427
24	5	1902	1902-0...	Terça-f...	Maio	19020...	190205	428
24	5	1902	1902-0...	Quinta...	Maio	19020...	190205	429
24	5	1902	1902-0...	Sábado	Maio	19020...	190205	430
24	5	1902	1902-0...	Segun...	Maio	19020...	190205	431
24	5	1902	1902-0...	Quarta...	Maio	19020...	190205	432
24	5	1902	1902-0...	Sexta-...	Maio	19020...	190205	433
24	5	1902	1902-0...	Domingo	Maio	19020...	190205	434
24	5	1902	1902-0...	Terça-f...	Maio	19020...	190205	435
24	5	1902	1902-0...	Quinta...	Maio	19020...	190205	436
24	5	1902	1902-0...	Sábado	Maio	19020...	190205	437
24	5	1902	1902-0...	Segun...	Maio	19020...	190205	438
24	5	1902	1902-0...	Quarta...	Maio	19020...	190205	439
24	5	1902	1902-0...	Sexta-...	Maio	19020...	190205	440
24	6	1902	1902-0...	Domingo	Junho	19020...	190206	441
24	6	1902	1902-0...	Terça-f...	Junho	19020...	190206	442
24	6	1902	1902-0...	Quinta...	Junho	19020...	190206	443
24	6	1902	1902-0...	Sábado	Junho	19020...	190206	444
24	6	1902	1902-0...	Segun...	Junho	19020...	190206	445
24	6	1902	1902-0...	Quarta...	Junho	19020...	190206	446

num_dia	num_...	num_...	data_...	nom_dia	nom_...	ano_...	ano_...	sk_data
25	4	1902	1902-0...	Quinta...	Abril	19020...	190204	422
25	4	1902	1902-0...	Sábado	Abril	19020...	190204	423
25	4	1902	1902-0...	Segun...	Abril	19020...	190204	424
25	4	1902	1902-0...	Quarta...	Abril	19020...	190204	425
25	5	1902	1902-0...	Sexta-...	Maio	19020...	190205	426
25	5	1902	1902-0...	Domingo	Maio	19020...	190205	427
25	5	1902	1902-0...	Terça-f...	Maio	19020...	190205	428
25	5	1902	1902-0...	Quinta...	Maio	19020...	190205	429
25	5	1902	1902-0...	Sábado	Maio	19020...	190205	430
25	5	1902	1902-0...	Segun...	Maio	19020...	190205	431
25	5	1902	1902-0...	Quarta...	Maio	19020...	190205	432
25	5	1902	1902-0...	Sexta-...	Maio	19020...	190205	433
25	5	1902	1902-0...	Domingo	Maio	19020...	190205	434
25	5	1902	1902-0...	Terça-f...	Maio	19020...	190205	435
25	5	1902	1902-0...	Quinta...	Maio	19020...	190205	436
25	5	1902	1902-0...	Sábado	Maio	19020...	190205	437
25	5	1902	1902-0...	Segun...	Maio	19020...	190205	438
25	5	1902	1902-0...	Quarta...	Maio	19020...	190205	439
25	5	1902	1902-0...	Sexta-...	Maio	19020...	190205	440
25	6	1902	1902-0...	Domingo	Junho	19020...	190206	441
25	6	1902	1902-0...	Terça-f...	Junho	19020...	190206	442
25	6	1902	1902-0...	Quinta...	Junho	19020...	190206	443
25	6	1902	1902-0...	Sábado	Junho	19020...	190206	444
25	6	1902	1902-0...	Segun...	Junho	19020...	190206	445
25	6	1902	1902-0...	Quarta...	Junho	19020...	190206	446

Figura B.3 Comparação dos resultados das consultas SQL