UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

GUSTAVO BARANCELLI MOREIRA BANDEIRA

DWARF BATT: UMA FERRAMENTA DE TESTE BASEADO EM DEFEITOS PARA DATA WAREHOUSE

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA

2015

GUSTAVO BARANCELLI MOREIRA BANDEIRA

DWARF BATT: UMA FERRAMENTA DE TESTE BASEADO EM DEFEITOS PARA DATA WAREHOUSE

Trabalho realizado para a conclusão do Curso de Bacharelado em Sistemas de Informação do Departamento Acadêmico de Informática da Universidade Tecnológica Federal do Paraná.

Orientadora: Prof.ª Dr.ª Maria Cláudia Figueiredo Pereira Emer

CURITIBA

2015

Dedico este trabalho a todas as crianças adultas que assim como eu um dia sonharam em se tornar cientistas.

AGRADECIMENTOS

Agradecer é uma tarefa complicada, na qual em diversos momentos cometemos injustiças, vezes por esquecimento, vezes por não atribuir a devida atenção.

Agradeço aos meus pais, Adriane e Manoel pelo amor, educação, incentivo e suporte constante nos meus estudos.

Agradeço aos meus irmãos Fernando e Beatriz, que mesmo com a nossa grande diferença de idade, sempre estão por perto para descobrir e ensinar algo novo.

Agradeço aos meus colegas de trabalho Itelvina Oliveira e Rafael José, pela colaboração e companheirismo no desenvolvimento do trabalho.

Agradeço aos meus colegas e amigos, por todos os momentos vividos durante o curso e pela aprendizagem adquirida nesses anos.

Agradeço à Universidade Tecnológica Federal do Paraná pela oportunidade de formação e por toda a estrutura disponibilizada.

Agradeço aos professores do Dainf, por todos os ensinamentos, atividades e eventos realizadas durante a graduação.

Agradeço à Instituição A, pelo apoio financeiro e colaboração em propiciar a realização desta pesquisa fornecendo ajuda e infraestrutura para o desenvolvimento.

Por fim e mais importante, agradeço à professora Maria Claudia, minha orientadora, por todos os incentivos, ensinamentos e broncas que me levaram na direção correta para a realização deste trabalho.

"Até parece que foi ontem minha mocidade
Com diploma de sofrer de outra Universidade
Minha fala nordestina, quero esquecer o francês
E vou viver as coisas novas, que também são boas
O amor, humor das praças cheias de pessoas
Agora eu quero tudo, tudo outra vez
Belchior"

RESUMO

BANDEIRA, Gustavo. DWARF BATT: UMA FERRAMENTA DE TESTE BASEADO EM DEFEITOS PARA DATA WAREHOUSE. 62 f. Trabalho de Conclusão de Curso – Curso de Bacharelado em Sistemas de Informação, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

O processo de análise de dados é crucial para o processo de tomada de decisão em instituições. Nesse contexto, a estrutura de armazenamento de dados baseada em *Data Warehouses* possibilita uma visualização mais dinâmica do processo de negócio. No entanto, para a criação e validação destes bancos de dados são necessários testes. Este trabalho apresenta um estudo sobre o critério de teste análise de mutantes no âmbito de *Data Warehouses*. O objetivo do trabalho é o desenvolvimento de uma ferramenta de teste baseado em defeitos para *Data Warehouse*. A ferramenta foi desenvolvida na linguagem Java e a contribuição deste trabalho é o estudo na área de teste de software e, por consequência, a ferramenta de teste. Para validar a ferramenta, foi realizado um estudo de caso com dados reais, que apresentaram resultados preliminares promissores.

Palavras-chave: *Data Warehouse*, Processo ETL, Teste de Software, Testes baseados em Defeitos, SQL, Análise de Mutantes

ABSTRACT

BANDEIRA, Gustavo. DWARF BATT: A DATA WAREHOUSE FAULT BASED TESTING TOOL. 62 f. Trabalho de Conclusão de Curso – Curso de Bacharelado em Sistemas de Informação, Universidade Tecnológica Federal do Paraná. Curitiba, 2015.

The data analysis process is crucial to the decision-making process in institutions. In this context, a storage structure based on Data Warehouses enables a more dynamic view of the business process. However, to create and validate these databases, tests are needed. This paper presents a study of the mutation analysis technique in Data Warehouses. The tool was developed in the Java language and the contribution of this work is the software test study and, consequently, the testing tool. To validate the tool, a study case with real data was applied, resulting promising preliminary results.

Keywords: Data Warehouse, ETL process, Software testing, Fault-based Testing, SQL, Mutation Analysis

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE SIGLAS

BI	*Business Intelligence*
DW	*Data Warehouse*
PPGCA	Programa de Pós Graduação em Computação Aplicada
UTFPR	Universidade Tecnológica Federal do Paraná
SQL	*Structured Query Language*
ETL	*Extraction, Transformation and Load*
SEL	*Select Clause*
JOI	*Join Clause*
AGR	*Aggregate Function*
tOpCp	Troca de Operador de Comparação
ROR	*Relational Operator Replacement*
tOpMT	Troca de Operador Matemático
AOR	*Arithmetic Operator Replacement*
tOpCj	Troca de Operador Lógico
LCR	*Logic Connector Replacement*
iNot	Inserção de Operador de Negação
rNot	Remoção de Operador de Negação
UOI	*Unary Operator Insertion*
ABS	*Absolute Value Insertion*
BTW	*Between Predicate*
LKE	*Like Predicate*
tPoAt	Troca de Posição de Atributo
rAtr	Retirada de Atributo
iAtr	Inserção de Atributo
tAtr	Troca de Atributo
tPoVr	Troca de Posição de Valor
tVr	Troca de Valor
tNmTb	Troca de Nome de Tabela
tFuAg	Troca de Função de Agregação
API	*Applications Programming Interface*
BDT	Base de Dados de Teste

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTO

O gerenciamento de informações vem há vários anos sendo uma ferramenta para as organizações no desenvolvimento de seus processos de negócio. A Inteligência Empresarial (BI -*Business Intelligence*), permite, a partir de dados e informações importantes da empresa, a obtenção de uma vantagem estratégica sobre seus concorrentes, aprimorando o processo de tomada de decisão (MUNAWAR et al., 2011).

Sistemas de BI, para funcionar de maneira correta, necessitam de acesso a uma vasta quantidade de dados, os quais normalmente não estão presentes em bancos de dados convencionais. Para isso, empresas começaram a utilizar *Data Warehouses* (KIMBALL; ROSS, 2013).

A criação de um DW (*Data Warehouse*) é um processo complexo de modelagem e organização de dados. A partir disso, surge a necessidade da execução de testes nesses ambientes, visto que erros no procedimento podem comprometer o processo de análise e decisão, gerando gastos à organização.

Segundo Sommerville (2003), o teste de software é uma atividade essencial para aumentar o nível de confiança no momento de utilização de um software. As técnicas utilizadas para a execução de testes são baseadas em diferentes princípios e análises, modeladas por meio de perspectivas distintas.

Neste trabalho a técnica de teste baseado em defeitos é investigada. Conforme a teoria de DeMillo et al. (1978), o bom programador tende a construir um sistema logicamente correto. No entanto, pequenos erros no momento de escrita do código podem ser cometidos, mesmo muitas vezes não comprometendo completamente a execução do software. Defeitos deste tipo podem ser encontrados utilizando o critério de teste análise de mutantes.

O critério de teste análise de mutantes se constitui da criação de códigos chamados

mutantes, sendo estes quase idênticos ao código original, apenas com alguma pequena alteração pontual, com o objetivo de comparar os resultados gerados por ambos.

A utilização do teste de mutação em consultas SQL, aplicado em um ambiente de *Data Warehouse* contribui para a detecção de defeitos, auxiliando na descoberta de problemas na criação desse tipo de banco de dados, garantindo a integridade das informações extraídas e utilizadas nas análises estratégicas organizacionais.

1.2 JUSTIFICATIVA E MOTIVAÇÃO

A criação de *Data Warehouses* e *Data Marts* é feita por muitas empresas com o uso de sistemas de BI, com o objetivo de aprimorar a recuperação de dados para análise e tomada de decisão. Assim, a realização de testes, feita sobre o processo de construção dos DW, visa manter a integridade e a confiança no armazenamento dos dados, assim como aperfeiçoar o acesso e a seleção dos mesmos (MEKTEROVIC et al., 2011).

Processos como este são importantes para manter a estabilidade empresarial, a saúde financeira e melhorar o aproveitamento de recursos, tornando o processo de negócio mais eficiente.

A partir disso, a Instituição A[1] apresentou a necessidade de implantação do processo de BI em sua instituição, visando aprimorar o processo de decisão estratégica.

Este trabalho foi realizado em parceria com o PPGCA - Programa de Pós Graduação em Computação Aplicada - da UTFPR e com fomento da Instituição A. Ainda, este trabalho de conclusão de curso foi desenvolvido em conjunto com o trabalho de mestrado de Oliveira (2015).

1.3 OBJETIVOS

Nesta Seção são apresentados os objetivos deste trabalho.

1.3.1 OBJETIVO GERAL

O objetivo geral deste trabalho é o desenvolvimento de uma ferramenta para a realização de testes em um ambiente de *Data Warehouse*.

[1] A Instituição A é um órgão público, tendo seu nome oculto por questões de sigilo e segurança

1.3.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos são:

- Investigar o teste baseado em defeitos para *Data Warehouse*;
- Desenvolver uma ferramenta de teste baseada em defeitos para aplicação em um ambiente de *Data Warehouse*;
- Validar a ferramenta em um estudo de caso.

1.4 METODOLOGIA

A metodologia utilizada para a realização deste trabalho foi baseada nos seguintes itens:

1. Estudo teórico sobre conceitos de *Data Warehouse*, teste de software e geração de casos de teste baseados na análise de mutantes para consultas SQL. Foram apresentados os principais conceitos utilizados no trabalho, envolvendo as áreas de *Data Warehouse*, processo ETL, teste de software e análise de mutantes. Ainda, foram discutidos os principais trabalhos utilizados como base para o desenvolvimento deste.
2. Análise e projeto da ferramenta: através do estabelecimento de requisitos e casos de uso, foi planejado o processo de desenvolvimento da ferramenta. Toda a estrutura necessária para o desenvolvimento, tanto em questões de hardware quanto de software foi fornecida pela UTFPR e pela Instituição A.
3. Estudo de caso: com o objetivo de validar a ferramenta, um estudo de caso foi preparado com base em dados reais fornecidos pela Instituição A. O estudo de caso buscou replicar um processo real de utilização da ferramenta, de forma prática e eficiente.
4. Elaboração do Manual do Desenvolvedor: visto que a ferramenta será utilizada pela Instituição A, o manual foi elaborado para posteriores melhorias ou alterações na ferramenta, com o objetivo de facilitar o entendimento do funcionamento e da estrutura da ferramenta.

1.5 ORGANIZAÇÃO DO TRABALHO

O trabalho foi dividido da seguinte maneira: o Capítulo 2 apresenta conceitos e definições sobre as áreas de *Data Warehouse* e teste de software, assim como trabalhos relacionados quem ajudaram no desenvolvimento deste; O Capítulo 3 descreve a ferramenta desenvolvida, desde o processo de análise e projeto de sistema até uma explicação da estrutura da mesma; O Capítulo 4 descreve o estudo de caso aplicado no trabalho e os resultados obtidos com ele; Finalmente, o Capítulo 5 apresenta as considerações finais. Este trabalho contém um apêndice (Apêndice A), o Manual do Desenvolvedor, contendo diagramas e explicações detalhadas sobre a ferramenta desenvolvida.

2 REVISÃO TEÓRICA E ESTADO DA ARTE

Neste capítulo são descritos os conceitos fundamentais para a realização da pesquisa e desenvolvimento deste trabalho. A problemática é desenvolvida e são apresentados trabalhos relacionados que já foram desenvolvidos.

2.1 *DATA WAREHOUSE*

É notório o desenvolvimento e disseminação da área de *Data Warehouse* desde a criação de seu conceito, na década de 90. Segundo Kimball e Ross (2013), no ano de 2001, algumas empresas chegaram a marca de 50% de seus investimentos voltados para a área de *Business Intelligence* e *Data Warehouse*.

Inmon (2012) define *Data Warehouse* como um conjunto de dados integrados e orientados por assunto, armazenados de forma não volátil. Isso significa que primariamente há uma separação de dados em classes específicas, as quais guardam uma identidade e valores característicos de certo assunto. Por causa disso, é comum a utilização de *Data Marts*[1] para uma melhor organização e administração.

Laudon e Laudon (2013), com uma visão mais administrativa, comentam que *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.

Data Warehouses ainda têm como característica a temporalidade dos dados, apresentando uma linearidade nos dados armazenados. Por causa disso, raramente são feitas remoções ou atualizações de dados (WU; BUCHMANN, 1997).

Como funcionalidades para um *Data Warehouse*, Kimball e Ross (2013) destacam os seguintes pontos:

1. O *Data Warehouse* deve fornecer acesso aos dados corporativos organizacionais;

[1] *Data Marts* são subdivisões de um banco de dados, normalmente separadas por assunto.

2. Os dados do *Data Warehouse* devem ser consistentes;

3. Os dados do *Data Warehouse* podem ser separados e combinados usando-se qualquer medição possível do negócio (*slice and dice*);

4. O *Data Warehouse* não consiste apenas em dados, mas também em um conjunto de ferramentas para consultar, analisar e apresentar informações;

5. O *Data Warehouse* é o local em que são publicados dados confiáveis;

Além da perspectiva temporal, existem outras perspectivas diferentes para a análise de dados, conhecidas como dimensões. Com isso, a estrutura da base de dados de uma *Warehouse* é fundamentada no modelo Dimensional, diferentemente do tradicional modelo Relacional. Este modelo consiste na separação em estruturas de dois tipos: os Fatos e as Dimensões (KIMBALL; ROSS, 2013).

Fatos normalmente são compostos por elementos de tabelas normalizadas, as quais contém os dados relevantes e particulares de cada caso, e que estão presentes como os atributos principais nas bases de dados. Já as dimensões consistem de parâmetros para a seleção destes dados, filtrando os relacionados em específico de acordo com a relação que possui com a dimensão (WU; BUCHMANN, 1997). Um exemplo de modelagem relacional e sua respectiva versão dimensional é apresentado nas Figuras 1 e 2.

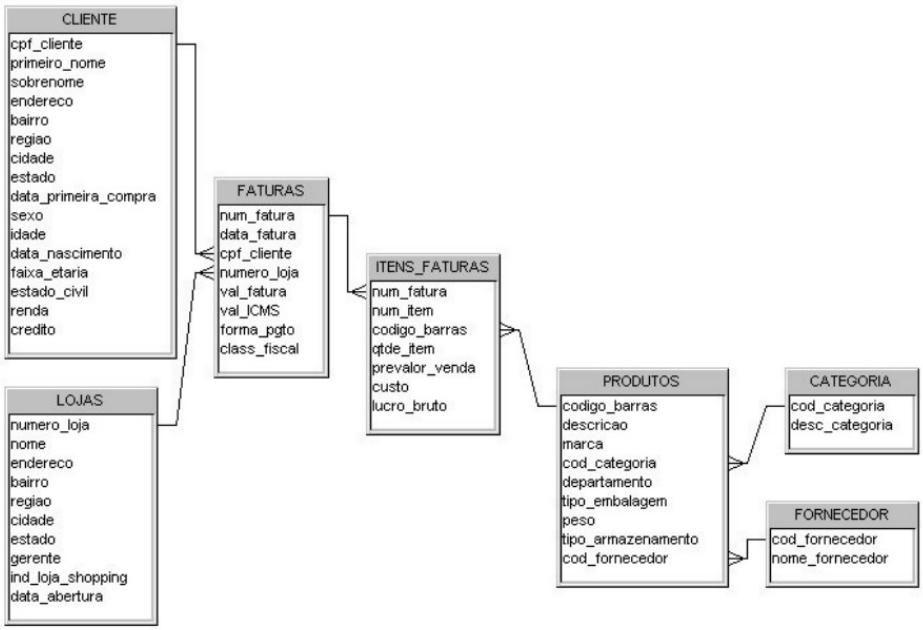


Figura 1: Exemplo de Banco Relacional.
Fonte: Hokama et al. (2004)

O modelo relacional ilustrado na Figura 1, possui uma tabela Faturas que contém todos os dados relacionados a uma fatura, com seus itens vendidos sendo armazenados na tabela Itens_Fatura. Para gerenciar as vendas, possui também um cadastro de Lojas, de Clientes, de Funcionários, de Produtos, de Categorias e de Fornecedores. Cada venda ocorrida representa uma linha na tabela Faturas e n linhas na tabela Itens_Fatura, sendo que n é o número de itens diferentes vendidos nessa transação.

Modelos relacionais tendem a trabalhar com normalização, reduzindo ao máximo o número de registros e eliminando redundâncias desnecessárias. Isso facilita na administração de recursos pontuais, sem o uso de muitos registros. Como o DW tende a se utilizar de muitos registros para a realização de análises, o modelo relacional pode comprometer o desempenho do sistema (HOKAMA et al., 2004).

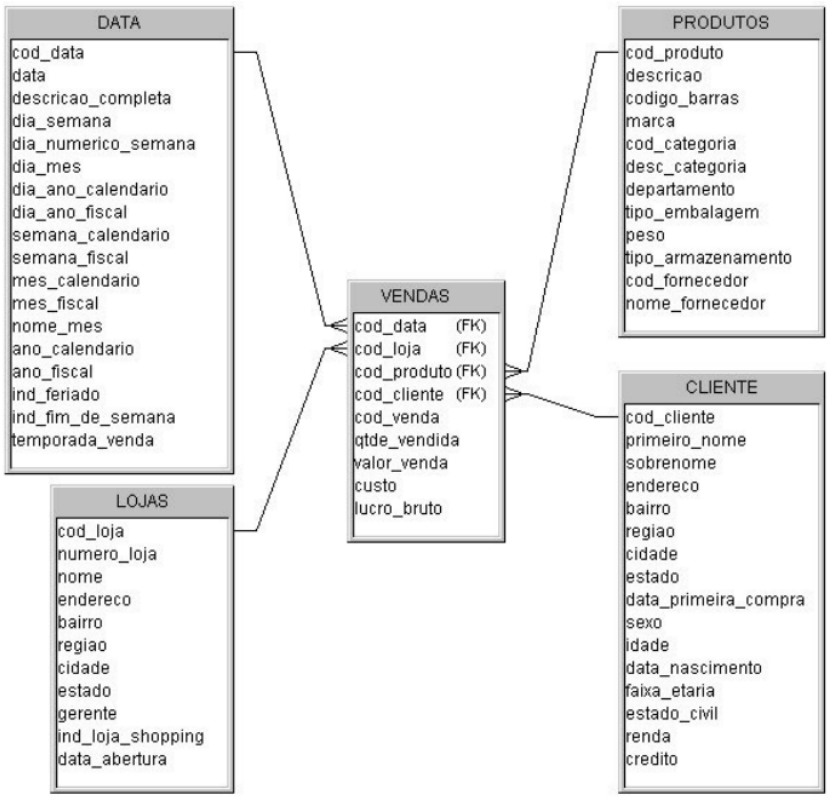


Figura 2: Exemplo de Banco Dimensional.
Fonte: Hokama et al. (2004)

Ilustrado na Figura 2 está o modelo dimensional da versão apresentada na Figura 1. Esse modelo possui uma tabela central chamada Vendas que armazena os dados principais a serem analisados da venda, como a quantidade e o lucro. Essa tabela tem relacionamento com as demais tabelas necessárias para identificar um item de venda com base no cliente, data, loja e produto.

A tabela Cliente possui os dados necessários para identificação do comprador, e dados importantes para filtrá-los, como idade, faixa etária, e renda. As tabelas Produto, Lojas e Data seguem o mesmo modelo, contendo dados de identificação atribuídos a determinada venda.

Neste esquema, a tabela Vendas é determinada como tabela fato, sendo as outras todas dimensões. Percebe-se que, para uma seleção ampla e detalhada de dados, o modelo dimensional acaba por se utilizar de menos junções de tabelas, melhorando a performance do sistema.

Para a transformação de uma base de dados relacional, para uma base de dimensional, é utilizado um método denominado ETL - como apresentado na Figura 3 - o qual será explicado na Seção seguinte.

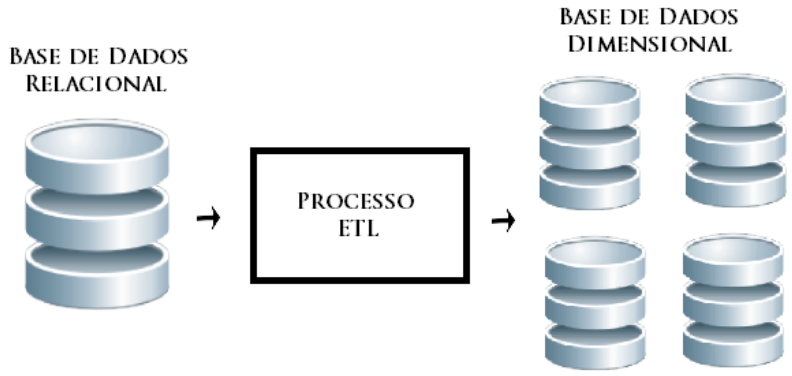


Figura 3: Transformação de modelos de Bancos de Dados.
Fonte: Autor

2.1.1 CRIAÇÃO DO *DATA WAREHOUSE*

O processo mais completo e demorado na construção de um *Data Warehouse* é o processo ETL (*Extraction, Transformation and Load*). Inmon (2012) afirma que apenas o processo ETL pode consumir até 80% do tempo de desenvolvimento do *DW*.

O processo basicamente é constituído por consultas SQL, que visam remodelar a estrutura dos dados e gerar o modelo dimensional. Por exemplo, a parte de Extração possui predominantemente consultas de seleção, enquanto que a carga possui predominantemente consultas de inserção.

A Figura 4 mostra o processo simplificado do ETL, 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 DW, possibilitando posteriormente a geração de *Data Marts*.

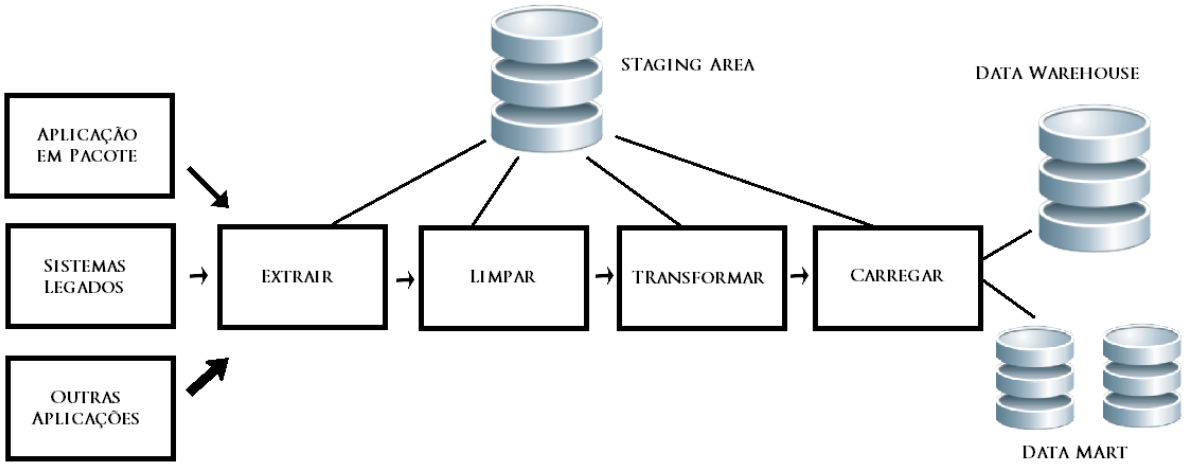


Figura 4: Processo ETL.
Fonte: Baseado em (TURBAN et al., 2010)

Kimball e Ross (2013) definem como principais requisitos para o processo ETL:

- 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;
- Verificar 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;

- Definir as entregas nas interfaces de usuário dos sistemas de BI.

Já Elgamal e Elbastawissy (2013) listam alguns desafios para testar sistemas DW:

- DW possuem consultas *ad-hoc*, então é quase impossível realizar testes antes do sistema DW ser desenvolvido, porque não é possível saber quais as consultas possíveis de 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, procurando armazenar informações detalhadas e de maneira cronológica;
- 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 testes são limitados, enquanto no DW, os casos de testes 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 TESTE DE SOFTWARE

A atividade de teste de software consiste de uma análise dinâmica do produto, sendo relevante para a identificação e eliminação de erros persistentes. Ainda, o conjunto das atividades de teste é significativo para as atividades de depuração, manutenção e estimativa de confiabilidade dos software (DELAMARO et al., 2007)

O processo de teste consiste na criação de um conjunto de casos de testes, baseado em algum critério de teste, com o objetivo de certificar que certa parte do programa não apresenta defeitos. O conjunto é composto, a princípio, apenas por dados de entrada,

seguido de uma análise feita com base no comportamento do sistema e nas informações obtidas pela saída do programa (MYERS, 2004). Pressman (2006) afirma que quanto melhores forem os resultados, maior será a confiança e a qualidade do sistema.

Segundo Sommerville (2003), as principais técnicas de teste de software são separadas em *Teste Caixa Preta* e *Teste Caixa Branca*. No entanto, *Teste Baseado em Defeitos* é uma técnica diferenciada, contendo elementos das duas técnicas anteriores.

Black Box Testing ou Teste Caixa Preta é uma técnica de teste funcional, a qual utiliza-se casos de teste[2] baseados na especificação do sistema. A criação de dados de entrada é feita a partir de classes, na qual há um conhecimento prévio sobre qual deverá ser a saída retornada pela programa, podendo ser considerados invariavelmente corretos ou incorretos. O nome provém da decisão de ignorar a estrutura interna do sistema, analisando apenas entradas e suas respectivas saídas (PRESSMAN, 2006).

White Box Testing ou Teste Caixa Branca é uma técnica de teste estrutural, na qual utiliza-se da estrutura interna lógica do programa para a criação dos casos de teste. Existem vários critérios existentes para sua utilização - entre eles, a *Complexidade Ciclomática* e o *Fluxo de Controle e de Dados* - cada um com diferentes propostas e objetivos (PRESSMAN, 2006).

A técnica de testes baseada em defeitos utiliza informações sobre os enganos mais frequentes cometidos no processo de desenvolvimento de software para realizar o teste (BARBOSA et al., 2001). O critério de Análise de Mutantes é um dos critérios da técnica baseada em defeitos e este é o foco deste trabalho, portanto será detalhado na próxima Seção.

2.3 ANÁLISE DE MUTANTES

Proposto por DeMillo et al. (1978), teste baseado em defeitos pelo critério análise de mutantes é fundamentado por dois princípios: *Hipótese do Programador Competente* e *Efeito de Acoplamento*.

O primeiro princípio consiste da premissa de que desenvolvedores tendem a escrever códigos próximos ao correto. Muitas vezes os códigos apresentam defeitos que alteram de forma drástica a saída do sistema, sendo estes encontrados em testes de caixa branca

[2]Casos de teste são instancias de utilização do software - como por exemplo entradas no sistema - que visam contemplar todas as possibilidades de utilização do sistema, com o objetivo de observar o comportamento do software.

ou caixa preta. No entanto, testes baseados em defeitos são capazes de encontrar defeitos mais complexos, nos quais se apresentam algoritmos sintaticamente corretos, mas semanticamente errados. Isso quer dizer que os códigos apresentam sua composição correta e "funcionam", mas não da forma como deveriam, porque sua lógica está comprometida.

A partir do descobrimento destes pequenos defeitos, conceitualmente considerados simples, podem ser encontrados defeitos mais complexos e de maior porte, constituindo o segundo princípio.

Com isso, DeMillo et al. (1978) propôs a geração de programas com pequenas alterações no programa de teste, resultando em cópias ligeiramente diferentes do original, denominadas mutantes - veja Figura 5, na qual o exemplo a esquerda contém a codificação original enquanto a direita possui a codificação com a mutação. Estes mutantes então são executados e seu resultado é comparado ao resultado do programa original.

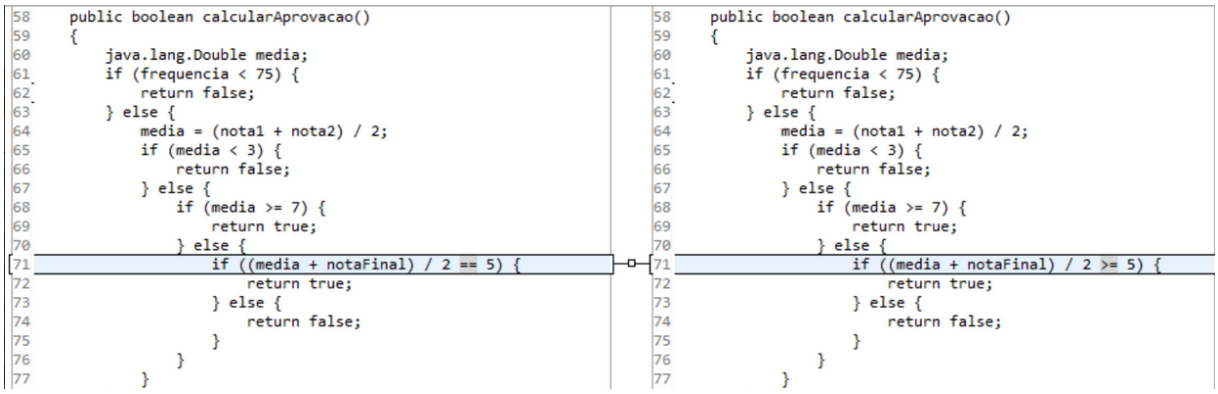


Figura 5: Exemplo de Programa Mutante.
Fonte: Autor

Caso o resultado de um mutante m seja diferente ao resultado do programa original p, temos que m é considerado "morto". No entanto, caso m seja igual, ele é considerado "vivo" e deve passar por outros testes, envolvendo outros casos de teste. Se mesmo assim m continuar obtendo resultados para "vivo", o mutante deve passar por uma análise humana, com chances de considerá-lo um mutante "equivalente", ou seja, um programa que tem o mesmo comportamento do original.

A quantificação de um teste baseado em mutantes é representada pelo escore de mutação - *Mutation Score* - um valor entre zero e um. Quanto mais próximo de um, significa que o conjunto de casos de teste é adequado para revelar os defeitos representados pelos programas mutantes.

$$EscMut(P,T) = \frac{K}{M-E} \tag{1}$$

Onde:

P: programa em teste;

T: conjunto de teste;

K: quantidade de mutantes mortos;

M: total de mutantes gerados;

E: total de mutantes equivalentes.

2.4 TRABALHOS RELACIONADOS

Nesta Seção são apresentados assuntos correlatos ao trabalho, assim como são feitas citações e exemplificações de trabalhos relacionados.

2.4.1 MUTAÇÕES EM CONSULTAS SQL

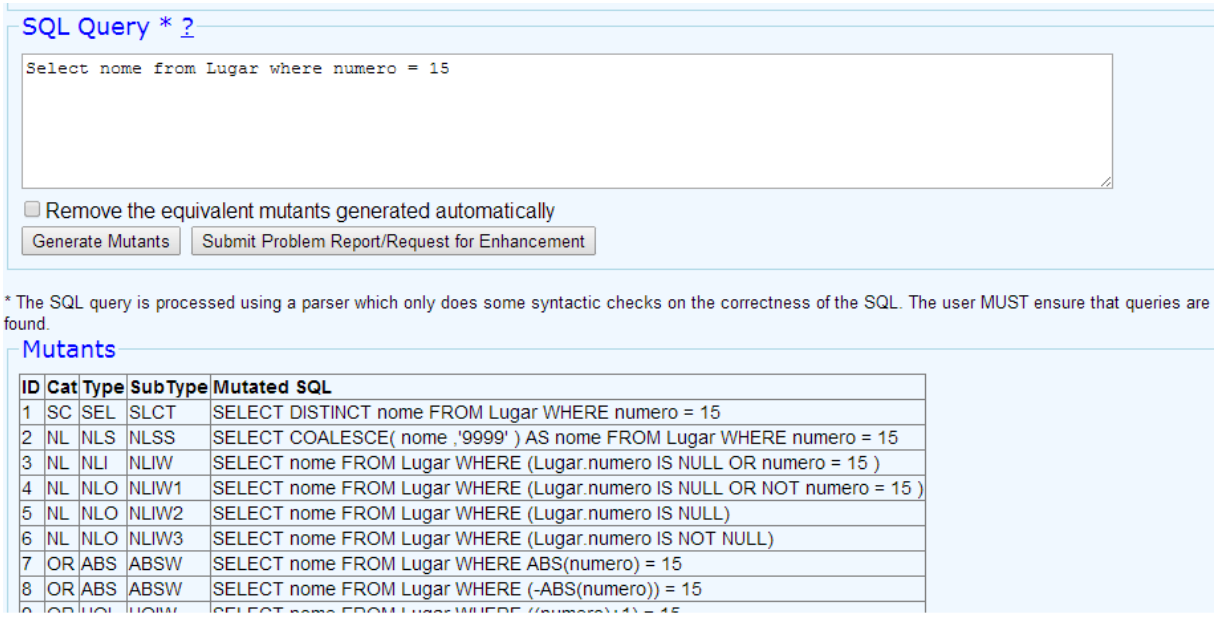
O uso de mutações para testes de consultas SQL é estudado em Tuya et al. (2006). O estudo apresenta diversos tipos de mutação para a aplicação sobre consultas em bancos de dados.

Neste estudo há o foco em consultas de seleção, com o operador SQL Select, o qual abrange diversas categorias e operadores cabíveis de mutação, como por exemplo para as cláusulas *Where* e *Join*. Essas categorias e suas devidas mutações serão utilizadas como base ao desenvolvimento e serão explicadas nas próximas Seções.

O estudo de Tuya et al. (2006) também traz consigo uma ferramenta de análise de mutantes própria, desenvolvida na linguagem de programação Java. O programa trabalha com base em uma consulta SQL de entrada, baseada em um banco de dados, que também deve ser fornecido. Um exemplo da versão *Web* da ferramenta é apresentado na Figura 6. Outros detalhes da ferramenta serão explicados na Seções seguintes.

Já Cabeça (2009) fez um estudo mais abrangente sobre mutações em consultas SQL. Não se restringindo apenas a consultas de seleção, também foram sugeridas consultas de manipulação de dados. No trabalho são descritos os operadores mutantes utilizados, baseados em consultas de inserção, atualização e remoção de dados - *Insert*, *Update* e *Delete*, respectivamente, na linguagem SQL 3.

Ainda, Cabeça (2009) descreve a ferramenta que desenvolveu e utilizou em seus estudos, mostrando exemplos de seu comportamento e os resultados que foram obtidos



ID	Cat	Type	SubType	Mutated SQL
1	SC	SEL	SLCT	SELECT DISTINCT nome FROM Lugar WHERE numero = 15
2	NL	NLS	NLSS	SELECT COALESCE(nome ,'9999') AS nome FROM Lugar WHERE numero = 15
3	NL	NLI	NLIW	SELECT nome FROM Lugar WHERE (Lugar.numero IS NULL OR numero = 15)
4	NL	NLO	NLIW1	SELECT nome FROM Lugar WHERE (Lugar.numero IS NULL OR NOT numero = 15)
5	NL	NLO	NLIW2	SELECT nome FROM Lugar WHERE (Lugar.numero IS NULL)
6	NL	NLO	NLIW3	SELECT nome FROM Lugar WHERE (Lugar.numero IS NOT NULL)
7	OR	ABS	ABSW	SELECT nome FROM Lugar WHERE ABS(numero) = 15
8	OR	ABS	ABSW	SELECT nome FROM Lugar WHERE (-ABS(numero)) = 15

Figura 6: Versão Web do Sql Mutation.
Fonte: (TUYA et al., 2006)

quando aplicada em uma base própria.

Vale ressaltar também que, em ambos os casos descritos, as consultas foram feitas para bases de dados convencionais, utilizando o modelo relacional de dados.

2.4.2 OPERADORES DE MUTAÇÃO EM SQL

Nesta Seção são apresentados os operadores de mutação de Tuya et al. (2006) e Cabeça (2009).

2.4.2.1 OPERADORES DE MUTAÇÃO PARA CLÁUSULAS SQL

Definido em Tuya et al. (2006), operadores de mutação para cláusulas SQL visam demonstrar defeitos na essência de cada uma das principais cláusulas SQL.

- ***SEL - Select Clause***: Cada ocorrência de uma das seguintes palavras chave *Select* ou *Select Distinct* é trocada pela outra.
- ***JOI - Join Clause***: Cada ocorrência de uma das seguintes palavras chave do tipo *Join*, *Inner Join*, *Left Outer Join*, *Right Outer Join*, *Full Outer Join* e *Cross Join* é trocada por todas as outras. Quando há a troca de uma palavra chave por *Cross Join*, uma cláusula *On* é adicionada e sua correspondente condição é criada a partir das *Primary Keys* das tabelas dentro do *Join*.

- ***AGR - Aggregate Function***: Cada ocorrência de uma das seguintes palavras chave de funções de agregação *Max*, *Min*, *Avg*, *Sum*, *Count* é trocada por todas as outras. A troca das palavras chave deve levar em consideração os tipos de dados nos argumentos da função. Caso o tipo de dado seja caractere, as funções *Sum* e *Avg* não devem ser substituídas. Cabeça (2009) descreve esse operador como *Troca de Função de Agregação*, mas o classifica como Operador de Miscelânea.

2.4.2.2 OPERADORES DE MUTAÇÃO PARA OPERADORES DE SQL

Definidos em Cabeça (2009), operadores de mutação para operadores de SQL visam representar os defeitos criados pelo programador ao usar incorretamente operadores SQL.

Operadores SQL normalmente são aplicados em cláusulas *where*, mas a criação de casos mutantes pode ser realizada em qualquer parte do comando em que seja possível a realização da alteração.

- ***tOpCp - Troca de Operador de Comparação***: Cada ocorrência de um dos operadores de comparação (=,<=,>=,<,>,<>) é trocado por cada um dos outros operadores. Esse operador também é definido pelo nome ROR - *Relational Operator Replacement*, em Tuya et al. (2006);

- ***tOpMT - Troca de Operador Matemático***: Cada ocorrência de um dos operadores matemáticos (+,-,*,/,%) é trocada por cada um dos outros operadores. Esse operador também é definido pelo nome AOR - *Arithmetic Operator Replacement*, em Tuya et al. (2006);

- ***tOpCj - Troca de Operador Lógico***: Cada ocorrência de um dos operadores lógicos é trocada por cada um dos outros operadores do qual seja possível a alteração. Esse operador também é definido pelo nome LCR - *Logic Connector Replacement*, em Tuya et al. (2006), no qual admite que operadores lógicos e operadores conjuntivos apresentam as mesmas características para a permutação;

- ***iNot - Inserção de Operador de Negação***: Insere o operador de negação *NOT* em conjunto a outros operadores que permitem sua utilização (*IS NULL, BETWEEN, IN, LIKE, EXISTS, UNIQUE, ALL, ANY e EQUAL*);

- ***rNot - Remoção de Operador de Negação***: remove o operador de negação *NOT* em conjunto a outros operadores que permitem sua remoção;

Tuya et al. (2006) também define operadores de mutação para operadores de SQL. Os operadores já apresentados não serão listados.

- ***UOI - Unary Operator Insertion***: Cada ocorrência de uma expressão aritmética a um número e é trocada por -e, e+1 e e-1. Referências a números não devem ser alteradas dentro de cláusulas *ORDER BY*, *GROUP BY* ou *EXISTS*;
- ***ABS - Absolute Value Insertion*** Cada ocorrência de uma expressão aritmética a um número e é trocada por -ABS(e) e ABS(e). Referências a números não devem ser alteradas dentro de cláusulas ORDER BY, GROUP BY ou EXISTS;
- ***BTW - Between Predicate***: Cada ocorrência de uma expressão na forma a BETWEEN x AND y é trocada por $a > x$ AND $a <= y$ e $a => x$ AND $a < y$. Não deverão ser criados mutantes no caso de NOT BETWEEN;
- ***LKE - Like Predicate***: Cada ocorrência de uma expressão LIKE possui infinitas possibilidades de mutações. No entanto são restritas ao comportamento dos *wildcards* {%,_} (o símbolo de porcentagem se refere a qualquer *string*, enquanto que o *underscore* se refere a um único caractere). Cada ocorrência de um *wildcard* gera: (1) a troca do *wildcard* pelo outro, (2) a remoção do *wildcard*, (3) a remoção do primeiro caracter anterior ao *wildcard* (caso este não esteja no início da expressão) e (4) remoção do primeiro caractere posterior ao *wildcard*. Caso não exista nenhum *wildcard* no início da expressão então adicionar um *wildcard*, repetindo o processo para o final da expressão.

2.4.2.3 OPERADORES DE MUTAÇÃO DE MISCELÂNEA

Segundo Cabeça (2009) operadores de miscelânea podem ser utilizados em diversas cláusulas e com diversas funcionalidades. Por exemplo, operadores de miscelânea são os mais utilizados em consultas do tipo *insert* e *delete*.

- ***tPoAt - Troca de Posição de Atributo***: A cada ocorrência de um atributo, em uma lista ordenada de atributos, há a permutação por outro da mesma lista.
- ***rAtr - Retirada de Atributo***: A cada ocorrência de um atributo, em uma lista ordenada de atributos, há a retirada do atributo.
- ***iAtr - Inserção de Atributo***: Em uma lista ordenada de atributos, há a inserção de um atributo por outro que seja compatível.

- ***tAtr - Troca de Atributo***: Em uma lista ordenada de atributos, há a troca por outro atributo, que seja compatível.

- ***tPoVr - Troca de Posição de Valor***: A cada ocorrência de um valor, em uma lista ordenada de valores, há a permutação por outro da mesma lista.

- ***tVr - Troca de Valor***: A cada ocorrência de um valor, em uma lista ordenada de valores, há a troca por outro valor compatível (sendo numérico ou alfanumérico) e por NULL.

- ***tNmTb - Troca de Nome de Tabela***: A cada ocorrência de uma tabela, há a troca pelo nome de outra tabela presente no banco de dados.

- ***tFuAg - Troca de Função de Agregação***: A cada ocorrência de uma função de agregação (*SUM, AVG, COUNT, MAX, MIN, FIRST* e *LAST*) há a permutação por outra.

3 A FERRAMENTA DWARF BATT

A proposta deste trabalho é estender o uso da técnica baseada em defeitos para o contexto de *Data Warehouse*, com o foco no processo ETL. Para a aplicação desta técnica por meio da análise de mutantes, se faz necessário o desenvolvimento de uma ferramenta de teste.

Neste capítulo são apresentados o funcionamento do processo de teste e as técnicas utilizadas para o desenvolvimento da ferramenta.

3.1 DESCRIÇÃO DO PROBLEMA

A problemática deste trabalho está envolvida no desenvolvimento de uma ferramenta, a qual deve ser capaz de executar a técnica de teste baseado em defeitos, seguindo o critério de análise de mutantes em ambientes de *Data Warehouse*.

Logo, o sistema tem como objetivo auxiliar o programador a encontrar defeitos em consultas SQL aplicadas a bancos de dados *Data Warehouse*, facilitando o processo de análise, reconhecimento e tratamento de erros.

Esse processo é feito por meio de uma entrada primária, para conexão com o banco de dados, e entradas esporádicas, das consultas a serem testadas. Ambas as entradas devem ser feitas pelo usuário. Com o resultado, o usuário tem condições de analisar e ponderar as saídas, constituídas por listas de consultas e cálculos gerados pelo sistema.

Os operadores implementados na ferramenta estão listados na Tabela 1. A descrição mais detalhada foi feita na Seção 2.4.2 deste trabalho.

Tabela 1: Operadores utilizados na ferramenta

Categoria	**Sigla**	**Descrição**
Cláusulas SQL	SEL	Select Clause
	JOI	Join Clause
	AGR	Aggregate Function
Operadores de SQL	tOpCp	Troca de Operador de Comparação
	tOpMt	Troca de Operador Matemático
	tOpCj	Troca de Operador Lógico
	iNot	Inserção de Operador de Negação
	rNot	Remoção de Operador de Negação
	UOI	Unary Operator Insertion
	ABS	Absolute Value Insertion
	BTW	Between Predicate
	LKE	Like Predicate
Miscelânea	tPoAt	Troca de Posição de Atributo
	rAtr	Retirada de Atributo
	iAtr	Inserção de Atributo
	tAtr	Troca de Atributo
	tPoVr	Troca de Posição de Valor
	tVr	Troca de Valor
	tNmTb	Troca de Nome de Tabela
	tFuAg	Troca de Função de Agregação

3.2 DESCRIÇÃO DA FERRAMENTA

A *DWarF BaTT* (*Data Warehouse Fault-Based Testing Tool*) possibilita o teste de consultas do tipo Select, Insert, Delete e Update, de acordo com o padrão SQL 3, inseridas em bancos de dados do Microsoft SQL Server.

A ferramenta foi implementada na linguagem Java com o uso da API (*Applications Programming Interface*) SQLRules (TUYA et al., 2006) usada para detectar e mapear o esquema de banco de dados conectado e do SQLJDBC4 (*SQL Java Database Connectivity*) usado para manipular e consultar informações em bases de dados do Microsoft SQL Server.

3.3 ARQUITETURA DA FERRAMENTA

A arquitetura da ferramenta e a comunicação entre os seus principais módulos são descritos e apresentados na Figura 7:

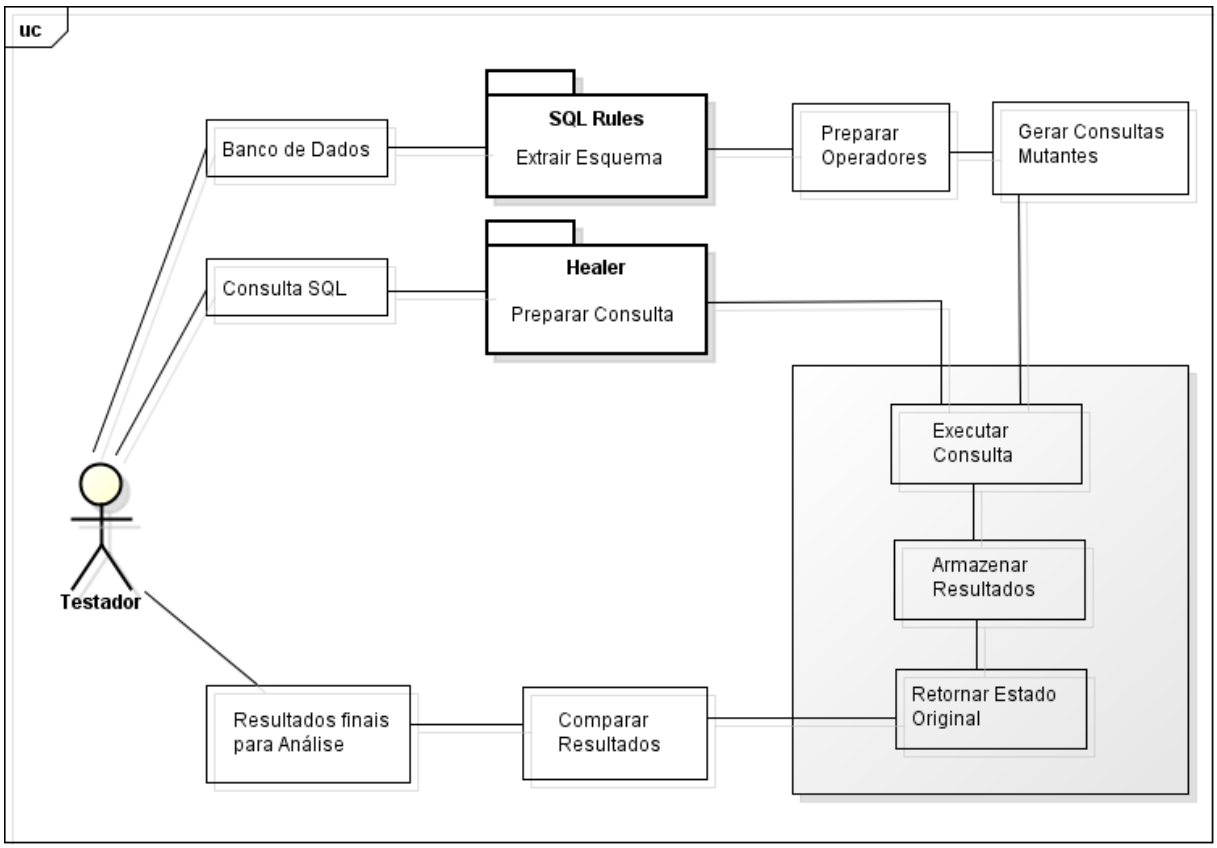


Figura 7: Arquitetura da Ferramenta.
Fonte: Autor

- Banco de Dados: Primeira entrada, com os dados referentes a conexão ao banco de dados e os dados do logon que será utilizado.
- Consulta SQL: Segunda entrada, com a consulta SQL a qual será testada.
- SQL Rules: Extrai informações sobre o esquema do banco de dados conectado, como nomes de tabelas e seus campos, suas interações, quantidade de elementos e tipos de elementos.
- *Healer*: Prepara a consulta para a aplicação dos operadores, através da listagem e detecção de suas estruturas.
- Preparar Operadores: Estabelece os operadores que devem ser utilizados de acordo com o tipo da consulta e das cláusulas presentes nas consultas.
- Gerar Consultas Mutantes: Cria as consultas mutantes com base na consulta original.
- Executar Consultas: No banco de dados, a consulta original e as consultas mutantes são executadas.

- Armazenar Resultados: Armazena temporariamente os resultados obtidos através da execução das consultas.
- Retornar Estado Original: Realiza o retorno do banco de dados ao estado inicial, para que outra consulta possa ser executada de maneira correta.
- Comparar Resultados: Compara o resultado de cada consulta mutante com o da consulta original, gerando assim seu estado.
- Resultados para Análise: Dispõe ao usuário os resultados obtidos através do processo, permitindo-o filtrá-las da maneira que desejar.

3.4 REQUISITOS FUNCIONAIS E NÃO-FUNCIONAIS

Larman (2005) propõe que os requisitos devem ser organizados a partir de suas funcionalidades, ou seja, são primeiramente listados os requisitos funcionais e, em sequencia, atribuindo a cada um seus próprios requisitos especiais - requisitos que atendem as necessidades não-funcionais, como por exemplo interfaces e interações.

O padrão para a apresentação dos requisitos foi baseado em Larman (2005), sendo ligeiramente modificado para adequação a proposta.

1. **Conectar Banco de Dados**	
Conectar ao banco de dados fornecido pelo usuário	
Requisitos Especiais:	
Nome	Descrição
1.1 Acesso ao Banco	Recebe como entrada a URL do banco de dados
1.2 Acesso ao Logon	Recebe como entrada o logon e sua senha
1.3 Mensagem de Erro	Abre uma janela de erro, em caso de falha na conexão

Quadro 3.1: Requisito 1

2. **Reconhecer Consulta**	
Reconhecer consulta fornecida pelo usuário	
Requisitos Especiais:	
Nome	Descrição
2.1 Edição Manual	Habilita edição manual do campo
2.2 Leitura de Arquivo	Encontra a consulta em um arquivo de texto
2.3 Armazenar Consulta	Mantém a consulta disponível para visualização

Quadro 3.2: Requisito 2

3. **Executar Consulta**	
Executar a consulta no banco de dados conectado	
Requisitos Especiais:	
Nome	Descrição
3.1 Execução	Executa junto a geração de mutantes
3.2 Mensagem de Erro	Abre uma mensagem de erro em caso de falha na execução

Quadro 3.3: Requisito 3

4. **Gerar Consultas Mutantes**	
Gerar as consultas mutantes com base na consulta original	
Requisitos Especiais:	
Nome	Descrição
4.1 Execução	Executa os operadores sobre a consulta inserida no campo
4.2 Atualizar Lista	Atualiza a lista da base de teste
4.3 Visualização	Mostra a consulta completa ao selecionar elemento na lista

Quadro 3.4: Requisito 4

5. **Executar Consulta Mutante**	
Executar consultas da base de teste e armazenar seu resultado	
Requisitos Especiais:	
Nome	Descrição
5.1 Execução	Executa junto a geração de mutantes

Quadro 3.5: Requisito 5

6. **Comparar Resultado**	
Compara o resultado da consulta original com cada consulta do banco de teste e atualiza seu estado	
Requisitos Especiais:	
Nome	Descrição
6.1 Execução	Executa junto a geração de mutantes
6.2 Atualizar Lista	Atualiza a lista da base de teste

Quadro 3.6: Requisito 6

7. **Trocar Estado**	
Habilita a troca de estado pelo usuário	
Requisitos Especiais:	
Nome	Descrição
7.1 Edição Manual	Habilita edição manual
7.2 Atualizar Lista	Atualiza a lista da base de teste

Quadro 3.7: Requisito 7

8. **Calcular Escore de Mutação**	
Realiza o cálculo do Escore de Mutação	
Requisitos Especiais:	
Nome	Descrição
8.1 Execução Automática	Executa automaticamente ao reconhecer mudança
8.2 Atualizar Lista	Atualiza a lista da base de teste

Quadro 3.8: Requisito 8

3.5 CASOS DE USO

Nesta Seção são apresentados o diagrama e os casos de uso criados para base do desenvolvimento da ferramenta.

3.5.1 ESPECIFICAÇÕES DE CASOS DE USO

A partir da análise dos requisitos elaborados, foram criados 8 casos de uso para o sistema. A apresentação destes é baseada em Larman (2005), com ligeiras alterações para adequação ao proposto. O diagrama de casos de uso é apresentado na Figura 8. Nesta Seção são apresentadas as descrições de três casos de uso principais: Executar Consulta original, Gerar Consultas Mutantes e Executar Consultas Mutantes. A descrição detalhada dos outros casos de uso pode ser vista no Apêndice A.

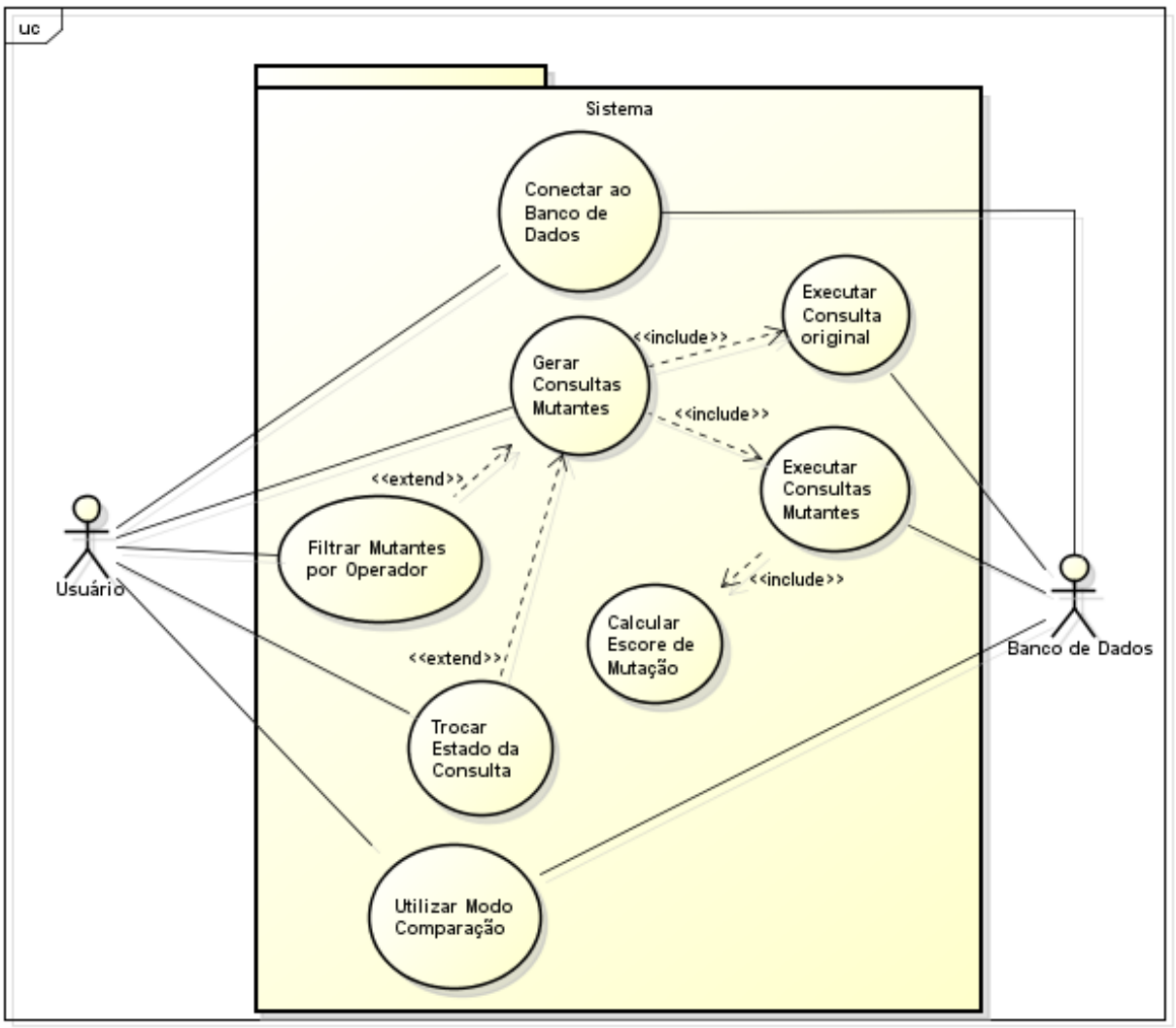


Figura 8: Diagrama de Caso de Uso.
Fonte: Autor

Executar Consulta Original.

- **Escopo:** Reconhecer a consulta introduzida pelo usuário e executá-la no banco de dados, armazenando o resultado obtido.
- **Nível:** Sub-função
- **Interessados e Interesses:**
  - Sistema: necessita uma conexão válida para realização de suas outras operações. - Usuário: validação da consulta que foi colocada como entrada.
- **Ator Principal:** Usuário.
- **Pré-condições:** Conexão efetiva com o banco de dados do usuário.
- **Garantia de Sucesso:** Armazenamento do resultado obtido pela consulta colocada como entrada pelo usuário.
- **Requisitos Associados:** 2 e 3
- **Variações tecnológicas:** A estrutura da consulta pode não estar correta, apresentando resultados não condizentes com as necessidades do usuário, antes mesmo da realização dos testes pelo sistema.

Fluxo Principal:

- **1.** A consulta é inserida pelo usuário;
- **2.** O sistema reconhece a consulta e executa no banco de dados conectado;
- **3.** O resultado obtido através da consulta é armazenado pelo sistema, para utilização posterior.

Fluxo Alternativo:

- **3.** A consulta retorna um resultado inválido ou apresentando algum tipo de erro;

  3.1 O sistema retorna uma mensagem de erro;

  3.2 O sistema retorna a tela principal;

Gerar Consultas Mutantes.

- **Escopo:** Aplicar operadores de mutação na consulta original, gerando as consultas que servirão para a base de testes.
- **Nível:** Sub-função
- **Interessados e Interesses:**
  - Sistema: necessita de conexão e uma consulta original válida para realização de suas outras operações.
  - Usuário: entender os dados aplicados a consulta de teste gerada e suas diferenças para a original.
- **Ator Principal:** Usuário.
- **Pré-condições:** Entrada de consulta validada.
- **Garantia de Sucesso:** Criação da lista de consultas mutantes para visualização na tela do sistema.
- **Requisitos Associados:** 4
- **Variações tecnológicas:** Existem várias nomenclaturas e formas de categorizar diferentes operadores de mutação para consultas em bancos de dados.

Fluxo Principal:

- **1.**A consulta validada é recebida pelo sistema;
- **2.**O sistema aplica o operador de mutação com base na consulta recebida;
- **3.**São geradas as consultas mutantes e armazenadas na base de testes;
- **4.**As consultas são listadas e categorizadas.

Fluxo Alternativo:

- **2.** O operador não se aplica a consulta recebida;

  2.1 O sistema passa ao próximo operador;

  2.1 O fluxo retorna ao passo 2.

Executar Consultas Mutantes

- **Escopo:** Executar as consultas da base de teste e atribuir seu resultado de acordo com a comparação com a consulta original.
- **Nível:** Sub-função
- **Interessados e Interesses:**
  - Sistema: comparar os resultados das consultas mutantes com o resultado da consulta original. - Usuário: identificação do estado da consulta de teste, para uma posterior análise.
- **Ator Principal:** Usuário.
- **Pré-condições:** Base de teste preenchida.
- **Garantia de Sucesso:** Atribuição dos valores "Vivo"ou "Morto"para as consultas mutantes da base de testes.
- **Requisitos Associados:** 4, 5 e 6
- **Variações tecnológicas:** Alguns operadores podem acabar não gerando consultas mutantes realmente efetivas, devido a diferentes formações de consultas. Isso pode comprometer resultados e análises posteriores. Ainda, existe ambiguidade entre diferentes tipos de operadores, podendo gerar consultas iguais para a análise.

Fluxo Principal:

- **1.** O sistema executa uma consulta da base de teste que ainda não possui classificação;
- **2.** O sistema armazena o resultado obtido pela consulta;
- **3.** O resultado é comparado com o armazenado da consulta original;
- **4.** O sistema atribui o estado de acordo com o resultado da comparação e passa para a próxima etapa.

Fluxo Alternativo:

- **4.** O sistema atribui o estado de acordo com o resultado da comparação.

  4.1 O fluxo retorna ao passo 1;

3.6 DIAGRAMA DE CLASSES

As principais classes e suas relações são apresentados no diagrama da Figura 9. Mais informações referentes as classes e seus métodos, estão no Apêndice A, o Manual do Desenvolvedor. Figuras com telas do sistema estão na próxima Seção, Utilização do Sistema.

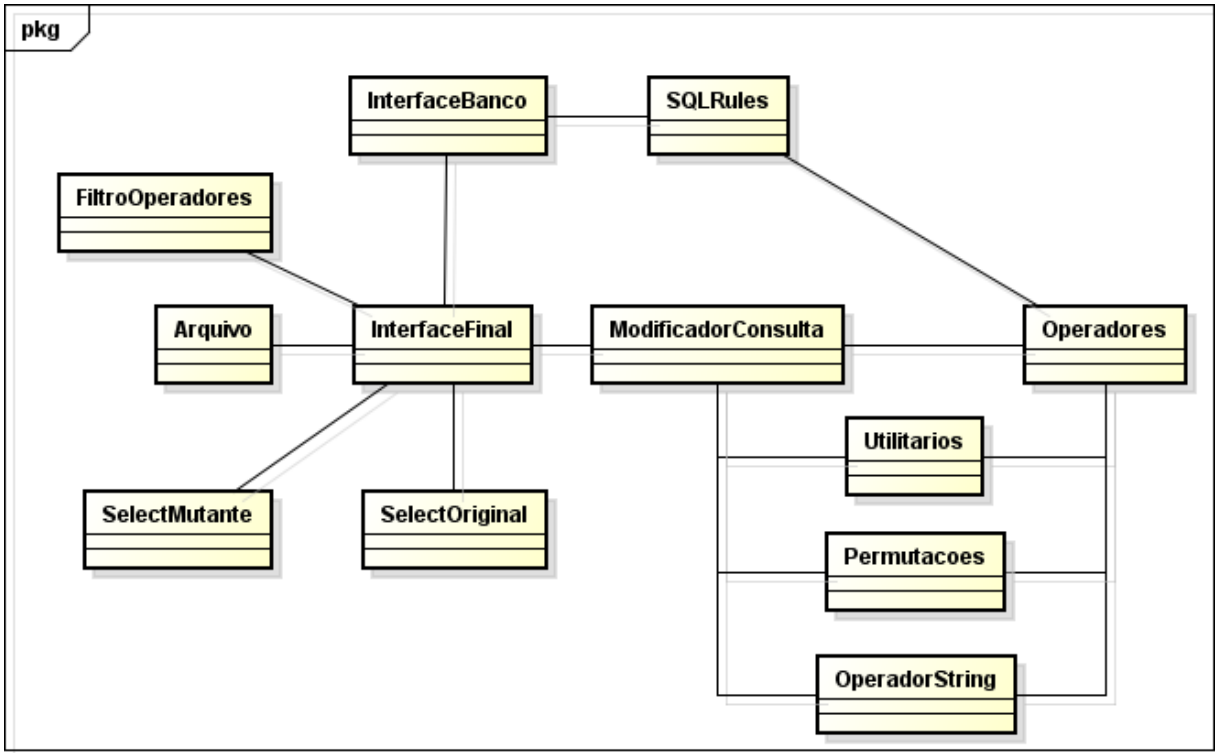


Figura 9: Diagrama de Classes Simplificado da Ferramenta.
Fonte: Autor

- **Classe InterfaceBanco**: Classe Interface responsável por armazenar o conteúdo de conexão ao banco de dados. A interface aparece ao iniciar o sistema e pode ser aberta a qualquer momento caso haja necessidade.
- **Classe InterfaceFinal**: Interface principal do sistema. Nela o usuário pode interagir com todas as funcionalidades do sistema.
- **Classe ModificadorConsultas**: Classe responsável pelo *heal* da consulta, deixando-a pronta para o uso dos operadores.
- **Classe Operadores**: Classe que contém os algoritmos dos Operadores de mutação. Seus métodos são utilizados por outras classes. Destro destes métodos, ainda, é realizada a execução da consulta no banco de dados, para definir o estado de cada mutante.

- **Classe OperadorString**: Classe auxiliar, a qual provê métodos de manipulação dos elementos de uma *String*. Seu objetivo é retornar os dados corretamente para a criação da consulta mutante. A *String* sempre está como parâmetro dos métodos.
- **Classe Utilitários**: Classe com métodos auxiliares, utilizados por outras classes, como por exemplo métodos de ordenação.
- **Classe SelectOriginal**: Classe interface, a qual apresenta os resultados de comparação de uma consulta mutante e da consulta original. A interface é aberta ao clicar na linha referente a consulta desejada.
- **Classe SelectMutante**: Classe interface, a qual apresenta os resultados de uma consulta mutante selecionada. A interface é aberta ao clicar na linha referente a consulta desejada.
- **Classe Permutações**: Classe responsável por criar permutações para utilização dos operadores.
- **Classe Defeitos**: Classe Interface utilizada para filtrar os resultados apresentados na tabela de mutantes.

3.7 UTILIZAÇÃO DO SISTEMA

A ferramenta possui uma interface na qual o usuário é livre para escolher qual banco de dados deve acessar, qual consulta deve ser testada e quais resultados são realmente interessantes para cada situação.

A conexão com o banco de dados é feita por meio da interface apresentada na Figura 10, na qual o usuário deve entrar com os dados do banco (*Host*, *Port* e *Nome*, e dados de um *login* válido e com as devidas permissões (*Nome de Usuário* e *Senha*).

Caso a conexão não seja devidamente estabelecida, a ferramenta exibe uma mensagem de erro, como pode ser visto na Figura 11.

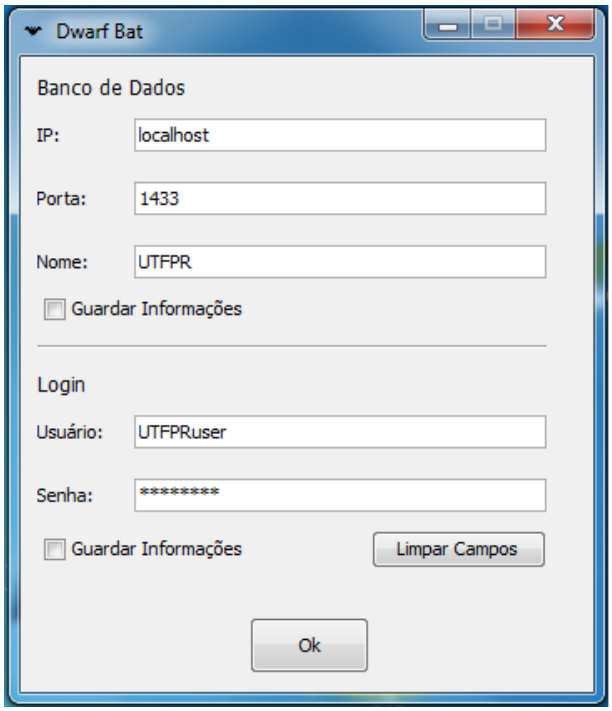


Figura 10: Interface de Conexão ao banco de dados.
Fonte: Autor

Figura 11: Mensagem de erro.
Fonte: Autor

A Figura 12 mostra a interface principal do sistema. Nela estão os seguintes elementos:

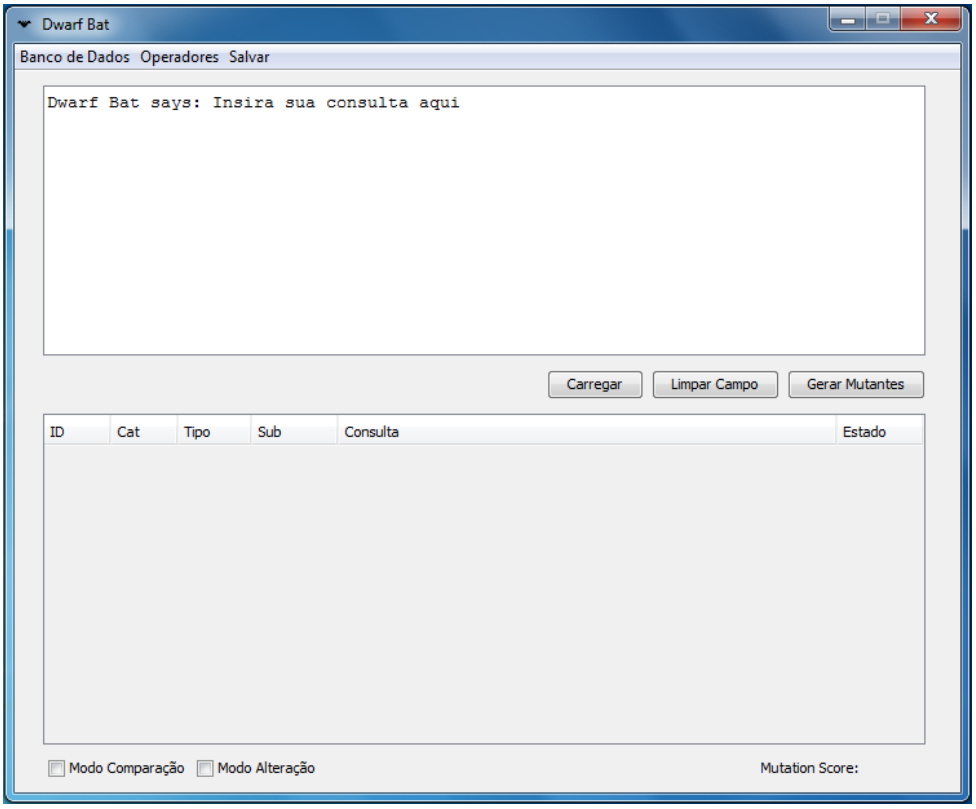


Figura 12: Interface Principal.
Fonte: Autor

- Botão Carregar: Permite o usuário selecionar um arquivo que contenha a consulta que deseja testar.
- Botão Limpar Campo: Limpa o campo de consulta.
- Botão Gerar Mutantes: Aplica todo o processo de teste baseado em defeitos a partir da consulta e do banco de dados fornecidos.
- Tabela de Mutantes: É preenchida pela geração dos mutantes. Seus campos indicam as categorias e tipos dos mutantes gerados, os próprios mutantes e seu estado final em comparação a consulta original. Um exemplo da tabela preenchida pode ser visto na Figura 13.
- Botão Operadores: Permite selecionar quais operadores devem ser apresentados na tabela de Mutantes. A interface de controle é apresentada na Figura 14.

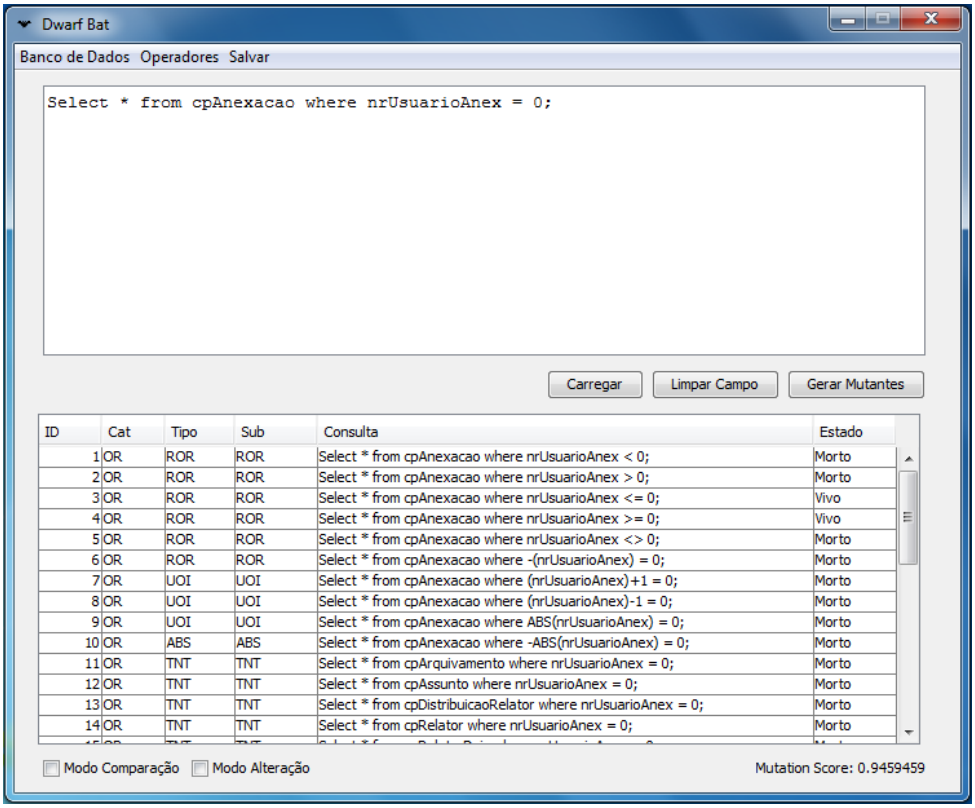


Figura 13: Tabela de Mutantes preenchida.
Fonte: Autor

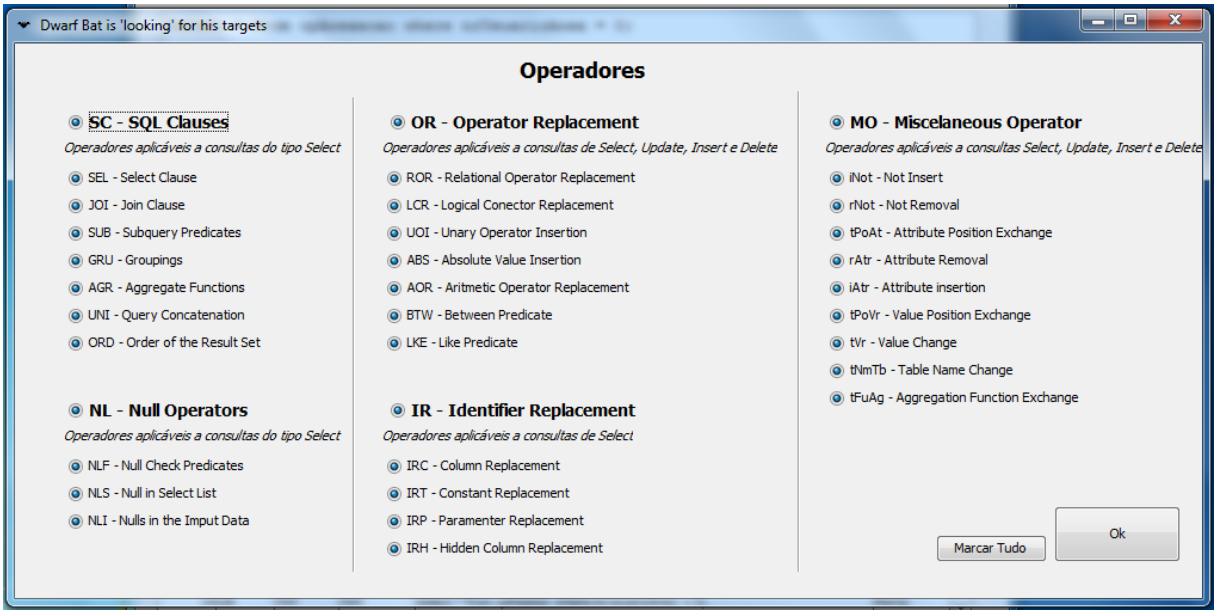


Figura 14: Interface de Operadores.
Fonte: Autor

Ainda, a funcionalidade permite o retorno dos operadores removidos por meio da própria Tabela de Mutantes e atualiza o Mutation Score automaticamente, como pode ser visto na Figura 15.

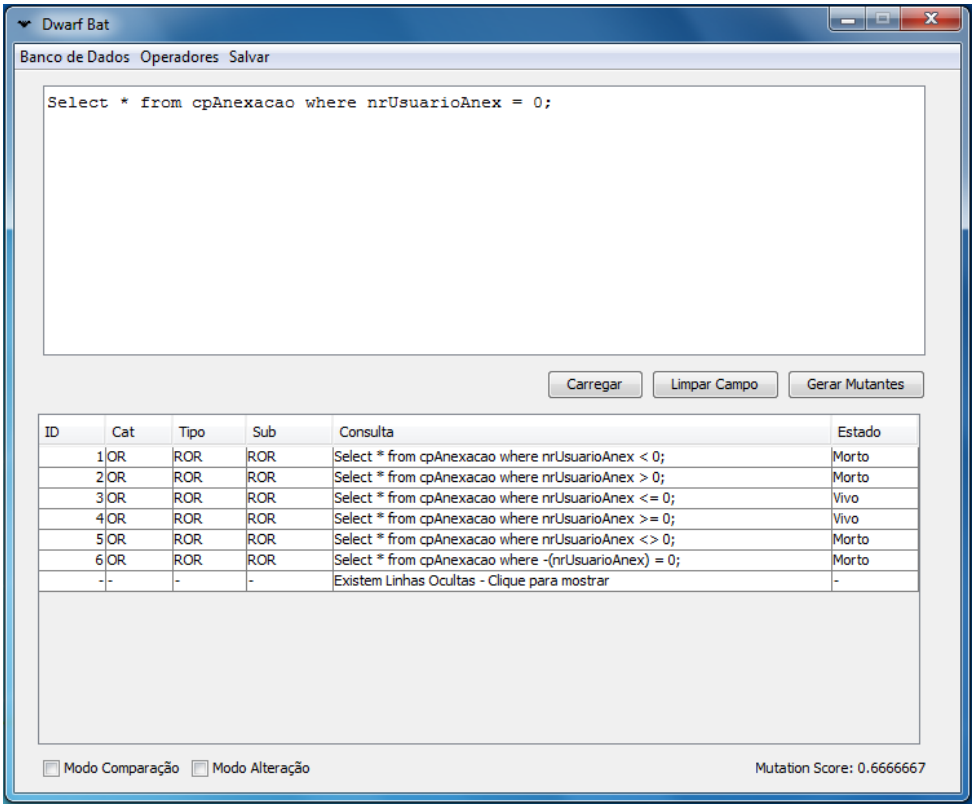


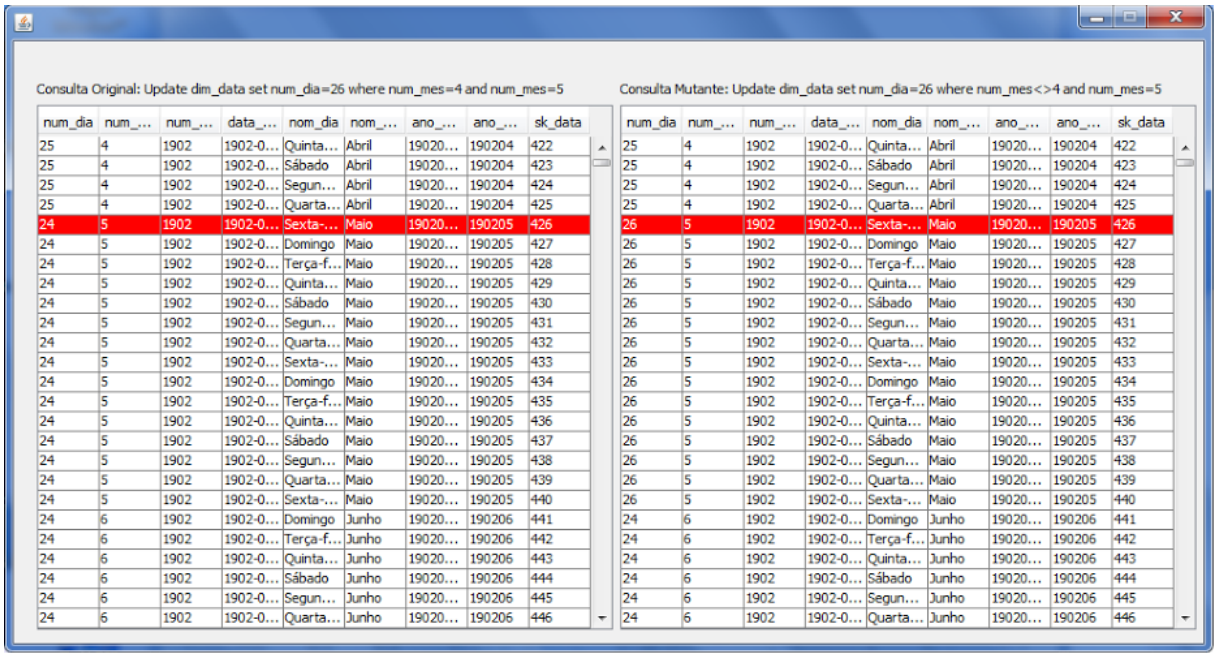
Figura 15: Tabela de Mutantes com elementos ocultos.
Fonte: Autor

- Botão Modo Comparação: Quando ativado, permite que o usuário selecione uma linha da Tabela de Mutantes, abrindo uma nova janela - Figura 16. Essa janela apresenta então os resultados obtidos pela execução da consulta original e da consulta mutante, para análise do usuário. Além disso, caso o mutante possua estado Morto, estará indicado o local que gerou este estado, ou seja, a primeira diferença entre os resultados.

- Botão Banco de Dados: Abre a interface de conexão ao banco de dados, em caso de possíveis necessidades de alterações.

- Botão Salvar: Salva os resultados obtidos na Tabela de Mutantes.

- Botão Modo Alteração: Permite que o usuário selecione uma linha da Tabela de Mutantes e altere seu estado final, alternando entre as opções Vivo, Morto e Equivalente. Essas opções então alteram automaticamente o Mutation Score.



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
26	5	1902	1902-0...	Sexta-...	Maio	19020...	190205	426
26	5	1902	1902-0...	Domingo	Maio	19020...	190205	427
26	5	1902	1902-0...	Terça-f...	Maio	19020...	190205	428
26	5	1902	1902-0...	Quinta...	Maio	19020...	190205	429
26	5	1902	1902-0...	Sábado	Maio	19020...	190205	430
26	5	1902	1902-0...	Segun...	Maio	19020...	190205	431
26	5	1902	1902-0...	Quarta...	Maio	19020...	190205	432
26	5	1902	1902-0...	Sexta-...	Maio	19020...	190205	433
26	5	1902	1902-0...	Domingo	Maio	19020...	190205	434
26	5	1902	1902-0...	Terça-f...	Maio	19020...	190205	435
26	5	1902	1902-0...	Quinta...	Maio	19020...	190205	436
26	5	1902	1902-0...	Sábado	Maio	19020...	190205	437
26	5	1902	1902-0...	Segun...	Maio	19020...	190205	438
26	5	1902	1902-0...	Quarta...	Maio	19020...	190205	439
26	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

Figura 16: Modo Comparação.
Fonte: Autor

4 ESTUDO DE CASO

Com necessidades internas, a Instituição A visa o processo de criação de um *Data Warehouse* a partir de sua base de dados relacional. Para isso, há a necessidade de uma ferramenta auxiliar para a descoberta de defeitos nesse processo.

O sistema denominado Controle de Trâmites é responsável pelo controle de fluxo dos processos na Instituição A, prestando 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 Microsoft 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 da Instituição A.

Para a realização do teste foram utilizadas 30 tabelas da base de dados relacional e 27 tabelas do DW, as quais foram copiadas e disponibilizadas em servidor remoto pela Instituição A. Todas as tabelas, tanto da fonte de dados como do DW, foram duplicadas 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 Seções 4.1, 4.2 e 4.3.

4.1 PLANEJAMENTO DO ESTUDO DE CASO

Neste estudo de caso foram utilizados 20 comandos de manipulação SQL retirados das *procedures* do processo de Carga dos Dados (*Load*) do processo ETL.

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.

Após a execução das consultas mutantes nas instâncias da base de dados, a

DWarF BaTT realiza uma análise automática do resultado das consultas avaliando se os mutantes resultaram em "Mortos" ou "Vivos". Para isso, a ferramenta realiza uma comparação entre os resultados gerados pela execução da consulta original e entre 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 para este último o testador deverá analisar se o mutante é equivalente ou não.

Caso seu estado seja determinado como equivalente, ou seja, o mutante continuou "Vivo" durante as execuções, significa que o mutante possui o mesmo comportamento do original. A partir disto, o testador decide se deve ou não continuar os testes, sempre levando em consideração o escore de mutação alcançado para o teste.

A qualidade 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.

Vale ressaltar que a ferramenta só aplica os operadores de mutação SQL se for possível a sua aplicabilidade aos comandos SQL, pois existem operadores de mutação SQL não aplicáveis a determinadas consultas.

Ainda, só são computadas no *Mutation Score* consultas válidas. Se por algum acaso um operador crie uma consulta que resulte em algum erro de execução, o sistema não computará seu resultado, por não se aplicar a uma consulta válida.

O estudo foi dividido em três rodadas. Isso se deve ao fato de que apenas ao fim da terceira rodada de testes foram alcançados os números de escore de mutação satisfatórios para o término. Esse número pode variar de acordo com os objetivos do teste, caso a caso.

A Figura 17 mostra o diagrama simplificado do ciclo realizado no estudo de caso. Primeiramente, as vinte consultas são executadas no sistema; seus resultados então são analisados, principalmente os mutantes de estado Vivo. De acordo com as análises, são inseridos dados na base com o objetivo de matar aqueles mutantes classificados como vivos e em sequencia o ciclo se repete.

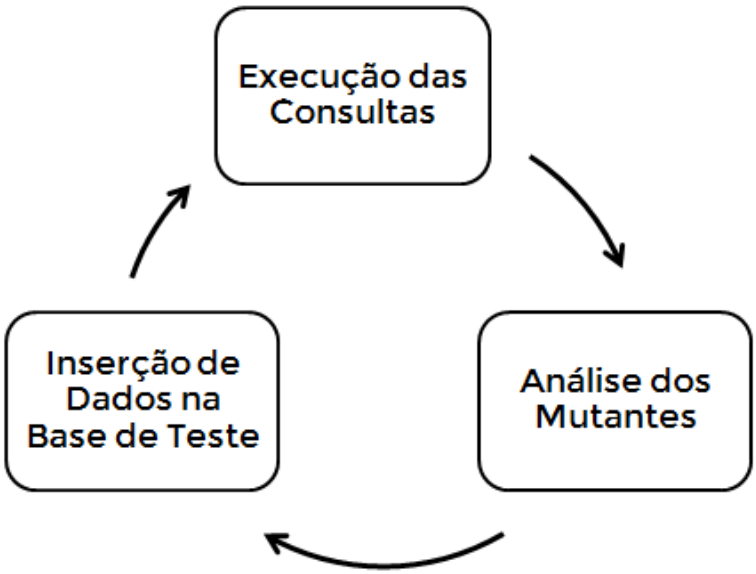


Figura 17: Ciclo do Estudo de Caso.
Fonte: Autor

4.2 PRIMEIRA RODADA

Na primeira execução, as consultas são desenvolvidas no sistema baseando-se na cópia fiel do banco de dados, sem haver qualquer alteração nos dados ou no esquema do mesmo. Os resultados da primeira execução são apresentados na Tabela 2.

De acordo com as analises realizadas, os principais comportamentos apresentados foram:

- Mutantes vivos gerados de operadores de alteração lógica (LCR em Tuya et al. (2006)) geravam resultados idênticos pelo fato do operador *OR* ser menos restritivo que o operador *AND*. Para matar esses mutantes, foram inseridos novos dados na base de teste.

- Mutantes vivos gerados pelo operador SEL tiveram suas diferenças no Select para o Select Distinct, ou seja, a ausência de um ou mais aparições iguais na seleção final dos dados. Para matá-los, foram adicionadas ou removidas instancias de dados que se enquadravam nas restrições impostas pela consulta.

- Mutantes vivos gerados pelo operador ROR apresentaram um efeito particular. Sempre que houve a troca sem o comprometimento do resultado final percebeu-se que a aplicação entrava dentro de um parâmetro de operação lógica OR e que - em todos os casos obtidos - o problema era apresentado no outro parâmetro, tornando sua

Tabela 2: Primeira Execução

	Mutantes		
Consulta SQL	Vivos	Mutantes	Escore de Mutação
1	5	76	0,94
2	5	76	0,94
3	3	78	0,96
4	5	76	0,94
5	10	79	0,89
6	4	76	0,95
7	5	70	0,93
8	5	70	0,93
9	5	70	0,93
10	2	73	0,97
11	2	4	0,67
12	3	60	0,95
13	3	47	0,94
14	1	50	0,98
15	1	51	0,98
16	6	109	0,95
17	2	80	0,98
18	4	49	0,92
19	3	32	0,93
20	0	35	1

reparação singular inútil. Para matá-lo foi necessário adequar a base para ambos os parâmetros da consulta.

- Mutantes vivos gerados pelos operadores de inserção e remoção de NOT apresentaram resultados similares ao operador ROR, no qual em maior parte apareciam ligadas a parâmetros de operadores lógicos. Para matá-los também foi realizada a inserção de dados de ambos os parâmetros necessários.

4.3 SEGUNDA RODADA

A Tabela 3 apresenta os resultados obtidos na segunda execução. Percebe-se que o *Mutation Score* de boa parte das consultas aumentou, revelando uma eficiência nas inserções propostas. No entanto algumas consultas ainda apresentaram resultados vivos, referentes aos operadores SEL, JOI, ROR e TFA.

- Mutantes vivos gerados pelo operador SEL foram ocorrências não distintas que passaram despercebidas no momento de análise. Basicamente foi realizado o mesmo

processo para resolução, adicionando-se a conferencia de recorrência de casos possíveis, sendo aplicada a inserção de dados necessária para cada caso.

- Mutantes vivos gerados pelo operador ROR também foram casos recorrentes no banco de dados. O processo para resolução foi o mesmo, adicionando-se a conferencia de recorrência de casos possíveis, sendo aplicada a inserção de dados necessária para cada caso.

Tabela 3: Segunda Execução

	Mutantes		
Consulta SQL	Vivos	Mutantes	Escore de Mutação
1	1	80	0,99
2	1	80	0,99
3	0	81	1
4	0	81	1
5	0	89	1
6	0	81	1
7	0	75	1
8	0	75	1
9	0	75	1
10	1	74	0,99
11	2	4	0,67
12	1	62	0,98
13	0	50	1
14	1	50	0,98
15	1	51	0,98
16	0	115	1
17	1	81	0,99
18	1	52	0,98
19	1	34	0,97
20	0	35	1

4.4 TERCEIRA RODADA

A terceira execução apresenta os resultados finais do estudo de caso, os quais podem ser observados na Tabela 4.

Tabela 4: Terceira Execução

	Mutantes		
Consulta SQL	Vivos	Mutantes	Escore de Mutação
1	1	80	0,99
2	1	80	0,99
3	0	81	1
4	0	81	1
5	0	89	1
6	0	81	1
7	0	75	1
8	0	75	1
9	0	75	1
10	1	74	0,99
11	2	4	0,67
12	1	62	0,98
13	0	50	1
14	1	50	0,98
15	1	51	0,98
16	0	115	1
17	0	82	1
18	0	53	1
19	0	35	1
20	0	35	1

4.5 CONCLUSÃO DO ESTUDO DE CASO

Os resultados finais do estudo de caso são apresentados na Tabela 5, a qual 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.

Foram gerados um total de 1336 mutantes, distribuídos em 10 classes de operadores de mutação SQL diferentes.

Na Figura 18 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*).

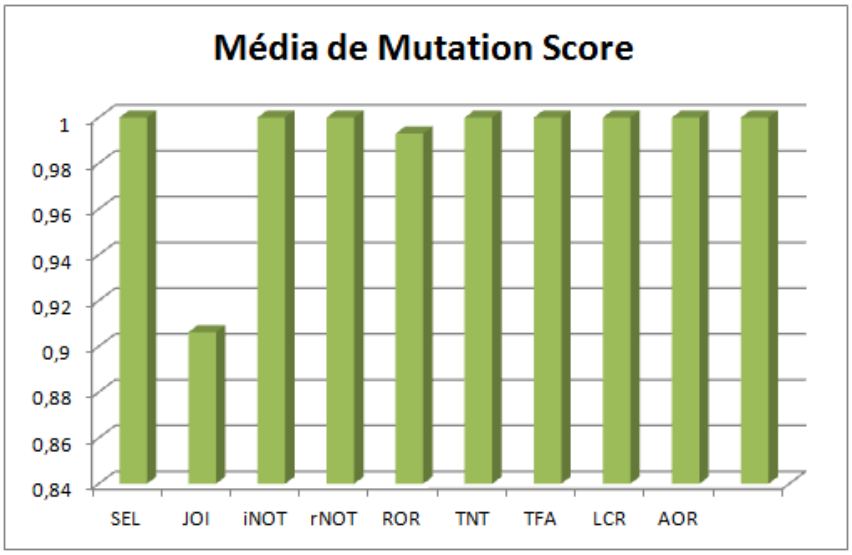


Figura 18: Média de Escore de Mutação por Operador
Fonte: Autor

Na Figura 19 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).

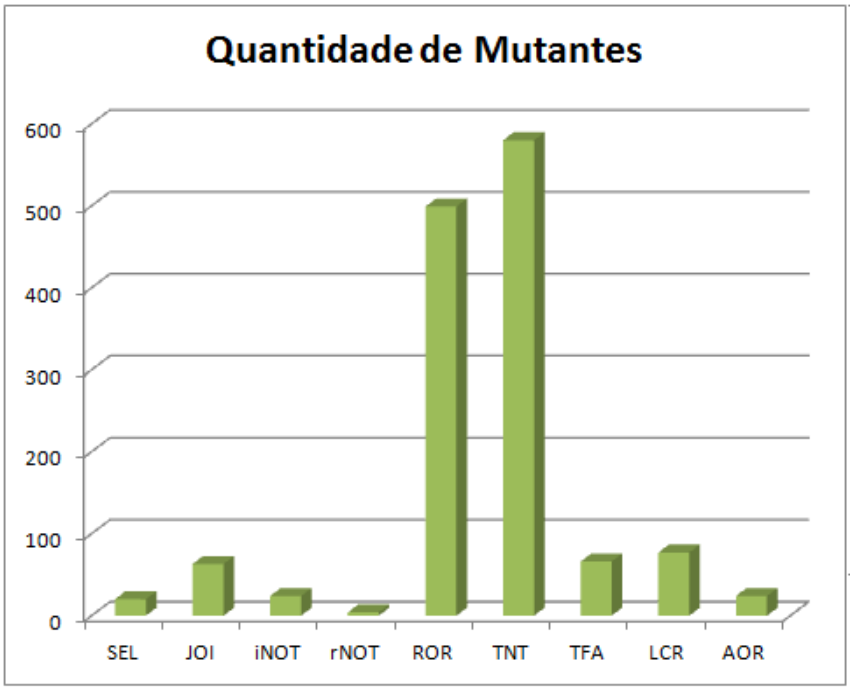


Figura 19: Quantidade de Mutantes por Operador.
Fonte: Autor

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.

Tabela 5: Mutantes por Consulta

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

Continua na próxima página

Tabela 5: Mutantes por Consulta

Consulta SQL	Operador	Vivos	Mortos	Escore de Mutação
	tNmTb	0	29	1
5	SEL	0	1	1
	JOI	0	4	1
	iNOT	0	2	1
	ROR	0	40	1
	tFuAg	0	6	1
	LCR	0	7	1
	tNmTb	0	29	1
6	SEL	0	1	1
	JOI	0	4	1
	iNOT	0	1	1
	ROR	0	35	1
	tFuAg	0	6	1
	LCR	0	5	1
	tNmTb	0	29	1
7	SEL	0	1	1
	JOI	0	4	1
	iNOT	0	1	1
	ROR	0	30	1
	tFuAg	0	6	1
	SEL	0	1	1
	LCR	0	4	1
	tNmTb	0	29	1
8	SEL	0	1	1
	JOI	0	4	1
	iNOT	0	1	1
	ROR	0	30	1
	tFuAg	0	6	1
	LCR	0	4	1
	tNmTb	0	29	1
9	SEL	0	1	1
	JOI	0	4	1

Continua na próxima página

Tabela 5: Mutantes por Consulta

Consulta SQL	Operador	Vivos	Mortos	Escore de Mutação
	iNOT	0	1	1
	ROR	0	30	1
	tFuAg	0	6	1
	LCR	0	4	1
	tNmTb	0	29	1
10	SEL	0	1	1
	JOI	1	3	0,75
	iNOT	0	1	1
	ROR	0	30	1
	tFuAg	0	6	1
	LCR	0	4	1
	tNmTb	0	29	1
11	SEL	0	1	1
	JOI	2	2	0,5
	LCR	0	1	1
12	SEL	0	1	1
	JOI	1	3	1
	iNOT	0	1	1
	ROR	0	25	1
	LCR	0	3	1
	tNmTb	0	29	1
13	SEL	0	1	1
	BTW	0	2	1
	iNOT	0	2	1
	rNOT	0	2	1
	ROR	0	10	1
	LCR	0	4	1
	tNmTb	0	29	1
14	SEL	0	1	1
	JOI	0	4	1
	iNOT	0	1	1
	ROR	1	14	0,93

Continua na próxima página

Tabela 5: Mutantes por Consulta

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

Continua na próxima página

Tabela 5: Mutantes por Consulta

Consulta SQL	**Operador**	**Vivos**	**Mortos**	**Escore de Mutação**
	tNmTb	0	29	1
20	SEL	0	1	1
	ROR	0	5	1
	tNmTb	0	29	1

5 CONSIDERAÇÕES FINAIS

Este trabalho teve como objetivo o desenvolvimento de uma ferramenta de testes, visando revelar defeitos cometidos no processo de criação e uso de bancos de dados, com foco em *Data Warehouse*. Os defeitos foram detectados por meio da técnica baseada em defeitos com o uso do critério de teste análise de mutantes, a partir de operadores de mutação propostos na literatura anteriormente.

O processo de teste executado pela ferramenta consistiu em aplicar a análise de mutantes em consultas SQL que são usadas para integração de dados em um DW. A proposição desta ferramenta teve por objetivo auxiliar a criação de um DW, de modo aumentar a confiabilidade nos dados do DW, que podem ser utilizados para apoio de decisões estratégicas de uma determinada empresa. A ferramenta possui 20 operadores implementados, os quais podem ser aplicados a consultas do tipo Insert, Delete, Update e Select.

Para validação da ferramenta foi executado um estudo de caso com dados reais extraídos da base de dados de um órgão público, denominada no trabalho como Instituição A. Os resultados obtidos neste estudo de caso permitiram concluir que a ferramenta pode contribuir para a construção de um *DW* com dados confiáveis, já que foi possível aplicar a análise de mutantes nas consultas SQL que realizam o ETL do DW, encontrando os defeitos representados pelo mutantes.

Além disso, diferentemente da ferramenta de Tuya et al. (2006), a *DWarF BaTT* é capaz de executar consultas de modificação dos dados, não se restringindo somente ao Select, tornando a ferramenta uma contribuição tanto para o meio acadêmico quanto para a Instituição A.

5.1 TRABALHOS FUTUROS

Como trabalho futuro tem-se:

- Refinamento da Ferramenta, como o conserto de *bugs* e a implementação de novas funcionalidades.
- Implementação de novos operadores de mutação
- Utilização de outros tipos de banco de dados, como MySQL e Oracle
- Tradução do Sistema para C#, devido as necessidades de uso exigidas pela Instituição A

REFERÊNCIAS

BARBOSA, E.; MALDONADO, J.; VINCENZI, A. **Towards the determination of sufficient mutant operators for C, Software Testing, Verification and Reliability**. [S.l.]: *, 2001.

CABEçA, A. G. **Análise de Mutantes em Aplicações SQL de Banco de Dados**. 2009.

DELAMARO, M. E.; MALDONADO, J. C.; JINO, M. **Introdução ao Teste de Software**. Rio de Janeiro: Elsevier, 2007.

DEMILLO, R.; LIPTON, R.; SAYWARD, F. **Hints on test data selection: Help for the practicing programmer**. April 1978. Computer.

ELGAMAL, N.; ELBASTAWISSY, G.-E. A. **Data Warehouse testing**. 2013. Proceedings of the Joint EDBT/ICDT.

HOKAMA, D. D. B. et al. **A Modelagem de Dados no ambiente Data Warehouse**. 2004.

INMON, W. H. **Building the Data Warehouse**. 4ª. ed. Indianapolis, Indiana: Wiley, 2012.

KIMBALL, R.; ROSS, M. **The Data Warehouse Toolkit**. Third. [S.l.]: Wiley, 2013.

LARMAN, C. **Utilizando UML e Padrões**. 3. ed. [S.l.]: Bookman, 2005.

LAUDON, K. C.; LAUDON, J. P. **Sistemas de Informação Gerenciais**. [S.l.]: Prentice Hall, 2013.

MEKTEROVIC, I.; BRKIC, L.; BARANOVIC, M. **A generic procedure for integration testing of ETL procedures**. 2011. Automatika.

MUNAWAR, M.; SALIM, N.; IBRAHIM, R. **Towards Data Quality into the Data Warehouse Development**. December 2011. IEEE Ninth International Conference.

MYERS, G. J. **The Art of Software Testing**. New York: Wiley, 2004.

OLIVEIRA, I. **Teste Baseado em Defeitos para Ambientes de Data Warehouse**. Curitiba: [s.n.], 2015. 148 p. Dissertaçào de Mestrado - PPGCA, Universidade Tecnológica Federal do Paraná - UTFPR.

PRESSMAN, R. S. **Engenharia de Software**. 6ª. ed. Rio de Janeiro: Mc Graw-Hill, 2006. 720 p.

SOMMERVILLE, I. **Engenharia de Software**. 6ª. ed. São Paulo: Addison-Wesley, 2003.

TURBAN, E. et al. **Business Intelligence: A Managerial Approach**. 2. ed. [S.l.]: Prentice Hall, 2010.

TUYA, J.; SUáREZ-CABAL, M. J.; RIVA, C. de la. Mutating database queries. **Science Direct**, 2006.

WU, M. C.; BUCHMANN, A. P. **Research Issues in Data Warehousing**. March 1997. Proceesings of the German Database Conference.

APÊNDICE A – MANUAL DO DESENVOLVEDOR

.

Documentação do Software de Análise de Mutantes SQL

Brasil

Curitiba, 2015

.

Documentação do Software de Análise de Mutantes SQL

Universidade Tecnológica Federal do Paraná

Departamento Acadêmico de Informática - DAINF

PPGCA

Brasil

Curitiba, 2015

Sumário

1 Introdução

O sistema tem como objetivo auxiliar a tomada de decisão no momento de criar e executar uma consulta SQL em um banco de dados. Através da execução de diferentes casos de teste, o sistema pode ajudar a encontrar falhas ou defeitos na criação ou manutenção de bancos de dados, com foco na utilização em ambientes *Data Warehouse*.

1.1 Objetivos Específicos

- Descobrir o máximo possível de erros nos casos de teste (operadores de mutação)
- Prover base matemático estatística para a tomada de decisão (Mutation Score e filtro de mutantes)
- Gerar conteúdo para o entendimento dos erros (comparação de tabelas resultantes)

1.2 Utilização do Sistema

A ferramenta possui uma interface na qual o usuário é livre para escolher qual banco de dados deve acessar, qual consulta deve ser testada e quais resultados são realmente interessantes para cada situação.

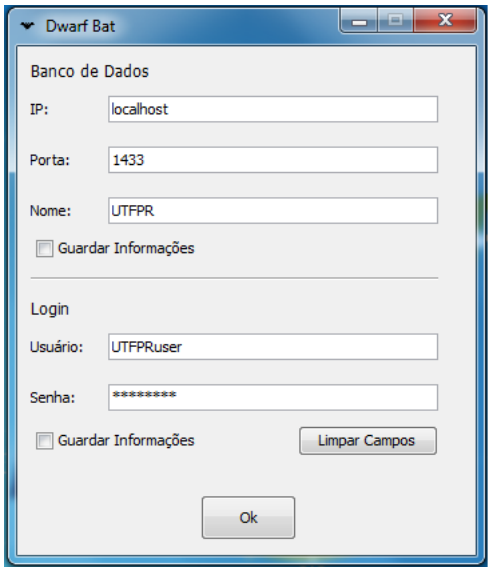


Figura 1 – Interface de Conexão ao banco de dados.
Fonte: Autor

A conexão com o banco de dados é feita através da interface apresentada na Figura 1, na qual o usuário deve entrar com os dados do banco (*Host*, *Port* e *Nome*, e dados de um login válido e com as devidas permissões (*Nome de Usuário* e *Senha*).

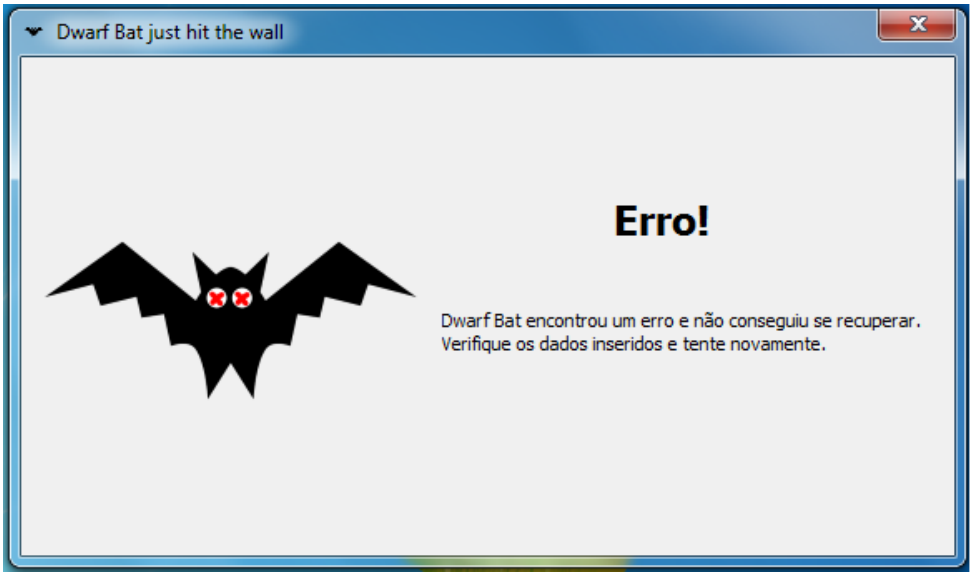


Figura 2 – Mensagem de erro.
Fonte: Autor

Caso, a conexão não seja devidamente estabelecida, a ferramenta mostrará uma mensagem de erro, como pode ser visto na Figura 2.

A Figura 3 mostra a interface principal do sistema. Nela estão os seguintes elementos:

- Botão Carregar: Permite o usuário selecionar um arquivo que contenha a consulta que deseja testar.
- Botão Limpar Campo: Limpa o campo de consulta.
- Botão Gerar Mutantes: Aplica todo o processo de teste baseado em defeitos a partir da consulta e do banco de dados fornecidos.

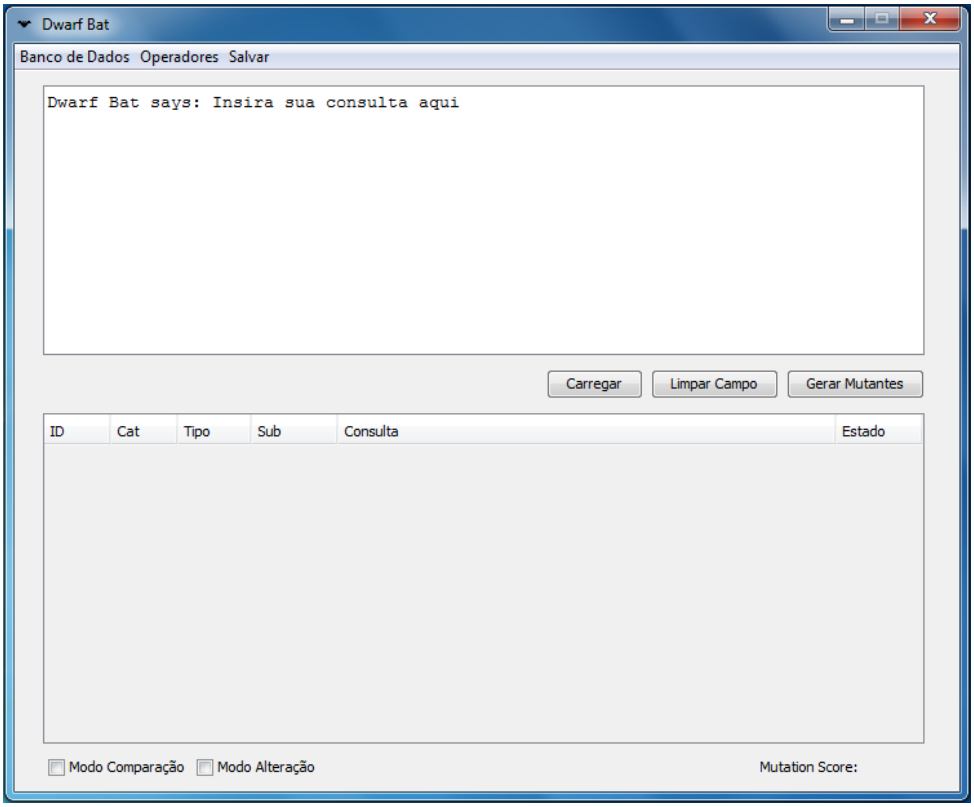


Figura 3 – Interface Principal.
Fonte: Autor

- Tabela de Mutantes: É preenchida pela geração dos mutantes. Seus campos indicam as categorias e tipos dos mutantes gerados, os próprios mutantes e seu estado final em comparação a consulta original. Um exemplo da tabela preenchida pode ser visto na Figura 4.

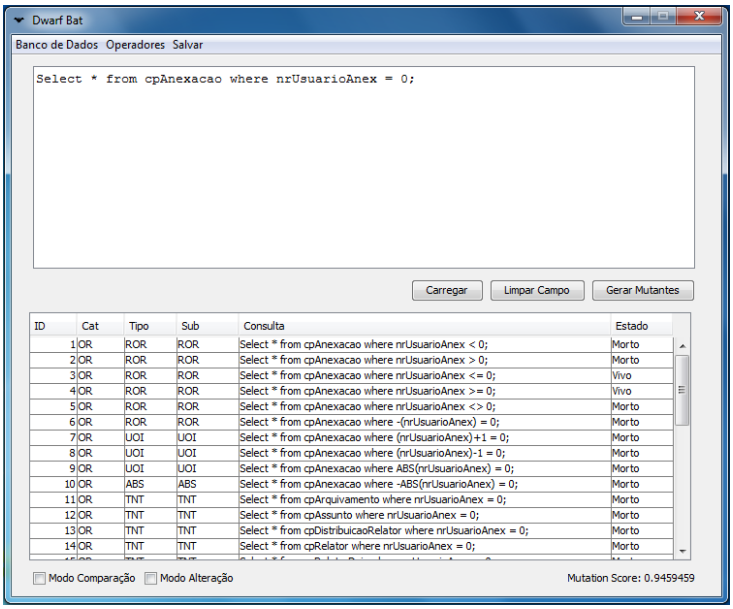


Figura 4 – Tabela de Mutantes preenchida.
Fonte: Autor

- Botão Operadores: Permite selecionar quais operadores devem ser apresentados na tabela de Mutantes. A interface de controle é apresentada na Figura 5.

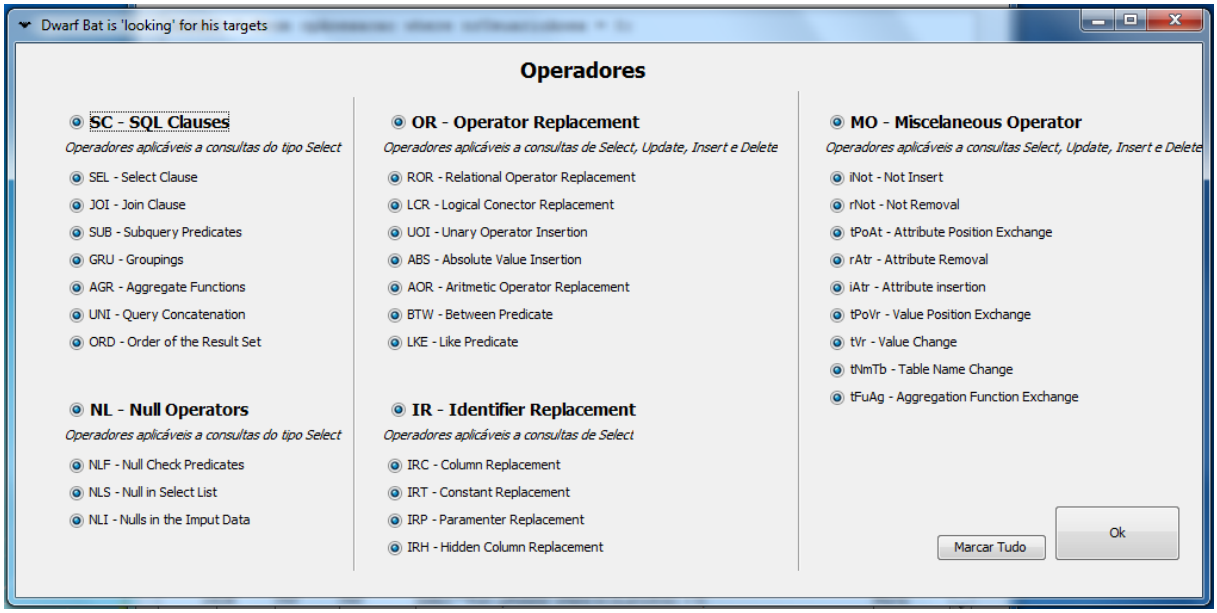


Figura 5 – Interface de Operadores.
Fonte: Autor

Ainda, a funcionalidade permite o retorno dos operadores removidos através da

própria Tabela de Mutantes e atualiza o Mutation Score automaticamente, como pode ser visto na Figura 6.

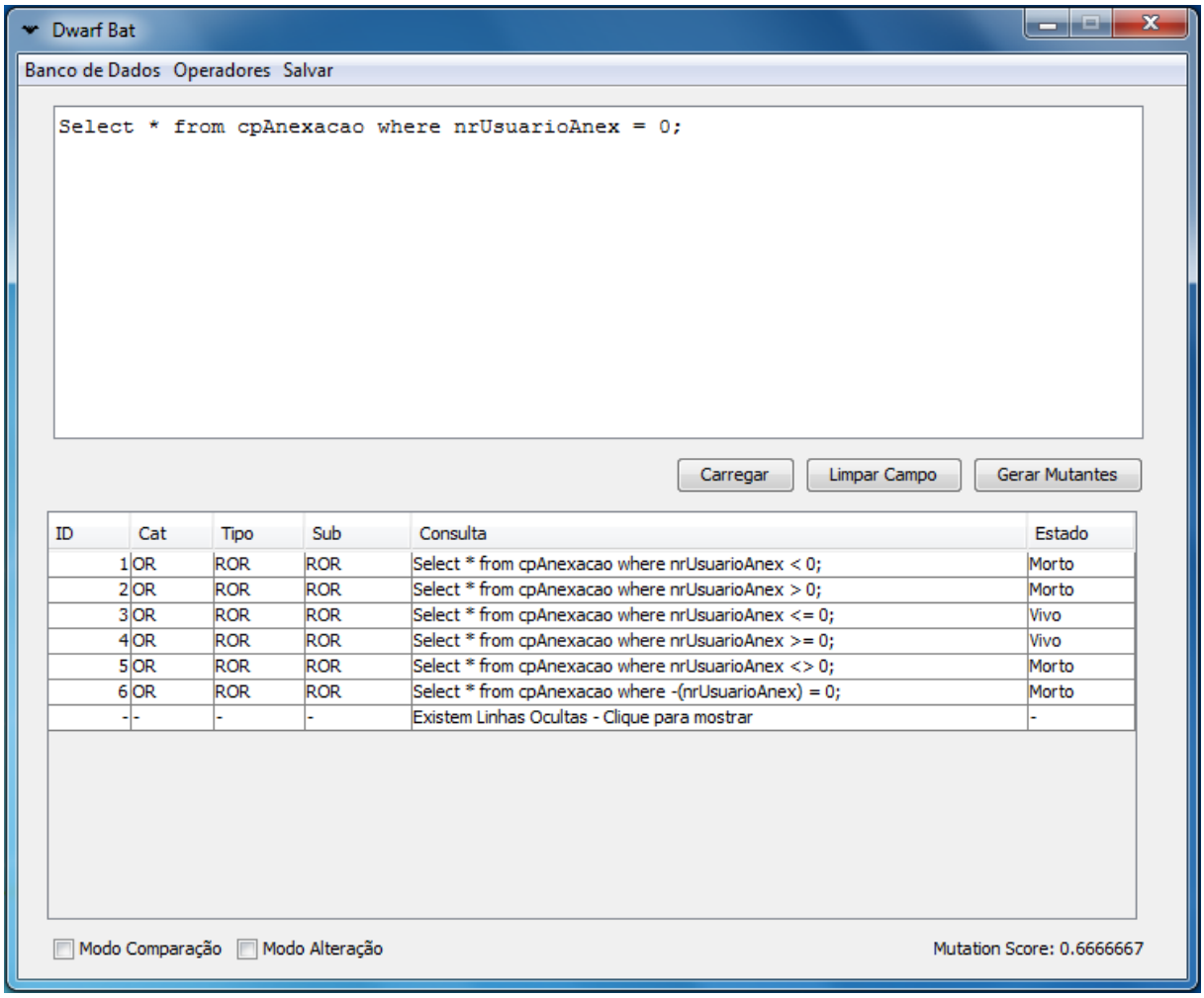


Figura 6 – Tabela de Mutantes com elementos ocultos.
Fonte: Autor

- Botão Banco de Dados: Abre a interface de conexão ao banco de dados, em caso de possíveis necessidades de alterações.
- Botão Salvar: Salva os resultados obtidos na Tabela de Mutantes.
- Botão Modo Comparação: Quando ativado, permite que o usuário selecione uma linha da Tabela de Mutantes, abrindo uma nova janela, com o resultado da consulta mutante e o resultado da consulta original, como pode ser visto na Figura 7. Além disso, caso o mutante possua estado Morto, estará indicado o local que gerou este estado.
- Botão Modo Alteração: Permite que o usuário selecione uma linha da Tabela de Mutantes e altere seu estado final, alternando entre as opções Vivo, Morto e Equivalente. Essa opções então alteram automaticamente o Mutation Score.

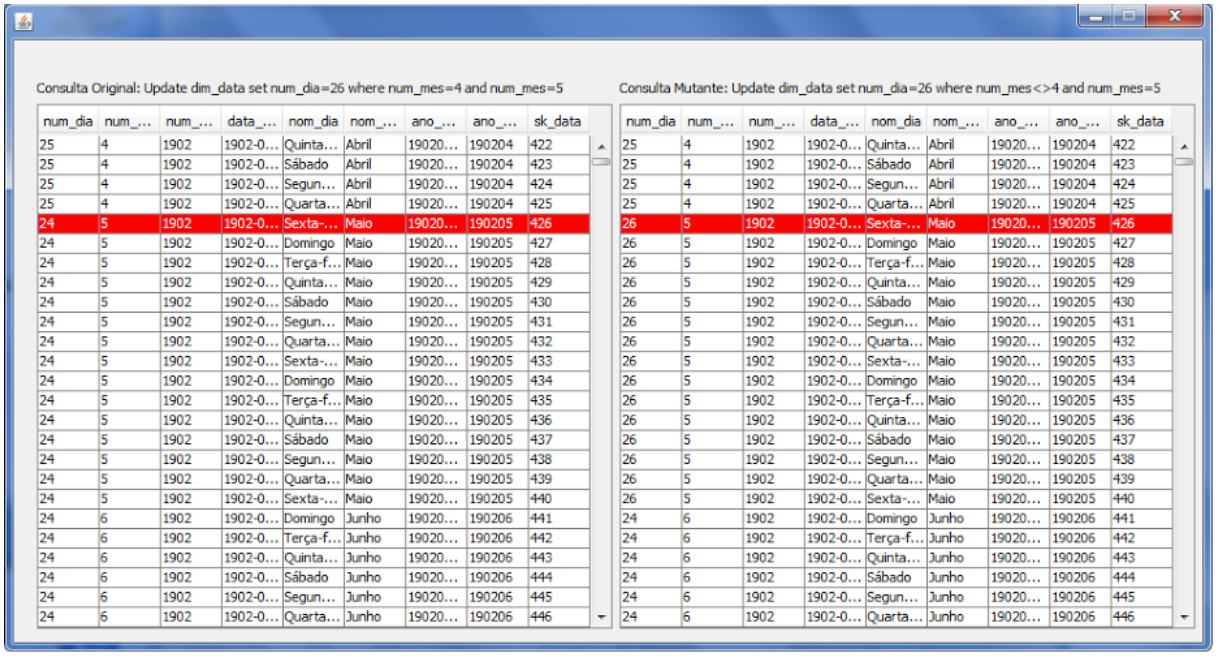


Figura 7 – Modo Comparação.
Fonte: Autor

1.3 Perguntas Frequentes

1. **Em que tipos de banco de dados serão aplicados os testes?**

   Os testes serão aplicados na criação de em um banco de dados do tipo *Data Warehouse*. Os testes também podem ser executados em bancos de dados relacionais, caso necessário.

2. **O sistema será usado especificamente para um banco, ou funciona de maneira dinâmica?**

   O sistema deve ser projetado para ser utilizado apenas em bancos de dados Microsoft SQL Server. No entanto, existe a possibilidade de usar diferentes bancos, mas não de forma simultânea.

3. **Em que parte do processo de ETL serão aplicadas os testes?**

   O sistema deve ser capaz de testar uma grande variedade de consultas das três partes do processo, mas o foco são nas consultas de Carga (Loading).

4. **Que tipo de profissional pode operar o sistema?**

   O sistema deve ser operado por um administrador do banco de dados ou um especialista em testes, capaz de discernir se uma consulta realmente apresentará problemas caso seja indicada pelo sistema.

5. **Qual é o retorno do sistema?**

O sistema deve resultar consultas mutantes e o seu estado em comparação à consulta original. Caso o resultado verdadeiro de um estado seja equivalente, este será marcado como vivo, o qual deve ser analisado por um especialista e/ou inserido em outros casos de teste podendo então ser convertido para equivalente.

6. **Há outros sistemas da Instituição A que se integrarão com o sistema?**

   O sistema pode ser utilizado em outros bancos de dados, desde que também se utilizem do Microsoft SQL Server e suas respectivas consultas sejam compatíveis ao sistema. Algumas palavras chaves não são contempladas pelo sistema, mas nota-se que isso não compromete o resultado final de cada mutante/consulta, apenas ignora possíveis criações de outros mutantes. Posteriormente, o sistema terá a integração do sistema de análise de dados.

7. **Quais permissões são necessárias para o logon ter acesso a todos os recursos do sistema?**

   Há a necessidade do logon criar e deletar tabelas e reconhecer todos os campos e atributos de todas as tabelas.

8. **Algum tipo de dado é guardado?**

   Cada consulta a ser testada deve gerar resultados em tempo razoável, com o objetivo de não necessitar guardar um log dos resultados obtidos anteriormente. Há a possibilidade de extrair os dados diretamente da interface, caso o usuário deseje armazená-los.

9. **Há permissões diferentes para uso do sistema?**

   O usuário será capaz de utilizar todas as funcionalidades do sistema. No entanto, caso o logon não tenha todos as permissões necessárias, as funcionalidades terão um comportamento errôneo, o qual pode comprometer os testes.

10. **Há integração on-line?**

    Caso o banco de dados a ser acessado não esteja na mesma máquina do sistema (não recomendado), há necessidade de uma conexão capaz de alcançar o banco de dados. Não existe uma versão web, que realize as mesmas funções ou funções integradas do sistema.

11. **É necessário utilizar um banco especificamente para os testes ou o sistema pode ser utilizado diretamente no banco original?**

    Não há a necessidade de criar um outro banco, pois as funcionalidades são reversíveis. No entanto, é muito recomendável que seja utilizado um banco idêntico ao que será testado, pois o sistema se utiliza de comandos como *drop table* e *delete*, os quais podem eventualmente ocasionar problemas.

12. **O usuário pode manipular os resultados gerados pelo sistema, caso seja de seu interesse?**

    Sim, existem filtros para manipular apenas os tipos de consultas desejadas pelo usuário. Ainda, caso seja confirmado pelo usuário que um mutante tem seu estado como equivalente, o mesmo pode ser alterado na própria interface.

13. **Outros profissionais usarão o mesmo sistema?**

    O sistema apresenta apenas uma instancia, a qual não armazena resultados obtidos, não havendo necessidade para usuários utilizarem o sistema simultaneamente.

14. **Há um espaço para possíveis anotações com relação a notas do usuário?**

    Não, as anotações devem ser feitas em softwares separados.

2 Requisitos do Sistema

2.1 RF 001 - Conectar ao banco de Dados

Tabela 1 – Requisito 1

1. **Conectar Banco de Dados**	
Conectar ao banco de dados fornecido pelo usuário	
Requisitos Especiais:	
Nome	Descrição
1.1 Acesso ao Banco	Recebe como entrada a URL do banco de dados
1.2 Acesso ao Logon	Recebe como entrada o logon e sua senha
1.3 Mensagem de Erro	Abre uma janela de Erro, em caso de falha na conexão

2.2 RF 002 - Reconhecer consulta

Tabela 2 – Requisito 2

2. **Reconhecer Consulta**	
Reconhecer consulta fornecida pelo usuário	
Requisitos Especiais:	
Nome	Descrição
2.1 Edição Manual	Habilita edição manual do campo
2.2 Leitura de Arquivo	Encontra a consulta em um arquivo de texto
2.3 Armazenar Consulta	Mantém a consulta disponível para visualização

2.3 RF 003 - Executar consulta

Tabela 3 – Requisito 3

3. **Executar Consulta**	
Executar a consulta no banco de dados conectado	
Requisitos Especiais:	
Nome	Descrição
3.1 Execução	Executa junto a geração de mutantes
3.2 Mensagem de Erro	Abre uma mensagem de erro em caso de falha na execução

2.4 RF 004 - Gerar Base de teste

Tabela 4 – Requisito 4

4. **Gerar Base de Teste**	
Gerar as consultas mutantes com base na consulta original	
Requisitos Especiais:	
Nome	Descrição
4.1 Execução	Executa os operadores sobre a consulta inserida no campo
4.2 Atualizar Lista	Atualiza a lista da base de teste
4.3 Visualização	Mostra a consulta completa ao selecionar elemento na lista

2.5 RF 005 - Executar consulta mutante

Tabela 5 – Requisito 5

5. **Executar Consulta Mutante**	
Executar consultas da base de teste e armazenar seu resultado	
Requisitos Especiais:	
Nome	Descrição
5.1 Execução	Executa junto a geração de mutantes

2.6 RF 006 - Comparar resultado

Tabela 6 – Requisito 6

6. **Comparar Resultado**	
Compara o resultado da consulta original com cada consulta do banco de teste e atualiza seu estado	
Requisitos Especiais:	
Nome	Descrição
6.1 Execução	Executa junto a geração de mutantes
6.2 Atualizar Lista	Atualiza a lista da base de teste

2.7 RF 007 - Troca de estado

Tabela 7 – Requisito 7

7. **Troca de Estado**	
Habilita a troca de estado pelo usuário	
Requisitos Especiais:	
Nome	Descrição
7.1 Edição Manual	Habilita edição manual
7.2 Atualizar Lista	Atualiza a lista da base de teste

2.8 RF 008 - Calcular Escore de Mutação

Tabela 8 – Requisito 8

8. **Calcular Escore de Mutação**	
Realiza o cálculo do Escore de Mutação	
Requisitos Especiais:	
Nome	Descrição
8.1 Execução Automática	Executa automaticamente ao reconhecer mudança
8.2 Atualizar Lista	Atualiza a lista da base de teste

3 Casos de Uso

3.1 Conectar ao banco de Dados

- **Escopo:** Estabelecer a conexão com um banco de dados específico, o qual servirá de base para o teste de consultas.
- **Nível:** Objetivo de Usuário
- **Interessados e Interesses:**
  - Usuário: deseja estabelecer a conexão com seu banco de dados de forma simples e específica. - Sistema: necessita uma conexão válida para realização de suas outras operações.
- **Ator Principal:** Usuário.
- **Pré-condições:** O ususário acaba de iniciar o sistema.
- **Garantia de Sucesso:** Conexão ao banco de dados válido efetuada com sucesso, através de um logon capaz de realizar as devidas operações.
- **Requisitos Associados:** 1
- **Variações tecnológicas:** Diferentes sistemas de gerenciamento de bancos de dados possuem formas de conexão distintas, e se agregam a diferentes operações, podendo haver conflitos entre tipos diferentes de estruturas.

Fluxo Principal:

- **1.**O sistema é iniciado pelo usuário;
- **2.**O usuário entra com os dados referentes ao banco de dados;
- **3.**O sistema estabelece a conexão ao banco de dados;
- **4.**O sistema abre a tela principal.

Fluxo Alternativo:

- **3.** O sistema não estabelece a conexão ao banco de dados;

  3.1 O sistema emite uma mensagem de erro;

  3.2 O fluxo retorna ao passo 2;

3.2 Processar Consulta Original

- **Escopo:** Reconhecer a consulta introduzida pelo usuário e executá-la no banco de dados, armazenando o resultado obtido.
- **Nível:** Sub-função
- **Interessados e Interesses:**
  - Sistema: necessita uma conexão válida para realização de suas outras operações. - Usuário: validação da consulta que foi colocada como entrada.
- **Ator Principal:** Usuário.
- **Pré-condições:** Conexão efetiva com o banco de dados do usuário.
- **Garantia de Sucesso:** Armazenamento do resultado obtido pela consulta colocada como entrada pelo usuário.
- **Requisitos Associados:** 2 e 3
- **Variações tecnológicas:** A estrutura da consulta pode não estar correta, apresentando resultados não condizentes com as necessidades do usuário, antes mesmo da realização dos testes pelo sistema.

Fluxo Principal:

- **1.** A consulta é inserida pelo usuário;
- **2.** O sistema reconhece a consulta e executa no banco de dados conectado;
- **3.** O resultado obtido através da consulta é armazenado pelo sistema, para utilização posterior.

Fluxo Alternativo:

- **3.** A consulta retorna um resultado inválido ou apresentando algum tipo de erro;
  - **3.1** O sistema retorna uma mensagem de erro;
  - **3.2** O sistema retorna a tela principal;

3.3 Gerar Consultas Mutantes

- **Escopo:** Aplicar operadores de mutação na consulta original, gerando as consultas que servirão para a base de testes.
- **Nível:** Sub-função
- **Interessados e Interesses:**
  - Sistema: necessita de conexão e uma consulta original válida para realização de suas outras operações. - Usuário: entender os dados aplicados a consulta de teste gerada e suas diferenças para a original.
- **Ator Principal:** Usuário.
- **Pré-condições:** Entrada de consulta validada.
- **Garantia de Sucesso:** Criação da lista de consultas mutantes para visualização na tela do sistema.
- **Requisitos Associados:** 4
- **Variações tecnológicas:** Existem várias nomenclaturas e formas de categorizar diferentes operadores de mutação para consultas em bancos de dados.

Fluxo Principal:

- **1.**A consulta validada é recebida pelo sistema;
- **2.**O sistema aplica o operador de mutação com base na consulta recebida;
- **3.**São geradas as consultas mutantes e armazenadas na base de testes;
- **4.**As consultas são listadas e categorizadas.

Fluxo Alternativo:

- **2.** O operador não se aplica a consulta recebida;

  2.1 O sistema passa ao próximo operador;

  2.1 O fluxo retorna ao passo 2.

3.4 Executar as Consultas Mutantes

- **Escopo:** Executar as consultas da base de teste e atribuir seu resultado de acordo com a comparação com a consulta original.
- **Nível:** Sub-função
- **Interessados e Interesses:**
  - Sistema: comparar os resultados das consultas mutantes com o resultado da consulta original. - Usuário: identificação do estado da consulta de teste, para uma posterior análise.
- **Ator Principal:** Usuário.
- **Pré-condições:** Base de teste preenchida.
- **Garantia de Sucesso:** Atribuição dos valores "Vivo"ou "Morto"para as consultas mutantes da base de testes.
- **Requisitos Associados:** 4, 5 e 6
- **Variações tecnológicas:** Alguns operadores podem acabar não gerando consultas mutantes realmente efetivas, devido a diferentes formações de consultas. Isso pode comprometer resultados e análises posteriores. Ainda, existe ambiguidade entre diferentes tipos de operadores, podendo gerar consultas iguais para a análise.

Fluxo Principal:

- **1.** O sistema executa uma consulta da base de teste que ainda não possui classificação;
- **2.** O sistema armazena o resultado obtido pela consulta;
- **3.** O resultado é comparado com o armazenado da consulta original;
- **4.** O sistema atribui o estado de acordo com o resultado da comparação e passa para a próxima etapa.

Fluxo Alternativo:

- **4.** O sistema atribui o estado de acordo com o resultado da comparação.

  4.1 O fluxo retorna ao passo 1;

3.5 Calcular Escore de Mutação

- **Escopo:** Calcular automaticamente o Escore de Mutação de acordo com o dados presentes nos estados das consultas da base de teste.
- **Nível:** Sub-função
- **Interessados e Interesses:**
  - Sistema: calcular o Escore de Mutação. - Usuário: Identificação do Escore, para análises.
- **Ator Principal:** Usuário.
- **Pré-condições:** Classificação da base de teste.
- **Garantia de Sucesso:** Número entre 0 e 1 gerado de acordo com os dados provenientes da base de teste.
- **Requisitos Associados:** 8
- **Variações tecnológicas:** Existem outras fórmulas para o cálculo de índices, os quais possuem propriedades e funções parecidas.

Fluxo Principal:

- **1.** O sistema recolhe os dados da base de teste;
- **2.** O sistema realiza o cálculo e imprime na tela.

3.6 Trocar Estado de Consulta

- **Escopo:** Trocar manualmente o estado de uma consulta da base de teste, com o objetivo de aperfeiçoar o cálculo do Escore de Mutação.
- **Nível:** Objetivo de Usuário
- **Interessados e Interesses:**
  - Sistema: necessita estados para a realização do cálculo do Escore de Mutação. - Usuário: Aperfeiçoar o cálculo do Escore de Mutação e remover possíveis incompatibilidades no sistema.
- **Ator Principal:** Usuário.
- **Pré-condições:** Classificação da base de teste completa.
- **Garantia de Sucesso:** Capacidade de alterar qualquer estado de consulta da base de teste.
- **Requisitos Associados:** 7
- **Variações tecnológicas:** Só aceita a opção de mudar entre "Vivo", "Morto"e "Equivalente", os quais são as classificações utilizadas para cálculo do Escore de Mutação.

Fluxo Principal:

- **1.** A lista de consultas é preenchida;
- **2.** O usuário entra com dado referente ao estado da consulta;
- **3.** O sistema recalcula o Escore de Mutação.

-

3.7 Filtrar Mutantes por Operador

- **Escopo:** Omite a utilização de operadores não desejáveis, com o objetivo de aperfeiçoar o cálculo do Escore de Mutação.
- **Nível:** Objetivo de Usuário
- **Interessados e Interesses:**
  - Sistema: necessita dos elementos da tabela para a realização do cálculo do Escore de Mutação. - Usuário: Aperfeiçoar o cálculo do Escore de Mutação e remover possíveis incompatibilidades no sistema.
- **Ator Principal:** Usuário.
- **Pré-condições:** Classificação da base de teste completa.
- **Garantia de Sucesso:** Capacidade filtrar qualquer elemento da tabela de mutantes.
- **Requisitos Associados:** 8
- **Variações tecnológicas:** Varia de acordo com os operadores utilizados em cada consulta.

Fluxo Principal:

- **1.** A lista de consultas é preenchida;
- **2.** O usuário seleciona os operadores que deseja, e a tabela de mutantes é atualizada;
- **3.** O sistema recalcula o Escore de Mutação.

Fluxo Alternativo:

- **2.** O usuário seleciona os operadores que deseja, e a tabela de mutantes é atualizada;

  2.1 O usuário seleciona a opção de apresentar novamente os resultados omitidos, e a tabela de mutantes é atualizada;

  2.2 O fluxo retorna ao passo 3.

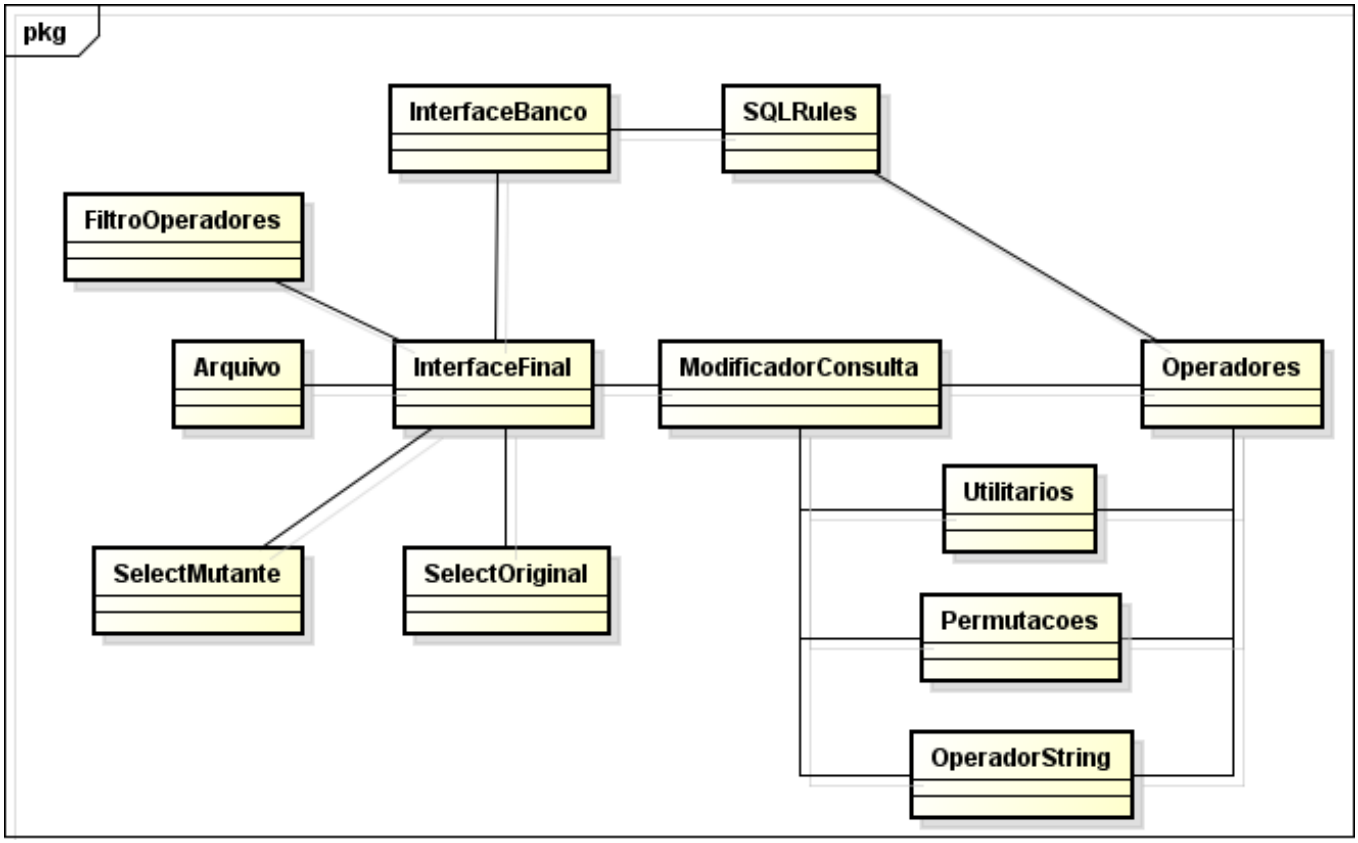
3.8 Utilizar Modo Comparação

- **Escopo:** Apresenta uma tela com os resultados da consulta original e da consulta mutante selecionada.
- **Nível:** Objetivo de Usuário
- **Interessados e Interesses:**
  - Sistema: necessita dos elementos da tabela para a apresentação dos resultados. - Usuário: Encontrar a diferença entre os resultados das consultas selecionadas.
- **Ator Principal:** Usuário.
- **Pré-condições:** Classificação da base de teste completa.
- **Garantia de Sucesso:** Apresentação de todos os dados referentes as consultas.
- **Requisitos Associados:** 8
- **Variações tecnológicas:** Varia de acordo com as tabelas presentes no banco de dados e as consultas inseridas.

Fluxo Principal:

- **1.** A opção Modo Comparação é selecionada;
- **2.** O usuário seleciona uma das consultas mutantes na tabela de mutantes;
- **3.** O sistema mostra os resultados referentes as consultas selecionadas.

4 Diagrama de Classes



Classe OperadorString: Classe auxiliar, a qual provê métodos de manipulação dos elementos de uma *String*. Seu objetivo é retornar os dados corretamente para a criação da consulta mutante. A *String* sempre está como parâmetro dos métodos.

- **Método firstBefore**: Dada uma posição p da *String* e um caractere c, o método retorna o número da aparição anterior mais próxima de c em relação a p.
- **Método firstAfter**: Dada uma posição p da *String* e um caractere c, o método retorna o número da aparição posterior mais próxima de c em relação a p.
- **Método replaceAt**: Dada uma sequencia de caracteres sc, um número n e uma sequencia de caracteres nova sc', o método retorna a *String* com sc' no lugar da enésima aparição de sc na *String*.
- **Método count**: Dada uma sequência de caracteres sc, o método retorna o número de aparições de sc na *String*.

- **Método getUntil**: Dada uma posição *p* e até três opções de sequencias de caracteres *sc*, o método retorna a sequencia de caracteres de *p* até a primeira aparição posterior de um *sc* na *String*.

- **Método getUntil**: Dada uma posição *p* e até duas opções de posições *p'*, o método retorna a sequencia de caracteres de *p* até a primeira aparição posterior de um *p'* na *String*.

- **Método nthOccurrence**: Dada uma sequencia de caracteres *sc* e um número *n*, o método retorna a posição da enésima aparição de *sc* na *String*.

- **Método indexOfIgnoreCase**: Dada uma sequencia de caracteres *sc*, o método retorna a posição da primeira aparição de *sc* na *String*, ignorando diferenças entre letras maiúsculas ou minúsculas, nos momentos de comparação.

- **Método replaceFirstIgnoreCase**: Dada uma sequencia de caracteres *sc* e uma sequencia de caracteres nova *st*, o método retorna a *String* com *st* no lugar da enésima aparição de *sc* na *String*, ignorando diferenças entre letras maiúsculas ou minúsculas nos momentos de comparação.

Classe Operadores: Classe que contém os algoritmos dos Operadores de mutação. Seu métodos são utilizados por outras classes. Destro destes métodos, ainda, é realizada a execução da consulta no banco de dados, para definir o estado de cada mutante.

- **Método alterSEL**: Procura por casos em que o operador SEL seja viável de execução. Caso encontre, gera seus respectivos mutantes.

- **Método anterJOI**: Procura por casos em que o operador JOI seja viável de execução. Caso encontre, gera seus respectivos mutantes.

- **Método alterROR**: Procura por casos em que o operador ROR seja viável de execução. Caso encontre, gera seus respectivos mutantes.

- **Método alterUOI**: Procura por casos em que o operador UOI seja viável de execução. Caso encontre, gera seus respectivos mutantes.

- **Método alterLCR**: Procura por casos em que o operador LCR seja viável de execução. Caso encontre, gera seus respectivos mutantes.

- **Método alterAOR**: Procura por casos em que o operador AOR seja viável de execução. Caso encontre, gera seus respectivos mutantes.

- **Método alterSEL**: Procura por casos em que o operador SEL seja viável de execução. Caso encontre, gera seus respectivos mutantes.

- **Método alterBTW**: Procura por casos em que o operador BTW seja viável de execução. Caso encontre, gera seus respectivos mutantes.
- **Método alterNOT**: Procura por casos em que os operadores iNOT, rNOT e NOT sejam viáveis de execução. Caso encontre, gera seus respectivos mutantes.
- **Método alterATR**: Procura por casos em que os operadores PATR, RATR e TATR sejam viáveis de execução. Caso encontre, gera seus respectivos mutantes.
- **Método alterVAL**: Procura por casos em que os operadores PVAL e TVAL sejam viáveis de execução. Caso encontre, gera seus respectivos mutantes.
- **Método alterTNT**: Procura por casos em que o operador TNT seja viável de execução. Caso encontre, gera seus respectivos mutantes.
- **Método alterTFA**: Procura por casos em que o operador TFA seja viável de execução. Caso encontre, gera seus respectivos mutantes.
- **Método alterLKE**: Procura por casos em que o operador LKE seja viável de execução. Caso encontre, gera seus respectivos mutantes.
- **Método getTables**: Retorna as tabelas existentes no banco de dados. Método auxiliar, usado em outros métodos da classe.
- **Método tableElements**: Preenche as estruturas auxiliares com os atributos e seus tipos de uma determinada tabela. Essas estruturas são utilizadas em outros métodos da classe.
- **Método getSelectOriginal**: Preenche estruturas auxiliares com os resultados obtidos através da execução da consulta principal no banco de dados. Essas estruturas são utilizadas por outros métodos da classe.
- **Método mutantState**: Retorna o estado de uma consulta mutante. Utiliza os resultados obtidos pelo método getSelectOriginal.

Classe InterfaceFinal: Interface principal do sistema. Nela o usuário poderá interagir com todas as funcionalidades do sistema.

- **Método createMutants**: Desencadeia a sequencia para a criação das consultas mutantes. Primeiramente utiliza a classe ModificadorConsulta e posteriormente a classe Operadores.
- **Método rowRemove**: Omite linhas nos resultados apresentados na tabela de mutantes. Esse método age conforme preferências do usuário.
- **Método rowInsertAll**: Insere todas as linhas que estavam omitidas.
- **Método warn**: Insere uma linha de alerta na tabela de mutantes, indicando que existem linhas ocultas.
- **Método clean**: Limpa o campo de consulta.
- **Método load**: Abre a janela de auxilio no carregamento de arquivos.
- **Método mutantTable**: Ao receber um clique em uma linha da tabela de mutantes, é aberta uma outra interface, para auxílio no entendimento da consulta mutante.

Classe Utilitários: Classe com métodos auxiliares, utilizados por outras classes, como por exemplo métodos de ordenação.

- **Método quickSort**: algoritmo de ordenação vetorial simples.
- **Método partition**: método auxiliar utilizado no quickSort.
- **Método attributeChecker**: confirma se um atributo é de um tipo específico, conforme a necessidade de uso.

Classe SelectOriginal: Classe interface, a qual apresenta os resultados de comparação de uma consulta mutante e da consulta original. A interface é aberta ao clicar na linha referente a consulta desejada.

- **Método theKiller**: método que marca a linha da primeira diferença encontrada entre os resultados da consulta mutante e da consulta original. O método só é realizado com sucesso caso o mutante esteja morto. Se não, é ignorado.
- **Método fillMutant**: insere os elementos na tabela referente a consulta mutante.
- **Método fillOriginal**: insere os elementos na tabela referente a consulta original.

Classe SelectMutante: Classe interface, a qual apresenta os resultados de uma consulta mutante selecionada. A interface é aberta ao clicar na linha referente a consulta desejada.

- **Método theKiller**: método que marca a linha da primeira diferença encontrada entre os resultados da consulta mutante e da consulta original. O método só é realizado com sucesso caso o mutante esteja morto. Se não, é ignorado.
- **Método fillMutant**: insere os elementos na tabela referente a consulta mutante;

Classe Permutações: Classe responsável por criar permutações para utilização dos operadores.

- **Método permuta**: dois métodos com o mesmo nome são apresentados na classe. Os dois trabalham em conjunto de forma recursiva, realizando o processo de permuta de posições.

Classe ModificadorConsultas:Classe responsável pelo *heal* da consulta, deixando-a pronta para o uso dos operadores.

- **Método setup**: Método complexo. Primeiramente separa cada tipo de consulta por seu operador primário. Para cada operador, então implementa regras de suporte que não danificam ou comprometem a consulta, com o objetivo de facilitar a leitura e manipulação no sistema.

Classe InterfaceBanco: Classe Interface responsável por armazenar o conteúdo de conexão ao banco de dados. A interface aparece ao iniciar o sistema e pode ser aberta a qualquer momento caso haja necessidade.

Classe Erro: Classe Interface genérica para apresentar algum possível erro. Futuramente a classe pode ser explorada para identificar os diferentes erros que podem ocorrer.

Classe Dados: Classe para armazenamento e acesso rápido a dados de necessidade global.

Classe Defeitos: Classe Interface utilizada para filtrar os resultados apresentados na tabela de mutantes.

- **Método buttonSC**: Marca/desmarca as opções que fazem parte de lacuna SC;
- **Método buttonNL**: Marca/desmarca as opções que fazem parte de lacuna NL;
- **Método buttonOR**: Marca/desmarca as opções que fazem parte de lacuna OR;

- **Método buttonIR**: Marca/desmarca as opções que fazem parte de lacuna IR;
- **Método buttonMOM**: Marca/desmarca as opções que fazem parte de lacuna MOM;
- **Método buttonOk**: Realiza as devidas alterações na tabela de mutantes.
- **Método buttonMarcar**: Marca/desmarca todas as opções;