UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

ANA LETÍCIA HERCULANO DA SILVA

Técnicas e Critérios de Testes em uma Aplicação de Banco de Dados Relacioanal

Estudo de Caso

Goiânia
2017

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

AUTORIZAÇÃO PARA PUBLICAÇÃO DE TRABALHO DE CONCLUSÃO DE CURSO EM FORMATO ELETRÔNICO

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: Técnicas e Critérios de Testes em uma Aplicação de Banco de Dados Relacioanal – Estudo de Caso

Autor(a): Ana Letícia Herculano da Silva

Goiânia, 03 de Julho de 2017.

Ana Letícia Herculano da Silva – Autor

Cássio Leonardo Rodrigues – Orientador

Dr. Edmundo Sérgio Spoto – Co-Orientador

ANA LETÍCIA HERCULANO DA SILVA

Técnicas e Critérios de Testes em uma Aplicação de Banco de Dados Relacioanal

Estudo de Caso

Trabalho de Conclusão apresentado à Coordenação do Curso de Sistemas de Informação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação.

Área de concentração: Teste de Software.

Orientador: Prof. Cássio Leonardo Rodrigues

Co-Orientador: Prof. Dr. Edmundo Sérgio Spoto

Goiânia
2017

ANA LETÍCIA HERCULANO DA SILVA

Técnicas e Critérios de Testes em uma Aplicação de Banco de Dados Relacioanal

Estudo de Caso

Trabalho de Conclusão apresentado à Coordenação do Curso de Sistemas de Informação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação, aprovada em 03 de Julho de 2017, pela Banca Examinadora constituída pelos professores:

Prof. Cássio Leonardo Rodrigues
Instituto de Informática – UFG
Presidente da Banca

Prof. Dr. Edmundo Sérgio Spoto
Instituto de Informática – UFG

Profa. Luciana de Oliveira Berretta
Instituto de Informática – UFG

Ana Letícia Herculano da Silva

Graduanda em Sistemas de Informação na UFG - Universidade Federal de Goiás. Durante sua graduação, participou do projeto de homologação de PAF-ECF (Programa Aplicativo Fiscal - Emissor de Cupom Fiscal), prestando consultoria em homologação e teste de software para empresas de todo o Brasil. Possui certificação CTFL (Certified Tester, Foundation Level) e atuou em empresas goianas de software, especificamente na área de teste de software. Possui experiência com sistemas de escrituração fiscal, documentos fiscais eletrônicos (NF-e, CT-e e afins), GRP (Government Resource Planning) e ITSM (IT Service Management). Atualmente trabalha como Analista de Testes em um sistema voltado para glosa de convênios hospitalares.

Dedico este trabalho ao meu querido Prof. Dr. Edmundo Sérgio Spoto, por ter me apresentado a área de testes, pela qual hoje sou apaixonada, e ao Danilo Guimarães, que é minha inspiração diária dentro da área de Tecnologia da Informação.

Agradecimentos

Agradeço primeiramente aos meus pais, Umbelina Luzia Herculano e José Pereira da Silva, que incontestavelmente sempre me nutriram de amor, carinho e atenção para que eu pudesse alcançar meus objetivos. Também aos meus irmãos, Gustavo e Júlia Pereira Herculano, por me esperarem acordados, a chegar em casa a noite após a aula para um abraço e beijo de boa noite, os quais eu amo incondicionalmente.

Agradeço imensamente ao meu marido Danilo Guimarães Justino Lemes pela força, paciência, companheirismo e amor depositados para que eu tenha superado todos os desafios. Sem seu apoio eu não conseguiria.

Deixo minha gratidão aos amigos que conquistei durante essa jornada: a Daniel Melo, que nunca mediu esforços para me ajudar, com sua sabedoria, paciência e sensatez admirável. Ao Bruno Nogueira, sempre disposto a colaborar com seu conhecimento e sua incrível agilidade. A Jéssica Milene, que amenizou o sofrimento dessa jornada com seu carisma contagiante. Vocês são história, que carrego comigo com bastante carinho.

Agradeço ao meu co-orientador Prof. Dr. Edmundo Sérgio Spoto, que com sua dedicação, experiência e conhecimento, soube me guiar com confiança de bons resultados, e aprendizados para uma vida inteira. Sem sua instrução não poderia ter alcançado este objetivo.

Agradeço a Deus pela sabedoria a mim concedida e a saudável vida de minha avó, Erci Dias Herculano, que sempre zelou pelo meu bem.

Talvez não tenha conseguido fazer o melhor, mas lutei para que o melhor fosse feito. Não sou o que deveria ser, mas Graças a Deus, não sou o que era antes.

Marthin Luther King Jr.,
Pastor, ativista político, notório líder do movimento dos direitos civis dos negros e Nobel da Paz de 1964.

Resumo

Silva, Ana Letícia Herculano da. **Técnicas e Critérios de Testes em uma Aplicação de Banco de Dados Relacioanal**. Goiânia, 2017. 45p. Relatório de Graduação. Instituto de Informática, Universidade Federal de Goiás.

A garantia de qualidade de um software é importante para agregação de valor aos clientes que o utilizam. Uma das fases para a obtenção dessa qualidade é a de Testes de Software. O testes de software pode ser aplicado em diferentes níveis e de diferentes técnicas. A proposta para aumentar a confiabilidade do software é aplicar técnicas de testes funcional ou baseado em erros ou estrutural com base nas dependência de dados e integridade das informações trabalhadas no Banco de Dados. Muitas vezes todas as técnicas podem ser aplicadas de forma complementar, já que nenhuma inclui a outra. Neste Trabalho foi aplicado um estudo de caso de um software real, onde é aplicado técnicas e critérios de testes funcional e teste de mutantes em uma Aplicação de Banco de Dados Relacional. São apresentados a contextualização teórica, um estudo de caso, os resultados obtidos e uma análise geral desses resultados.

Palavras–chave

Teste de Software, Banco de Dados, Teste Funcional e Teste de Mutantes

Abstract

Silva, Ana Letícia Herculano da. **<Work title>**. Goiânia, 2017. 45p. Relatório de Graduação. Instituto de Informática, Universidade Federal de Goiás.

The book is on the table.

Keywords

Software Testing, Database, Fuctional Testing, Mutant Testing

Sumário

Lista de Figuras

Lista de Tabelas

CAPÍTULO 1

Introdução

A onipresença de software ao redor do globo é indiscutível. Consumimos (e produzimos) em uma escala imensurável pelos mais otimistas de tempos passados. Temos contato

A qualidade de software aos poucos vem ocupando seu espaço no processo de desenvolvimento de um software. O testes de software em si, é a principal etapa para que a qualidade seja garantida. Foram realizados alguns trabalhos e pesquisa com base nas diversas técnicas de testes, importantes para a realização deste trabalho em si.

Souza [21] realizou uma pesquisa a qual foi a base para alguns critérios de testes aplicados neste trabalho. Ela propôs alguns critérios de testes em Banco de Dados Relacional baseados na especificação de requisitos através da UML. Para apoiar a aplicação dos conjuntos de critérios propostos, foi desenvolvido uma abordagem denominada mapeamento Conceitual de Informação.

1.1 Objetivos

O objetivo deste trabalho é mostrar diferentes técnicas de testes que podem ser aplicadas em um banco de dados relacional e apontar quais benefícios cada técnica pode trazer na detecção de defeitos. Mostrar que as duas técnicas utilizadas neste trabalho são complementares para que uma aplicação de banco de dados relacional possa alcançar níveis mais satisfatórios de qualidade. Os objetivos específicos são:

- Mostrar como cada técnica funciona;
- Mostrar como são gerados os planos de Testes de cada Técnica;
- Mostrar a execução de cada técnica em um BDR;
- Realizar uma análise de cada técnica, comparando os resultados obtidos;

1.2 Motivação

Grande parte dos testes aplicados atualmente levam em consideração apenas o código fonte do software ou o executável. Poucos se preocupam em realizar testes no Banco de Dados e nas partes do código que se relacionam com o Banco de Dados. O exemplo de ABDR utilizado neste trabalho, foi extraído de uma empresa que não realizam as técnicas de testes no BDR. Sendo assim foi com esse intuito gerar essa demonstração aplicando as técnicas de testes funcional e de mutantes aplicados em Banco de Dados, para assim manter a integridade de seus dados.

1.3 Organização do trabalho

Esse trabalho foi organizado da seguinte forma:

- No Capítulo 1 foi apresentado uma introdução sobre testes em Banco de Dados que são referenciados por vários trabalhos anteriores,
- No Capítulo 2 foram colocados as principais teorias que serão utilizadas sobre Teste de Software,
- No Capítulo 3 foram elencadas as características essenciais de um Banco de Dados, com foco em Banco de Dados Relacional,
- No Capítulo 4 foram apresentadas as técnicas de testes de Banco de Dados Relacional,
- No Capítulo 5 foram apresentados um estudo de caso,
- No Capítulo 6 são mostrados os planos de testes das duas técnicas utilizadas (Funcional e Mutantes),
- No Capítulo 7 são apresentados os resultados obtidos e, por fim,
- no Capítulo 8 são apresentadas as conclusões e trabalhos futuros.

1.4 Lista de abreviaturas e siglas

ABDR Aplicação de Banco de Dados Relacional

BD Banco de Dados

BDR Banco de Dados Relacional

CPU Central Processing Unit

DB Database

DBMS Database Management System

DDL Data Definition Language

DER Diagrama Entidade Relacionamento

DML Data Manipulation Language

FK Foreign Key

GPU Graphical Processing Unit

MER Modelo Entidade Relacionamento

RDBMS Relational Database Management System

SGBD Sistema Gerenciador de Banco de Dados

SQL Structured Query Language

CAPÍTULO 2

Teste de Software

2.1 Conceitos de Teste de Software

Com o passar do tempo, o desenvolvimento de software requer **níveis** de qualidade cada vez maiores, a estabilidade do produto em um momento crítico - por exemplo- ganha a confiança do cliente e do produto. Uma das garantias da qualidade esperada é o teste do software.

Embora durante o processo de desenvolvimento de software possam ser utilizadas técnicas, critérios e ferramentas a fim de evitar que erros sejam introduzidos no produto de software, a atividade de teste continua sendo de fundamental importância para eliminação de erros que persistem (Maldonado, 1991)[10]. Por isso, o teste de software é um elemento crítico para a garantia da qualidade do produto e representa a última revisão de especificação, projeto e codificação (Pressman, 2000) [14].

O teste de software consiste em uma verificação dinâmica de que o programa provê comportamentos esperados, dado um conjunto finito de casos de testes (SWEBOK V3, 2014) [23].

O processo de teste focaliza os aspectos lógicos internos do software, garantindo que todos os comandos sejam testados (teste estrutural), e os aspectos externos funcionais; isto é, conduz testes para descobrir erros e garantir que entradas definidas produzirão resultados reais, que concordam com os resultados exigidos (Pressman, 2002)[15].

Os testes podem ser aplicados em diferentes níveis através dos processos de construção e de manutenção. Os níveis podem ser distinguidos com base no propósito, ou no alvo do que se quer testar. O alvo do teste pode ser uma simples função (ou método), um módulo, um conjunto de módulos ou até mesmo o sistema inteiro.

O nível mais elementar de alvo de teste é o chamado teste unitário, ou teste de unidade. Consiste basicamente em checar o comportamento de uma função isoladamente. Tipicamente, testes unitários ocorrem com acesso ao código-fonte e com o suporte de ferramentas de debugging. Frequentemente, mas nem sempre é verdadeiro, os próprios desenvolvedores são responsáveis pela condução do teste unitário.

O próximo alvo são os testes de integração, onde um sistema é colocado em uma situação onde depende de um outro sistema, ou um conjunto de sistemas. Um exemplo básico de integração é checar se a camada de persistência de um software está se comunicando adequadamente com o software gerenciador do banco de dados.

E por último, o teste de sistema é o mais abrangente de todos e foca no comportamento do sistema como um todo, de forma que as integrações estejam coesas e que os resultados esperados sejam alcançados. Testes de sistema também são comumente utilizados para avaliar aspectos não-funcionais do sistema, tais como segurança, performance e tolerância à falhas. Interfaces externas, equipamentos de hardware e ambientes operacionais são geralmente exercitados durante esse tipo de teste.

Para a realização de testes é preciso criar casos de testes especificando as entradas de dados que devem ser implantadas no sistema, cada entrada gera uma determinada saída com declarações do que está sendo testado no sistema e os dados de testes são as entradas geradas automaticamente. A saída é analisada e comparada com o resultado esperado ou previsto por quem entende o que o sistema executa. Se apontar a presença de defeitos é depurado e feito correção do mesmo (Sommerville, 2008)[20].

2.2 Técnicas e Critérios de Teste de Software

É importante ressaltar que as técnicas de teste devem ser vistas como complementares e a questão está em como utilizá-las de forma que as vantagens de cada uma sejam melhor exploradas, possibilitando uma estratégia que leve a uma atividade de teste de boa qualidade, ou seja, eficaz e de baixo custo (Howden, 1987)[8]. Segundo Modesto (2006)[11], as técnicas e critérios de teste fornecem ao desenvolvedor uma abordagem sistemática e teoricamente fundamentada para se conduzir e avaliar a qualidade do teste de software. Dentre as várias técnicas existentes, pode-se destacar as técnicas de teste estrutural, funcional e baseada em erro.

Apresenta-se, a seguir, uma definição para cada uma das técnicas citadas acima (Pressman, 2002)[15]:

- Teste Funcional: Também chamado de caixa-preta ou comportamental, focaliza os requisitos funcionais do software. O teste caixa-preta permite ao engenheiro de software derivar conjuntos de condições de entrada que vão exercitar plenamente todos os requisitos funcionais de um programa.
- Teste Estrutural: Também chamado de caixa-branca, é um método de projeto de casos de teste que usa a estrutura de controle do projeto procedimental para derivar casos de teste. Usando métodos de teste caixa-branca, o engenheiro de software pode derivar casos de teste que: (1) garantam que todos os caminhos independentes

de um módulo tenham sido exercitados pelo menos uma vez; (2) exercitam todas as decisões lógicas em seus lados verdadeiros e falsos; (3) executam todos os ciclos nos seus limites e dentro de seus intervalos operacionais; e (4) exercitam as estruturas de dados internas para garantir a sua validade.

- Baseada em erros: O objetivo do teste baseado em erros em um sistema é projetar testes que tenham uma grande probabilidade de descobrir erros sutis. Como o produto ou sistema deve satisfazer a requisitos do cliente, o planejamento preliminar necessário para realizar este tipo de teste começa com o modelo de análise. O testador procura erros sutis (aspectos da implementação do sistema que podem resultar em defeitos), para determinar se esses erros existem, casos de teste são projetados para exercitar o projeto ou código.

Neste trabalho será abordada apenas a técnica de Teste Funcional, sendo que a mesma servirá como base para a proposta: Teste Funcional em Banco de Dados.

Formalmente, critérios de teste definem qual é o conjunto de elementos requeridos do software que devem ser exercitados (Rocha et al, 2001)[18].

De um modo bem específico, serve para direcionar a atividade de teste e tomar decisões relativas ao teste. Os critérios de teste podem ser usados de duas maneiras:

- Critério de seleção (também chamado de critério de geração): quando o critério é utilizado para selecionar um conjunto de dados de teste;
- Critério de adequação (também chamado de critério de cobertura): quando o critério é utilizado para avaliar a qualidade de um conjunto de dados de teste. (Modesto - 2006)[11]

2.3 Teste Funcional

O teste funcional avalia o sistema do ponto de vista do usuário, isto é, não considera a estrutura interna ou a forma de implementação do sistema, olhando-se o Software apenas através de suas interfaces. Sendo este o único tipo de teste possível quando não se dispõem do código-fonte do sistema. Os erros encontrados através do teste funcional são: erros de interfaces, funções incorretas, erros na estrutura de dados ou no acesso a dados externos, erros de desempenho e erros de iniciação ou finalização (Crespo, et al., 2000)[4].

Dadas às entradas, as saídas são examinadas, se as saídas não são aquelas previstas, pode-se dizer que o teste detectou com sucesso um problema no Software. O problema enfrentado pelos responsáveis dos testes é selecionar as entradas que tenham grande possibilidade de provocarem comportamento anômalo. Além de demonstrar a operacionalidade das funcionalidades do sistema, e adequação da saída em relação à

entrada o teste caixa preta serve também para demonstrar que a integridade da informação externa, por ex.: uma base de dados é mantida (Pressman, 2005)[16].

Na Técnica de teste funcional ss critérios mais conhecidos são Particionamento de Equivalência, análise do Valor Limite e Grafo Causa-Efeito. Esses critérios de testes podem ser aplicados em todas as fases de testes , e não levam em consideração a implementação.

Segundo Pressman (2006)[17], o critério Particionamento em Classes de Equivalência, é um método de teste caixa-preta, que divide o domínio de entrada por meio das condições de especificações de um determinado programa em classes de dados (classes de equivalência), das quais casos de testes são derivados. Este critério divide-se, portanto, em duas funções: identificar classes de equivalência; definir os casos de teste a partir das classes de equivalência.

De acordo com Rocha et al. (2004)[18], a identificação das classes de equivalência é feita derivando-se cada condição de entrada, obtida na especificação, para a criação de dois tipos de classes: válidas (entradas válidas esperadas do programa); e inválidas (outros possíveis valores associados à condição, ou seja, entradas não esperadas) (Myers, 1976)[12].

Pressman (2006)[17] apresenta algumas definições para as classes de equivalência. Se uma variável de entrada exigir o:

- Uso de intervalos: uma classe válida e duas inválidas são definidas, ou seja, um valor inválido seria bem abaixo do limite inferior e bem acima do limite superior;
- Uso de valor específico: uma classe válida e duas inválidas são definidas; ou seja, o próprio valor (válido) e um valor inferior e outro superior (inválido);
- Uso de um elemento de um conjunto: uma classe válida (dentro do conjunto) e uma inválida (fora do conjunto) são definidas; ? Uso de booleano: uma classe válida (V ou F) e uma inválida (diferente de V ou F) são definidas.

Após a identificação das classes válidas e inválidas, cada uma delas é enumerada para serem elaborados os casos de testes. Para Pressman (2006)[17] e Rocha et al. (2004), o Particionamento em Classes de Equivalência busca produzir casos de teste que descubram diversas classes de erros, podendo-se desta forma, reduzir o número total de casos.

Já a Análise do Valor Limite é um outro critério de teste caixa-preta que utiliza as abordagens do Particionamento em Classes de Equivalência como complemento, assim, tornando-o mais sistemático Pressman (2006)[17].

De acordo com Meyers (2004)[13], a experiência mostra que casos de teste que exploram condições limites têm uma maior probabilidade de encontrar defeitos. Tais condições correspondem a valores que estão exatamente sobre ou imediatamente acima ou abaixo dos limitantes das classes de equivalência.

O critério Análise do Valor Limite também exercita as condições de entrada, deriva-se também, casos de teste para o domínio de saída (Myers, 1976)[12].

De acordo com Pressman (2006)[17], as diretrizes para a Análise do Valor Limite são semelhantes ao critério de Particionamento em Classes de Equivalência, a seguir:

- Se uma condição de entrada especificar um intervalo limitado pelos valores a e b, casos de teste devem ser projetados com valores a e b, logo acima e logo abaixo de a e b;
- Se uma condição de entrada especificar diversos valores, são criados casos de teste para exercitar valores mínimos e máximos. Os valores logo abaixo e logo acima do mínimo e do máximo são testados;

Aplicação para as condições de saída, da primeira e segunda diretriz; ? Se as estruturas internas de dados do programa têm limites identificados, deve ser projetado um caso de teste para exercitar essa estrutura de dados no seu limite.

Com essas condições, o testador que aplica essas diretrizes, o teste será mais sistemático e completo, tendo uma maior probabilidade de encontrar defeitos.

2.4 Teste de Mutação

O Teste de mutação ou análise de Mutantes, como também é conhecido, é um critério de testes da técnica baseada em defeitos. Nessa técnica são utilizados defeitos típicos do processo de implementação de software para que sejam derivados os requisitos de testes.No caso de testes de mutação, o programa ou o script que está sendo testado é alterado diversas vezes incluindo um pequeno defeito, gerando um mutante. Desta forma cria-se um conjunto de programas alternativos ou mutantes, como se defeitos fossem inseridos no programa original. O trabalho do testador é escolher casos de testes que mostrem a diferença de comportamento entre o programa original e o programa mutante, quando existir uma entrada de dados. Assim como nas demais técnicas, cada mutante determina um subdomínio do domínio de entrada, formado por aqueles dados capazes de distinguir o comportamento do programa original e do mutante. A diferença, nesse caso, é que cada subdomínio está claramente relacionado com a capacidade de revelar um defeito específico. (Delamaro et al. 2007)[6]

Segundo Javier (Tuya et al 2006)[24] a análise de mutação consiste em gerar um grande número de programas alternativos chamados mutantes, cada um com um defeito simples que consiste em uma única mudança sintática no programa original.

Em (Cabeça, et al, 2008)[2] foram gerados operadores de mutação com base em 5 itens importantes que sempre estão presentes nas aplicações de Banco de Dados sendo:

Operador Matemáticos (+, -, *, /, %), Operadores de Comparação (<, >, =, <>), Operador Conjuntivos, Operadores Lógicos (`and`, `or`) e Operador de Negação (`not`).

Os mutantes são criados transformando o código fonte usando um conjunto de regras definidas (operadores de mutação) que são desenvolvidos para induzir alterações de sintaxe simples com base em erros que os programadores geralmente fazem ou para forçar metas de teste comuns. Cada mutante é executado com os dados de teste e quando produz uma saída incorreta (a saída é diferente da do programa original), o mutante é dito ser morto. Um caso de teste é dito ser eficaz se ele mata alguns mutantes que ainda não foram mortos por qualquer um dos casos de teste executados anteriormente. Alguns mutantes sempre produzem a mesma saída que o programa original, portanto nenhum caso de teste pode matá-los. Diz-se que estas mutações são mutantes equivalentes

CAPÍTULO 3

Banco de Dados

Atualmente todas as empresas que utilizam algum software para administração de seus negócios precisam manter seus dados armazenados e protegidos de alguma maneira. O acesso a esses dados muitas vezes necessitam de agilidade e confiabilidade, para quem o utiliza.

Elmasri e Navathe (2005)[7], explicam que Banco de Dados é uma coleção de dados com relacionados, ele é projetado, construído e são inseridos dados nele. Os dados são fatos que podem ser gravados e possui um significado implícito. O Banco de Dados pode ser de qualquer tamanho e sua complexidade pode ser variável. Um SGBD é uma coleção de programas que permite ao usuário criar, acessar e modificar esses programas. Ele facilita a definição, construção, manipulação e compartilhamento do BD entre aplicações e usuários diversos. Algumas funções importantes de um SGBD é a proteção do sistema com relação à falhas, funcionamento e tipos de acessos não autorizados e também a manutenção do BD (Elmasri; Navathe, 2005)[7]).

Em 1970 Edgar Frank Codd[3], da IBM, publicou um artigo sobre o modelo relacional baseado em álgebra relacional e cálculos matemáticos, mas somente na década de 80 esse modelo foi disponibilizado, era o Oracle e o SQL/DS (Structured Query Language/Data System) da IBM (International Business Machines). Hoje em dia existem outros tipos de modelo relacional da Oracle, IBM e Microsoft. O SQL/DS atualmente é o DB2 da IBM, o Oracle, o Access da Microsoft, entre outros (Elmasri; Navathe, 2005)[7].

Segundo Spoto(2000)[22] O modelo relacional representa uma base de dados como uma coleção de relações. Informalmente, cada relação pode ser considerada como uma tabela. Existem importantes diferenças entre relações e arquivos; a principal delas é a formação estruturada de seus componentes. Quando uma relação é tratada como uma tabela de valores, cada linha na tabela representa uma coleção de valores de dados relacionados. Esses valores podem ser interpretados como fatos que representam uma entidade do mundo real ou um relacionamento. O nome da tabela e os nomes das colunas são usados para ajudar a interpretar os valores em cada linha da tabela. Por exemplo, uma tabela com nome `EMPLOYEE` armazena valores relacionados aos empregados de uma empresa, onde cada linha contém os dados sobre um determinado empregado. Os

nomes das colunas `emp_id`, `emp_fname`, e `emp_lname` interpretam os valores de dados específicos para cada linha,baseados nos valores de cada coluna. Todos os valores de uma coluna devem ser do mesmo tipo de dados.

No modelo relacional existem quatro termos que são utilizados: domínio, tupla, atributo e relação. Alguns autores definem um domínio `D`, como um conjunto de valores atômicos, sendo a menor unidade de dado e individual no Modelo Relacional (Elmasri; Navathe, 2005)[7]. Portanto, um domínio é um conjunto de tais valores, todos do mesmo tipo, pois para cada atributo existe um conjunto de valores permitidos, que é o domínio desse atributo (Souza, 2008)[21].

Outra definição a se destacar é de uma relação. Uma relação `R (A1, A2, A3, ..., An)`, o elemento A1 refere-se aos atributos de uma relação, que é o nome do papel desempenhado por um domínio `D`. Uma relação `R (A1, A2, A3, ..., An)`, indicada por `r(R)` é um conjunto de várias tuplas `r{t1, t2, ..., tn}`. Cada `n`-tuplas é uma lista ordenada de `n` valores, cada valor é um elemento do domínio (A) ou um valor `null` (Elmasri; Navathe, 2005)[7].

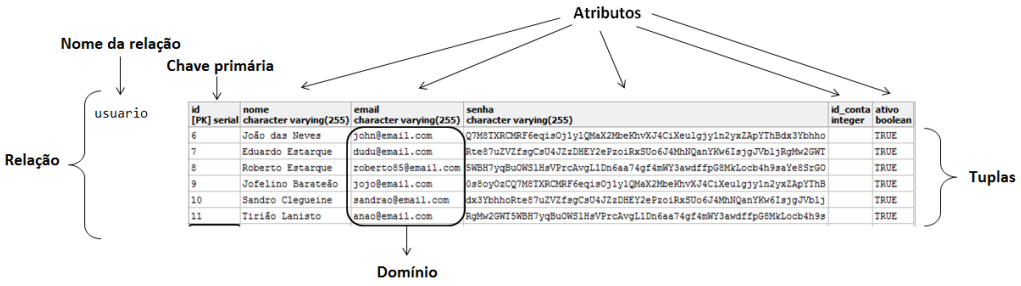


Figura 3.1: *Principais elementos de uma relação*

As chaves em um modelo de BD relacionais são importantes, pois são elas que distinguem unicamente uma tupla na relação, ou seja, nenhum par de tupla pode ter o mesmo valor para todos os atributos. São atributos que podem ser nomeados como super-chave, candidata, primária e estrangeira, e, caracterizam-se como uma propriedade da relação inteira, e não das tuplas individuais (Silberschatz et al., 2006)[19].

3.1 Restrições de Banco de Dados

3.1.1 Restrições de Domínio

3.1.2 Restrições de Integridade Semântica

3.1.3 Restrições de Integridade Referencial e de Chave Primária

Veja a Figura3.2:

Restrições	Objetivos	Módulos a serem assegurados
Estruturais de Relacionamentos	A restrição de razão de cardinalidade assegura o número máximo que a instância de relacionamentos das entidades pode participar. A restrição de cardinalidade mínima ou de participação assegura o número mínimo de instâncias de relacionamento em que cada entidade pode participar, ou seja, determina se a existência de uma entidade depende de outra entidade.	Relacionamento binário entre conjuntos de entidades (relações/tabelas) A e B.
Domínio	Assegurar se: os tipos de domínios de atributos são compatíveis (ou válidos); as consultas são relevantes; e se domínios podem conter valores nulos (*null*).	Domínios (tipos e semântica) de atributos de uma mesma relação/tabela. Ou de uma consulta de mais relações.
Chave Primária	Assegurar que: em duas tuplas distintas não podem ter o mesmo valor para todos os seus atributos chaves; e as chaves não podem conter valores nulos (*null*).	Atributos chaves de uma única relação/tabela.
Integridade Referencial e Chaves Estrangeiras	Assegurar que um conjunto de atributos em uma determinada relação apareça (referencia) outro conjunto de atributos em outra relação. Assim, pode-se ser mantida a consistência deste relacionamento.	Chave estrangeira em conjuntos de atributos em tuplas de relações/tabelas distintas (relacionamento entre duas relações) ou da mesma relação.
Integridade Semântica	Assegurar a semântica, tipos de dado, valores permitidos e transições de valores válidos para atributos.	Semântica dos atributos, em uma tabela ou relações entre tabelas.
Dependência Funcional	Assegurar o relacionamento de dependência funcional entre dois conjuntos de atributos X e Y a todo tempo.	Conjuntos de atributos dependentes em uma tupla de uma relação/tabela ou relacionamento entre elas..

Figura 3.2: *Tabela de critérios de Testes Funcionais para Banco de Dados propostos por Souza, 2008[21]*

3.2 Aplicações de Banco de Dados

3.3 Linguagem de Banco de Dados

CAPÍTULO 4

Técnicas de Testes aplicadas em Banco de Dados Relacional

4.1 Critérios de Teste Funcional em Aplicações de Banco de Dados

4.1.1 Critérios Para Testes Intra - Tabelas

4.1.2 Critérios Para Testes Inter-Tabelas

Veja a Figura 4.1:

Fluxo Funcional	Perspectivas dos critérios – Critério(s) de Teste Funcional	Sub-critérios	Exercícios dos ERs
Intra-Classe Inter-Classe	1. Baseados nas Estruturas de Relacionamento	*(c1) cardinalidades-máxima-intra-classe* *(c2) cardinalidades-máxima-inter-classe*	– cardinalidades máximas de relacionamentos.
		(c3) cardinalidades-mínimas-intra-classe *(c4) cardinalidades-mínimas-inter-classe*	– cardinalidades mínimas de relacionamentos.
	2. Baseados no Domínio de Atributos	*(c5) todos-os-valores-padrão*	– valores padrão do atributo (valores que iniciam um atributo).
		(c6) todos-os-domínios-atributos-intra-classe *(c7) todos-os-domínios-atributos-inter-classe*	– tipos de dados do atributo; – condições do atributo; – valores nulos e não nulos do atributo;
Sem Fluxo	3. Baseado nas Chaves Primárias	*(c8) todas-as-chaves-primárias*	– chaves primárias.
Intra-Classe Inter-Classe	4. Baseados na Integridade Referencial	*(c9) todas-as-chaves-estrangeiras-intra-classe* *(c10) todas-as-chaves-estrangeiras-inter-classe*	– cardinalidades de relacionamento; – chaves estrangeiras.
	5. Baseados na Integridade Semântica	*(c11) todas-as-semânticas-atributos-intra-classe* *(c12)-todas-as-semânticas-atributos-inter-classe*	– condições da semântica do atributo.
	6. Baseados na Dependência Funcional	*(c13) todos-os-atributos-dependentes-funcionalmente-intra-classe* *(c14) todos-os-atributos-dependentes-funcionalmente-inter-classe*	– condições de atributos dependentes.

Figura 4.1: *Tabela das perspectivas dos critérios de Souza (2008)[21] em comparação sobre os critérios propostos*

4.2 Técnica de Teste Mutante em Banco de Dados

CAPÍTULO 5

Estudo de Caso

O estudo de caso utilizado para exemplo é de um sistema de gestão comercial real em desenvolvimento pela empresa Oobj Tecnologia de Informação[5].

O sistema foi construído com a finalidade de controlar as licenças dos clientes e parceiros que utilizam outros produtos de software comercializados pela empresa. Será necessário realizar integrações com demais sistemas da empresa, tais como ERP, CRM, Financeiro, Operação, etc afim de garantir o cruzamento dos dados para tomada de decisão. Novos módulos comercializados serão ativados nesse sistema para que o cliente possa usufruir, bem como o bloqueio previsto em contrato em casos de inadimplência. O BD possui 14 (quatorze) tabelas que são:

- `empresa`,
- `filial`,
- `conta`,
- `plano`,
- `conta_plano`,
- `preco_modulo`,
- `matriz_empresa_modulo`,
- `usuario`,
- `grupo`,
- `usuario_grupo`,
- `grupo_permissao`,
- `schema_version`,
- `email_log` e
- `licencas_parametros`

O DER do software de Licenças é apresentado na Figura 5.1:

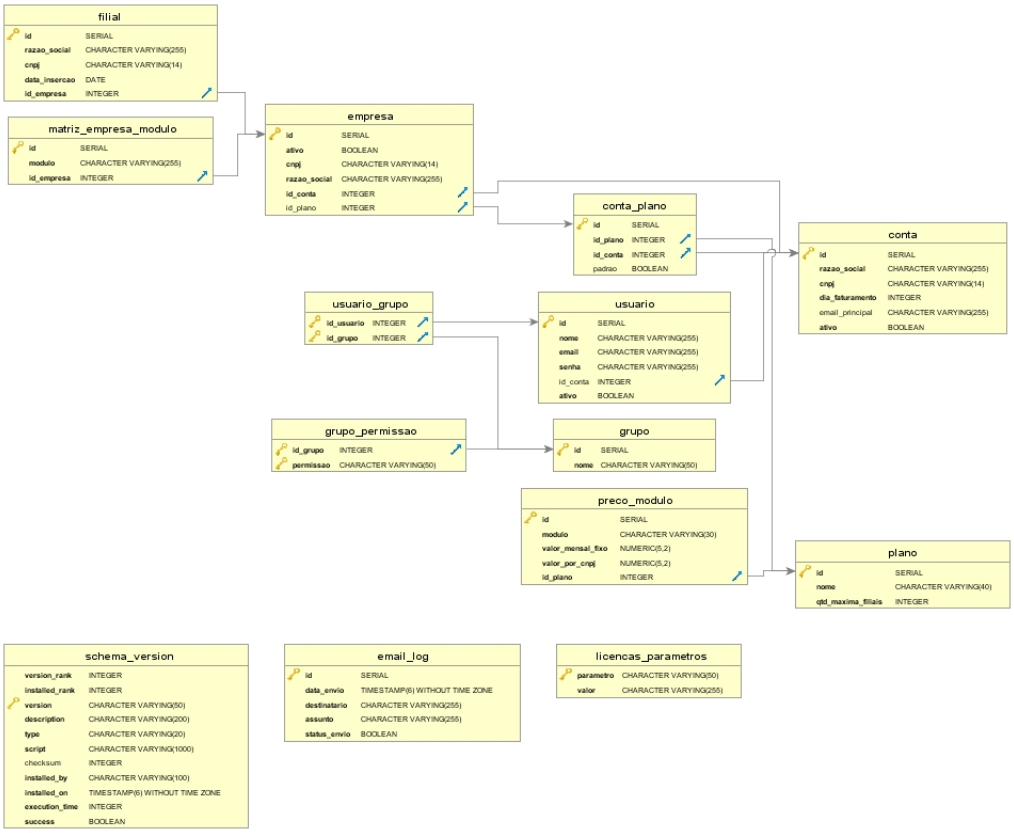


Figura 5.1: *Diagrama Entidade-Relacionamento do software objeto do estudo de caso*

De acordo com o DER apresentado na Figura 5.1, a tabela `empresa` está relacionada às empresas que irão receber a licença de uso. Geralmente são clientes ou outras empresas parceiras (software-houses), a tabela `filial` indica as filiais que uma empresa pode ter. Cada registro de filial está relacionada com um e apenas um registro de empresa.

Da mesma forma a tabela `conta` indica a conta na qual a empresa está inserida. Uma conta pode conter várias empresas, como por exemplo, quando um cliente possui um grupo de empresas distintas (CNPJ?s bases distintos), mas todas elas pertencem a mesma conta. Em contrapartida, uma empresa só pode pertencer a uma única conta.

A tabela `plano` determina a quantidade máxima de CNPJ?s que cada plano possuirá. Cada plano se adequa a realidade da empresa: micro, pequena, média, grande, multi-nacional etc.

Na tabela `conta_plano` fica a relação da conta com o plano na qual ela foi ?vinculada?, que tem o relacionamento direto com a tabela conta e com a tabela plano.

A tabela `preco_modulo` mostra qual valor que cada módulo tem para um determinado plano, para assim o pagamento ser efetuado de acordo com cada plano do cliente.

A tabela `matriz_empresa_modulo` relaciona os módulos que uma determinada empresa possui. Cada módulo pode ser um produto ou serviço diferente que a empresa

contratou.

A tabela `usuario` é o onde são armazenados os dados dos usuários do sistema.

A tabela grupo descreve um grupo de usuários. Geralmente, utilizado para agrupar usuários de um mesmo departamento ou função.

A tabela `usuario_grupo` indica em qual grupo o usuário está incluído.

A tabela `grupo_permissao` indica quais permissões de módulos que um grupo de usuários possui no sistema.

A tabela `schema_version` é a tabela para versionamento e controle de versão da base de dados. Ela é mantida automaticamente pelo framework Flyway[1].

A tabela `email_log` registra o histórico de alertas de emails enviados pelo sistema. Persiste informações tais como status do envio de email (sucesso ou falha), destinatário etc.

A tabela `licencas_parametros` armazena os parâmetros de configuração do sistema. É basicamente um mapa chave-valor (nome do parâmetro e seu respectivo valor)

A partir deste DER iremos realizar vários Casos de Testes Funcionais para exercitar alguns critérios de Teste Funcional de Banco de Dados de Souza[21]. Esses critérios serão apresentados no próximo capítulo. Bem como serão realizados alguns scripts de de Banco de dados para mostrar os efeitos do teste de mutantes em SQL, de forma a mostrar como o teste de Mutantes é realizado.

CAPÍTULO 6

Plano de Teste

6.1 Plano de Testes Baseado no Teste Funcional em BDR

6.1.1 Testes Intra - Tabelas

6.1.2 Testes Inter - Tabelas

CAPÍTULO 7

Resultados obtidos

7.1 Casos de Testes Intra-Tabelas

7.2 Casos de Testes Inter-Tabelas

7.3 Análise dos Resultados Obtidos na Técnica Funcional

De acordo com os testes realizados podemos observar que na técnica de Restrição de unicidade, o banco se manteve íntegro, todos os casos de testes foram executados com sucesso. A integridade referencial foi mantida para todos os casos de testes executados e , definidos no Plano de testes.

Já na Integridade de domínio tivemos algumas falhas, onde o banco não teve o comportamento esperado, como não existir um check em cada variável que tenha limites inferior e superior válidos dentro do escopo de cada tipo de dados por exemplos datas inválidas que não deveriam ser aceitas ou valores negativos quando a variável deve ser somente positiva tipo idade, essas regras foram definidas no Plano de testes . Essas falhas são comuns, e pelo percentual de erros, pode-se afirmar que o banco de dados, está 90% de qualidade atingida, apenas 10% foram defeitos ocorridos por esses motivos.

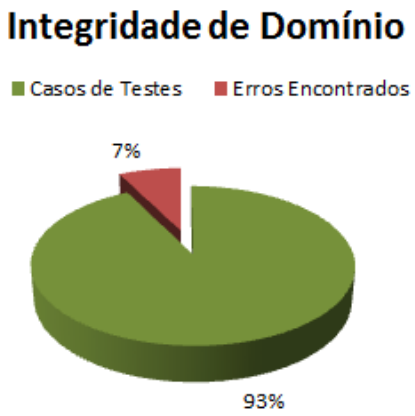
Na Tabela 7.1 podemos observar o resultados dos testes:

Tabela 7.1: *Comparação entre os 3 critérios utilizados*

Critério de Teste	Casos de Testes	Erros Encontrados	Porcentagem Eficaz
1 - Restrição de Unicidade	28	0	100%
2 - Integridade de Domínio	112	9	91,96%
3 - Integridade Referencial	15	0	100%
Total	**155**	**9**	**94,19%**

Nas Figuras 7.1(a), 7.1(b), 7.1(c) e 7.1(d), pode-se visualizar de uma melhor forma a porcentagem de casos de testes executados X os erros encontrados:

(a) *Restrição de Unicidade.*



(b) *Integridade de domínio.*

(c) *Integridade Referencial.*

(d) *Total.*

Figura 7.1: *Restrição de Unicidade (a), Integridade de Domínio (b), Integridade Referencial(c) e Total (d).*

7.4 Aplicação de Teste de Mutantes em Scripts de Banco de Dados (SQL)

Nesta seção será apresentado alguns scripts que serão aplicados os operadores de mutação descritos na seção Técnicas de Teste Mutante em Banco de Dados 4.2. Serão realizados 3 exemplos de scripts em SQL para criarmos os mutantes conforme os operadores de mutação dado em Tuya et al. (2006)[24]. Na Figura 7.2 mostra como os grupos de operadores de mutação são selecionados e quais tipos são aplicados neste trabalho.

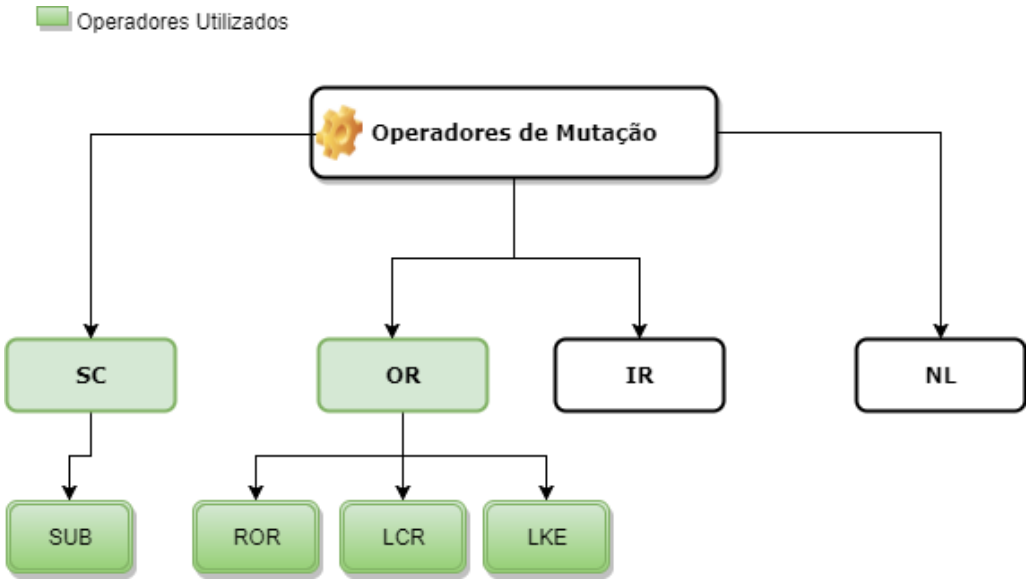


Figura 7.2: *Definição de Operadores Utilizados*

7.4.1 Caso 1

No Caso 1, serão aplicados os subtipos **ROR** e **LCR** do grupo **OR**, pré definidos na seção Técnicas de Teste de Mutante em Banco de Dados 4.2.

Query SQL: Exibe o

```
SELECT E.id, E.razao_social, F.id, F.razao_social, P.nome, CP.id, CP.id_plano, U.
FROM filial F, empresa E, conta_plano CP, plano P, usuario U, conta C
WHERE F.id_empresa = E.id and
E.id_plano = CP.id  and
P.id = CP.id_plano and
U.id_conta = C.id;
```

Exemplos dos Mutantes gerados no Caso 1:

M1.1 :

```
SELECT E.id, E.razao_social, F.id, F.razao_social, P.nome, CP.id, CP.id_plano, U.
FROM filial F, empresa E, conta_plano CP, plano P, usuario U, conta C
WHERE
F.id_empresa > E.id and
E.id_plano = CP.id  and
P.id = CP.id_plano and
U.id_conta = C.id;
```

M.1.5:

```
SELECT E.id, E.razao_social, F.id, F.razao_social, P.nome, CP.id, CP.id_plano, U.
FROM filial F, empresa E, conta_plano CP, plano P, usuario U, conta C
WHERE
F.id_empresa = E.id and
E.id_plano < CP.id  and
P.id = CP.id_plano and
U.id_conta = C.id;
```

M1.9

```
SELECT E.id, E.razao_social, F.id, F.razao_social, P.nome, CP.id, CP.id_plano, U.
FROM filial F, empresa E, conta_plano CP, plano P, usuario U, conta C
WHERE
F.id_empresa = E.id and
E.id_plano = CP.id  and
P.id <> CP.id_plano and
U.id_conta = C.id;
```

M1.13

```
SELECT E.id, E.razao_social, F.id, F.razao_social, P.nome, CP.id, CP.id_plano, U.
FROM filial F, empresa E, conta_plano CP, plano P, usuario U, conta C
WHERE
F.id_empresa = E.id or
E.id_plano = CP.id  and
P.id = CP.id_plano and
U.id_conta = C.id;
```

Na Tabela 7.2 mostra os resultados da execução de cada mutante gerado no Caso 1 mostrando que não houve nenhum mutante vivo. Ou seja todos mutantes deram resultados diferenciados em relação ao Script Original. Desta forma os erros plantados na geração de cada mutante mostrado nos exemplos acima e colocados no Apêndice, mostraram que no script original não existem esses respectivos defeitos, o que é muito importante e dá segurança para a aplicação.

7.4.2 Caso 2

No segundo exemplo,serão aplicados os subtipos **SUB** e **ROR**, do grupo **SC** e do grupo **OR**, respectivamente.

Query SQL Exibe o Valor pago por cada usuário ativo em cada plano;

```
SELECT u.nome, g.id, p.nome, pm.valor_mensal_fixo
FROM usuario U, grupo G, plano P, usuario_grupo UP, preco_modulo PM
WHERE U.id = UP.id_usuario and
G.id = UP.id_grupo and
PM.id_plano = P.id and
U.id in(SELECT distinct id from usuario WHERE ativo = true);
```

Exemplos dos Mutantes gerados no Caso 2:

M2.1

```
SELECT u.nome, g.id, p.nome, pm.valor_mensal_fixo
FROM usuario U, grupo G, plano P, usuario_grupo UP, preco_modulo PM
WHERE U.id > UP.id_usuario and
G.id = UP.id_grupo and
PM.id_plano = P.id and
U.id in(SELECT distinct id from usuario WHERE ativo = true);
```

M2.9

```
SELECT u.nome, g.id, p.nome, pm.valor_mensal_fixo
FROM usuario U, grupo G, plano P, usuario_grupo UP, preco_modulo PM
WHERE U.id = UP.id_usuario and
G.id = UP.id_grupo and
PM.id_plano <> P.id and
U.id  in(SELECT distinct id from usuario WHERE ativo = true);
```

M2.10

```
SELECT u.nome, g.id, p.nome, pm.valor_mensal_fixo
FROM usuario U, grupo G, plano P, usuario_grupo UP, preco_modulo PM
WHERE U.id = UP.id_usuario and
G.id = UP.id_grupo and
PM.id_plano = P.id and
U.id  not in(SELECT distinct id from usuario WHERE ativo = true);
```

Na Tabela 7.3 mostra o resultado dos mutantes gerados nos mutantes gerados no Caso 2 mostrados anteriormente. Nesse caso também ilustra que os defeitos colocados em cada mutante não está presente no Script original. No Caso 2 foram gerados 13 mutantes baseados nos operadores já comentados anteriormente. Os Mutantes são apresentados no Apêndice D.

Tabela 7.2: *Execução dos Mutantes Caso1*

Script Referenciado	Resultado
Original	56 Linhas
M1.1	63 Linhas
M1.2	217 Linhas
M1.3	280 Linhas
M1.4	63 Linhas
M1.5	217 Linhas
M1.6	280 Linhas
M1.7	49 Linhas
M1.8	175 Linhas
M1.9	224 Linhas
M1.10	32 Linhas
M1.11	136 Linhas
M1.12	168 Linhas
M1.13	19840 Linhas
M1.14	5160 Linhas
M1.15	10664 Linhas

Tabela 7.3: *Execução dos Mutantes Caso2*

Script Referenciado	Resultado
Original	288 Linhas
M2.1	3078 Linhas
M2.2	1254 Linhas
M2.3	4332 Linhas
M2.4	0 Linha
M2.5	152 Linhas
M2.6	1140 Linhas
M2.7	240 Linhas
M2.8	672 Linhas
M2.9	912 Linhas
M2.10	0 Linha
M2.11	0 Linha
M2.12	0 Linha
M2.13	0 Linha

7.4.3 Caso 3

No Terceiro exemplo é utilizado o subtipo **LKE** também pertencente ao grupo **OR**; A quantidade desses mutantes é reduzida devido combinação efetuada com 2 operadores, e a quantidade de dados no Banco.

Query SQL:

```
SELECT * from Usuario
WHERE nome Like 'A%';  // onde a letra A pode ser uma entrada de dado de uma ou m
```

Exemplos dos Mutantes gerados no Caso 3:

M3.1 :

```
SELECT * from Usuario
WHERE nome Like 'A_';
```

M3.2

```
SELECT * from Usuario
WHERE nome Like 'a%';
```

M3.3

```
SELECT * from Usuario
WHERE nome Like 'a_';
```

M3.4:

```
SELECT * from Usuario
WHERE nome Like 'C%';
```

M3.5:

```
SELECT * from Usuario
WHERE nome Like 'C_';
```

Na Tabela 7.4 mostra o resultado dos mutantes gerados nos mutantes gerados no Caso 3 mostrados anteriormente. Nesse caso também ilustra que os defeitos colocados em cada mutante não está presente no Script original. No Caso 3 foram gerados 5 mutantes baseados nos operadores já comentados anteriormente. Os Mutantes são apresentados no Apêndice D.

7.5 Análise dos Resultados obtidos em Testes de Mutante

Pode-se observar que o teste de mutantes não tem efeito direto como a técnica de testes funcional possui no Banco de Dados (no projeto do Banco), já o teste de mutante contribui em verificar que os scripts de SQL usados nas aplicações, estão corretos, não possuindo nenhum defeito gerado pelos operadores de mutação. Nos Mutantes gerados em nosso exemplo não ocorreram defeitos nos scripts, ou seja, não houve nenhum mutante equivalente (quando o mutante gera a mesma saída do que o script original).

Na Tabela 7.5 é apresentada a relação de quantidade de mutantes gerados por cada grupo de cada caso (scripts) utilizado como exemplo para ilustrar essa prática.

Tabela 7.4: *Execução dos Mutantes Caso3*

Script Referenciado	Resultado
Original	1 Linha
M3.1	0 Linha
M3.2	2 Linhas
M3.3	0 Linha
M3.4	6 Linhas
M3.5	0 Linha

Tabela 7.5: *Relação de Mutantes gerados nos exemplos*

Operador	Caso 1	Caso 2	Caso 3	Total
OR	15	9	5	29
SC	0	1	0	1
				30

CAPÍTULO 8

Conclusão e trabalhos futuros

Referências Bibliográficas

[1] BOXFUSE GMBH. **Flyway**. https://flywaydb.org/. Accessado em: 2016-12-03.

[2] CABEÇA, A. G.; LEITÃO-JR, P. J. M. **Análise de Mutantes em Aplicações SQL de Banco de Dados**, 2008.

[3] CODD, E. F. **A Relational Model of Data for Large Shared Data Banks**, 1970.

[4] CRESPO, . **?**, 2000.

[5] DA INFORMAÇÃO, O. T. **Oobj Tecnologia da Informação**. http://www.oobj.com.br/. Accessado em: 2017-12-03.

[6] DELAMARO, M. E; MALDONADO, J. C. J. M. **Teste de mutação**. In: *Introducao ao Teste de Software*, p. 77–118, Rio de Janeiro, 2007. Elsevier.

[7] ELMASRI, RAMEZ; NAVATHE, S. **Sistemas de banco de dados**. São Paulo, sixth edition, 2005.

[8] HOWDEN. **?**

[9] KORTH, HENRY F.; SILBERSCHATZ, A. S. S. **Sistema de banco de dados**. Makron Books, São Paulo, 1999.

[10] MALDONADO, J. C. **?**, 1991.

[11] MODESTO, L. R. **Teste funcional baseado em diagramas da uml**. Dissertação de mestrado em ciência da computação, Centro Universitário Eurípides de Marília (UNIVEM), Marília, São Paulo, Brasil, 2006.

[12] MYERS, G. J. **?** ?, ?, 1976.

[13] MYERS, G. J. **The art of software testing**. John Wiley & Sons, Inc, Hoboken, New Jersey, 2004.

[14] PRESSMAN, R. **Software Engineering?** 2000.

[15] PRESSMAN, R. **Software Engineering?** 2002.

[16] PRESSMAN, R. **Software Engineering?** 2005.

[17] PRESSMAN, R. **Software Engineering?** 2006.

[18] ROCHA, T. **?**, 2001.

[19] SILBERSCHATZ, J. **Silberschatz ?**, 2006.

[20] SOMMERVILLE, I. **Engenharia de software**. Pearson, São Paulo, 2008.

[21] SOUZA, J. P. D. **Teste funcional em aplicação de banco de dados relacional baseado nos diagramas da uml**, 2008.

[22] SPOTO, E. S. **Teste estrutural de programas de aplicação de banco de dados relacional**. Tese de doutorado em engenharia elétrica, Faculdade de Engenharia Elétrica e de Computação da Universidade Estadual de Campinas - FEEC/UNICAMP, Campinas, 2000.

[23] SWEBOK. **SWEBOK V3 Guide - Software Engineering Body of Knowledge**, 2014.

[24] TUYA, J.; SUÁREZ-CABAL, M. R. C. **Mutating database queries, information and software technology**, 2006.

APÊNDICE A

Script de criação da base de dados

Apêndicess são iniciados com o comando `\apendices`.

APÊNDICE B

Script de Inserção inicial no Banco

Texto do Apêndice B.

Apêndices são iniciados com o comando `\apendices`.

APÊNDICE C

Scripts dos casos de testes

Texto do ApÃªndice C.

APÊNDICE D

Scripts dos Testes de Mutantes

Texto do ApÃªndice C.