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.

Dedicatória do trabalho a alguma pessoa, entidade, etc.

# Agradecimentos

<Texto com agradecimentos àquelas pessoas/entidades que, na opinião do autor, deram alguma contribuíção relevante para o desenvolvimento do trabalho.>

<Epígrafe é uma citação relacionada com o tópico do texto>

**<Nome do autor da citação>**,
*<Título da referência à qual a citação pertence>*.

# Resumo

Silva, Ana Letícia Herculano da. **Técnicas e Critérios de Testes em uma Aplicação de Banco de Dados Relacioanal**. Goiânia, 2017. 33p. 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. 33p. 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 [19] 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

Objetivos

## 1.2 Motivação

Motivacao

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)[8]. 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) [12].

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) [21].

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)[13].

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)[18].

## 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)[6]. Segundo Modesto (2006)[9], 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)[13]:

- 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)[16].

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)[9]

## 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)[3].

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)[14].

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)[15], 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)[16], 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)[10].

Pressman (2006)[15] 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)[15] 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)[15].

De acordo com Meyers (2004)[11], 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)[10].

De acordo com Pressman (2006)[15], 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)[4]

Segundo Javier (Tuya et al 2006)[22] 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)[1] 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)[5], 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 sistema gerenciador de banco de dados (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)[5]).

Em 1970 Edgar Frank Codd[2], 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)[5].

Segundo Spoto(2000)[20] 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)[5]. 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)

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 rt1,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)[5].

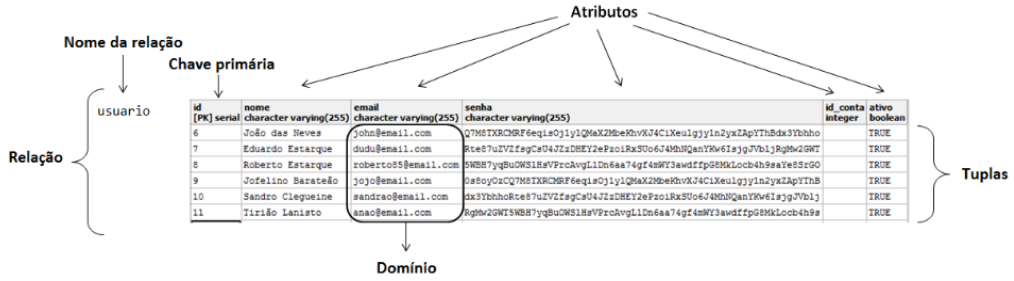


**Figura 3.1:** *Principais elementos de uma relação*

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)[17].

## 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

## 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

## 4.2 Técnica de Teste Mutante em Banco de Dados

CAPÍTULO 5

# Estudo de Caso

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

CAPÍTULO 8

# Conclusão e trabalhos futuros

# Referências Bibliográficas


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

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

[3] CRESPO. **Crespo**, 2000.

[4] 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.

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

[6] HOWDEN. **Howden**.

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

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

[9] 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.

[10] MYERS, G. J. **The art of software testing**. John Wiley & Sons, Inc, Hoboken, New Jersey, 1976.

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

[12] PRESSMAN, R. **Software Engineering**. 2000.

[13] PRESSMAN, R. **Software Engineering**. 2002.

[14] PRESSMAN, R. **Software Engineering**. 2005.

[15] PRESSMAN, R. **Software Engineering**. 2006.

[16] ROCHA, T. **Rocha**, 2001.

[17] SILBERSCHATZ, J. **Silberschatz**, 2006.

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

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

[20] 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.

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

[22] 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.