Análise de Mutantes em Aplicações SQL de Banco de Dados

Andrea G. Cabeça[1], Plínio S Leitao-Junior[2], Mario Jino[1]

[1]Faculdade de Engenharia Elétrica e de Computação - Universidade Estadual de Campinas (UNICAMP) – Campinas, SP - Brasil

[2]Instituto de Informática – Universidade Federal de Goiás - Goiânia, GO - Brasil.

{andrea.cabeca@kpocs.com.br, plinio@inf.ufg.br, jino@dca.fee.unicamp.br}

Abstract. *Testing database applications is crucial for ensuring high quality software as undetected faults can result in unrecoverable data corruption. SQL is the most widely used interface language for relational database systems. Our approach aims to achieve better tests by selecting fault-revealing databases. We use mutation analysis on SQL statements and discuss two scenarios for applying strong and weak mutation techniques. Experiments using real applications, real faults and real data were performed to: (i) evaluate the applicability of the approach, and (ii) compare fault-revealing abilities of input databases.*

Resumo. *O teste de aplicações de banco de dados é crucial para assegurar a alta qualidade do software, pois defeitos não detectados podem resultar em corrupção irrecuperável dos dados. SQL é a mais amplamente utilizada interface para sistemas de banco de dados. Nossa abordagem visa a alcançar melhores testes pela seleção de bases de dados reveladoras de defeitos. Usamos a análise de mutantes em comandos SQL e discutimos dois cenários para aplicar as técnicas de mutação forte e fraca. Experimentos usando aplicações reais, defeitos reais e dados reais foram conduzidos para: (i) avaliar a aplicabilidade da abordagem; e (ii) comparar as capacidades de revelação de defeitos de bases de dados de entrada.*

1. Introdução

A incapacidade humana para executar tarefas e comunicar-se com perfeição é um dos fatores motivadores para que o desenvolvimento de software seja acompanhado por esforços direcionados à garantia de qualidade [Deutsch, 1979]. Da perspectiva de software, podemos dizer que eles são expressivos tanto no meio industrial quanto no acadêmico. Em cada fase do processo de desenvolvimento, encontramos dezenas de metodologias e guias para auxiliar o profissional na busca da qualidade, na direção da confiabilidade e da corretude do software. Nesse contexto, o teste de software aparece como um elemento crítico para a busca da qualidade.

A análise de mutantes ou teste de mutação [DeMillo, 1978] é uma técnica (critério) que consiste em inserir pequenas alterações em um programa em teste, resultando em programas ligeiramente diferentes do original, chamados mutantes. As

alterações são determinadas por meio de regras, denominadas *operadores de mutação*, que visam a caracterizar um determinado modelo de defeitos. Dada uma entrada, o comportamento do programa original P é comparado com o de cada programa mutante M. Em tempo de execução, se o comportamento de M for distinto do de P, diz-se que M está "morto". Se, para um conjunto de caso de testes T, M apresentar o mesmo comportamento de P, diz-se que M está "vivo" e uma análise adicional deverá ser feita para determinar: (i) se M é equivalente a P; ou (ii) se T não é bom o suficiente para "matar" M. Após a execução dos mutantes, o *escore de mutação EscM* é calculado; *EscM* é a razão entre a quantidade de mutantes mortos e a quantidade de mutantes não equivalentes vivos (gerados menos os equivalentes); varia entre 0 e 1 e dá uma indicação da qualidade dos casos de teste escolhidos para exercitar o programa [DeMillo, 1980].

De acordo com Kapfhammer e Soffa (2003), uma aplicação de banco de dados é um programa cujo ambiente sempre contém uma ou mais bases de dados. Em geral, a Aplicação de Banco de Dados Relacional (ABDR) pode ser composta por vários programas codificados em diversas linguagens como C e Java, sendo sua principal característica a manipulação de dados por comandos que acessam as bases de dados controladas por um Sistema Gerenciador de Banco de Dados (SGBD) [Spoto, 2000]. No âmbito da ABDR, as bases de dados integram o conjunto de dados de entrada dos testes; assim, um caso de teste é formado por: (i) dados de entrada para o programa; (ii) conjunto de instâncias de bases de dados; e (iii) a saída esperada e instâncias esperadas das bases para estes dados de entrada.

A SQL (Structured Query Language) é uma linguagem padrão utilizada para a comunicação com um banco de dados relacional, representando a forma de acesso das aplicações às bases de dados; é a linguagem mais usada pelos SGBDs comerciais. Técnicas de teste para programas convencionais e até mesmo para a geração de bases de dados têm sido propostas, implementadas e avaliadas, tais como em Chan e Cheung (1999) e Chays e Deng (2003), mas ainda há carência de abordagens sistemáticas de testes na busca pela melhoria da qualidade das aplicações de banco de dados que usam a SQL [Tuya et al., 2007; Chan et al., 2005 e Leitao-Junior et al., 2005].

Este trabalho explora o teste de mutação para a SQL no contexto de ABDRs, estabelecendo operadores de mutação para a linguagem e, sobretudo, experimentando cenários próprios de aplicações reais. Tais cenários são complementares e representam etapas pertinentes à atividade de teste, a saber: (i) teste isolado de comandos SQL; e (ii) teste de seqüências SQL geradas a partir de aplicações de banco de dados. Em ambos os cenários, são empregados operadores de mutação que constituem uma evolução dos propostos na literatura e que representam defeitos comumente inseridos por programadores SQL. A análise empírica dos operadores em tais cenários tem o suporte de uma ferramenta, a qual cobre a geração de mutantes e a avaliação do teste, permitindo a análise da eficácia dos conjuntos de teste (bases de dados de entrada).

A Seção 2 apresenta definições relativas ao teste de mutação aplicado a SQL. A definição dos cenários de mutação forte e de mutação fraca é apresentada na Seção 3. Os operadores de mutação são introduzidos na Seção 4. A Seção 5 explora as características da ferramenta desenvolvida para o suporte à abordagem. A Seção 6 descreve os experimentos que objetivam a investigação da aplicabilidade e da

capacidade de detecção de defeitos dos operadores de mutação e o seu comportamento nos cenários estudados. A Seção 7 discute brevemente alguns trabalhos relacionados, que também exploram o teste de mutação em aplicações SQL. A Seção 8 apresenta as conclusões e contribuições do trabalho.

2. Teste de Mutação Aplicado a SQL

Nesta seção são introduzidos conceitos pertinentes à análise de mutantes nas aplicações que utilizam a SQL. A Subseção 2.1 apresenta o conceito de Unidade de Programa com SQL e suas abordagens para teste; tal conceito é útil na introdução dos cenários de teste para o experimento. A Subseção 2.2 estende as definições do teste de mutação para aplicações que utilizam a linguagem de manipulação SQL.

2.1 Unidade de Programa com SQL

Uma unidade de programa é dita *Unidade de Programa com SQL (UPS)*, se possuir comandos da SQL para o acesso à base de dados. Para a avaliação de UPSs, identificamos duas abordagens de teste:

(Ti) o resultado do teste é a saída *O* da UPS e o estado final DB_f (instância de saída) da base de dados;

(Tii) o resultado do teste é a saída *O* da UPS e a seqüência *S* de comandos da SQL, tal que a aplicação de *S* à instância de entrada da base de dados produz o estado final DB_f da base de dados.

Ambas as abordagens podem ser aplicadas ao teste de UPSs. Na segunda abordagem (Tii), vale ressaltar que a seqüência *S* é oriunda da execução da UPS a partir do dado de teste: valores de entrada e instância de banco de dados.

2.2 Definições

Esta subseção estende as definições do teste de mutação para aplicações que utilizam a linguagem de manipulação SQL. Considere *P* uma unidade de programa com SQL.

Definição 1: O ***Dado de Teste*** *ID* para a execução de *P* é definido como $ID = \{ I , DB_i \}$, tal que: *I* representa os dados de entrada de *P*; e DB_i é a instância inicial da base de dados.

Definição 2: O ***Caso de Teste*** *TC* para a execução de *P* é definido como $TC = \{ ID , O , S \}$ ou $TC = \{ ID , O , DB_f \}$, tal que: *ID* constitui o dado de teste para *P*; *O* é a saída da execução de *P* a partir de *ID*, *S* é a seqüência de comandos SQL oriunda da execução de *P* a partir de *ID*; e DB_f é a instância final da base de dados obtida aplicando-se a seqüência *S* à instância Db_i em *ID*.

Definição 3: O ***conjunto de operadores de mutação*** aplicáveis à seqüência *S* de comandos SQL é definido como $MO = \{ mo_1, mo_2, ..., mo_x \}$, $x >= 1$, em que cada operador $mo_i \in MO$, $1 \leq i \leq x$, pode ser aplicado a algum comando de *S*.

Definição 4: A partir de *S* e *MO*, pode-se derivar o ***conjunto de mutantes*** $M = \{mo_{11}, mo_{12}, ..., mo_{i1}, mo_{i2}, ..., mo_{ij}, ..., mo_{x1}, mo_{x2}, ...\}$, tal que: (1) mo_{ij} é o j-ésimo mutante obtido a partir do operador de mutação $mo_i \in MO$; e (2) $|M| \geq x$, sendo *x* a cardinalidade do conjunto *MO*.

Definição 5: A partir do conjunto M, é derivado o ***conjunto de mutantes possíveis*** M*, tal que $|M^*| \leq |M|$. Um mutante é possível se for sintaticamente válido.

Ao aplicar um dado de teste *ID* à unidade *P*, é obtida a seqüência *S* de comandos SQL. Da seqüência *S* é derivado o conjunto $MO = \{ mo_1, mo_2, ..., mo_x \}$ de operadores de mutação, donde se constroem *x* mutantes, $x >= 0$. Aplica-se DB_i a cada $mo_i \in MO$ e determina-se o número *y* de mutantes mortos, $y \leq |MO|$. O *escore de mutação EscM = y / |MO|* determina o percentual de mutantes mortos dentre os mutantes existentes. Se *EscM=1*, significa que todos os mutantes foram mortos (situação ideal). Dado o conjunto *C* de casos de teste, deve-se selecionar o subconjunto *C** de *C* que é composto dos casos de teste de C que resultaram nos maiores valores para o *escore EscM*.

3. Cenários

Nesta seção, são introduzidos os cenários dos experimentos para o teste de mutação aplicado a Unidades de Programa com SQL. O foco são o teste e a análise de seqüências de comandos SQL, ou de componentes dessas seqüências.

3.1 Cenário 1

Neste cenário, seqüências SQL são analisadas como um átomo, caracterizando o aspecto de Mutação Forte [Woodward, 1993] para o teste.

Uma seqüência *S'* de comandos SQL é obtida pela aplicação dos dados de teste *ID'* (dados de entrada e instância inicial da base de dados) a uma UPS. A seqüência *S'* e um conjunto de mutantes, os quais foram gerados a partir de *S'*, são executados usando a instância de entrada da base de dados em *ID'*. A instância de banco de dados produzida pela execução de *S'* é comparada com as instâncias de banco de dados oriundas das execuções dos mutantes de *S'*. O resultado dessa comparação produz o *escore de mutação EscM*, referente ao dado de teste *ID'*.

O *escore de mutação EscM* obtido demonstrará o valor relativo do dado de teste *ID'* em relação a outros dados de teste para o teste da UPS. O foco da avaliação é a comparação de dados de teste, onde o teste de mutação os avaliará em sua eficiência sob diversos aspectos, tais como a capacidade de revelar classes de defeitos e a cobertura de comandos SQL a serem exercitados. Vale ressaltar que qualquer variação na instância de entrada da base de dados caracterizará um outro dado de teste, denominado *ID''* (cuja aplicação à UPS produz a seqüência *S''* de comandos SQL, possivelmente distinta de *S'*), o qual é passível de comparação com *ID'*.

3.2 Cenário 2

Neste cenário, seqüências de comandos SQL são também consideradas como objetos do teste, mas são analisadas em termos de seus componentes, caracterizando o aspecto de Mutação Fraca [Woodward, 1993] para o teste.

Geralmente, um componente da seqüência é constituído por um único comando de acesso à base de dados, tais como SELECT e INSERT, mas pode ser composto por um grupo de comandos que denotem algum aspecto particular da seqüência (por exemplo, um subcaminho ou uma classe de defeitos).

Os mutantes são gerados a partir da aplicação de operadores de mutação aos comandos do componente em questão. O *escore de mutação EscM* indica o valor relativo entre instâncias de entrada da base de dados para o teste do componente. Vale ressaltar que esta abordagem é pertinente à comparação de instâncias de base de dados para: (i) o teste de uma UPS particular, pois apóia a seleção de bases potencialmente reveladoras de defeito pela análise de componentes de seqüências; (ii) o teste de seqüências SQL oriundas de outras fontes, tais como *scripts* SQL de uso geral, *triggers* e *stored procedures*.

4. Operadores de Mutação

A definição dos operadores de mutação em SQL visa à avaliação de instâncias de entrada de bases de dados, com respeito à sua capacidade em revelar defeitos, que estejam presentes no esquema da base de dados ou nos comandos SQL de acesso à base de dados.

Nosso objetivo foi cobrir todos os comandos pertencentes ao padrão SQL 3 [ANSI, 2003], cujas sintaxes sejam passíveis de mutação. Embora alguns estudos, como o de Pönighaus (1995), demonstrem que o comando SELECT é o mais utilizado nas aplicações comerciais, não há registros sobre a relevância dos demais comandos. Ou seja, isso não significa que nos demais comandos da linguagem SQL não existam defeitos, nem que os defeitos nos demais comandos sejam pouco freqüentes ou que não sejam tão graves quanto os defeitos nos comandos de consulta. Este trabalho define operadores de mutação para a maioria dos comandos da SQL, em adição aos comandos de consulta e atualização de dados.

Os operadores foram divididos em 5 categorias definidas conforme a funcionalidade dos comandos SQL, à exceção da categoria "Miscelânea" com operadores que podem ser utilizados em comandos de funcionalidades distintas. Por exemplo, um operador de mutação definido para um operador matemático, tal como "+", é classificado na categoria *Operadores de Mutação para Operadores de SQL*. Um operador definido para alterar o parâmetro de entrada ou saída de uma procedure de SQL, tal como "in" ou "out", pertence à categoria *Operadores de Mutação para Funções e Procedimentos*. Todos os operadores definidos para diversos comandos da SQL, tais como INSERT e DELETE, pertencem à categoria *Miscelânea*, pois podem caracterizar defeitos diferentes, dependendo da funcionalidade do comando ao qual ele foi aplicado. As próximas subseções apresentam as 5 categorias de operadores de mutação.

4.1 Operadores de Mutação para Operadores de SQL

O objetivo desta categoria de operadores é a caracterização dos erros normalmente criados pelo programador devido ao uso incorreto dos operadores de SQL.

Foram definidos operadores de mutação para os 5 tipos de operadores mais comuns em SQL – operadores matemáticos, de comparação, lógicos, conjuntivos e de negação, visando a caracterizar defeitos em: desvios de fluxos; repetições em estruturas condicionais; número de *tuplas* (número a mais ou a menos de *tuplas* na seleção, inserção, exclusão ou atualização); inserção, atualização e exclusão de dados persistentes; e visualização ou cálculo de dados. Os 5 operadores pertencentes a esta

categoria são: *Troca de Operador Matemático (tOpMt), Troca de Operador de Comparação (tOpCp), Troca de Operador Conjuntivo (tOpCj), Troca de Operador de Lógico (tOpLg), Inserção de Operador de Negação (iNot), e Retirada de Operador de Negação (rNot).*

4.2 Operadores de Mutação Miscelânea

Os operadores desta categoria são caracterizados pela sua aplicação em muitos comandos com funcionalidades distintas.

São aplicáveis a diversos comandos da SQL, tais como INSERT e DELETE, e podem caracterizar defeitos em: inserção, atualização ou exclusão de dados persistentes; visualização ou cálculo de dados; definição de atributos e variáveis; e número de tuplas. Os 15 operadores pertencentes a esta categoria são: *Troca de Posição de Atributo* (tPoAt); *Retirada de Atributo* (rAtr); *Inserção de Atributo* (iAtr); *Troca de Atributo* (tAt); *Troca de Posição de Valor* (tPoVr); *Troca de Valor* (tVr); *Troca de Tipo de Variável* (tTpVar); *Troca de Nome de Tabela* (tNmTb); *Troca de Nome de ROLE* (tNmRole); *Inserção de ROLE* (iRole); *Retirada de ROLE* (rRole); *Troca de Nome de Cursor* (tNmCursor); *Troca de Função de Agregação* (tFuAg); *Troca de Intersecção* (tInSec); e *Troca de Join* (tJoin).

4.3 Operadores de Mutação para Fluxo de Controle

O objetivo desta categoria de operadores é abordar o fluxo de controle nos comandos SQL, por modificações em estruturas condicionais e de repetição.

Operadores desta categoria caracterizam os seguintes defeitos: execução indevida de comandos e comandos não executados. Os 6 operadores pertencentes a esta categoria são: *Troca de Bloco de Comandos nas estruturas de condição e repetição (tBlCmEstRep); Retirada de Comando do Bloco de Repetição/Condição (rCmBlRep); Inserção de Comando do Bloco de Repetição/Condição (iCmBlRep); Troca de Posição do Leave no Bloco de Comandos (tPosLeave); Retirada de LEAVE (rLeave); e Inserção de LEAVE (iLeave).*

4.4 Operadores de Mutação para Controle de Transações.

Esta categoria de operadores está focada nos defeitos associados ao controle das transações e às permissões de acesso por parte dos usuários.

Os defeitos nesta categoria são normalmente cometidos pelos usuários no momento da execução dos comandos e na definição do esquema de permissões de acesso, pela ausência ou excesso de permissão de execução de comandos para usuários. Os 12 operadores pertencentes a esta categoria são: *Inserção de COMMIT (iCM); Retirada de COMMIT (rCM); Inserção de ROLLBACK (iRb); Retirada de ROLLBACK (rRb); Troca de COMMIT por ROLLBACK (tCmRb); Troca de ROLLBACK por COMMIT (tRbCm); Troca de Nome do SAVEPOINT (tNmSP); Troca de Permissão (tPerm); Troca de Privilégio (tPriv); Troca de GRANT por REVOKE (tGrRe); Troca de REVOKE por GRANT (tReGr); e Troca de Nome de Usuário (tNmUsr).*

4.5 Operadores de Mutação para Funções, Procedimentos e Triggers

Esta categoria visa a caracterizar o uso incorreto de valores e de chamadas externas: execução errônea de função, *trigger*, view ou procedimento; retorno de valores incorretos; ausência de retorno de valores; execução de trigger em evento incorreto. Os 5 operadores pertencentes a esta categoria são: *Troca de Nome da Função, Procedimento, VIEW ou TRIGGER (tNm); Troca Posição de Retorno da Função (tPoReFu); Retirada de Retorno da Função (rReFu); Troca de Parâmetros da PROCEDURE (in por out) (tPaPro); e Troca de Evento na TRIGGER (tEv).*

5. Ferramenta para a Automação

O número elevado de mutantes criados a partir de uma classe de operadores torna necessários o desenvolvimento e o uso de uma ferramenta para dar suporte à criação, execução e análise de mutantes (mortos, vivos e equivalentes).

Quatro aspectos foram observados na automação da análise de mutantes para SQL: (1) captura de dados: envolve a obtenção dos dados de entrada e de saída, incluindo: os comandos SQL executados pela linguagem hospedeira, os dados de entrada (incluindo o estado da base de dados) e os resultados obtidos pela execução dos comandos SQL; (2) definição dos operadores de mutação a serem aplicados nos comandos SQL; (3) geração e execução dos mutantes; e (4) análise dos resultados.

Para os mutantes vivos, a análise de uma possível equivalência é realizada pelo testador com o apoio da ferramenta, e a decisão pode ser armazenada na ferramenta. Os dados dos mutantes são armazenados na base de testes, permitindo que cada grupo de comandos e seus mutantes sejam re-executados a qualquer tempo.

A ferramenta permite a visualização e a armazenagem de informações sobre o experimento, tais como: (i) identificação da seqüência de comandos executados para um determinado caso de teste; (ii) operadores de mutação aplicados; (iii) mutantes gerados para cada operador; (iv) resultado da avaliação do mutante; (v) comandos SQL original e modificado para cada mutante; (vi) resultado da execução dos comandos original e modificado, incluindo os dados persistentes envolvidos; (vii) mensagens do sistema gerenciador de banco de dados; e (viii) número de tuplas selecionadas, modificadas, número de operações de escrita e de leitura. Em adição, relatórios são gerados sobre o experimento com informações do projeto: operadores aplicados; quantidade de mutantes gerados; índices de mutantes mortos, vivos e equivalentes por operador, projeto, caso de teste, comando SQL ou categoria de operador.

6. Experimentos

Foram conduzidos três experimentos utilizando aplicações e bases de dados reais, com o objetivo de: investigar a aplicabilidade e a habilidade de detecção de defeitos dos operadores de mutação propostos, validar a aplicação da automação com a ferramenta desenvolvida e investigar os 2 cenários propostos para a aplicação do teste de mutação.

Os Experimentos 1 e 2 exploram o Cenário 1 (Subseção 3.1). O Experimento 3 explora o Cenário 2 (Subseção 3.2).

Nos experimentos foram aplicados operadores de mutação nos quatro comandos de manipulação da SQL – INSERT, SELECT, UPDATE e DELETE, pois são os mais

comuns em sistemas comerciais de informação. Foram empregados os *Operadores para SQL* (Seção 3.1) e os de *Miscelânea* (Seção 3.2), por representarem os mais aplicáveis aos comandos selecionados.

O Experimento 1 utilizou uma aplicação de gestão de equipamentos de informática. Suas principais funcionalidades são o controle de: entrada e saída de equipamentos (software e hardware); materiais; custo, amortização, emissão de notas, fatura e outras operações. A aplicação foi desenvolvida, sob demanda, para uma empresa de médio porte atuante no comércio varejista e está implantada e em funcionamento em 25 filiais, sendo utilizada por aproximadamente 4 usuários por filial. A equipe de desenvolvimento da aplicação possui perfil pleno (não é composta de pessoal inexperiente). O analista de testes foi o responsável pelo projeto dos casos de testes, os quais visam o teste de regras de negócio e de aspectos de funcionamento básico da aplicação (operações de inclusão, alteração, navegação e exclusão), com cobertura de um caso de uso.

O Experimento 2 utilizou um software de controle de empréstimo de materiais, cujas principais funcionalidades são: controle de entrada e saída, e reserva de materiais (livros, cds, revistas, etc.); integração com outros sistemas da empresa pelo envio de notificações. A aplicação foi desenvolvida, sob demanda, para uma empresa de médio porte atuante no comércio varejista e está implantada e em funcionamento em 30 filiais; é utilizada por aproximadamente 40 usuários por filial. A equipe de desenvolvimento da aplicação possui perfil *senior*. O analista de testes foi o responsável pelo projeto dos casos de testes, os quais visam o teste de regras de negócio e de aspectos de funcionamento básico da aplicação (operações de inclusão, alteração, navegação e exclusão), com cobertura de um caso de uso.

O Experimento 3 utilizou uma aplicação cuja principal função é exemplificar padrões de desenvolvimento como telas, componentes e relatórios. Esta aplicação é utilizada internamente em uma empresa multinacional de tecnologia de informação. A base de dados utilizada neste sistema é denominada *Adventure Works*: um banco de dados exemplo do sistema gerenciador de banco de dados SQL Server. Não há informações sobre a equipe de desenvolvimento do software. Os casos de testes utilizados neste experimento foram criados por um analista de testes com perfil *senior*. Foram utilizados 42 comandos de manipulação SQL retirados de procedimentos da aplicação em teste.

Tabela 1: Características das bases de dados utilizadas nos experimentos.

Experimento	Base	Ambiente	Descrição das Bases de Dados
1	1	Testes	Utilizada por desenvolvedores e testadores para executar testes de unidades, funcionais, de integração e de sistemas.
	2	Produção	Dados da produção, sem nenhuma alteração de testadores ou desenvolvedores. As persistências de dados são reais.
2	1	Produção	Dados da produção, sem nenhuma alteração de testadores ou desenvolvedores. As persistências de dados são reais.
	2	Produção Reduzido	Dados da produção com redução de persistências de dados de 30%, realizada através de ferramenta proprietária, que busca reduzir o volume de dados das bases sem, no entanto, perder a qualidade dos dados de testes.
3	1	*Adventure Works*	Dados disponíveis no banco de dados de exemplo sem nenhuma alteração.

	2	*Adventure Works* Reduzido	Dados disponíveis no banco de dados de exemplo com redução de persistências de dados de 50%, realizada através de uma ferramenta proprietária, que busca reduzir o volume de dados das bases sem prejuízo para a qualidade dos dados de testes.

As características das bases de dados utilizadas nos experimentos são descritas na Tabela 1. Tais bancos de dados basearam as instâncias de entrada para as bases de dados e caracterizam o uso de dados reais nas situações de produção e de testes. Note que, para cada experimento, são descritas duas bases de dados. O objetivo da análise empírica é comparar as duas bases de dados dentro do cenário de cada experimento.

6.1 Seleção dos Operadores

Devido ao grande número de mutantes gerados para os experimentos, alguns operadores foram aplicados em apenas 10% dos casos. Para a seleção dos operadores, o estudo de mutação seletiva [Mathur, 1991; Offutt et al., 1996] foi adaptado. A Mutação Seletiva consiste em selecionar um número reduzido de operadores de mutação que sejam realmente diferentes dos demais. A não aplicação de um operador que resulta em um grande número de mutantes reduz o custo do teste.

Tabela 2: Operadores Aplicados no Experimento 1

Operador	Observações
tOpMt	Aplicado às permutas do operador multiplicação pelos operadores adição, subtração e divisão. Todas as trocas possíveis para este operador foram utilizadas.
tOpCp	Aplicado às permutas dos operadores igualdade, maior e menor pelos demais operadores comparativos. Todas as trocas possíveis para estes operadores foram utilizadas.
tOpCj, iNot, rNot	Todas as trocas possíveis foram aplicadas.
tPoAt, tAt	Aplicado para 10% dos atributos.
tVr	Aplicado para 10% dos valores em condições
tPoVr	Aplicado para 10% dos atributos em atualizações.
tNmTb	Aplicado para 10% das tabelas.
tJoin	Aplicado às permutas de INNER JOIN pelos outros tipos de JOIN. Todas as trocas possíveis para este operador foram utilizadas.
tFuAg	Aplicada à permuta de COUNT por SUM. Todas as trocas possíveis para este operador foram utilizadas.

Tabela 3: Operadores Aplicados no Experimento 2

Operador	Observações
tInSec	Aplicado às permutas de INTERCEPT e EXCEPT pelas outras intersecções. Todas as trocas possíveis para estes operadores foram utilizadas.
tOpCp	Aplicado às permutas dos operadores igualdade, maior e menor pelos demais operadores comparativos. Todas as trocas possíveis para estes operadores foram utilizadas.
tOpCj, iNot, rNot	Todas as trocas possíveis foram aplicadas.
tPoAt, tAt	Aplicado para 10% dos atributos.
tVr	Aplicado para 10% dos valores em condições
tPoVr	Aplicado para 10% dos atributos em atualizações.
tNmTb	Aplicado para 10% das tabelas.
tJoin	Aplicado às permutas de INNER JOIN pelos outros tipos de JOIN. Todas as trocas possíveis para este operador foram utilizadas.
tFuAg	Aplicado à permuta de COUNT por SUM. Todas as trocas possíveis para este operador foram utilizadas.

Neste estudo, a seleção foi realizada pelo cálculo de *escores de mutação* para os seguintes casos: (1) foram gerados 100% dos mutantes e os testes foram executados; (2) foram gerados mutantes com apenas 10% dos operadores selecionados e os testes foram executados. Os *escores de mutação* foram comparados e os valores foram muito próximos, demonstrando que os mutantes excluídos podem ser removidos, pois não revelarão defeitos diferentes daqueles revelados pelos mutantes que permanecerão no experimento.

O critério de seleção de operadores para redução considerou também a experiência do testador que apontou quais seriam os operadores mais interessantes, considerando as características do software e o conhecimento da equipe de desenvolvimento.

As Tabelas 2, 3 e 4 listam os operadores aplicados em cada um dos três experimentos.

Tabela 4: Operadores Aplicados no Experimento 3

Operador	Observações
tInSec	Aplicado às permutas de INTERCEPT e EXCEPT pelas outras intersecções. Todas as trocas possíveis para estes operadores foram utilizadas.
tOpMt	Aplicado às permutas do operador adição pelo operador subtração. Todas as trocas possíveis para este operador foram utilizadas.
tOpCp	Aplicado às permutas dos operadores igualdade, maior e menor pelos demais operadores comparativos. Todas as trocas possíveis para estes operadores foram utilizadas.
tOpCj, iNot, rNot	Todas as trocas possíveis foram aplicadas.
tPoAt, tAt	Aplicado para 10% dos atributos.
tVr	Aplicado para 10% dos valores em condições
tPoVr	Aplicado para 10% dos atributos em atualizações.
tNmTb	Aplicado para 10% das tabelas.
tJoin	Aplicado às permutas de INNER JOIN pelos outros tipos de JOIN. Todas as trocas possíveis para este operador foram utilizadas.
tFuAg	Aplicado às permutas de AVG e MIN por MAX, COUNT, MIN, AVG e SUM.
tOpLg	Todas as trocas possíveis para tAllAny, tAnyAll, tExUn foram aplicadas.

6.2 Procedimento de Realização dos Experimentos

A realização do experimento ocorreu em 3 passos, sendo o primeiro passo diferente para o Experimento 3 pois, ao contrário dos outros, utilizou o Cenário 2 (Seção 3.2):

Passo 1: O testador entra com uma combinação de instâncias do banco de dados (Bases 1 e 2) e com valores de entrada parametrizados (dados de entrada dos casos de testes projetados pelo testador) em uma UP (nos Experimentos 1 e 2), que obtém como saída comandos de manipulação da SQL, que por sua vez, selecionam ou modificam registros em suas instâncias alvos do BD. No Experimento 3, tais dados são aplicados para executar uma seqüência de comandos SQL, que são retirados de *procedures*; estes alteram e modificam as instâncias alvos do BD.

Passo 2: A ferramenta armazena informações das alterações e modificações juntamente com os comandos executados; os operadores de mutação selecionados são aplicados automaticamente pela ferramenta nos comandos SQL coletados e os mutantes gerados são executados (os comandos originais são gerados antes do mutante). Após a

execução, as informações sobre as alterações e seleções feitas são armazenadas e comparadas com os comandos de manipulação original.

Passo 3: A ferramenta automaticamente classifica os mutantes em mortos ou vivos. No caso de mutantes vivos a ferramenta demonstra os resultados tanto do comando original quanto do mutante, para que o testador decida se o mutante é equivalente ou não. A ferramenta permite que o mutante seja re-executado isoladamente para análise. O *escore de mutação EscM* é gerado para cada caso de teste executado no experimento e seus valores são avaliados.

6.3 Resultados dos Experimentos

A Figuras 1, 2 e 3 apresentam os *escores de mutação* obtidos nos Experimentos 1, 2 e 3, respectivamente, para as bases de dados 1 e 2 de cada experimento.

Foi observado, nos Experimentos 1 e 2, uma variação na quantidade de comandos SQL entre suas bases de dados: 79 e 85 comandos (Experimento 1), 71 e 66 comandos (Experimento 2), para as bases de dados 1 e 2, respectivamente. Essa variação ocorreu devido ao fato de as instâncias das bases de dados exercitarem partes diferentes do software em teste. O Experimento 3 foi realizado com os mesmos 42 comandos SQL para ambas as bases, devido às características do cenário do experimento (Cenário 2).

Na maioria dos casos de teste dos Experimentos 1 e 3, os *escores de mutação* alcançados para a base de dados 1 foram superiores ou iguais aos obtidos para a base de dados 2. No Experimento 2, a base de dados 2 obteve melhores *escores de mutação*. Assim, os Experimentos 1 e 2 sugerem que as bases de dados reduzidas, as quais foram preparadas pelas empresas para a realização de testes (base de dados 1 no Experimento 1, e base de dados 2 no Experimento 2), possuem habilidade igual ou melhor de detecção de defeitos do que as bases de dados de produção. Por outro lado, a redução simples de uma base de dados pode causar perda em sua habilidade de detecção de defeitos (base de dados 2 no Experimento 3).

No Experimento 1, a aplicação dos operadores de *Troca de Posição*, tais como *tPoAt* e *tPoVr*, mostrou que as bases de dados possuem um esquema bem definido, uma vez que todos os mutantes definidos a partir desses operadores foram mortos por ambas as bases, pois as restrições de dados estão preparadas para não receber dados inválidos. Por outro lado, a aplicação desses operadores para comparar a qualidade dos dados entre as duas bases é ineficiente, pois o esquema (meta-dado) e os *escores de mutação* obtidos são iguais para as bases de dados.

No Experimento 2, a base de dados reduzida (base de dados 2) exercitou menos código (5 comandos a menos) e revelou 2 defeitos a menos em relação à base de dados de produção (base de dados 1). Contudo, apenas a base de dados reduzida revelou alguns defeitos e o *escore de mutação* para esta base de dados foi mais elevado do que o da base de produção para a maioria dos operadores de mutação. Na Figura 2, observa-se que o *escore de mutação* para ambas as bases de dados é similar em muitos casos de teste, podendo indicar que a base reduzida possui eficiência de detecção de defeitos, no mínimo, similar à da base de produção.

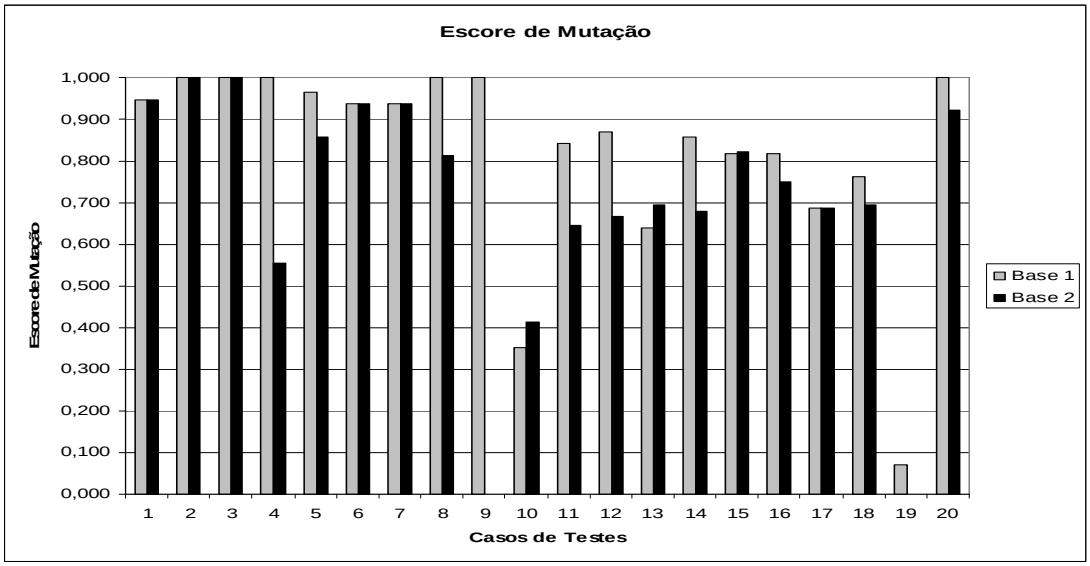

Figura 1: *Escores de mutação* para as bases de dados 1 e 2 do Experimento 1.

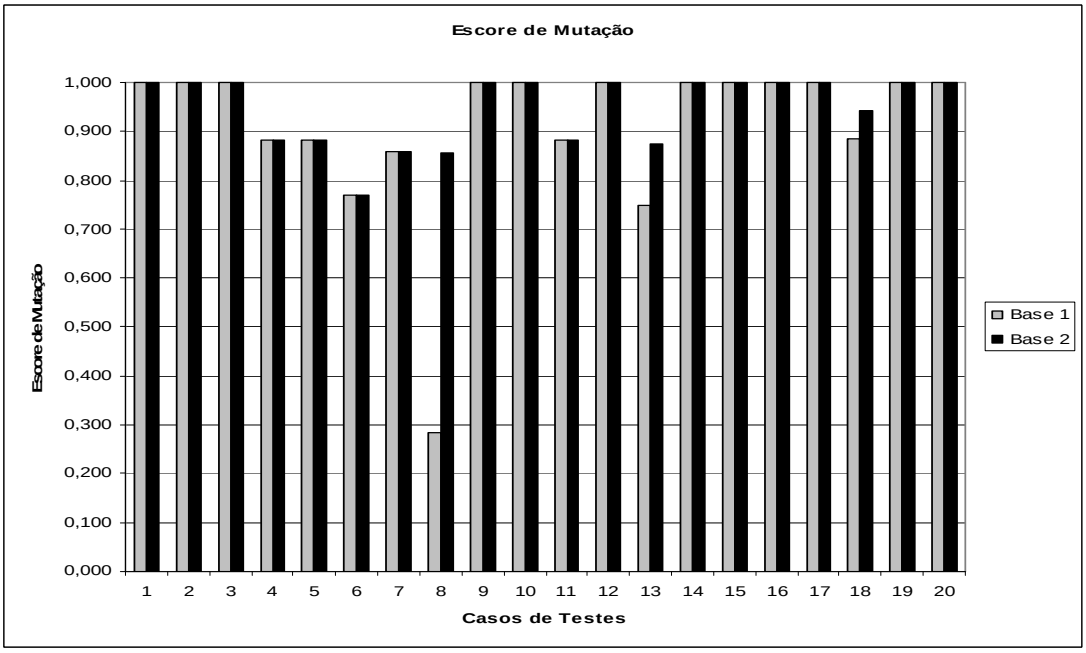

Figura 2: *Escores de mutação* para as bases de dados 1 e 2 do Experimento 2.

No Cenário 1 (Experimentos 1 e 2), as seqüências de comandos SQL resultantes do emprego de bases de dados diferentes podem ser distintas. Essa possibilidade não invalida a análise para a comparação das bases de dados, pois os *escores de mutação* representam valores relativos com respeito a defeitos comuns no software (operadores de mutação). O Cenário 2 possui um efeito de comparação mais forte, pois a seqüência de comandos SQL (e qualquer de seus componentes) será igual, independentemente das bases de dados de entrada. Assim, do Experimento 3, conclui-se que a base de dados 1 é superior à base de dados 2 para a detecção de defeitos.

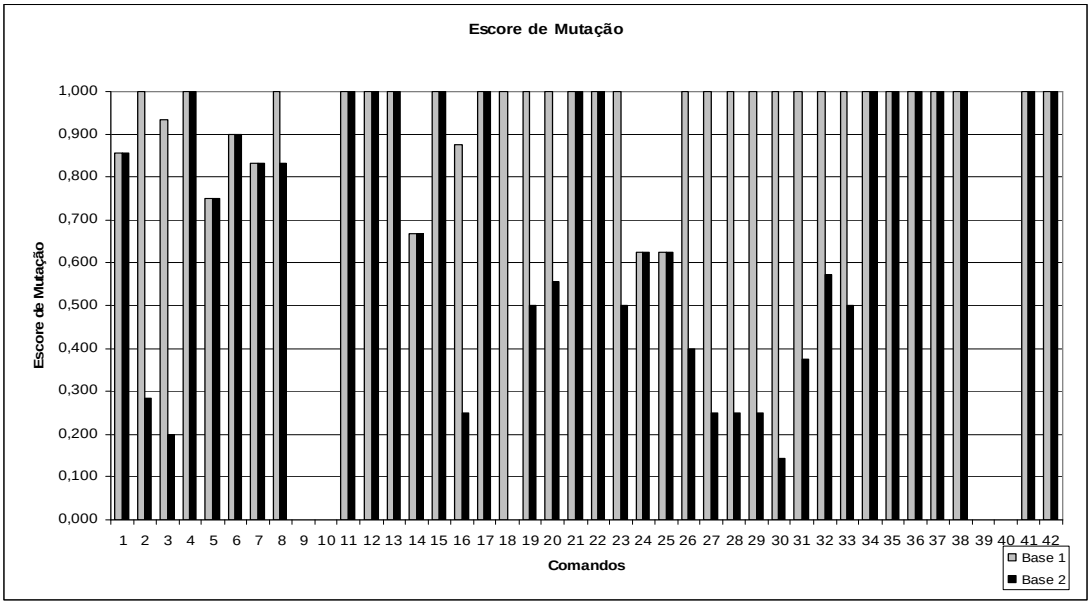

Figura 3: *Escores* de mutação para as bases de dados 1 e 2 do Experimento 3.

6.4 Considerações Finais

O operador de *troca de join* conseguiu atingir, no Experimento 3, ao contrário dos outros dois experimentos, *escores de mutação* distintos de zero. Se a base de dados for boa o suficiente para revelar este tipo de defeito, ele será revelado. Assim, dos 2 primeiros experimentos conclui-se que as bases de dados eram ruins e não que o operador é difícil de matar. Contudo, a seleção dos operadores de *troca de join* deve ser feita cautelosamente, devido aos seus baixos *escores de mutação*.

Um aspecto favorável às bases de dados reduzidas é o tempo de processamento; mesmo com a utilização de poucos operadores de mutação, o tempo de processamento das bases de produção foi muito superior ao das bases reduzidas.

7. Trabalhos Relacionados

Na aplicação da análise de mutantes em aplicações de banco de dados com SQL, temos conhecimento de apenas dois estudos, com as seguintes características: (i) criação de operadores de mutação baseado em apenas um comando de manipulação; e (ii) uso de dados sintéticos.

Em Tuya et al. (2007), são propostos operadores de mutação para comandos de consulta SQL (SELECT) envolvendo mutações para: as principais cláusulas da SQL, condições e expressões, manipulação de valores nulos e substituição de parâmetros, tuplas e constantes. Os mutantes foram gerados a partir de comandos estáticos de consulta SQL. O experimento ocorreu em uma base de dados que sofreu alterações conforme a aplicação dos operadores. A detecção de mutantes equivalentes e a criação de casos de testes adicionais são realizadas de forma manual; não se indica se existe automação. O artigo abrange também o conceito de redução de custos, utilizando a técnica de redução do número de mutantes por mutação seletiva [Mathur, 1991] e redução do tamanho do conjunto de teste por ordenação de mutantes [Rothermel e Elbaum, 2003; Rothermel et al., 2001; Elbaum et al., 2002]. O experimento não inclui

comandos de consulta SQL complexos, tais como sub-consultas e utilização de múltiplas tabelas, e não utiliza dados reais. Segundo os autores, a ausência de dados reais torna o experimento incerto em relação à representatividade de cada consulta, em termos das combinações de características que podem ser encontradas no mundo real, impossibilitando a comparação da aplicabilidade dos operadores com sistemas reais.

Chan et al. (2005) propõem operadores de substituição para a mutação da SQL. A abordagem explora informações semânticas do modelo de dados conceitual, tais como relações entre os atributos armazenados e derivados para a criação dos mutantes, de acordo com a família de operadores de substituição definida. A mutação ocorre nos comandos SQL embutidos na linguagem hospedeira. O artigo não mostra resultados da aplicabilidade dos mutantes e nem apresenta uma ferramenta ou uma abordagem de como automatizar a geração, execução e avaliação dos mutantes. Os operadores de mutação definidos pelo artigo são apenas para consulta (comando SELECT) e o motivo de tal escolha não é explicitado no trabalho. O artigo é pioneiro na aplicação da análise de mutantes para comandos SQL.

8. Conclusões

O objetivo principal do trabalho é a melhoria da qualidade dos testes de aplicações de banco de dados, pela seleção de bases de dados mais reveladoras de defeitos, por meio da aplicação da análise de mutantes em comandos da linguagem SQL.

A motivação para este trabalho é a constatação de que apenas dois estudos buscam o mesmo objetivo [Tuya et al., 2007; Chan et al., 2005] e, mesmo assim, tais pesquisas limitam-se ao estudo de comandos estáticos de consulta da linguagem SQL e a experimentos utilizando apenas dados sintéticos e sem suporte à automação.

As principais contribuições deste trabalho são:

- definições para o teste de mutação aplicado a SQL;
- definição e caracterização de cenários para a aplicação do teste de mutação em SQL, nos níveis de mutação fraca e forte;
- definição e categorização de operadores de mutação para SQL;
- análise experimental usando aplicações reais e dados reais, cobrindo os cenários definidos e apresentando resultados promissores para a aplicação de teste de mutantes em aplicações SQL, com respeito à detecção de defeitos e à aplicabilidade da técnica.

Uma peculiaridade dos cenários definidos é a mutação de seqüências de comandos da SQL, as quais são oriundas da execução de aplicações. Tais seqüências caracterizam o código da aplicação que manipula os dados persistentes. A abordagem permite o teste de aplicações com montagem dinâmica de comandos da SQL.

Uma conclusão é a constatação de que a técnica permite a comparação de bases de dados nos cenários de mutação forte e fraca. Nos experimentos foram utilizadas bases de dados de produção e bases de dados reduzidas, as quais são dedicadas à atividade de teste. A aplicação da técnica permitiu avaliar a qualidade das bases de dados reduzidas.

Alguns trabalhos futuros são:

- estender a análise experimental a outras aplicações reais e dados reais;
- implantar a técnica nas empresas que colocaram à disposição os programas e dados reais para o experimento; inicialmente, motivar a avaliação de todas as bases de dados reduzidas derivadas de bases de dados de produção, para aquilatar a sua eficácia na atividade de teste.

9. Referências Bibliográficas

ANSI SQL-3 STANDARD, www.ansi.org. Acesso em: 12 de jun. 2008.

Chan,W.K., Cheung, S.C., and Tse, T.H. (2005) "Fault-based testing of database application programs with conceptual data model". Proceedings of the 5th Intl. Conference on Quality Software, IEEE Computer Society Press, pg 187–196, Los Alamitos, CA.

Chan, M. Y. and Cheung, S. C. (1999) "Testing Database Applications with SQL Semantic". Proceedings of the 2nd Intl. Symposium on Cooperative Database Systems for Advanced Applications, Springer, pg 363-374, Singapore.

Chays, D. and Deng, Y. (2003) "Demonstration of AGENDA Tool Set for Testing Relational Database Applications". Proceedings of the 25th Intl. Conference on Software Engineering, pg 802-803, Portland, Oregon.

DeMillo, A.R., Lipton, R.J., and Sayward, F.G (1978)."Hints on test data selection: Help for the practicing programmer". IEEE Computer, Vol.11, nº 4, pg 34-41.

DeMillo, A.R. (1980) "Mutation analysis as a tool for software quality assurance". Proceedings of COMPSAC 80, Chicago, IL, pg 390-393.

Deutsch, M. (1979) "Verification and Validation" in Software Engineering (R. Jensen and C.Tonies, Eds.), Prentice-Hall, 1979, pg 329-408.

Elbaum, S., Malishevssky, A.G., and Rothermel, G. (2002). "Test case priorization: A family of empirical studies". IEEE Trans. on Software Engineering 28 (2), pg 159–182.

Kapfhammer, G.M. and Soffa, M.L. (2003). "A Family of Test Adequacy Criteria for Database-Driven Applications". European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE. Helsinki, Finland, pg 98-107.

Leitao-Junior, P.S., Vilela, P.R.S., and Jino, M. (2005). "Mapping Faults to Failures in SQL Manipulation Commands". Proceedings of the 3rd ACS/IEEE Intl. Conference on Computer Systems and Applications (AICCSA-05), pg 1-4, Egito, Cairo.

Mathur, A.P. (1991). "Performance, effectiveness and reliability issues in software testing". 15th Annual International Computer Software and Application Conference, Tokio, Japan, IEEE Computer Society Press, pg 604-605.

Offutt, A.J, Lee, A., Rothermel, G., Untch, R.H and Zapf, C. (1996) "An experimental determination of sufficient mutant operators". ACM Trans. on Software Engineering Methodology, Vol.5, nº 2 pages 99-118.

Pönighaus, R. (1995) "'Favourite' SQL-Statements – an empirical analysis of SQL-Usage in commercial applications". Proceedings of the 6th Intl. Conference on Information Systems and Management of Data, Lecture Notes in Computer Science, Vol. 1006, Springer, pg 75–91.

Rothermel, G. and Elbaum, S. (2003) "Putting your best tests forward". IEEE Software 20 (5), pg 74–77.

Rothermel, G., Untch, R.H., and Harrold, M.J. (2001). "Prioritizing test cases for regression testing". IEEE Trans. on Software Engineering 27 (10), pg 929–948.

Spoto, E.S. (2000). "Teste Estrutural de Programas de Aplicação de Banco de Dados Relacional". Tese de Doutorado, DCA/FEEC/UNICAMP, Campinas, SP, Brasil.

Tuya, J., Suárez-Cabal, M.S. and de la Riva, C. (2007) "Mutation Database queries". Information and Software Technology 49, pg 398-417.

Woodward, M.R. (1993). "Mutation Testing – its Origin and Evolution". Information and Software Technology, Vol. 35, nº 3, pg 163–169.