Geração de Base de Dados para o Teste de Aplicações de Banco de Dados pelo Emprego da Computação Evolucionária

Bruno Braz Silveira[1], Plínio Sá Leitão-Júnior[1], Mariana Soller Ramada[1], Beatriz Proto Martins[1]

[1]Instituto de Informática – Universidade Federal de Goiás (UFG)
Caixa Postal 131 – 74.001-970 – Goiânia – GO – Brazil

bbrazsilveira@gmail.com,{plinio,mariana,beatrizmartins}@inf.ufg.br

Abstract. *This paper focuses on the problem of generating data for test execution in SQL statements on the context of database applications. Given the big amount of defects in SQL instructions of an application, it is necessary to generate, within the domain of attributes in a database schema, tuples sets with quality, which can help to detect the presence of most defects. For this, it was applied the principles of Evolutionary Computation, through meta-heuristic Genetic Algorithms, by evolving test data. In addition, it was used Analysis of SQL Mutants for assessing the quality of the test data. In addition, it was used Analysis of SQL Mutants for assessing the quality of the test data.*

Resumo. *Este artigo enfoca o problema da geração de dados para a execução de testes em instruções SQL no contexto de aplicações de banco de dados. Dado o grande número de defeitos em instruções SQL de uma aplicação, deve ser gerado, dentro do domínio dos atributos de um esquema de banco de dados, um conjunto de tuplas com qualidade, que consiga auxiliar na detecção da maioria dos defeitos. Para isso, foram aplicados os princípios da Computação Evolucionária, através da meta-heurística Algoritmos Genéticos, ao evoluir os dados de teste. Além disso, foi utilizada a Análise de Mutantes SQL para a avaliação da qualidade dos dados de teste.*

1. Introdução

A Inteligência Computacional (IC) busca o desenvolvimento de sistemas inteligentes que imitem aspectos do comportamento humano, tais como: aprendizado, percepção, raciocínio, evolução e adaptação [Engelbrecht 2007]. Computação Evolucionária (CE) é um paradigma de IC, que busca reproduzir processos naturais de evolução, onde é usado o conceito de sobrevivência [Jong 2006]. Como os processos e os produtos de engenharia de software são dependentes de decisões humanas, o uso de técnicas de CE pode agregar qualidade à esses elementos.

A introdução de defeitos no software é inerente ao seu processo de desenvolvimento. Desta forma, o programa ou sistema é executado durante a fase de teste com a finalidade de encontrar defeitos, que passaram despercebidos durante a fase de desenvolvimento [Myers 1979]. Entretanto, a própria atividade de teste possui incertezas, como, por exemplo, o fato de somente um subconjunto de elementos do domínio de entrada ser selecionado para o teste. Isso introduz uma taxa de incerteza por não ser garantido que todos os defeitos serão revelados.

O contexto desta pesquisa refere-se às incertezas inerentes ao teste de software, especificamente durante o planejamento e a aplicação do teste, em atividades tais como a seleção de dados de teste, execução do teste e a verificação dos resultados. É também abordado o teste de software envolvendo persistência de dados através da Aplicação de Banco de Dados (ABD). ABDs fazem uso de algum tipo de linguagem para acessar os dados, onde se destaca a SQL (*Structured Query Language*).

O problema em questão refere-se em como gerar bases de dados para o teste de aplicações de banco de dados a um baixo custo e com eficácia para a descoberta de defeitos em instruções SQL. Os dados de teste para essa classe de aplicações constituem bases de dados que serão lidas pelo software em teste, o que denota elevada complexidade ao problema, devido às "infinitas" possibilidades para a geração dessas bases de entrada [Tuya et al. 2009].

A geração de base de dados de entrada para o teste de ABD pode, em geral, ser representado por um problema de otimização ou busca. Assim, é possível obter resultados importantes para dados de teste aplicando-se a CE com um custo computacional relativamente baixo.

1.1. Objetivos

O objetivo principal da pesquisa é testar ABDs, visando à descoberta de potenciais defeitos em instruções em SQL. São aplicados conceitos da CE para a geração de bases de entrada e, além disso, é utilizada a Análise de Mutantes para medir a qualidade do conjunto de bases geradas.

Como a CE abrange uma grande quantidade de algoritmos, o objetivo principal da pesquisa pode ser decomposto nos seguintes objetivos específicos: 1) Representar base de dados como indivíduos que podem ser criados e evoluídos por meio da aplicação de algoritmos da CE; 2) Desenvolver algoritmos e suporte computacional para a aplicação da CE na geração de bases de teste de ABDs, segundo a meta-heurística *Algoritmos Genéticos (AG)*; 3) Empregar Análise de Mutantes em experimento para avaliar e comparar bases de dados de teste geradas pelo uso de Algoritmos Genéticos (AG) e aleatoriamente.

2. Fundamentação Teórica

2.1. Análise de Mutantes

Uma abordagem usada para aumentar a eficácia dos testes de instruções SQL é a Análise de Mutantes [Derezinska 2007], que objetiva mensurar a qualidade de um conjunto de bases de entrada. Para tal, são produzidas várias versões da instrução SQL, denominadas mutantes, cada uma com uma pequena modificação na sintaxe da instrução original. As bases de entrada são então aplicadas à instrução original e a todos os seus mutantes.

Um conjunto de bases de entrada é promissor para revelar defeitos se o resultado obtido pela execução da instrução original difere dos resultados dos seus mutantes. Ou seja, o conjunto de bases de entrada foi capaz de detectar a diferença entre a instrução original e os seus mutantes (nesse caso o mutante é "morto"). Essa técnica vem sendo

utilizada com resultados satisfatórios, como uma evidência de que os dados de teste são reveladores de defeitos.

2.2. Algoritmos Genéticos

A meta-heurística Algoritmos Genéticos é um dos métodos mais utilizados para a resolução de problemas de otimização. Utiliza conceitos da Genética, como população, geração, reprodução e mutação. O funcionamento dessa meta-heurística se resume, basicamente, no emprego de três operações genéticas: o *cruzamento* (*crossover*), no qual as informações estruturais de duas soluções são cruzadas a fim de gerar novas soluções; a *mutação*, processo pelo qual algumas alterações aleatórias podem ser realizadas nas soluções geradas; e a *seleção*, que é responsável pela escolha dos indivíduos que serão submetidos às operações genéticas. Após isso, as soluções atuais são avaliadas para a determinação de quais sobreviverão para a próxima iteração (geração) [Jong 2006].

2.3. Trabalhos Relacionados

Gupta et al. (2010) trabalham a geração de dados de teste para matar mutantes SQL considerando as mutações nas cláusulas JOIN e nos operadores relacionais. Shah, et al. (2011), além dos JOINs e dos operadores relacionais, acrescentam a geração de dados de teste para matar os mutantes dos comandos de agregação. Os resultados de ambos os trabalhos foram considerados eficientes para as classes de mutantes envolvidas, porém não utilizam meta-heurísticas nem evolução dos dados de testes.

Tuya, et. al (2009) apresentam uma estratégia de redução de um banco de dados de produção para realização de teste, através de regras de cobertura de algumas instruções SQL e usando a Análise de Mutantes como método de avaliação.

Nota-se que Gupta et al. (2010) e Shah et al. (2011) são trabalhos muito semelhantes, que tratam sobre a geração de bancos de dados de teste, baseados em mutantes de determinadas cláusulas SQL. Já em Tuya et al. (2009), considera-se um banco de dados existente e trabalha-se com a sua redução.

Almeida et al. (2013) aplicaram Programação Evolucionária (PE) para selecionar os dados a serem utilizados na avaliação de mutantes que podem ajudar a detectar defeitos nas instruções SQL de uma determinada aplicação. Os resultados foram obtidos a partir de experimentos que comparam Programação Evolucionária, Geração Aleatória e Algoritmo Genético

4. Metodologia

A metodologia envolve as seguintes etapas: 1) Definição da representação cromossômica dos dados de teste no contexto da Computação Evolucionária [Almeida et al. 2013]; 2) Projeto e implementação de suporte computacional para o emprego e avaliação da meta-heurística Algoritmos Genéticos no contexto de testes de ABDs; 3) Execução dos algoritmos atribuídos à meta-heurística evolucionária; 4) Análise de resultados, comparando com valores obtidos pelo Método Aleatório e pelo uso de Algoritmos Genéticos.

Um esquema de banco de dados foi utilizado na pesquisa, de modo que fosse palco à realização dos experimentos e à obtenção de seus resultados. Foi usado o Modelo Conceitual *Empresa* definido em Elmasri e Navathe (2012), muito difundido no meio acadêmico. Em adição, um Banco de Dados de Referência (BDR) para o referido esquema foi construído, utilizando-se as tuplas apresentadas no referido livro.

Um Banco de Dados de Domínios (BDD) foi construído, para armazenar diversos valores pertinentes ao domínio de cada um dos atributos do Modelo Conceitual *Empresa*. O BDD possui os valores que compõem o BDR, em adição a muitos outros valores dos domínios dos atributos do modelo. A ideia foi usar o BDD para gerar aleatoriamente Bases de Dados de Teste (BDTs).

Os experimentos foram elaborados para seguir as seguintes etapas: (**E0)** entendimento dos possíveis defeitos em instruções SQL; **(E1)** seleção do conjunto *S* de instruções SQL; **(E2)** geração de mutantes para as instruções contidas no conjunto *S*; **(E3)** geração aleatória de bases de teste (BDTs), a partir dos valores presentes no banco de dados de domínios (BDD); **(E4)** evolução da população *P*, buscando privilegiar a sobrevivência de indivíduos que podem revelar a presença do maior número dos potenciais defeitos nas instruções SQL do conjunto *S*.

Na etapa **E0** foi alcançado o entendimento dos possíveis defeitos em instruções SQL. Tal conhecimento é importante para entender o comportamento de instruções SQL defeituosas, tal como posto em Silveira e Leitão-Júnior (2013). Na etapa **E1** foram selecionadas 50 instruções SQL de Elmasri e Navathe (2012). Na etapa **E2** foi empregada a ferramenta *SQLMutation* [Tuya et al. 2006] para a geração dos mutantes para as instruções do conjunto *S*: 2559 mutantes foram gerados, com média aproximada de 50 mutantes por instrução SQL. Na etapa **E3**, a intenção foi gerar a população inicial de indivíduos. Um indivíduo é visto como uma base de dados para o teste das instruções SQL. Assim, a população *P* de indivíduos é composta pelo BDR e pelos BDTs gerados nesta etapa.

Os seguintes aspectos requerem realce na etapa evolucionária **E4**: 1) A população possui 10 indivíduos enquanto que a população inicial foi constituída por 9 BDTs gerados aleatoriamente (a partir do BDD) e pelo BDR; 2) A operação de mutação foi aplicada a 20% dos indivíduos da população a cada geração; a mutação consiste em alterar indivíduos existentes, selecionados aleatoriamente; ocorre a mudança de valores em algumas tuplas, substituindo por valores existentes no BDD; 3) A operação de cruzamento também foi aplicada a 20% dos indivíduos da população a cada geração; cada cruzamento ocorre da seguinte maneira: (i) selecionar 2 indivíduos, *X* e *Y*, aleatoriamente; e (ii) gerar novo indivíduo *Z*, considerando somente os valores do domínio dos indivíduos *X* e *Y*; 4) A seleção a cada geração envolveu a sobrevivência dos melhores indivíduos, ou seja, aqueles indivíduos que possuem o melhor escore de mutação:

Escore (indivíduo) = mutantes mortos / (mutantes - mutantes equivalentes)

As etapas **E3** e **E4** foram conduzidas utilizando suporte computacional, pela aplicação da ferramenta, escrita em linguagem Java, desenvolvida para os fins desta pesquisa.

4. Resultados

Os resultados obtidos são apresentados na Tabela 1. A primeira coluna identifica a geração, onde a Geração 0 refere-se ao estado inicial da população (antes da evolução). A segunda coluna exibe o escore obtido com o indivíduo denominado BDR, o banco de dados de referência extraído de Elmasri e Navathe (2012). A terceira e quarta colunas apresentam, respectivamente, o escore do melhor indivíduo, dentre os nove indivíduos restantes da população, e o escore médio desses indivíduos.

Geração	Escore BDR	Escore (melhor)	Escore (médio)
0	0,5842	0,5127	0,4093
1	0,5842	0,5315	0,4415
2	0,5842	0,5301	0,4312
3	0,5842	0,5569	0,4823
4	0,5842	0,5648	0,4956

Tabela 1. Escores de mutação obtidos para todas as instruções SQL.

Deve ser ressaltado que o BDR serviu como referência aos outros BDTs (nove indivíduos) da população, os quais foram gerados pelo método aleatório. Sobre os resultados apresentados na Tabela 1, observa-se o seguinte: 1) O método aleatório (indivíduos da Geração 0) foi o ponto de partida da população; os BDTs obtidos por esse método são inferiores ao BDR; 2) Quatro gerações foram criadas durante o processo de evolução da população; 3) O BDR possui escore superior a qualquer outro indivíduo, incluindo os escores obtidos para as gerações mais evoluídas; 4) Houve evolução positiva da população em geral durante o processo de evolução, pois o escore médio inicial era 0,4093 e evoluiu até 0,4956; 5); O melhor indivíduo em cada geração sempre foi bem superior a média da população.

Conclui-se que houve uma melhoria da população pela aplicação da meta-heurística Algoritmos Genéticos. Os BDTs evoluíram a cada geração, tornando-se mais eficazes para revelar a presença de possíveis defeitos em instruções SQL de consulta. Duas instruções (SQL-01 e SQL-02) do conjunto original de 50 instruções são apresentadas a seguir. As Tabelas 2 e 3 apresentam os resultados para as Instruções.

```
(SQL-01) SELECT Pnome, Unome FROM Funcionario
                    WHERE EXISTS (SELECT * FROM Dependente
                                  WHERE Cpf=Fcpf)
                                  AND EXISTS (SELECT *
                                              FROM Departamento
                                              WHERE Cpf=Cpf gerente)

(SQL-02) SELECT F.Unome AS Nome funcionario, S.Unome AS Nome supervisor
                    FROM Funcionario AS F, Funcionario AS S
                    WHERE F.Cpf supervisor=S.Cpf
```

Tabela 2. Escores de mutação obtidos para a instrução SQL-01.

Geração	Escore BDR	Escore (melhor)	Escore (médio)
0	0,8519	0,8704	0,7981
1	0,8519	0,8704	0,7981
2	0,8519	0,8704	0,7981
3	0,8519	0,8704	0,7981
4	0,8519	0,9124	0,8357

Tabela 3. Escores de mutação obtidos para a instrução SQL-02.

Geração	Escore BDR	Escore (melhor)	Escore (médio)
0	0,7727	0,7955	0,7943
1	0,7727	0,7955	0,7943
2	0,7727	0,7955	0,7943
3	0,7727	0,7955	0,7943
4	0,7727	0,7955	0,7943

Sobre os resultados apresentados nas Tabelas 2 e 3, é pertinente pontuar que em ambas as instruções os escores obtidos foram superiores aos escores do BDR. Os escores são distintos entre as instruções: SQL-01 obteve escores superiores em relação a SQL-02. Além disso, houve uma tendência a estabilidade dos valores, com respeito aos escores médio e do melhor indivíduo. A Instrução SQL-01 evoluiu os escores médio e de melhor indivíduo na Geração 4, esse comportamento (evolução gradual do indivíduo) é uma tendência observável nos dados da Tabela 1. Por fim, o escore de mutação pode ser usado como uma medida para a complexidade de cada instrução no contexto da atividade de teste.

Em síntese, há um grau de dificuldade para a obtenção de dados de teste para as instruções SQL, tal que cada instrução possui um custo próprio de geração de dados de teste. Através de uma análise dos resultados, foi identificado o **fator da restritividade** nas cláusulas *where* como um dificultador para se matar mutantes SQL. A restritividade é maior/menor quando há poucas/muitas tuplas na base de dados que atendem ao predicado da cláusula *where* da instrução. Dessa forma, as instruções foram divididas em dois conjuntos com relação ao BDR: instruções de Baixa Restritividade (BR) e de Alta Restritividade (AR).

As Tabelas 4 e 5 apresentam os resultados para os comandos BR e AR, respectivamente. Sobre os resultados apresentados em tais tabelas, observa-se que houve uma melhor evolução dos BDTs para instruções de baixa restritividade; em adição, os escores médio e do melhor indivíduo superaram o escore do BDR em todas as gerações. Além disso, instruções de alta restritividade resultam em escores inferiores (BDR e demais BDTs) em relação a instruções de baixa restritividade (ver as duas últimas colunas de ambas as tabelas).

Tabela 4. Escores de mutação obtidos para as instruções BR.

Geração	Escore BDR	Escore (melhor)	Escore (médio)
0	0,5276	0,5468	0,5211
1	0,5276	0,5612	0,5417
2	0,5276	0,5725	0,5589
3	0,5276	0,5563	0,5312
4	0,5276	0,5728	0,5689

Tabela 5. Escores de mutação obtidos para as instruções AR.

Geração	Escore BDR	Escore (melhor)	Escore (médio)
0	0,6142	0,4946	0,3501
1	0,6142	0,5012	0,3629
2	0,6142	0,5098	0,3718
3	0,6142	0,4998	0,3612
4	0,6142	0,5326	0,3959

Conclui-se que é mais difícil obter BDTs para revelar a presença de defeitos para instruções SQL com alta restritividade, em relação a instruções SQL com baixa restritividade.

5. Conclusões

Nesse trabalho, foi possível observar que a aplicação da Computação Evolucionária aliada à Análise de Mutantes é promissora para resolver o problema da Geração de Bases de Teste para a descoberta de defeitos em instruções SQL presentes em Aplicações de Banco de Dados. Especificamente, a meta-heurística Algoritmos Genéticos foi empregada no âmbito da evolução de indivíduos candidatos a dados de entrada para o teste de instruções SQL.

A caracterização do problema e de variáveis para a construção de solução por experimento, o desenvolvimento de ferramenta computacional para automatizar o experimento e a obtenção de resultados preliminares promissores a partir de análise empírica representam alguns dos aspectos importantes alcançados com esta pesquisa.

As conclusões ressaltadas foram que: 1) houve uma melhoria da população pela aplicação da meta-heurística Algoritmos Genéticos; os BDTs evoluíram a cada geração, tornando-se mais eficazes para revelar a presença de possíveis defeitos em instruções SQL de consulta; 2) há um grau de dificuldade para a obtenção de dados de teste para as instruções SQL, tal que cada instrução possui um custo próprio de geração de dados de teste; 3) é mais difícil obter BDTs para revelar a presença de defeitos para instruções SQL com alta restritividade, em relação a instruções SQL com baixa restritividade; 4) o escore de mutação pode ser usado como uma medida para a complexidade de cada instrução no contexto da atividade de teste.

Alguns desdobramentos futuros para a pesquisa são: 1) extensão da evolução dos BDTs pelo aumento do número de gerações; 2) definição de outros operadores para

mutação e cruzamento, que sejam específicos à solução do problema de geração de dados de teste para ABDs; 3) utilização de novos parâmetros para a análise empírica, tais como tamanho do indivíduo e emprego de medida de qualidade no conjunto da população (em vez de cada indivíduo); 4) disponibilização da ferramenta em ambiente Web; 5) expansão do escopo do problema, pelo tratamento da infactibilidade de indivíduos; 6) aplicação de computação paralela, para alcançar melhor desempenho e flexibilidade.

Referências

Derezinska, A. (2007) "An experimental case study to applying mutation analysis for sql queries", International Multiconference on Computer Science and Information Technology.

Engelbrecht, A. P. (2007) "Computational Intelligence". Willey, England.

Elmasri, R. and Navathe, S. B. (2012) "Sistemas de Banco de Dados", 6a Edição, Pearson.

Gupta, B. P., Vira, D., Sudarshan, S. (2010) "X-data: Generating test data for killing sql mutants". 26th International Conference on Data Engineering (ICDE).

Jong, K. A. D. (2006) "Evolutionary Computation A Unified Approach". Massachusetts Institute of Technology, Cambridge, MA.

Myers, G. J. (1979) "The Art of Software Testing", John Wiley & Sons.

Shah, S., Sudarshan, S., Kajbaje, S., Patidar, S., Gupta, B., Vira, D. (2011) "Generating test data for killing sql mutants: A constraint-based approach", IEEE 27th International Conference on Data Engineering (ICDE).

Silveira, B. B. and Leitão-Júnior, P. S. (2013) "Enumeração e Classificação de Defeitos em Consultas SQL", Congresso de Computação do Sul de Mato Grosso.

Tuya, J., Suárez-Cabal, M.J. and De Lariva, C. (2006) "SQLMutation: A tool to generate mutants of SQL database queries", 2nd Workshop on Mutation Analysis.

Tuya, J., Cabal, M. J. S., De Lariva, C. (2009) "Query-aware shrinking test databases", 2nd International Workshop on Testing Database Systems, DBTest, Rhode Island, USA.

Almeida, F., Leitão-Júnior, P. S., Vincenzi, A. M. R., Lucena, F. N. (2013) "Geração de Bases de Dados de Teste pela Aplicação de Programação Evolucionária". 7th Brazilian Workshop on Systematic and Automated Software Testing (SAST), 2013.