Medida de la cobertura de consultas SQL

María José Suárez Cabal, Javier Tuya

Departamento de Informática. Universidad de Oviedo
33271 - Gijón - Asturias
cabal@uniovi.es, tuya@lsi.uniovi.es

Abstract. La tarea de las pruebas del software puede mejorarse con su automatización, utilizando diferentes técnicas en función del criterio de la prueba. Normalmente, esta automatización se aplica para probar programas implementados en leguajes imperativos y estructurados. Sin embargo, el software desarrollado puede tener acceso a bases de datos mediante código SQL embebido en la aplicación. En este artículo se establece una forma de medir la cobertura de sentencias SQL utilizando el concepto de cobertura de múltiple condición, que se aplica tradicionalmente a los programas imperativos, para su posterior uso en la automatización del proceso de pruebas en software escrito en lenguaje SQL.

1. Introducción

La prueba del software es una parte de los procesos conocidos como Verificación y Validación (V&V)[19] en la ingeniería del software. Estos procesos son caros, típicamente se estima que cuestan al menos el 50% del coste total del software desarrollado. Dentro de estos procesos, expertos especializados o el propio programador diseña de forma manual el conjunto de pruebas unitarias. El uso de técnicas para automatizar partes de estos procesos permite probar el software de una forma más eficiente, reduciendo los tiempos y costes, y aumentando la calidad del producto final. También permite gestionar más eficazmente los casos de prueba diseñados y facilita las pruebas de regresión.

La automatización de la prueba del software, para programas escritos en leguajes imperativos, se puede estudiar desde varios puntos de vista en función del tipo de pruebas que se van a realizar: caja blanca o caja negra [15]. Existen diversos estudios y técnicas aplicados a programas escritos en leguajes imperativos que permiten la creación automatizada de casos de prueba: técnicas aleatorias [16]; técnicas estáticas que utilizan ejecución simbólica [5], programación lógica de restricciones [14], y lenguajes de especificación Z y VDML-SL [12] y pueden utilizar heurísticos [1] que optimicen la generación de casos de prueba; técnicas dinámicas que ejecutan los programas instrumentados previamente [8] tratando de probar la cobertura del código y pueden utilizar técnicas metaheurísticas [3] como algoritmos genéticos aplicables a cobertura de ramas [7][11][18] y a cobertura de caminos [9] y como recocido simulado [21]; por último técnicas que resultan de una combinación de las estáticas y de las dinámicas [17][6].

Las técnicas mencionadas se aplican a programas escritos en lenguajes imperativos. Es bastante común que estos programas tenga componentes con acceso a bases de datos mediante sentencias SQL embebidas en el código. Sin embargo, no hay demasiada información ni trabajos publicados en la bibliografía actual que aborden el tema de las pruebas del software y las bases de datos, y los existentes difieren bastante en sus enfoques.

El principal objetivo de este trabajo es establecer una medida de la cobertura de consultas SQL que pueda ser útil para la automatización de las pruebas.

En el capítulo 2 se indican las características que presenta la realización de pruebas del software en programas con acceso a bases de datos a través del lenguaje SQL, se citan algunos generadores de bases de datos de prueba comerciales y se comenta la bibliografía relacionada con el tema. A continuación, en el capítulo 3, se describe la información a tener en cuenta para medir la cobertura de una consulta SQL y cómo obtener dicha medida. En el capítulo 4 se presenta un caso de estudio en el que se aplica la medida a una consulta SQL y una base de datos real. El capítulo 5 cita las conclusiones obtenidas y los posibles caminos a seguir como líneas de investigación y trabajos futuros.

2. Pruebas de aplicaciones con bases de datos

Hay una serie de características a tener en cuenta en las pruebas del software cuando se llevan a cabo sobre aplicaciones que tienen acceso a bases de datos mediante consultas SQL.

La primera característica es que una misma base de datos va a ser utilizada por varias sentencias SQL, y por tanto, los datos que se diseñen y que posteriormente se carguen en la base de datos deben ser válidos y útiles para todas las consultas que se ejecuten. Las instancias de la base de datos que se utilicen en las pruebas van a determinar la calidad de los tests ya que se deben cubrir las situaciones planteadas en las consultas y evitar que se produzcan resultados no deseados [10].

Además, el código SQL puede tener constantes y ser parametrizado mediante variables que intervienen en las consultas. Por tanto, al igual que sucede con los lenguajes imperativos, es necesario generar los casos de prueba para estas entradas.

Otro punto a tener en cuenta es que, para una consulta SQL, la base de datos es entrada y salida a la vez: las instancias pueden cambiar mediante la ejecución del propio código SQL, por tanto en una etapa de las pruebas pueden influir los resultados de etapas anteriores.

Por último, se pueden establecer subconjuntos de la base de datos y consultas SQL sobre los que realizar parte de las pruebas.

2.1. Generadores comerciales de bases de datos

En el ámbito comercial, existen generadores de bases de datos, tales como TestByte 3, TestBase o DataTect, que ayudan a llevar a cabo la tarea de diseñar las instancias de para bases de datos de prueba. En general, tienen como función generar de forma aleatoria conjuntos de caracteres o números, dependiendo del tipo de dato de los

campos de la base de datos. Las más sofisticadas permiten al usuario definir conjuntos de datos para nombres, ciudades, rangos de valores y valores permitidos o prohibidos. El problema de estos generadores es que requieren información de entrada que debe proporcionar un usuario, el cual debe conocer la estructura de la base de datos.

2.2. Trabajos publicados

En la bibliografía se han encontrado varios trabajos relacionados con la generación de casos de pruebas para bases de datos y aplicaciones con sentencias SQL embebidas pero, generalmente, su principal objetivo es evaluar la eficiencia de los gestores de las bases de datos y no probar las aplicaciones. A continuación se describen brevemente algunos de estos artículos.

Uno de los estudios [20] ha sido desarrollado por el equipo de investigación de Microsoft. Su objetivo es evaluar sistemas gestores de bases de datos generando aleatoriamente sentencias SQL válidas. Estas consultas se ejecutan sobre varios sistemas gestores que mantienen bases de datos idénticas y se comparan los resultados obtenidos.

En otro trabajo [4], a partir de la estructura de la base de datos y sus restricciones de campo, se genera un conjunto de datos válidos e inválidos para intentar rellenar la base de datos. Los objetivos son comprobar si el gestor controla la consistencia de la información y obtener una base de datos inicial automáticamente.

En [2] se presenta el diseño de una herramienta cuyo objetivo es facilitar las pruebas de aplicaciones con bases de datos. Estaría formada por varios componentes que permitirían: generar datos de entrada para una base de datos satisfaciendo restricciones de integridad, ejecutar las consultas, comparar las salidas obtenidas con las esperados y comprobar el estado final de la base de datos. Según se indica, por el momento sólo han desarrollado el primer componente.

Por último, otro estudio presenta cómo poder generar instancias para una base de datos a partir de la semántica de sentencias SQL de un programa [22]. En él se describe una herramienta que genera un conjunto de restricciones que representan propiedades y son utilizadas para generar las instancias.

Como se puede observar, tal vez por las características citadas anteriormente, no hay muchos trabajos relacionados con el tema y, además, los enfoques que cada autor da a su trabajo son bastante diferentes.

3. Cómo medir la cobertura de sentencias SELECT

Para facilitar la tarea de las pruebas de software que incluyan código SQL se plantea como objetivo general construir una herramienta que, a partir de la estructura de la base de datos y de las consultas SQL a probar, genere de forma automática un conjunto de instancias con las que realizar la carga de la base de datos y un conjunto de valores para las variables existentes en el código SQL. Los criterios que se establecerán para obtener las salidas requeridas son, por un lado, minimizar las instancias de la base de datos y, por otro lado, cubrir el mayor número de situaciones de las consultas SQL, es decir tratar de alcanzar la mayor cobertura posible.

En este trabajo se desarrolla un subobjetivo necesario para, en posteriores trabajos, alcanzar el general. Se centra en cómo medir la cobertura de sentencias SQL, en concreto para sentencias SELECT.

La forma propuesta de medir la cobertura se basa en el criterio de cobertura de múltiple condición [15] que tradicionalmente se utiliza con lenguajes imperativos.

Para obtener esta medida se considerarán como entradas la estructura de la base de datos, la información contenida en ella y la sentencia SELECT. Una vez establecido cómo medir la cobertura habrá que evaluar la consulta y, como último paso, interpretar el resultado obtenido.

Debido a la variedad de consultas SQL que se pueden encontrar en una aplicación se ha restringido el estudio a las sentencias SELECT en las que intervienen únicamente campos de tablas de la base de datos.

Para la evaluación de la sentencia SELECT, se extraen las condiciones que se encuentran en la cláusula WHERE y aquellas que formen parte de "JOIN" en la cláusula FROM. Cada una de estas expresiones son niveles de un árbol, llamado árbol de cobertura, como se muestra en la Fig. 1. La expresión de cada nodo se evaluará y, en función de que el resultado sea cierto o falso, continuará la evaluación por una rama u otra del nodo.

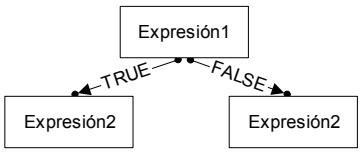

Fig. 1. Estructura del árbol de cobertura.

Cuando los operandos de las expresiones son columnas, la comparación es entre conjuntos de valores. Por tanto, en la evaluación, cada valor en la columna o campo del primer término se debe comparar con cada uno de los valores del segundo término de la expresión. La evaluación del nodo, una vez fijada una instancia del primer término, será:

- TRUE si se cumple la condición con alguna instancia del segundo término. En este caso se fijan las instancias de ambos términos para la evaluación de los nodos inferiores al dado, si los hubiera.
- FALSE si no se cumple la condición con ninguna instancia del segundo término. En este caso se fija únicamente la instancia del primer término para evaluar los siguientes nodos, si los hubiera.
- Se da un valor intermedio, NO TRUE, mientras no se cumpla la condición pero aún queden tuplas del segundo término por evaluar.

La evaluación del árbol terminará cuando no existan más instancias por evaluar o cuando todas las expresiones del árbol hayan tomado valores TRUE y FALSE, en cuyo caso se ha alzando el 100% de la cobertura de la consulta.

4. Caso de aplicación

La base de datos utilizada en el caso de aplicación fue proporcionada por una empresa siderúrgica como base de datos de carga de una aplicación software desarrollada mediante un proyecto de colaboración entre la empresa y la Universidad de Oviedo.

Un departamento de la empresa se dedica a la gestión de los cilindros utilizados en los trenes de laminación de fabricación de chapas de acero. Cada cilindro trabaja siempre en un tren y, en cada uno, los cilindros se organizan en cajas. De esta forma, un tren puede estar formado por una o varias cajas y dentro de cada caja hay varios cilindros. Por otro lado, después de un tiempo de trabajo en los trenes, los cilindros se desgastan y es necesario repararlos para lo que se extraerán de las cajas donde han trabajado y se sustituirán por otros distintos.

En la Fig. 2 se observa el modelo E-R con la información de los trenes, cajas y cilindros, las tablas asociadas a cada entidad con sus claves primarias, y la consulta SQL para obtener la información de todos los trenes, con sus cajas y cilindros.

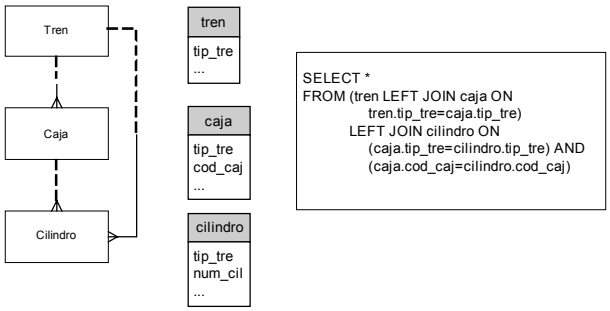

Fig. 2. Modelo E-R, claves primarias y sentencia SELECT de trenes, cajas y cilindros de laminación.

De la base de datos utilizada como carga cabe destacar la gran cantidad de información disponible: 20 trenes; 21 cajas; 1170 cilindros. Al ejecutar la consulta, el número de tuplas resultado de la sentencia SELECT es 1186.

Con la base de datos y la consulta propuestas anteriormente, al medir la cobertura de la sentencia SELECT se obtuvo como árbol de cobertura el que se muestra en la Fig. 3. Como se puede observar, no se alcanza el 100% de cobertura, indicando que todas las cajas, aún teniendo distinto "tipo de tren" que cualquier cilindro, coincide su "código de caja" con el "código de caja" de algún cilindro. Con esta información caben dos alternativas de interpretación de los datos que pueden mejorar los casos de prueba:

- No puede haber cajas cuyo "tipo de tren" y "código de caja" sean distintos a los de algún cilindro, en cuyo caso se debería intentar insertar en la base de datos este caso para comprobar que se controla dicha situación mediante código (disparador, procedimiento almacenado, …).
- Puede haber cajas con "tipo de tren" y "código de caja" distintos simultáneamente a los de todos los cilindros, en cuyo caso se debe insertar este caso que en la base de datos de prueba no existe.

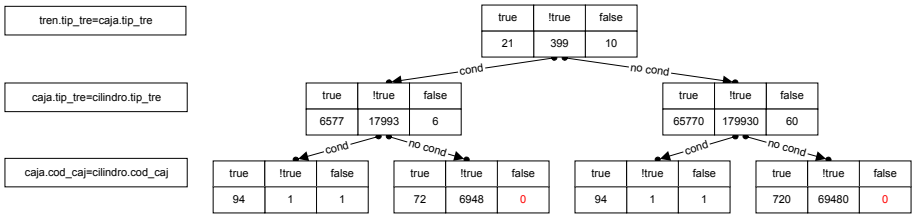

Fig. 3. Árbol de cobertura.

5. Conclusiones y trabajos futuros

Para finalizar, se indican las conclusiones obtenidas. En primer lugar, se ha establecido una medida de la cobertura de consultas SQL para el caso de sentencias SELECT. Como en las medidas de cobertura para lenguajes imperativos, es un indicador que ayuda a mejorar los casos de prueba. En el caso de aplicación planteado se ha detectado que, aunque se disponía de muchos datos reales del sistema en producción, no se llega a cubrir el total de la cobertura por la falta de algunos casos de prueba en la base datos.

En este capítulo se indican varios puntos que se han de tener en cuenta para las próximas líneas de trabajo a desarrollar:

- En primer lugar considerar los valores nulos y las restricciones en los campos de las tablas de la base de datos que, por el momento, no se tienen en cuenta.
- Será necesario seguir realizando más casos de estudio y aumentando la complejidad de las sentencias SQL que se analizan.
- Obtener la información, datos y estructura, de la base de datos, así como las expresiones de las consultas SQL de forma automática.
- Ofrecer al usuario una guía para interpretar la información de salida de forma que sirviera de ayuda para completar la información de la base de datos de prueba.
- Dar nuevas formas para medir la cobertura de sentencias, que podrá basarse en otras medidas tradicionales de la cobertura en lenguajes imperativos, como la cobertura decisión/condición o en otro concepto distinto.

Agradecimientos

Este trabajo ha sido financiado por el Ministerio de Ciencia y Tecnología (España) bajo el Plan Nacional de investigación, Desarrollo e Innovación, proyecto TIC2001-1143-C03-03 (ARGO).

El caso de estudio se ha basado en información proporcionada por el proyecto AITOR, financiado por la Comunidad Económica del Carbón y del Acero (CN-CECA-99-7210PR148) y por Aceralia Corporación Siderúrgica (CN-99-287-B1).

Referencias

[1] Burton, S., Clark, J., Galloway A. and McDermid, J. Automated V&V for high integrity systems: A targeted formal methods approach. 5th NASA Langley Formal Methods Workshop. 2000.

[2] Chays D., Dan S., Frankl, P.G., Vokolos, F.I. and Weyuker, E.J. A frame work for testing database applications. International Symposium on Software Testing and Analysis. ACM SIGSOFT. 2000

[3] Clarke, J., Dolado, J.J., Harman, M., Hierons, R., Jones, B., Lumkin, M., Rees, K. and Roper, M. Can software engineering be reformulated as search problem?. 2001

[4] Davies, R.A., Beynon, R.J.A. and Jones, B.F. Automating the testing of databases. 1st International Workshop of Automated Program Analysis, Testing and Verification. 2000

[5] DeMillo, R. A. and Offutt, A. J. Constraint-based automatic test data generation. IEEE Transactions on Software Engineering, 17(9). 1991.

[6] Ferguson, R. and Korel, B. The chaining approach for software test data generation. ACM Transactions on Software Engineering and Methodology, 5(1). 1996.

[7] Jones, B., Eyres, D. and Sthamer, H. Generating test data for ADA procedures using genetic algorithms. 1st IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems. 1995

[8] Korel, B. Automated software test data generation. IEEE Transactions on Software Engineering, 16(8). 1990.

[9] Lin, J. and Yeh, P. Automatic test data generation for path testing using GAs. Information Sciences 131. 2001.

[10] Mannila, H. and Räihä, K-J. Test data for relational queries. Symposium on Principles of Database Systems. ACM SIGACT-SIGMD. 1986.

[11] McGraw, G., Michael C. and Schatz, M. Generating software test data by evolution. Technical Report RSTR-018-97-01, RST Corporation. 1998.

[12] Meudec, C. Automatic Generation of software tests from formal specifications. Doctoral Thesis, Faculty of Science of The Queen's University of Belfast. 1997.

[14] Meudec, C. ATGen: Automatic test data generation using constraint logic programming and symbolic execution. Software Testing, Verification & Reliability 11(2). 2001.

[15] Myers, G. The art of software Testing. 1977.

[16] Ntafos, S. On random and partition testing. International Symposium on Software Testing and Analysis. ACM SIGSOFT.1998.

[17] Offutt, A. J., Jin, Z. and Pan, J. The dynamic domain reduction procedure for test data generation. Software-Practice and Experience, 29(2). 1999.

[18] Pargas, R.P., Harrold, M.J. and Peck, R.R. Test data generation using genetic algorithms. The Journal of Software Testing, Verification and Reliability, 9. 1999.

[19] Pressman, R.S. Ingeniería del Software. Un enfoque práctico. 4ª Edición. McGraw Hill. 1997.

[20] Slutz, D. Massive stochastic testing of SQL. 24th International Conference on Very Large Databases. 1998.

[21] Tracey, N., Clark, J. and Mander, K. Automated program flaw finding using simulated annealing. International Symposium on Software Testing and Analysis. ACM SIGSOFT. 1998.

[22] Zang, J., Xu, C. and Cheung, S. C. Automatic generation of database instances for white-box testing. 25th International Computer Software and Applications Conference. 2001.