Constraint Generation Tool for White-box Testing

Kshamata Shenoy[1], Madhuri Rao[2], S. S Mantha[3]

[1]Assistant Professor, BabaSaheb Gawde Institute of Technology, Mumbai, India

[2]Assistant Professor, Department of Information Technology Thadomal Shahani College Mumbai

[3]Chairman AICTE Prof CAD/CAM, Robotics VJTI College Mumbai. India

Abstract: ***Testing of database application is crucial for ensuring high software quality as undetected faults can result in unrecoverable data corruption. Conventionally database application testing is based upon whether or not the application can perform a set of predefined functions. While it is useful to achieve a basic degree of quality by considering the application to be a black box in the testing process white box testing is required for more thorough testing. However the semantics of the structural query language (SQL) statements embedded in database application are rarely considered in conventional white box testing techniques. In this paper we study the generation of constraints that respect the semantics of SQl statements embedded in a database application program. We have described a tool which generates a set of constraints. Database instance for program testing can be derived by solving the set of constraints using existing constraint solvers.***

Keywords: Database Application, SQL, Constraint Satisfaction Problem

1. Introduction

Database application program play an important role in commercial systems. These programs interact with a database system to realize a predefined logic in manipulating business data. While database application program realize application logic in some host language, database systems provide mechanisms for efficient access to and manipulation of massive volume of data. Database programs are often expected to exhibit high reliability. Faults if occurring in a database program can result in unrecoverable data corruption. Since not all database transactions can be unrolled, restoring data from backups cannot eradicate the errors.

Testing of database application is crucial for ensuring high software quality as undetected faults can result in unrecoverable data corruption. The problem of database application testing can be broadly partitioned into the problems of test cases generation, test data preparation and test outcomes verification. Among the three problems, the problem of test cases generation directly affects the effectiveness of testing.

Basically, a CSP is a problem composed of a finite set of variables, each of which is associated with a finite domain, and a set of constraints that restricts the values the variables can simultaneously take. The task is to assign a value to each variable satisfying all the constraints

CSPs are mathematical problems defined as a set of objects whose state must satisfy a number of constraints or limitations. CSPs represent the entities in a problem as a homogeneous collection of finite constraints over variables, which are solved by constraint satisfaction methods. CSPs are the subject of intense research in both artificial intelligence and operations research, since the regularity in their formulation provides a common basis to analyze and solve problems of many unrelated families. CSPs, require a combination of heuristics and combinational search methods to be solved in a reasonable time.

2. Testing Database Application

Database application programs play a central role in our information based society, so testing whether they behave correctly is of great importance. Because of the huge space of possible database states that must be considered, these programs pose new challenges to software testing. On the other hand, the structure of this state space and constraints on legitimate states expressed in the database schema and in additional business rules, offer an opportunity for automatic generation of tests.

Test Case Generation
Database management systems are widely used in many applications. The data stored in the databases is an important corporate asset and it is therefore important that the database system is error-free and stable. Many applications rely on the DBMS for actual implementation as well. The best way to ensure the reliability of this is to do DBMS testing regularly.

Testing can also help eliminate bugs early on and save a lot of time in implementing the system. DBMS testing, in general, is a labor intensive, time-consuming process, often performed manually. Automating DBMS testing not only reduces development costs, but also increases the reliability in the developed systems. White box testing techniques like statement testing, branch testing, condition coverage and path testing [11] can be employed to test some portions of or whole database application to attain a more complete testing of the application. White box testing lets testers examine the code in detail and make sure that at least a certain degree of test coverage such as execution of every statement has been achieved [11]

Test Data Preparation
A System is programmed by its data. Functional testing can suffer if data is poor, and good data can help improve functional testing. Good test data can be structured to improve understanding and testability. Its contents, correctly chosen, can reduce maintenance effort and allow flexibility.

Licensed Under Creative Commons Attribution CC BY

Preparation of the data can help to focus the business where requirements are vague.

Test Outcome Verification:

Verification and validation is the process of checking that a product, service, or system meets specifications and that it fulfills its intended purpose. Verification is a Quality control process that is used to evaluate whether or not a product, service, or system complies with regulations, specifications or conditions imposed at the start of a development phase. Verification can be in development, scale-up, or production. This is often an internal process.

Validation is a Quality assurance process of establishing evidence that provides a high degree of assurance that a product, service, or system accomplishes its intended requirements. This often involves acceptance of fitness for purpose with end users and other product stakeholders.

3. Basics of SQL

In this section, we overview some basic concepts of database application programs. Readers are referred to [4,9] for more details.

A. The Relational Model

Its central idea is to describe a database as a collection of predicates over a finite set of predicate variables, describing constraints on the possible values and combinations of values. The content of the database at any given time is a finite (logical) model of the database, i.e. a set of relations, one per predicate variable, such that all predicates are satisfied. A request for information from the database (a database query) is also a predicate.

The purpose of the relational model is to provide a declarative method for specifying data and queries. We directly state what information the database contains and what information we want from it, and let the database management system software take care of describing data structures for storing the data and retrieval procedures for getting queries answered.

Table 1: Information About Employees

Eid	Name	Age	Gender	Position	Salary
1	Suresh	38	Male	President	67000
2	Sakshi	36	Female	Secretary	45000
3	Vaibhav	22	Male	Manager	35000

Example 1. An Employee table may have the attribute eid, name, age, gender, position, etc. Each row gives information about a particular employee. A specific Employee table is given as Table1.

The relational model is closely related to the first-order predicate calculus. It is known that the relational algebra with the basic operators such as union and join has the same expressive power as first-order predicate logic [4]. In this paper, we focus on a subset of SQL and formulate it using the many-sorted predicate calculus. In the calculus, a table is treated as a sort which consists of all the rows of the table. Each column corresponds to a function. Let us look at Table1. We can consider employee as a set of three elements: el, e2 and e3.

The functions are defined as follows:
$eid(e_1) = 1$, $eid(e_2) = 2$, $eid(e_3) = 3$;
$Age(e_1) = 38$ $Age(e_2) = 36$ $Age(e_3) = 22$;

B. The Query language

Although we refer to the SQL language as a "query language," it can do much more than just query a database. It can define the structure of the data, modify data in the database and specify security constraints.

In relational databases, data are retrieved using SQL (Structured Query Language). A query statement of SQL maps one or several input tables into a result table. The basic structure of an SQL expression consists of three clauses: select, from and where.

The **select** clause corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the result of a query.

The **from** clause corresponds to the Cartesian-product of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.

The **where** clause corresponds to the selection predicate of the relational algebra. It consist of a predicate involving attributes of the relations that appear in the from clause.

A typical SQL query has the form

```
Select t1.a1, t2.b2
From t1, t2
Where a1=b1;
```

Example 1. (Cont'd.) The following is a simple query statement:

```
SELECT Eid
FROM Employee
WHERE position = 'manager' and
Gender. = 'female';
```

The result table has only one tuple whose Eid is 3.
More complicated queries may involve multiple tables. Other conditions (or even sub queries) can be added to the WHERE clause.

4. Database Instance Generation

A. The Problem

To test a program with embedded SQL statements, we need to find appropriate input data, which include a database instance. The conventional white-box testing methods are inadequate because the execution of the program is affected by the result of SQL statements. Let us consider the code fragment in Section III ***C***. The loop is not executed if the query results in an empty table. For the purpose of white-box testing, we need input data such that the loop is executed once or not executed at all. Thus, test data generation for database applications necessitates the tackling of the following general problem.

Licensed Under Creative Commons Attribution CC BY

Given an SQL statement and a property, find a database instance such that the result of executing the statement satisfies the property.

We understand that test data generation also comprises finding the values of ordinary program variables. But in this paper, we would like to focus on the database-specific aspects and the generation of a database instance.

We refrain from giving a language to describe the properties. The following are some examples of them:

(Pl) The result table is empty, i.e., it does not have any row.
(P2) The result table has a row which has a negative attribute value.

(Pl) and (P2) represent the scenario of null case and exception, respectively. These are two common properties that practitioners used to test database application programs. Let us denote the result table by Rst. Without loss of generality, we assume that it has a single column. The above properties can be represented by the following two formulas in the syntax of SQL:-

(p1) NOT EXISTS Rst

(p2) 0 > ANY Rst

To solve a CSP, we need to find the values of the variables such that all the constraints are satisfied. For the purpose of test data generation, we use a formula to describe the relationship between the input (database instance) and the output (result table). Solutions satisfying the formula can be computed using existing techniques and tools available in the CSP community.

For simplicity, we assume that there are two tables (tl and t2) and the attributes are denoted by a_0, a1, a2....... The selected attribute is a_s. Suppose table t_1 contains the rows t_{1i} (1<=i<=m) and table t_2 contains the rows t_{2j} (1<= j<= n). Let Cond denote the condition in the WHERE clause. We first consider the simpler case where condition does not involve sub queries. Our goal is to determine the values of the following entries such that some formula holds.

- $t_{1i}.a_k$, for each row i. and each attribute a_k of t_1
- $t_{2j}.a_l$, for each row j and each attribute a_l of t_2

The formula is generated based on an SQL statement and a property.

For example,

- If the property is "EXISTS Rst", the formula is $V_{i,j}$ $C^1(i, j)$;

- If the property is "const rop ALL Rst" (const is a numeric constant, rop is a relational operator such as >=), the formula is C^1 (i,j)→(const rop a^1_s)).

Here C^1 (i.j) is derived from Cond by changing t_1.ak to t_{1i}.ak and changing $t_2.a_l$ to $t_{2j}.a_l$. Note that $t_1.a_k$ and $t_2.a_l$ are used in the SQL statement while $t_{1i}.a_k$ and $t_{2j}.a_l$ denote unknowns of the constraint satisfaction problem. If the selected attribute a_s is from table t1, a^1_s is $t_{1i}.a_s$; and if a_s is from table t_2, is $t_{2j}.a_s$. If both t1 and t2 have the attribute a_s, the user should specify explicitly which one is to be selected.

B Constraint Solving

It is fairly easy to see that a CSP can be given an incremental formulation as a standard search problem as follows:

- **Initial state**: The empty assignment, in which all variables are unassigned.
- **Successor function**: A value can be assigned to any unassigned variable, provided that it does not conflict with previously assigned variables.
- **Goal test:** The current assignment is complete.

The input given to the tool consists of a Schema, SQl statement and an assertion and the required output is a set of constraints.

The constraints generated are further verified in constraint generation tool to see if they are satisfiable or not. If they are satisfiable, the desired database instances are obtained. This can be used for white-box testing which will assist the tester significantly.

Schema

The schema of a database system is its structure described in a formal language supported by the database management system (DBMS). In a relational database, the schema defines the tables, the fields, relationships, views, indexes, packages, procedures, functions, queues, triggers, types, sequences, materialized views, synonyms, database links, directories, Java, XML schemas, and other elements.

The schema is given as follows

Create table t1
(a1 int,a2 float,a3 int) [1];

Create table t2
(b1 int ,b2 int) [2];

There are two tables having 3 attributes and 2 attributes, respectively. We have specified that the first table has one row and the second table has two rows. They can be changed as per our requirements

Basic structure of SQL Queries

The basic structure of an SQL expression consists of three clauses: select, from and where.

The **select** clause corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the result of a query

The **from** clause corresponds to the Cartesian-product of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.

The **where** clause corresponds to the selection predicate of the relational algebra. It consist of a predicate involving attributes of the relations that appear in the from clause.

A typical SQL query has the form

```
Select t1.a1, t2.b2
From t1, t2
Where a1=b1;
```


Licensed Under Creative Commons Attribution CC BY

Assertion

An assertion is any condition that the database must always satisfy. Domain constraints and referential-integrity constraints are special forms of assertions. However there are many constraints that we cannot express by using only these special forms.

For example, "every loan has at least one customer who maintains an account with a minimum balance of $1000.00" must be expressed as an assertion. When an assertion is created, the system tests it for validity. If the assertion is valid, the any future modification to the database is allowed only if it does not cause that assertion to be violated.

An assertion is a predicate expressing a condition that we wish the database to be satisfied.

An assertion in SQL takes the form:

Create assertion < assertion-name > **check** < predicate>

Since SQL does not provide a "for all X, P(X)" construct (where P is a predicate), we are forced to implement the constraint by an equivalent construct, "not exist X such that not P(X)", that can be expressed in SQL.

When an assertion is created, the system tests it for validity. If the assertion is valid, then any future modification to the database is allowed only if it does not cause that assertion to be violated. This testing may introduce a significant amount of overhead if complex assertions have been made.

Assertion:

Exists Result

If result exists we can give an assertion as exists result.

C. An Automatic 'Tool' for Constraint Generation

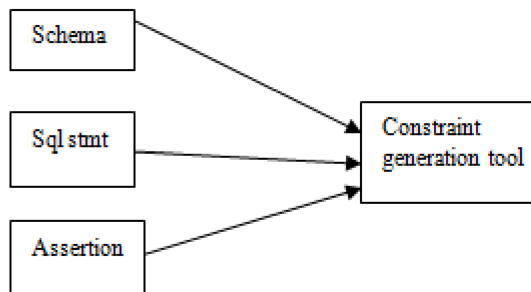

Figure 1: Constraint generation tool for white box testing

An automatic tool for generating constraints is as shown in the figure 1. The input of the tool consists of three parts: schema, SQL statement and assertion. The output is a set of constraints (which can be given to BONUS).

5. Results

Example 1. The input is like this:

Schema:

```
create table tblTest3(nNum int, vcName varchar(50))[1];
create table tblTest4(nNum int, vcName varchar(50))[2];
create table tblTest5(nNum int, vcName varchar(50))[3];
```

Sql statement:

```
select *
from tblTest3
inner join tblTest4
 on tblTest3.nNum = tblTest4.nNum
inner join tblTest5
 on tblTest4.nNum = tblTest5.nNum;
```

Assertion:

Exists Result

There are three tables having 2 attributes respectively. The first table has one row and the second table has two rows and the third table has three rows. They can be changed as per our needs. For instance we may look for two tables each having three rows. The keyword RESULT denotes the result of the query. In this example we are looking for three tables such that the result table is not empty. With this tool we get the following set of constraints:

Output:

```
int tblTest3_0_nNum;
varchar (50) tblTest3_0_vcName;
int tblTest4_0_nNum;
int tblTest4_1_nNum;
varchar(50) tblTest4_0_vcName;
varchar(50) tblTest4_1_vcName;
int tblTest5_0_nNum;
int tblTest5_1_nNum;
int tblTest5_2_nNum;
varchar(50) tblTest5_0_vcName;
varchar(50) tblTest5_1_vcName;
varchar(50) tblTest5_2_vcName;
bool b0 = (tblTest3_0_nNum = tblTest4_0_nNum);
bool b1 = (tblTest3_0_nNum = tblTest4_1_nNum);
bool b2 = (tblTest4_0_nNum = tblTest5_0_nNum);
bool b3 = (tblTest4_0_nNum = tblTest5_1_nNum);
bool b4 = (tblTest4_0_nNum = tblTest5_2_nNum);
bool b5 = (tblTest4_1_nNum = tblTest5_0_nNum);
bool b6 = (tblTest4_1_nNum = tblTest5_1_nNum);
bool b7 = (tblTest4_1_nNum = tblTest5_2_nNum);
{
AND
(OR (b0, b1),OR (b2, b3, b4, b5, b6, b7)
)
}
```

The above set of constraints has solutions which can be found by any constraint solvers.

6. Comparative Study

SQL is a ubiquitous language used in a wide range of applications for accessing the data stored in relational databases. However, the usual software testing techniques are not designed to address some important features of SQL.

Although many software testing techniques have been proposed and adopted in day to day industrial practice, these are not specifically tailored for handling the particularities of the Structured Query Language (SQL).Our constraint generated tool will be a definite advantage as a requirement for the tester. Database instances can then be generated then by solving these constraints.

Licensed Under Creative Commons Attribution CC BY

7. Conclusion

The testing of database application programs has not received much attention previously. In particular, few test data generation techniques consider the inclusion of database instances, which can be used for white-box testing. A prototype tool is also described. Its input consists of an SQL statement, the database schema definition, together with an assertion which represents the requirement of the tester. The output is a set of constraints which can be given to existing constraint solvers. If they are satisfiable, we obtain the desired database instances.

Certainly it is impossible to work out a fully automatic tool for test data generation, even when the program does not involve databases. But it is reasonable to expect that a powerful tool will assist the tester significantly. Currently the tool has some limitations. For instance it does not handle string variables. We need to represent them as integer instead.In the future we will improve the constraint generation tool described in this paper.

References

[1] B. Beizer, Softvare Testing Techniques (Second Edition) Van Nostrand Reinhold International Company Limited, New York, 1990. Chapter 1, pages 8, 10-1 1, 20-22,546,550.
[2] M.Y. Chan and S.C. Cheung, Testing Database aplications with SQL semantics, Proc. of the *2nd* Int'l Synzp. *On* Cooperative Database Systems for Advanced Applications (CODAS'99), 364-375, March, 1999.
[3] D. Chays, S. Dan, P.G. Frankl, F.I. Vokolos and E.J. Weyuker, A framework for testing database applications,Proc. Int'l Symp. on Software Testing and Analysis (ISSTA'OO), 147-157, August 2000.
[4] E.F. Codd, The Relational Model for Database Management:Version 2, Addison-Wesley, Reading, Mass., USA, 1990.
[5] P. G. Jeavons, D. A. Cohen, M. Gyssens,Closure Properties of Constraints, Journal of the ACM, 44(4): 527- 548, July 1997.
[6] B. K. Patel, Automated tools for database design and criteria for their selection for aerospace applications, IEEE Aerospace Applications Conference Digest,.
[7] M.Roper, SofhYare Testing, New York**,** TheMcGraw-Hill, Inc., 1994. Chapters 1-3, pages 32-96.
[8] E.Tsang, Foundations of Constraint Satisfaction Academic Press, London, 1993. Chapters 1, 2,6, 10,pages 1- 52,1577188,299.
[9] J.D. Ullman and J. Widom, A First Course in Database Systems, Prentice Hall, 1997.
[10] J. Zhang, Specification analysis and test datageneration by solving Boolean combinations of numeric constraints, Proc. of the first Asia-PaciJic ConJ: on Quality Software (APAQS), (eds.) T.H.Tse and T.Y.Chen, 267- 274,2000.
[11] Marc Roper, *Software Testing*, The McGraw-Hill Companies, Inc., 1994.
[12] M.Y.Chana and SC Cheung.Testing database applications with SQL semantics. In Proceedings of the 2nd International Symposium on Cooperative Database Systems for Advanced Applications (CODAS '99), pages 363–374. Wollongong, Australia, 1999.
[13] G.M. Kapfhammer and M. L. Soffa. A family of test adequacy criteria for database-driven applications. In Proceedings of the Joint 9th European Software Engineering Conference and 11th ACM SIGSOFT Symposium on the Foundation of Software Engineering (ESEC 2003 / FSE-11), pages 98–107. ACM Press, New York, 2003.
[14] R.A. Haraty, N. Mansour, and B. Daou. Regression testing of database applications. Journal of Database Management, 13 (2): 31–42, 2002.
[15] R.A. Haraty, N. Mansour, and B. Daou. Regression test selection for database applications. In volume 3 of Advanced Topics in Database Research, K. Siau (editor), pages 141–165. Idea Group, Hershey, Pennsylvania, 2004.

Licensed Under Creative Commons Attribution CC BY