Software Quality achieved through Coverage Metrics in Database Testing

Ms.A.Askarunisa [1]
Ms.P Prameela [2]
Dr.N.Ramaraj [3]

Thiagarajar College of Engineering, Madurai
Thiagarajar College of Engineering, Madurai
GKM Engineering College, Chennai
[1] (aacse)@tce.edu
[2] (prameela)@tce.edu
[3] nishanazer@yahoo.com

Abstract. Code coverage is often defined as a measure of the degree to which the source code of a program has been tested. Various metrics for measuring code coverage exist.The number of defects is an important measure of software quality which is widely used in industry. Software test coverage tools can easily and accurately measure the extent to which the software has been exercised .Both testing time and test coverage can be used as measures to model the defect finding process. However test coverage is a more direct measure of test effectiveness and can be expected to correlate better with the number of defects found.In majority of software applications,Database systems play an important role. In literature there is good amount of work done in testing database applications [3, 5, and 6]. The levels of quality, maintainability, testability,and stability of software can be improved by performing measures of the testing process and calculating its metrics. In most of the applications coverage metrics plays a major role in predicting the quality of the software.The accuracy of database testing is improved by calculating the coverage percentage of the frequently used SELECT statements.This paper proposes two algorithms for measuring coverage of the SQL queries. An analysis of the two algorithms also performed as to which metric is better so that it would help the test manager to take effective decisions.Our work in this paper details on importance of coverage metrics required to achieve quality of a database application and presents the various coverage metrics that are used for achieving the testing efficiency of a database application. We have considered six different database applications and calculated the various coverage metrics thereby achieving quality and efficiency in testing.

Keywords: Software Metrics, Database Testing, Test Cases, Coverage Tree, Command form metrics

(Received April 01, 2009 / Accepted July 04, 2009)

1 Introduction

Testing determines the validity of the computer solution to a business problem. Testing is used as the demonstration of the validity of the software at each stage in the system development life cycle. Database systems have major important and wide popularity among the worlds software industry. As a result they are becoming very complex.

Measurement plays a critical role in effective software development. It provides the scientific basis for software engineering to become a true engineering discipline. As the discipline has been progressing toward maturity, the importance of measurement has been gaining acceptance and recognition. The increase in both

application complexity and reliability expectation has contributed to great demands on software testing activities. Efficient and effective software testing is crucial within the software development and maintenance cycle. In Database testing metrics provide information to support a quantitative managerial decision-making. Among the various metrics like cost, execution time, reliability, complexity etc., Coverage metric is considered as the most important metric often used in software industries for testing [7]. Coverage analysis helps in the testing process by finding the areas of a database not exercised by a set of test cases, creating additional test cases to increase coverage, and determine the quantitative measure of the database. This metric indirectly helps in measuring the quality of the test process. The test manager uses this coverage metric in making decisions while selecting test cases for testing of the software . To ease the job of the database test manager and to achieve quality in testing databases, this paper proposes two algorithms that calculate the coverage of the database SELECT queries. An analysis between the two metrics is also performed and results obtained give an idea of which metrics would be better to the test manager.
In section2 review of the related work in database testing is detailed. Section3 describes the testing a database application. In section4 describes the measurement of coverage. Section5 details the implementation of coverage metric in testing database applications. Section6 described the conclusion and future work.

2 Related Work

In [1], Yuetang Deng et. al. has performed testing of database applications In their work, in order to check a state constraint that is not enforced by the DBMS, a tool named AGENDA (A (test) GENerator for Database Applications)creates temporary tables to store the relevant data and converts the assertion into a check constraint at attribute/row level on the temporary tables. In particular, constraints involving aggregation functions, constraints involving multiple tables, and dynamic constraints involving multiple database states are transformed into simpler constraints on temporary tables, and code to automatically insert relevant values into the temporary tables is generated and executed .

As an extension, in [2] , Eric Tang, Phyllis G. Frankl, Yuetang Deng extended AGENDA to aid in testing database applications. In addition to the tools that AGENDA currently had three additional tools were made to enhance testing and feedback [2]. They are the log analyzer, attribute analyzer, and query coverage.

In [3] , Gregory M. Kapfhammer, Mary Lou Soffa's work, a family of test adequacy criteria are used to assess the quality of test suites for database driven applications [3]. A unique representation of a database-driven application that facilitates the enumeration of database interaction associations was developed. These associations can reflect an application's definition and use of database entities at multiple levels of granularity.

In [5], dataflow and control flow analysis and the dependences between components of a database application were used to determine the components that should be tested when any change was produced and to minimize the set of test cases during regression testing, but its aim was not to design test cases.In [9], an automated tool that provides the measurement of the coverage of SQL queries is presented.In [8], an analysis that computes the corresponding testing requirements and an efficient technique for measuring coverage of these requirements is detailed.

Our work in this paper details on importance of coverage metrics required to achieve quality of a database application and presents the various coverage metrics that are used for achieving the testing efficiency of a database application. We have considered six different database applications and calculated the various coverage metrics thereby achieving quality and efficiency in testing.

3 Testing a Database Application

A statement of Structured Query Language(SQL) a powerful declarative query language, is embedded in programs written in any general-purpose language. This is referred to as the host language [5].

In most of the applications, queries and modifications of data are written as the embedded SQL statements. Other functionality like interacting with users and sending results to a graphical user interface is written in GPL (General Purpose Language). In this paper, we concentrate on testing this database application only and not the host language. Testing of database applications is different from the testing of structural programs like C, C++, java etc. [6].

The inputs for database applications involve both the user inputs and the database instances. In addition to checking the outcome with the expected result, programmers or testers should also check if the database is consistent and reflects the original environments during database application testing [4]. The steps involved in database application testing are the problems of Test Cases Generation, Test Data Preparation, and Test Execution and Output Verification. The tests should cover all the query situations and avoid producing undesired results so as to obtain their maximum possible coverage.

Problem Statement

The main aim of the paper is to:

- Define a measurement of coverage of SQL SELECT queries in relation to a database loaded with test data that can be used as an adequacy criterion to carrying out the testing of applications with access to databases.
- Present algorithms that calculate the coverage in order to help the software tester/manager.
- Extract a subset of database information that allows the obtainment of at least the same result as the original set for the established adequacy criterion

4 Software Metrics

Software metrics is defined as the current state of art in the measurement of software products and process [13].Measurement is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined unambiguous rules. Metrics strongly support software project management activities mainly test management. They relate to the four functions of management as follows:

1. Planning - Metrics serve as a basis of cost estimating, training planning, and resource planning, scheduling, and budgeting.
2. Organizing - Size and schedule metrics influence a project's organization.
3. Controlling - Metrics are used to status and track software development activities for compliance to plans.
4. Improving - Metrics are used as a tool for process improvement and to identify where improvement efforts should be concentrated and measure the effects of process improvement efforts.

4.1 Importance of Metrics

Deriving metrics in every phase of the SDLC has a major importance through out the life cycle of Software for management, Managers, Developers and Customers [14].

4.1.1 Metrics in Project Management

1. Metrics make the project's status visible.Managers can measure progress to discover if a project is on schedule or not.
2. Metrics focus on activity. Workers respond to objectives, and metrics provide a direct objective for improvement.
3. Metrics help to set realistic expectations. By assisting in estimation of the time and resources required for a project,metrics help managers set achievable targets.
4. Metrics lay the foundations for long-term improvement. By keeping records of what happens on various projects, the beneficial activities can be identified and encouraged, while the detrimental ones are rejected.
5. Metrics also help the management in reducing various resources namely people, time and cost in every phase of SDLC.

4.1.2 Metrics in Decision Making Metrics will not drive a return on investment unless managers use them for decision making. Some decisions in which software metrics can play a role include

- Product readiness to ship/deploy,
- Cost and schedule for a custom project,
- How much contingency to include in cost and schedule estimates?
- Where to invest for the biggest payback in process improvement, and
- When to begin user training.

Managers should demand supporting metrics data before making decisions such as these. For example, they can use fault-arrival-and-close-rate data when deciding readiness to deploy. Knowing the Overall project risk through metrics can help managers decide how much contingency to include in cost and schedule estimates.

4.2 Procedural (Traditional) Software Metrics

Support decision making by management and enhance return on the IT investment [15]. Once business goals have been identified, the next step is to select metrics that support them. Various types of Metrics [16] found in literature are:

Lines of Code namely Total Lines of Count(TLOC), Executable Lines of Count (ELOC),Comment Lines of Count (CLOC)

Hallstead's Metrics namely Program Length (N), Program Volume(V),Effort to Implement(E),Time to Implement (T),Number of Delivered Bugs (B).

Function point (FP) [17] is a metric that may be applied independent of a specific programming language, in fact, it can be determined in the design stage prior to the commencement of writing the program.

4.3 Object Oriented Metrics:

The most commonly cited software metrics to be computed for software with an object-oriented design are those proposed by Chidamber and Kemerer [17],[18]. Their suite of metrics consists of the following metrics: weighted methods per class, depth of inheritance tree, number of children, coupling between object classes, response for a class, and lack of cohesion in methods.

4.4 Coverage Metrics:

To measure how well the program is exercised by a test suite, coverage metrics are used. [14, 15] There exists a number of coverage metrics in literature. Following are descriptions of some types of coverage metrics.

Statement Coverage
This metric is defined as"The percentage of executable statements in a component that have been exercised by a test case suite."

Branch Coverage
This metric is defined as"The percentage of branches in a component that have been exercised by a test suite."

Loop Coverage
This metric is defined as"The percentage of loops in a component that have been exercised by a test suite."

Decision Coverage
This metric is defined as"The percentage of Boolean expressions in a component that have been exercised by a test suite." [19]

Condition Coverage
This metric is defined as"The percentage of decisions in a component that have been exercised by a test suite." [20]

Function Coverage
This metric is defined as"The percentage of functions in a component that have been exercised by a test suite."

Path Coverage
This metric is defined in [10] as"The percentage of paths in a component that have been exercised by a test suite."

5 Coverage Measurement for a Database

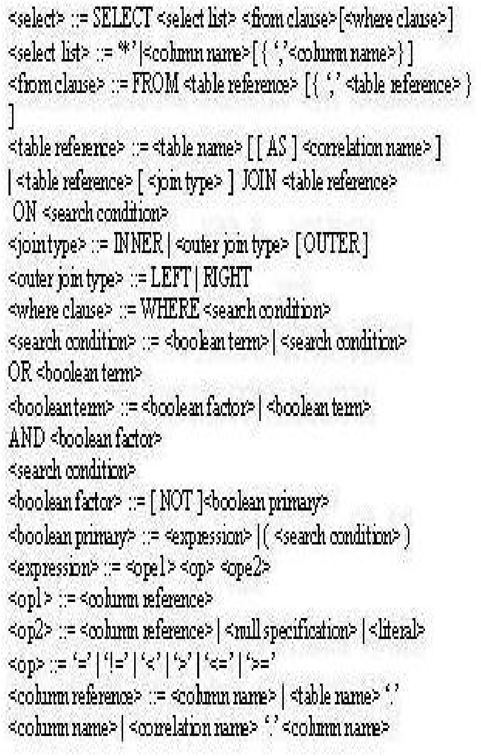
Various coverage's mentioned in section 4 are mostly applicable to software programs written in high level languages. These coverage metrics cannot be calculated for database applications. Hence there is a requirement of other new metrics exclusively for databases.

The approach proposed in this paper is to establish a way of measuring the coverage of an SQL query based on the coverage concept whereby the conditions take into account the true and false values during the explorations of their different combinations.

This paper proposes two algorithms that calculate the metrics exclusively for database applications viz. Coverage tree and command form. We have conducted experimental analysis in comparing the two coverage metrics for various database applications A database application contains many queries viz. DDL (Data definition language),DML (data Manipulation Language) statements and the various SELECT queries. In our work, we have concentrated only on testing and finding coverage's for SELECT queries.
The BNF notation for a SELECT query is as shown in Figure 1

5.1 Coverage Tree Metric

The algorithm to calculate the coverage tree metric, searches for SQL query situations covered with



```
<select> ::= SELECT <select list> <from clause> [<where clause>]
<select list> ::= '*' |<column name> [ { ','<column name> } ]
<from clause> ::= FROM <table reference> [ { ',' <table reference> }
]
<table reference> ::= <table name> [ [ AS ] <correlation name> ]
| <table reference> [ <join type> ] JOIN <table reference>
 ON <search condition>
<jointype> ::= INNER | <outer join type> [OUTER ]
<outer join type> ::= LEFT | RIGHT
<where clause> ::= WHERE <search condition>
<search condition> ::= <boolean term> | <search condition>
OR <boolean term>
<booleanterm> ::= <boolean factor> | <boolean term>
AND <boolean factor>
<search condition>
<boolean factor> ::= [ NOT ]<boolean primary>
<boolean primary> ::= <expression> | ( <search condition> )
<expression> ::= <ope1> <op> <ope2>
<op1> ::= <column reference>
<op2> ::= <column reference> | <null specification> | <literal>
<op> ::= '=' | '!=' | '<' | '>' | '<=' | '>='
<column reference> ::= <column name> | <table name> '.'
<column name> | <correlation name> '.' <column name>
```

Figure 1: Simplified BNF grammar of SELECT query

the data stored in the database. It evaluates the conditions of SELECT queries that are in the FROM clause, when they include JOIN, and in the WHERE clause. Moreover, the null values will be verified the same time as the conditions are evaluated. The flow of coverage algorithm is shown in Figure2.

In Figure 2, from the tables present in the database, SELECT queries are generated based on the reports required by the Test Manager. The SELECT queries are issued to the condition evaluation block where the conditions are evaluated based on the number of

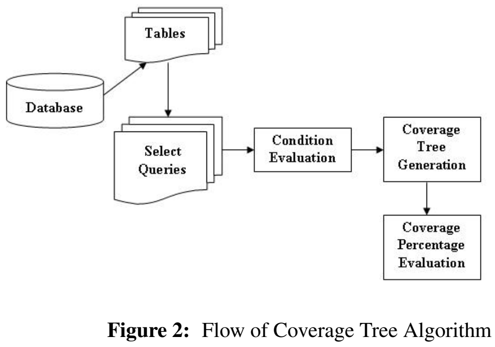


Figure 2: Flow of Coverage Tree Algorithm

conditions and the coverage tree constructed. The coverage percentages of the SELECT queries are calculated by the coverage percentage evaluation block.

The output obtained by the process is

The percentage of coverage of the SELECT query is determined using the coverage tree, achieving 100% coverage if all possible situations have been verified at a particular time.

5.1.1Condition Evaluation

Conditions are evaluated between sets of values, where the information in each field corresponds to a column from a table and several rows in the database as shown in Figure3 than between a single pair of values.

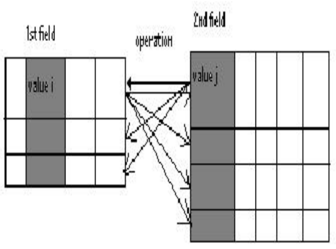


Figure 3: Operation between values of two fields.

5.1.2 Coverage Tree construction

A tree structure, called Coverage Tree (CT), is created prior to coverage evaluation, in which each level represents a condition of the query beginning with the conditions of the JOIN clause, if it exists, and then with those of the WHERE clause, in the same order in which they are found in the query. The node structure of the coverage tree is shown in below Fig 4.

Figure 4: Structure of node in a Coverage Tree

NL	NB	NR	FL	T	FR

Where:

NL - NULL LEFT
NR - NULL RIGHT
NB - NULL BOTH
FL - FALSE LEFT
FR - FALSE RIGHT
T - TREE NODE

The condition evaluation is as follows:

- A condition will be true if it is verified for a pair of values from the fields to compare. It is the same result if the condition is evaluated from left to right or from right to left.

- A condition will be false from left to right, Fl, if none of the values from the second field verifies the condition with a value from the first field. While there are values from the second field for comparing and the condition remains false, the evaluation is considered temporarily NOT true, because it is not true, although it is not yet known whether it is false.

- A condition will be false from right to left, Fr, if none of the values from the first field verifies the condition with a value from the second. As in the previous case, while there are values from the first field for comparing and the condition remains false, the evaluation is considered temporarily NOT true.

- A condition will have null values when a value from the first field is null, Nl, when a value from the second field is null, Nr, or when both values, from the first and second fields, are null, Nb.

5.1.3 Coverage evaluation from coverage Tree

The complete evaluation of the query is carried out by crossing over the tuples of the tables that participate in the conditions at each level of the coverage tree. The evaluation finishes when the entire tree has been covered, i.e. 100% coverage has been covered, or when there are no more values for comparing. For each particular node, the condition is evaluated for a tuples from the first field and another from the second, and:

- If the result is true, these tuples are fixed in order to evaluate the conditions of the lower levels of the tree via the T branch.

- If the result is false from left to right, only the tuples from the first field is fixed and, if it is false from right to left, the tuples from the second field is fixed, in order to evaluate the lower levels of the tree, via the branch at which the condition is false, Fl or Fr respectively.

From the evaluation the coverage measures are established and automatically calculated based on Theoretical coverage.

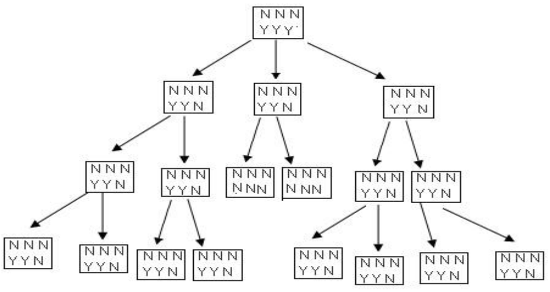


Figure 5: Sample Coverage Tree

Theoretical coverage:

This coverage takes into account all possible values present in every node of the coverage tree.

The percentage of theoretical coverage is calculated using the formula shown in equation (3).

$$\text{Coverage} = \frac{V*(s-1)}{P*(s^n-1)} * 100 \qquad (3)$$

V: number of cases (elements of a node) that it has been possible to verify (those marked with Y).
s: number of child-nodes that a node can have.
P: number of possible values that a condition can adopt once it is evaluated, which in the coverage measurement presented here will have six values (Nl, Nr, Nb, T, Fl, Fr).
n: number of levels of the coverage tree; i.e. the number of conditions in the query

The application of the above SQL coverage measurement is described below with a real database and an SQL query. We have used three databases namely Employee(EMP), Department(DEPT) and Location. The tables corresponding to the entities and their primary keys (PK) are shown in Figure6. The field "empid" of the tables "dept" and "location" should be a foreign key of the identically named field in the table "emp". Moreover, as it is not mandatory for location to stay in a dept, the field "depid" of the table "location" can be null, but the empid of location will always be not null.

For coverage implementation, we have considered the query as shown in Figure7. It obtains information about all Employees and their respective Department, if any, and the Location that are working in

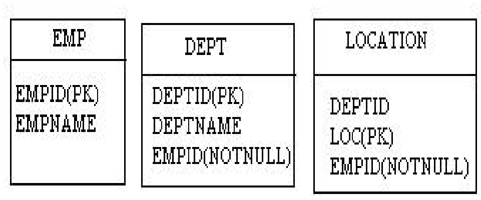
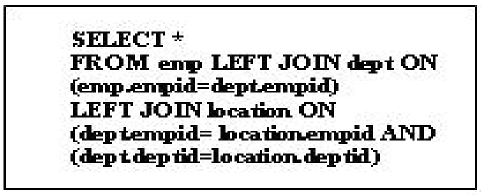


Figure 6: Tables, primary keys and possible null fields.

the organization at a particular moment.

It seems adequate to use two "LEFT JOIN"



```
SELECT *
FROM emp LEFT JOIN dept ON
(emp.empid=dept.empid)
LEFT JOIN location ON
(dept.empid= location.empid AND
(dept.deptid=location.deptid)
```

Figure 7: SELECT query for emp, dept and location.

clauses: one for the tables "emp" and "dept", so that all emp are obtained with or without dept and another for "dept" and "location", with the goal of obtaining all dept, with or without location.

From these tables, the coverage tree for the SELECT query using the coverage algorithm. The coverage tree is shown in Figure8. The first

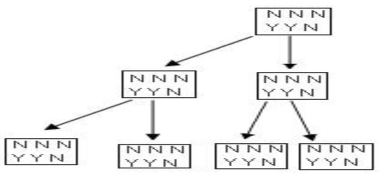


Figure 8: Coverage Tree for Select Query.

level node creation of the algorithm uses the two table's EMP and DEPT. It checks for the first condition creates the first node using the algorithm as given in section 5.1.3. For each value of T, FL, and FR it creates again nodes with the first condition result table and location table. The level of the tree depends on the number of condition in the query., as a result we have three levels. The coverage percentage of the query is calculated by the equation (3) using the coverage tree.
Similarly we have calculated the coverage's for various SELECT statements with different conditions and the results given in TABLE 1.From the TABLE it is clear that the coverage percentage for select queries for three conditions is 17.94%, for two conditions is25% and for one condition is 50% when three different tables are considered.

5.2 Command Form Algorithm

The second approach we have considered in calculating the coverage metrics is through command form algorithm. The command form algorithm

Table 1: Calculated Coverage Percentage by Coverage Tree

Tables	*Conditioninthequery*	*CoveragePercentage bycoveragetree*
One	*One*	50.0
One	*Two*	25.0
Three	*One*	2.56
Three	*two*	7.79
Three	*Three*	17.94

searches for SQL query situations covered with the data stored in the database. Evaluates the conditions of SELECT queries that are in the FROM clause, when they include JOIN, and in the WHERE clause. Moreover, the null values will be verified at the same time as the conditions are evaluated. The flow of coverage algorithm is shown in Figure9.

From the tables present in the database, select queries are generated based on the reports required by the test manager. The select queries are issued to the command form generation block where the conditions are evaluated and the command form constructed. This coverage information is used to calculate the coverage percentages of the select queries.

5.2.1 Command Form

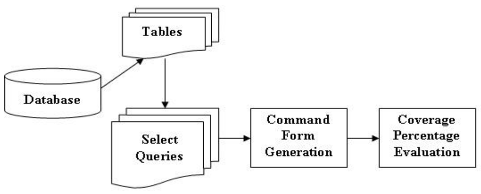


Figure 9: Flow of Command form Algorithm

The main challenge when generating command forms is the accurate identification of the possible SQL commands that could be issued at a given database interaction point. The set of testing requirements for Command form coverage consists of all the command forms for the database interaction points in the application under test. The main goal is to exercise the interactions between an application and its underlying database. Command forms represent a model of the database application at the right level of abstraction. They model all the possible commands that the application can generate and execute on the database. Therefore, the number of command forms exercised by a test suite is likely to be a good indicator of the thoroughness of the testing of the interactions between the application and its database. The command form for the sample query:

SELECT * FROM EMP WHERE empid=10;
is shown in Figure10.

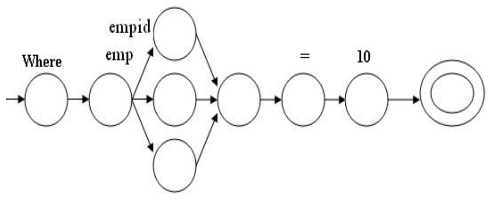


Figure 10: Command form for the sample select query

We have developed an automated tool that generates the command form for any given sql query and also calculates the command form coverage metric.

5.2.2 Coverage evaluation from Command Form

To measure the adequacy of a test suite with respect to our coverage criterion, we monitor the execution of the application and determine which command forms are exercised. We consider a command form associated with a database interaction point to be covered if, during execution, an SQL command that corresponds to the command form is issued at that point. The algorithm for calculating command form coverage is as follows. The coverage information is collected by inserting a call to a monitor immediately before each database interaction point. At runtime, the monitor is invoked with two parameters: the string that contains the actual SQL command to be executed and a unique identifier for the interaction point. First, the monitor parses the command string into a sequence of SQL tokens. Second, using the interaction point's identifier, it retrieves the corresponding SQL command form model. To find which command form corresponds to the command string, the monitor traverses the model by matching SQL tokens and transition labels until it reaches an accepting state. At the end of the traversal, the path followed corresponds to the command form covered by the command string, and the ID of the command form is given by the sum of the edge values associated with the transitions in the traversed path. At this point, the monitor adds to the set of coverage data a pair consisting of the ID of the covered command form and the ID of the database interaction point. Given a set of coverage data, the database command form coverage measure can be expressed as in equation (4).

The number of command forms covered

$$\text{Coverage} = \frac{\text{Number of command forms covered}}{\text{Total number of command forms}} * 100 \qquad (4)$$

is simply the number of unique entries in the coverage data. The total number of command forms is given by the sum of each database interaction point's maximum command form ID. All command form IDs that do not appear in the coverage data correspond to command forms that were not covered during testing. Given an ID, we can easily reconstruct the string representation of the corresponding command form and show it to the testers. For the tables shown in Figure6 and the SELECT query shown in Figure7, the command form coverage is as shown in Figure11.The command form creation for the database considers two tables - employee and department. It takes into account the first condition present in the SELECT query and continues with second, third conditions and so on.

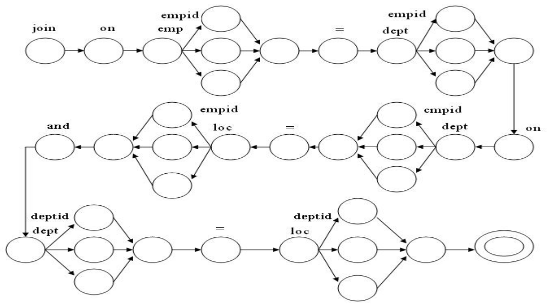


Figure 11: Command Form for Select Query

The coverage percentage is then calculated using equation (4) by counting the number of command forms covered for the select queries. TABLE 2 details some command form coverage's calculation for various SELECT queries with three, two and one condition.

From the TABLE 2 it is clear that the coverage percentage for select queries for three conditions is 68.42%, for two conditions is 69.23% and for one condition is 71.42% when three different tables are considered.

Table 2: Calculated Coverage Percentage by Command form

$Tables$	$Conditioninthequery$	$Coverage$ $ByCommandForm$
One	One	78.11
One	Two	77.64
$Three$	One	71.42
$Three$	two	69.29
$Three$	$Three$	68.42

6 Experimental Databases used for Coverage Metrics Computation

For experimental purpose we have considered four databases created by the under graduate students and two database from UCI Machine Learning Repository [http://archive.ics.uci.edu/ml/].The details of the above databases are shown in TABLE 3 as follows.

In TABLE 3, for testing the various databases

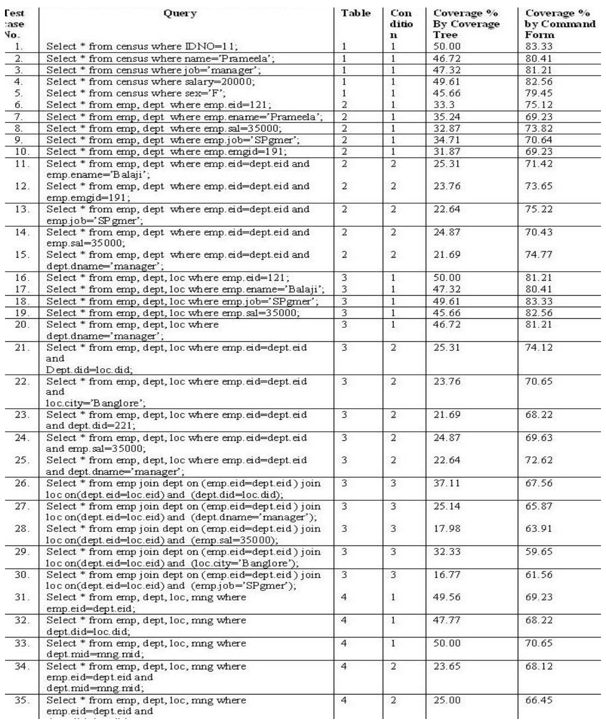
Table 3: Details of the Databases

DB $Application$	A	R	P K	F K	SQ	$Size$ (KB)
$Census(UCI)$	14	682	4	2	25	62
$Employee$	12	425	5	1	30	53
$Department$	10	321	3	1	15	46
$Location$	8	425	2	1	20	51
Tax	11	250	3	1	18	32
$University$ $DataSet(UDS)$	17	285	7	3	22	37

we have considered 15 testcases for department database and 22 testcases from university dataset etc. For each and every database, the testcases were executed using the DbUnit tool. DbUnit is an open source Framework created by Manuel Laflamme. This is a powerful tool for simplifying Unit Testing of the database operations [11]. It extends the popular JUnit test framework that puts the database into a known state while the test executes. This strategy helps to avoid the problem that can occur when one test corrupts the database and causes subsequent test to fail. DbUnit provides a very simple XML based mechanism for loading the test data, in the form of data set in XML file, before a test runs. Moreover the database can be placed back into its pre-test state at the completion of the test [12].

7 Result Analysis

For calculating the various coverage metrics like coverage tree and command form we have considered test cases (SELECT queries) which have conditions based on one or more tables as shown in Fig-



Test case No.	Query	Table	Condition	Coverage % By Coverage Tree	Coverage % by Command Form
1.	Select * from census where IDNO=11;	1	1	50.00	83.33
2.	Select * from census where name='Prameela';	1	1	46.72	80.41
3.	Select * from census where job='manager';	1	1	47.32	81.21
4.	Select * from census where salary=20000;	1	1	49.61	82.56
5.	Select * from census where sex='F';	1	1	45.66	79.45
6.	Select * from emp, dept where emp.eid=121;	2	1	33.3	75.12
7.	Select * from emp, dept where emp.ename='Prameela';	2	1	35.24	69.23
8.	Select * from emp, dept where emp.sal=35000;	2	1	32.87	73.82
9.	Select * from emp, dept where emp.job='SPgmer';	2	1	34.71	70.64
10.	Select * from emp, dept where emp.emgid=191;	2	1	31.87	69.23
11.	Select * from emp, dept where emp.eid=dept.eid and emp.ename='Balaji';	2	2	25.31	71.42
12.	Select * from emp, dept where emp.eid=dept.eid and emp.emgid=191;	2	2	23.76	73.65
13.	Select * from emp, dept where emp.eid=dept.eid and emp.job='SPgmer';	2	2	22.64	75.22
14.	Select * from emp, dept where emp.eid=dept.eid and emp.sal=35000;	2	2	24.87	70.43
15.	Select * from emp, dept where emp.eid=dept.eid and dept.dname='manager';	2	2	21.69	74.77
16.	Select * from emp, dept, loc where emp.eid=121;	3	1	50.00	81.21
17.	Select * from emp, dept, loc where emp.ename='Balaji';	3	1	47.32	80.41
18.	Select * from emp, dept, loc where emp.job='SPgmer';	3	1	49.61	83.33
19.	Select * from emp, dept, loc where emp.sal=35000;	3	1	45.66	82.56
20.	Select * from emp, dept, loc where dept.dname='manager';	3	1	46.72	81.21
21.	Select * from emp, dept, loc where emp.eid=dept.eid and Dept.did=loc.did;	3	2	25.31	74.12
22.	Select * from emp, dept, loc where emp.eid=dept.eid and loc.city='Banglore';	3	2	23.76	70.65
23.	Select * from emp, dept, loc where emp.eid=dept.eid and dept.did=221;	3	2	21.69	68.22
24.	Select * from emp, dept, loc where emp.eid=dept.eid and emp.sal=35000;	3	2	24.87	69.63
25.	Select * from emp, dept, loc where emp.eid=dept.eid and dept.dname='manager';	3	2	22.64	72.62
26.	Select * from emp join dept on (emp.eid=dept.eid) join loc on(dept.eid=loc.eid) and (dept.did=loc.did);	3	3	37.11	67.56
27.	Select * from emp join dept on (emp.eid=dept.eid) join loc on(dept.eid=loc.eid) and (dept.dname='manager');	3	3	25.14	65.87
28.	Select * from emp join dept on (emp.eid=dept.eid) join loc on(dept.eid=loc.eid) and (emp.sal=35000);	3	3	17.98	63.91
29.	Select * from emp join dept on (emp.eid=dept.eid) join loc on(dept.eid=loc.eid) and (loc.city='Banglore');	3	3	32.33	59.65
30.	Select * from emp join dept on (emp.eid=dept.eid) join loc on(dept.eid=loc.eid) and (emp.job='SPgmer');	3	3	16.77	61.56
31.	Select * from emp, dept, loc, mng where emp.eid=dept.eid;	4	1	49.56	69.23
32.	Select * from emp, dept, loc, mng where dept.did=loc.did;	4	1	47.77	68.22
33.	Select * from emp, dept, loc, mng where dept.mid=mng.mid;	4	1	50.00	70.65
34.	Select * from emp, dept, loc, mng where emp.eid=dept.eid and dept.mid=mng.mid;	4	2	23.65	68.12
35.	Select * from emp, dept, loc, mng where emp.eid=dept.eid and	4	2	25.00	66.45

Figure 12: Calculation of the Various Coverage Metrics

ure 12.From Figure 12 it is clear that as the number of tables and the conditions in the SELECT queries increases, the coverage percentage for both coverage tree and command form gets decreased. For example considering testcase4, 5 which contains 1 table 1 condition,has a coverage tree percentage of 49.61%, 45.66% and command form percentage of 82.56%, 79.45% respectively.For testcase23,24 which has 2 Figure 12 conditions, the percentages is reduced to 21.69%,24.87% and 68.22%,69.63% respectively. The results of two

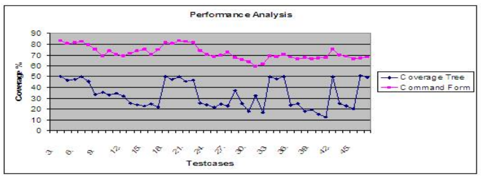


Figure 13: Performance analysis of two Coverage algorithms

Coverage algorithms namely coverage tree and command form were analyzed for different kinds of select statements with varying tables and conditions as shown in TABLE4 and the graphical representation in Fig 12.

8 Conclusion and Future Enhancement

Well-designed metrics with documented objectives can help an organization obtain the information it needs to continue to improve its software products, processes, and services while maintaining a focus on what is important.In this paper we have evaluated the two coverage algorithms namely coverage tree and command form for database applications and from the analysis it is clear that the coverage percentage calculated using command form is better than the coverage tree algorithm. Thus for database testing these coverage metrics may help the test manager to calculate the effectiveness of the various test cases there by achieving quality in the testing process.This work can be extended by calculating the coverage for other DML statements for database applications. In order to attain the better accuracy of testing, use of SQL mutation operators for SQL queries that covers a wide range of the SQL syntax and semantics can be computed.

9 References

[1] Yuetang Deng, Phyllis Frankl, David Chays. Testing Database Transactions with AGENDA. ICSE'05, May 15-21, 2005, St. Louis, Missouri, USA.2005 ACM.

[2] Eric Tang,Phyllis G. Frankl,Yuetang Deng, Test Coverage Tools for Database Applications. Proceedings of MASPLAS'06 Mid Atlantic Student Workshop on Programming Languages and Systems Rutgers University, April 29, 2006.

[3] Gregory M. Kapfhammer, Mary Lou Soffa. A Family of Test Adequacy Criteria for Database-Driven Applications. ESEC/FSE'03, September 1-5, 2003, Helsinki, Finland.2003 ACM .

[4] Daou, B., Haraty, R.A. and Mansour, N. Regression testing of database applications. Symposium of Applied Computing.ACM. 2001

[5] M.Y.Chan and S.C.Cheung, Testing Database Applications with SQL Semantics,Proceedings of 2nd International Symposium on Cooperative Database Systems for Advanced Applications(codas'99), March 1999,pp 363-374.

[6] W.K. Chan, S.C. Cheung and T.H. Tse, "Fault-Based Testing of Database Application Programs with Conceptual Data Model", Proceedings of the 5th International Conference on Quality Software, 2005.IEEE

[7] Zhu, H., Hall, P. A. V., May, J. H. R. Software Unit Test Coverage and Adequacy. ACM Computing Surveys, 49(4) 366-427. 1997

[8] William G.J. Halfond and Alessandro Orso. Command-Form Coverage for Testing Database Applications. 21st IEEE International Conference on Automated Software Engineering (ASE'06) 2006 IEEE

[9] María José, Suárez-Cabal and Javier Tuya, Using an SQL coverage measurement for testing database applications, ACM SIGSOFT Software Engineering Notes, Volume 29, Issue 6, Pages: 253 262, November 2004.

[10] Woodward, M.R., Hedley, D. and Hennell, M.A., "Experience with Path Analysis and Testing of Programs", IEEE Vol. SE-6, No. 3, pp. 278-286, May 1980

[11] www.dbunit.org

[12] www.sourceforge.net

[13] Norman E Fenton, Martin Neil, "Software Metrics: Roadmap".

[14] Qaiser Durrani, " Role of Software Metrics in Software Engineering and Requirements Analysis ", 2005 IEEE

[15] Grady, R.B, Practical Software Metrics for ProjectManagement and Process Improvement,1992.

[16] Roger S. Pressman, Software Engineering: A Practitioner's Approach

[17] Dekkers, C. Demystifying function points: Let's understand some terminology. IT Metrics Strategies, Oct. 1998.

[18] Chidamber, S. R. and Kemerer, R. F. A metrics suite for object-oriented design. IEEE Trans. Software Eng. 20, 6 (June 1994), 476-493.

[19] John Joseph Chilenski and Steven P. Miller, "Applicability of Modified Condition/Decision Coverage to Software Testing", Software Engineering Journal, Sept 1994, Vol. 9, No. 5, pp.193-2000.

[20] Howden, "Weak Mutation Testing and Completeness of Test Sets", IEEE Trans. Software Eng., Vol.SE-8, No.4, July 1982, pp.371-379