Efficient Processing of Test Database Reduction Scheme

DH. Srinivasan[1], G.V. Padma Raju[2]
[1]PG scholar, [2]Professor,
[12]Department of Computer science and Engineering, SRKR Engineering College, Bhimavaram

Abstract- Functional testing of uses that procedure the data put away in databases frequently requires a cautious plan of the test database. The bigger the test database, the more troublesome it is to create and keep up tests and also to load and reset the test information. This paper exhibits a way to deal with decrease a database as for an arrangement of SQL queries and a scope foundation. The reduction procedures look through the lines in the underlying database that add to the scope so as to locate a delegate subset that fulfills an indistinguishable scope from the underlying database. The approach is computerized and effectively executed against vast databases and complex queries. The assessment is done more than two genuine applications and an outstanding database benchmark. The outcomes demonstrate an expansive level of reduction and in addition versatility in connection to the measure of the underlying database and the time expected to play out the reduction.

Keywords- Test database reduction, test coverage of code, test design.

I. INTRODUCTION

Database applications include the management of a lot of information put away and sorted out in numerous tables. This information are normally overseen utilizing an outsider segment called the Database Management System (DBMS) that provides superior and a high level of adaptability and constancy. The application can get to the put away information utilizing some sort of query language. In spite of the persistent improvements in new advancements, for example, NoSQL databases and perseverance systems, applications dealing with the information utilizing Relational DBMS and the Structured Query language (SQL) are universal in practically all mechanical and business divisions. Testing software applications includes an essential movement that comprises of expounding test cases; each having sets of test case preconditions, inputs and expected yields [3]. The tester needs to provide enough significant contributions to order to practice the application code however much as could reasonably be expected. On the off chance that the application includes a database, the elaboration of test databases is a deciding variable.

On a few events, the test database might be by a long shot the most imperative segment of the information, (for example, reports, logical queries or dashboards). Making a test database includes various specialized and commonsense difficulties. The test database ought to contain enough significant information to sufficiently practice the application under test. Be that as it may, populating the test database turns into a troublesome undertaking in light of the very interrelated nature of tables. Test databases ought to be kept little in order to encourage:

1) The productivity of the reset of the test database,
2) The blame confinement and investigating of fizzled tests,
3) The test yield assessment when a test produces numerous yields from the database, and
4) The upkeep and extensible of test contents. Consider, for instance, the accompanying situation: A database contains orders made by clients. Each order has the data about the client and the warehouse that will supply the merchandise.

This data is put away in a fundamental table (order) with the order ID (oid), client ID (cid), warehouse ID (wid) and the order status. The warehouse table incorporates its ID (wid) and its name. Another revealing module is a work in progress and one of the reports comprises in showing every single dropped order (status = 'C') and the warehouse name. The engineer makes the report in light of the accompanying query:

```
SELECT o.oid, o.status, c.cid, w.name
FROM order o, warehouse w
WHERE o.wid =w.wid AND o.status = 'C'
```

The test necessities for this report incorporate making test databases with orders with status 'C' and other distinctive statuses. Likewise, as the warehouse is doled out in the wake of entering an order, there must be orders in the test database that have been dropped when the task of a warehouse. Making test databases needs a trade-off between the nature of the information from the testing perspective and down to earth issues identified with populating and stacking the test database. The tester may embrace diverse methodologies that range from 1) beginning from a formerly populated database (e.g., a duplicate of the production database) to 2) beginning from an unfilled database. In the event that testing is finished utilizing a production database, the real outcomes must be looked at numerous rows in the answer to guarantee they meet the particular. Specifically it ought to be watched that every single detailed line are incorporated and there are no overlooked rows. For this situation the query isn't right as it disregards dropped orders that don't have a warehouse relegated yet. The wellspring of the blame in the query is that the join between tables ought to be a left join. It ought to be composed as:

```
SELECT o.oid, o.status, c.cid, w.name FROM order o
LEFT JOIN warehouse w ON o.wid = w.wid
WHERE o.status = 'C'
```

In addition, if the test is additionally robotized, its execution will require a reset of the database to separate this test from others that change the database, which is additional tedious as the span of the database grows.

The second methodology is to begin from a vacant database. The tester is allowed to make a content to populate a test database containing just the rows that satisfy the test prerequisites. The examination of the real outcomes is less demanding as fewer rows at the yield must be checked and whatever is left of the database is speedier. Be that as it may, the tester needs to indicate each line and its qualities (counting all segments in the tables included, which are rearranged in the illustration) and to populate extra tables to guarantee referential respectability. A middle system that constitutes a trade-off between the above would comprise of extricating a subset of the information that satisfies the test prerequisites from the production database and creating a content to populate the test database with this subset. This is a reduction of the production database. On the off chance that it is made consequently, this would encourage the testing as it contains few, however important information (that cover the test necessities of the query). It is less demanding to check the genuine outcomes (they contain less rows) and simpler to populate and stack the test database (the content would be consequently made).

The extent of this paper identifies with this middle system: Given a database, create a littler database containing significant information to empower its utilization as a test database. To achieve this, 1) we begin from an underlying database (that can be taken from a production database in the wake of muddling confidential information) and an arrangement of queries that have been issued to the database (which can be taken from the execution log enlisted by the DBMS). 2) In order to have the capacity to choose significant test information from the underlying database we utilize a test paradigm called SQL Full Predicate Coverage (SQLFpc) [4] which is a variation of Modified Condition/Decision Coverage (MCDC) [5],[6] particularly custom fitted for SQL. Given a SQL query and a test database, the SQLFpc paradigm characterizes an arrangement of test necessities, each spoke to as a coverage rule (composed as a SQL articulation). The execution of the tenets against the underlying database decides if the test necessities for the query are met. 3) Then the information which fulfill every coverage rule are recovered, decreased to a subset and embedded into another database (at first unfilled) which constitutes the lessened test database.

II. LITERATURE SURVEY

Writing overview is the most critical advance in software improvement process. Before building up the instrument it is important to decide the time factor, economy n organization quality. Once these things r fulfilled, ten subsequent stages are to figure out which working system and language can be utilized for building up the apparatus. Once the developers begin assembling the instrument the software engineers require parcel of outside help. This help can be gotten from senior developers, from book or from sites. Before building the system the above consideration are considered for building up the proposed system.

"An analysis of the effectiveness of different coverage criteria for testing relational database schema integrity constraints", These fundamentally vital limitations guarantee the cognizance of relational information in a database, protecting it from rules that could disregard prerequisites, for example, "usernames must be interesting" or "the host name can't be absent or obscure." This article is the first to propose coverage criteria, got from rationale coverage criteria that set up various levels of testing for the plan of trustworthiness requirements in a database mapping. These range from basic criteria that command the testing of effective and unsuccessful INSERT proclamations into tables to further developed criteria that test the definition of complex respectability limitations, for example, multi-section PRIMARY KEYs and self-assertive CHECK imperatives. Because of various seller understandings of the structured query language (SQL) particular as to how trustworthiness limitations ought to really work by and by, our criteria vitally represent the hidden semantics of the database management system (DBMS).

"Program-input generation for testing database applications using existing database states", Utilizing a current database state is alluring since it has a tendency to be illustrative of certifiable items' qualities, identifying issues that could cause disappointments in true settings. In any case, to cover a particular program-code divide (e.g., square), fitting project inputs additionally should be created for the given existing database state. To address this issue, in this paper, a novel approach that produces program contributions for accomplishing high code coverage of a database application, given a current database state. This approach utilizes emblematic execution to track how program inputs are changed before showing up in the executed SQL queries and how the imperatives on query comes about influence the application's execution. One critical test in our concern setting is the hole between program-input limitations got from the program and from the given existing database state; fulfilling the two sorts of requirements is expected to cover a particular program-code parcel. Our approach incorporates novel query definition to bridge this hole. We fuse the information instantiation segment in our system to manage

the case that no powerful program input esteems can be accomplished. We decide how to produce new records and populate them in the new database state with the end goal that the code along the way can be secured.

"The Impact of Equivalent, Redundant and Quasi Mutants on Database Schema Mutation Analysis", Since the relational database is an imperative part of certifiable software and the diagram assumes a noteworthy part in guaranteeing the nature of the database, relational construction testing is basic. This paper presents strategies for enhancing the effectiveness and exactness of transformation examination, a set up strategy for evaluating the nature of test cases for database mappings. Utilizing a DBMS-free unique portrayal, the displayed strategies consequently identify and expel mutants that are either comparable to the first blueprint, excess as for different mutants, or bothersome in light of the fact that they are valid for certain database systems. Applying our systems for incapable mutant evacuation to an assortment of compositions, a considerable lot of which are from genuine sources like the U.S. Bureau of Agriculture and the Stack Overflow site, uncovers that the introduced static examination of the DBMS-free portrayal is various orders of size speedier than a DBMS-particular technique.

"Automated testing for SQL injection vulnerabilities: an input mutation approach", Web administrations are progressively embraced in different spaces, from fund and e-government to online networking. As they are based over the web advancements, they endure likewise a phenomenal measure of assaults and abuses like the Web. Testing to recognize such vulnerabilities previously making web administrations open is pivotal. Introduce in this paper a mechanized testing approach, to be specific μ4SQLi, and its supporting arrangement of change administrators. μ4SQLi can create viable data sources that prompt executable and destructive SQL proclamations. Executability is key as generally no infusion helplessness can be abused.

"Generating Test Data to Distinguish Conjunctive Queries with Equalities", the utilization of databases in software systems has expanded the significance of unit testing the queries that frame the interface to these databases. Transformation investigation is an intense testing system that has been adjusted to test database queries. In this paper we address each of the three of these difficulties by adjusting comes about because of the rich writing on query modifying. We confine regard for the class of conjunctive queries with equities. As an end-result of this confinement, we give a calculation that perceives equal mutants, creates a test database that recognizes each nonequivalent mutant, and applies to subjective transformations, as long at the change is additionally a conjunctive query with equities. The paper introduces the test database age calculation and demonstrates that it is sound and finish for conjunctive queries with uniformities.

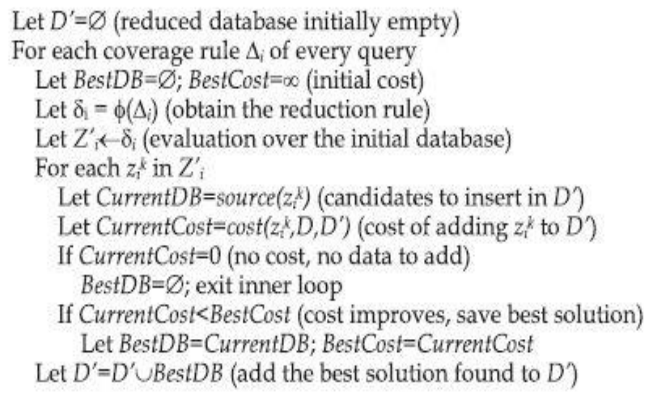
"Guided test generation for database applications via synthesized database interactions", Testing database applications regularly requires the age of tests comprising of both program sources of info and database states. As of late, a testing procedure called Dynamic Symbolic Execution (DSE) has been proposed to decrease manual exertion in test age for software applications. In any case, applying DSE to produce tests for database applications faces different specialized difficulties. For instance, the database application under test needs to physically interface with the related database, which may not be accessible for different reasons. The program inputs whose qualities are utilized to frame the executed queries are not treated emblematically, posturing challenges for creating valid database states or suitable database states for accomplishing high coverage of query-result-rule code. To address these difficulties, in this article, we propose an approach called SynDB that combines new database associations to supplant the first ones from the database application under test.

Problem Definition:- The system is to begin from an empty database. The tester is allowed to make a content to populate a test database containing just the rows that satisfy the test necessities. The correlation of the real outcomes is simpler as fewer rows at the yield must be checked and the reset of the database is speedier. Be that as it may, the tester needs to indicate each line and its qualities (counting all sections in the tables included, which are improved in the case) and to populate extra tables to guarantee referential respectability.

The Relational Model:- The relational model was first created by Codd [15] and characterizes the establishments of information stockpiling and querying that is actualized in the present business relational database management systems. The documentation utilized as a part of this paper is that displayed by the creator in the second form of the relational model [16], alluded to as RM/V2, with a few adjustments required for resulting segments.

Relations and Attributes:- Given a set An of attributes A1;. . .;Am, a relation R is a subset of the Cartesian result of their domain, meant as R(A1; . . .Am) or just R(A) or R. At the end of the day, a relation R(A) is a set of tuples of the attributes in A. In SQL a relation is a table or view, attributes are segments and tuples are rows. For every relation one or more attributes are primary keys which interestingly identify each tuple in this relation. The space of attributes incorporates exceptional imprints to reference absent or inapplicable attributes which are demonstrated as NULL in business relational DBMS. This prompts a three-esteemed logic of predicates.

Reduction Rules and Procedures:- This section points of interest how the coverage rules are transformed into the reduction guidelines and how tuples coming about because of the assessment of the reduction govern are chosen to acquire the diminished database.



```
Let $D'=\varnothing$ (reduced database initially empty)
For each coverage rule $\Delta_i$ of every query
   Let $BestDB=\varnothing$; $BestCost=\infty$ (initial cost)
   Let $\delta_i = \phi(\Delta_i)$ (obtain the reduction rule)
   Let $Z'_i \leftarrow \delta_i$ (evaluation over the initial database)
   For each $z_i^k$ in $Z'_i$
      Let $CurrentDB=source(z_i^k)$ (candidates to insert in $D'$)
      Let $CurrentCost=cost(z_i^k,D,D')$ (cost of adding $z_i^k$ to $D'$)
      If $CurrentCost=0$ (no cost, no data to add)
         $BestDB=\varnothing$; exit inner loop
      If $CurrentCost<BestCost$ (cost improves, save best solution)
         Let $BestDB=CurrentDB$; $BestCost=CurrentCost$
   Let $D'=D' \cup BestDB$ (add the best solution found to $D'$)
```

Fig. 2. Algorithm to select the reduced database with lowest cost.

Join and Select Operators: The instance of a RVE (query) that performs joins and a further determination of joined tuples is the least difficult one and permit the introduction of the establishments of the reduction approach. We initially show it with a case. Case one Consider an underlying database D =fR;Sg which contains two relations R(A0,A1) and S(B0,B1,B2), being A0 and B0 primary keys and a RVE D which speaks to a coverage manage characterized as:

D: = R[A0 = B1]S [A1 < 14](A1;B2)

In SQL: SELECT A1; B2 FROM R INNER JOIN S ON R: A0 =S.B1 WHERE R:A1 < 14

Reduction Transformation and Reduction Rule: A reduction transformation (ϕ) transforms a RVE (coverage rule) into a reduction administer (d) such that relation Z' got subsequent to assessing d over D permits identifying all source tuples of D. In the case, this is expert by incorporating the primary keys in the projection. The outcome is another RVE called reduction rule.

d :=R[A0 = B1]S [A1 < 14] (A0,B0,A1,B2)

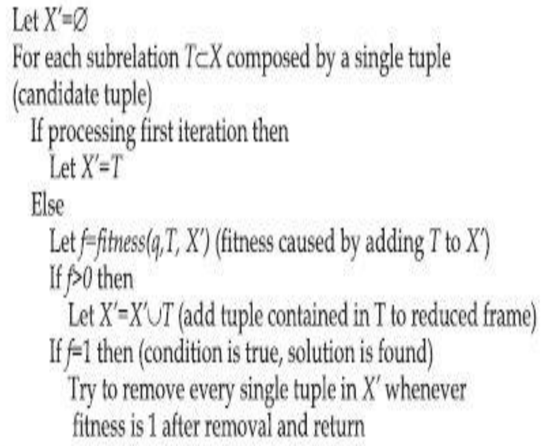
In SQL: SELECT A0; B0; A1; B2 FROM R INNER JOIN S ON R.A0=S.B1 WHERE R.A1 < 14

Reduction Procedure: A reduction technique chooses a little subset of tuples of Z' d and finds the source tuples in D to get the diminished database D0 = {R', S'} The reduction system must utilize some sort of methodology such that the lessened database is as little as would be prudent. This depends on the cost of including each tuple of Z' to D' estimated as far as the quantity of new tuples that must be added to the lessened database. Essentially here Z' contains two tuples. Both of them will create tuples in R' and S' with cost 2. Every one of them might be chosen (for instance, the primary tuple of Z' in the figure). The reduction strategy is incremental, being executed for every coverage govern, with the goal that it considers tuples that are as of now in the diminished database. For instance, if a tuple with A0 = 3 as of now exists in D', the cost of the second tuple in Z' will be 1 (a lower cost) as just a single new tuple with B0 = 2 would should be added to the diminished database.

Coverage Gain and Loss: As the reduction method just adds tuples to the diminished database at each progression, when a query has just determination and join operators, there are no coverage decides that are secured at some progression and wind up revealed at a later advance (coverage loss). In any case, the inverse isn't valid: a decide that isn't shrouded in the underlying database may wind up canvassed in the lessened database (coverage given). This is the situation of principles with external joins: Initially, ace relations may have no less than one related line in their detail; along these lines, decides that require an ace with no detail are not secured by the underlying database. Notwithstanding, as the reduction procedure chooses just a couple of rows from the underlying database, the lessened database may contain circumstances in which there is a column in an ace table with no related line in the detail, prompting a coverage given.

Framed Relations: As the encircling hides the tuples and primary keys that are gathered in each casing, the initial step is to ungroup the casing by joining the relation Z acquired by assessing the coverage administer with the original relation R utilizing the gathering attributes (A1) as the joining attributes, and after that ordering by the gathering attributes of Z. The resulting relation (Z') reveals the frames.

Reduction of Frames: Hunt based algorithms have been already utilized for various software building issues and specifically, software testing. A principal issue is the meaning of a wellness work that is limited in order to locate the best arrangement among various candidate arrangements.



```
Let $X'=\varnothing$
For each subrelation $T \subset X$ composed by a single tuple
(candidate tuple)
   If processing first iteration then
      Let $X'=T$
   Else
      Let $f=fitness(q,T,X')$ (fitness caused by adding $T$ to $X'$)
      If $f>0$ then
         Let $X'=X' \cup T$ (add tuple contained in T to reduced frame)
      If $f=1$ then (condition is true, solution is found)
         Try to remove every single tuple in $X'$ whenever
          fitness is 1 after removal and return
```

Fig. 4. Algorithm to reduce a frame.

The wellness work incorporates an idea of separation or cost that measures how far a candidate arrangement is (i.e., a set of contributions) from fulfilling some paradigm (e.g., make genuine or false a given condition or decision). The separation for relational articulations is assessed utilizing fetched capacities. Wellness works for the most part incorporate the cost and an extra term called approach level.

Distances for base predicates: Give X<Z' a chance to be a single original frame that is being decreased and X' the lessened edge (at first vacant). Give pi a chance to be a base predicate, which may contain references to attributes or total capacities over attributes however not logical articulations. The separation d(pi, X0) over the relation X' is figured utilizing the Tracey capacities.

Consider, for instance, a predicate p:= sum(a)>=8 assessed over a relation with three tuples {(1), (2), (3)}. The assessment of the separation figures the term sum(a) which gives 6. At that point the separation is 8– 6 = 2.

Efficiency Optimizations: Coverage rules for a query are intended to acquire a subset of rows that fulfill a given test prerequisite for a query. This produces fewer rows than the original query and in that capacity a quicker handling. In any case, the transformations that acquire the reduction rules acquaint an overhead due with the extra joins used to decide the base keys. Various improvements are made after the reduction rules have been produced:

Frame removal: In the event that a rule does not have any select after an edge, the edges are evacuated as the reduction method just needs a solitary tuple from the casing to cover the rule.

Using SQL windowing functions2 for frames: It is material just if the DBMS supports such capacities. At the point when this improvement is connected the condition PARTITION BY . . . OVER is utilized as opposed to joining the source and gathering relations.

Move sub queries to join clauses: On the off chance that an uncorrelated scalar sub query shows up at a conjunctive choice predicate, the condition containing the sub query is utilized for joining the source and base relations. This causes the DBMS to recover the information faster.

Convert non-scalar to scalar sub queries: In the above case, if the total capacity is avg, max or min and the relational operator is < or <=, an extra condition is added to the join predicate to force the sub query to come back to the most extreme esteem (on the other hand if operator is > or >=). This reductions the quantity of tuples that are recovered from the database and also avoids potential coverage losses.

Simplification of sub query rules: Some coverage rules for uncorrelated sub queries incorporate a principle query and a where condition with the sub query inside of an exists logical predicate. As no less than one line that fulfills the principle query has been acquired when handling the past rules, the present rule is streamlined by expelling the fundamental query.

Limiting the Size of the Reduction Relation:
A littler size of the reduction relation is accomplished by changing the reduction rule to indicate a point of confinement in the quantity of tuples of various casings. Four distinct cases can be determined for:

Queries without frames: To limit the tuples retrieved by the reduction rule.

Queries with frames: To limit the tuples retrieved by the group relation.

Frames: To limit the tuples retrieved by the source relation.

Sub queries: To limit the tuples retrieved by the source relation.

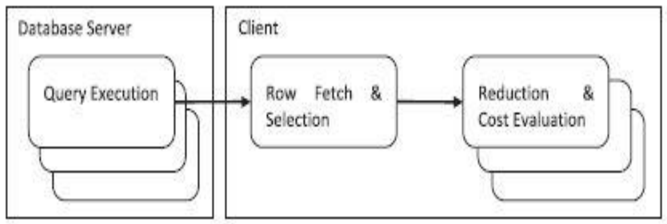


Fig.1: Parallelizing the reduction procedures.

This is expert by encasing the SQL of the reduction rule under the provision PARTITION BY . . . OVER. This permits ruleling the span of each edge by determining its most extreme number of tuples. Note that this streamlining must be utilized on the off chance that it is supported by the DBMS. The specific sentence structure relies upon the specific DBMS vendor particular. For example, in SQL server the TOP keyword is utilized after SELECT. In Oracle, the ROWNUM exceptional section is included a condition of the WHERE statement. Restricting the outcome measure is a trade-off amongst cost and quality. When restricting the outcome set size the proficiency enhances as less information is perused from the dataset. In any case, this suggests less information might be reused when performing the reductions thus a bigger diminished database might be created.

III. CONCLUSION

We have displayed an approach for the reduction of test databases that takes a set of SQL queries and an underlying database, and produces a lessened test database that preserves the SQLFpc coverage. The approach can deal with complex queries covering a substantial set of SQL builds and their blends. The outcomes demonstrated that a high level of reduction can be accomplished with few coverage misfortunes and some coverage picks up. Also, it is versatile in relation to the extent of the underlying database and the reduction time. Moreover, the entire way to deal with the reduction of a database is completely mechanized and a few improvements are incorporated. The regular target situation is created by applications that depend on SQL queries for handling complex business rules. A decreased test database might be made for testing or creating particular queries, prompting a beginning stage of the test database to finish the tests. While considering the entire application, an extensive set of queries might be extricated from the database sign in order to make a lessened database covering these queries, which thus might be utilized as a beginning stage for finishing tests or performing other support tasks. This is the primary potential advantage of the approach that adds to a reduction of the time spent on the assignment of making test databases. A second potential advantage is to permit a lessening of the seasons of stacking test databases, while keeping an agent set of information to practice the queries of the applications, prompting speedier test execution. Furthermore, having little test databases adds to make the undertaking of checking the real outcomes less demanding when creating and testing queries, adding to a speedier and more dependable test comes about examination.

IV. FUTURE ENHANCEMENT

Our future work will focus on two territories. To begin with, to finish the assessment and down to earth use at the application level to incorporate the capacity to decrease test databases into the engineer and tester workflows. This will infer testing how the reduction performs utilizing different DBMS. Second, to utilize the reduction standards to address NoSQL databases in order to provide support for testing with regards to the improvement of uses that control information utilizing these innovations.

V. REFERENCES

[1]. A. B. M. Moniruzzaman and S. A. Hossain, “NoSQL Database:New era of databases for big data analytics - classification, characteristics and comparison,” Int. J. Database Theory Appl., vol. 6,no. 4, pp. 1–14, Aug. 2013.

[2]. Information Technology - Database Languages – SQL, Int. Standards Organisation ISO/IEC 9075, 1999.

[3]. Software and Systems Engineering - Software Testing - Part 1: Concepts and Definitions, Int. Standards Organisation ISO/IEC/IEEE 29119–1:2013, 2013.

[4]. J. Tuya, M. J. Su_arez-Cabal, and C. de la Riva, “Full predicate coverage for testing SQL database queries,” Softw. Testing, Verification Rel., vol. 20, no. 3, pp. 237–288, Sep. 2010.

[5]. RTCA Inc., DO-178-B: Software Considerations in Airborne Systems and Equipment Certification, Radio Technical Commission for Aeronautics (RTCA), 1992.

[6]. J. J. Chilenski, “An investigation of three forms of the modified condition decision coverage (MCDC) criterion,” U.S. Dept. Transp., Federal Aviation Administration, Washington, DC, USA, Tech. Rep. DOT/FAA/AR-01/18, Apr. 2001.

[7]. J. Tuya, M. J. Suarez-Cabal, and C. de la Riva, “Query-aware shrinking test databases,” in Proc. 2nd Int. Workshop Testing Database Syst., Jun. 2009, pp. 6:1–6:6.

[8]. S. Yo and M. Harman, “Regression testing minimization, selection and prioritization: A survey,” Softw. Testing, Verification Rel., vol. 22, no. 2, pp. 67–120, Mar. 2012.

[9]. G. Rothermel, M. J. Harrold, J. Ronne, and C. Hong, “Empirical studies of test suite reduction,” Softw. Testing, Verification Rel., vol. 4, no. 2, pp. 219–249, Dec. 2002.

[10]. E. Engstr€om, M. Skoglund, and P. Runeson, “Empirical evaluations of regression test selection techniques: A systematic review,” in Proc. 2nd ACM-IEEE Int. Symp. Empirical Softw. Eng.Meas., 2008, pp. 22–31.

[11]. W. E. Wong, J. R. Horgan, S. London, and A. P. Mathur, “Effect of test set minimization on fault detection effectiveness,” Softw. Practice Experience, vol. 28, no. 4, pp. 347–369, Apr. 1998.

[12]. W. E. Wong, J. R. Horgan, A. P. Mathur, and A. Pasquini, “Test set size minimization and fault detection effectiveness: A case study in a space application,” J. Syst. Softw., vol. 48, no. 2, pp. 79–89, Oct. 1999.

[13]. G. Rothermel, M. J. Harrold, J. Ostrin, and C. Hong, “An empirical study of the effects of minimization on the fault detection capabilities of test suites,” in Proc. Int. Conf. Softw. Maintenance,1998, pp. 34–43.

[14]. G. Rothermel, M. J. Harrold, J. Ronne, and C. Hong, “Empirical studies of test suite reduction,” Softw. Testing, Verification Rel., vol. 4, no. 2, pp. 219–249, Dec. 2002.

[15]. E. F. Codd, “A relational model of data for large shared data banks,” Commun. ACM, vol. 13, no. 6, pp. 377–387, Jun. 1970.

[16]. E. F. Codd, The Relational Model for Database Management – Version 2. Reading, MA, USA: Addison-Wesley, 1990.

[17]. J. Tuya, M. J. Su_arez-Cabal, and C. de la Riva, “Mutating database queries,” Inf. Softw. Technol., vol. 49, no. 4, pp. 398–417,Apr. 2007.

[18]. G. Kaminski, U. Praphamontripong, P. Ammann, and J. Offutt, “A logic mutation approach to selective mutation for programs and queries,” Inf. Softw. Technol., vol. 53, no. 10, pp. 1137–1152,Oct. 2011.

[19]. W. K. Chan, S. C. Cheung, and T. H. Tse, “Fault-based testing of database application programs with conceptual data model,” in Proc. 5th Int. Conf. Quality Softw., 2005, pp. 187–196.

[20]. C. J. Wright, G. M. Kapfhammer, and P. McMinn, “Efficient mutation analysis of relational database structure using mutant schemata and parallelisation,” in Proc. 8th Int. Workshop Mutation Anal., 2013, pp. 63–72.

[21]. C. J. Wright, G. M. Kapfhammer, and P. McMinn, “The impact of equivalent, redundant and quasi mutants on database schema mutation analysis,” in Proc. 14th Int. Conf. Quality Softw., Oct.2014, pp. 57–66.

[22]. J. Tuya, M. J. Su_arez-Cabal, and C. de la Riva, “SQLMutation: A tool to generate mutants of SQL database queries,” in Proc. 2nd Workshop Mutation Analy., 2006, p. 1.

[23]. C. Zhou and P. G. Frankl, “Mutation testing for java database applications,” in Proc. 2nd Int. Conf. Softw. Testing Verification Validation,2009, pp. 396–405.

[24]. C. Zhou and P. G. Frankl, “JDAMA: Java database application mutation analyser,” Softw. Testing, Verification Rel., vol. 21, no. 3,pp. 241–263, Sep. 2011.

[25]. C. Zhou and P. G. Frankl, “Inferential checking for mutants modifying database states,” in Proc. 4th Int. Conf. Softw. Testing Verification Validation, 2011, pp. 259–268.