Efficient Software Testing Technique based on Hybrid Database Approach

Humma Nargis Aleem[1], Prof. Dr. Mirza Mahmood Baig[2], Dr. Muhammad Mubashir Khan[3]
NED University of Engineering and Technology, Karachi, Pakistan

***Abstract*—In the field of computer science, software testing is referred as a critical process which is executed in order to assess and analyze the performance and risks existing in software applications. There is an emphasis on integrating specific approaches to carry out testing activities in an effective mode; the efficient strategy being explored recently is adopting hybrid database approach. For this purpose, a hybrid algorithm will be proposed to ensure the functionality and outcomes of testing procedure. The technical processes and its impact on current methodology would help to evaluate its effectiveness in software testing through which specific conclusions could be drawn. The findings of the research will elaborate effectiveness of the proposed algorithm that would be used in software testing. It would be suggested that new technology is easier and simple to assess and analyze the reliability of the software. Basically, hybrid database approach comprises of traditional and modern techniques that are deployed in order to achieve testing outcomes. It is explored from various testing methods that challenges have been identified related to focusing on traditional techniques due to which hybrid approach is now being developed in most of the areas. In the light of addressing these concepts, the paper aims to investigate the complexity and efficiency of hybrid database approach in software testing, as well as its scope in the IT industry.**

Keywords—Software testing; database testing; hypothetical database testing; traditional database testing; test case(s); grey box testing; software quality assurance

I. Introduction

Computer science is a vast field which is distributed across innumerable sections in order to address different technicalities. Software testing is one of the fields in Computer Science which is referred as incorporating critical processes for assessing and analyzing possible risks and performance of a software [1]. A number of professional software testers across the world have explored variety of mechanisms that are performed to test vulnerabilities, as well as the efficiency which are the core areas of software testing. In the recent IT industry, the major concern that software developers and testers have reflected is the incompetent approaches being applied in the field of testing that hinders functionalities while business needs are not appropriately catered as they should be [2].

The IT industry has always performed strategically in providing utmost facilities to the business so that no technical issues could affect their performance and productivity [3]. In every day technical activities, a number of software techniques are adopted and used according to the specifications so that significant outcomes could be met. However, difficulty is faced when appropriate methods are not satisfying technical needs due to which businesses are affected at large [4]. All of these professional ensure that developed software are bug-free because they follow specific software development life cycle in order to make sure that each component of the software is developed under full consideration. Among these phases, software testing is also applied which is actually performed to assure quality and necessary fixes that are done to improve its functions [5].

Nowadays, reliable software development needs are not properly reviewed due to which businesses, as well as consumers, are facing difficulty in taking benefits from its use [6, 7]. Software testing has been given more attention in every aspect but due to outdated methods and techniques, certain technical needs are not properly fixed. In this regard, concerns have been placed to improve traditional methods by integrating modern approaches in order to improve software testing approaches. In the current methods, database approach is getting more attention due to its reliability and efficiency to fulfill testing needs thus, hybrid approaches have become research's focus recently. The following study is developed to address the need to overcome the problem(s) and introduce hybrid approach in software database testing [8, 9].

A. Paper Structure

Section II presents a brief description of utilization of software testing methodologies along with their key research contributors under the umbrella of literature review. Section III put forward the proposed software testing methodology based on hybrid database approach. Section IV emphasizes more specifically on the algorithm and execution of proposed software testing approach in an illustrative manner with a brief on limitations of the study. Section V and Section VI compares the performance of proposed methodology with the methodology based on traditional database testing approach by considering various parameters for testing goals and its accomplishment. Section VII concludes the proposed methodology better than the traditional approaches in terms of performance and foresees more refined methodologies even better. It also foresees future aspects of this research in terms of quantum computing and machine learning perspectives.

II. Literature Review

A. Software Testing

According to Bajaj, Kamini Simi [2], the process of software testing is not complicated but its approaches have increased its complexity to the greatest extent. The author further sheds light on the definition of software testing in terms of evaluation process in which the software is tested to ensure whether it is developed to meet system originality or not. Furthermore, the author also adds that the process of software testing comprises of validation and verification aspects that checks if the developed software is meeting certain criteria defined by the user [10]. The analysis of the study determines another important part of software testing which include results that defines the major difference between actual and expected result.

B. *Existing Testing Methods*

In software testing, the pre-defined traditional methods are recognized in almost every technical area and thus, their functionalities and approaches varies with the level of testing method. Based on the study of Arnicans, Guntis, and Vineta Arnicane [5], fundamental software testing methods incorporated in every aspect are black box testing, white box testing, and grey box texting. Different forms of database testing types and techniques have already been developed that are being preferably used according to the suitability and applicability of the specific type of database on a specific platform. The generally discussed types of database testing techniques are in the form of structural (internal) database testing, non-functional (external) database testing and functional (logical / conceptual) database testing [11, 12].

C. *Black Box Testing*

The paper proposed by Jamil, Muhammad Abid, Muhammad Arif, Normi Sham Awang Abubakar, and Akhlaq Ahmad [1] describes that black box testing only performs testing measures in evaluating software's functionalities rather than focusing on its implementation in detail. It is identified that black box software testing is appropriate at every level of SDLC in order to examine the bugs and errors within major functionalities. The basic function of the testing method is to assess the required functions and compares it with user requirements to verify if the application is developed according to desired needs [13, 14]. The following existing method is efficient in finding adequate functionalities by testing each phase at their minimum and maximum case value. Jamil, Muhammad Abid, Muhammad Arif, Normi Sham Awang Abubakar, and Akhlaq Ahmad [1] also explain that black box testing is one of the simplest and widespread methods which are mainly carried out by professionals across the globe.

D. *White Box Testing*

Muşlu, Kıvanç, YuriyBrun, and Alexandra Meliou [3] define the significance of white box testing in terms of its effectiveness and important functions. Basically, white box testing is one of the approaches that are famous for testing internal structure of the developed software. It is also evident that to perform white box testing, IT industry requires specific programming skills and knowledge as a pre-requisite in order to develop test cases. Another study provides more information regarding white box testing [15]. In the study, the method is also illustrated as clear box or glass box testing due to the fact that it validates and verifies internal mechanism to satisfy development process. In addition, white box testing is also known to be applied to different levels, such as unit, integration, and even system testing. It is also explained that among all other testing methodologies, white box testing is excellent due to its nature and complexity [16].

E. *Grey Box Testing*

With respect to grey box testing, Arnicans, Guntis, and Vineta Arnicane [5] define that it is hybrid in nature because it accompanies all the basic requirements and functions that are performed by black box and white box testing. As the approach carries advantages of both black box and white box testing, grey box testing is vitally used across different areas in order to evaluate vulnerabilities and security of the developed software. Inputs are provided from the front interface of the application in order to verify back-end data structure through debugging process which reveals internal culpabilities of database schema [17].

III. PROPOSED METHODOLOGY

To display the functions and phases of testing, following research proposes an algorithm for hybrid software testing database approach which can be used to develop an efficient and effective testing methodology for software developers [18]. Secondary sources are used to collect specific information regarding testing methods and emergence of new technology. For this purpose, different scholarly articles and tech blogs were reviewed. The research is designed on the basis of addressing technical processes and its impact on software development to ensure the effectiveness and efficiency of designed procedure. Illustrated "Fig. 1" beneath depicts the proposed methodology [19].

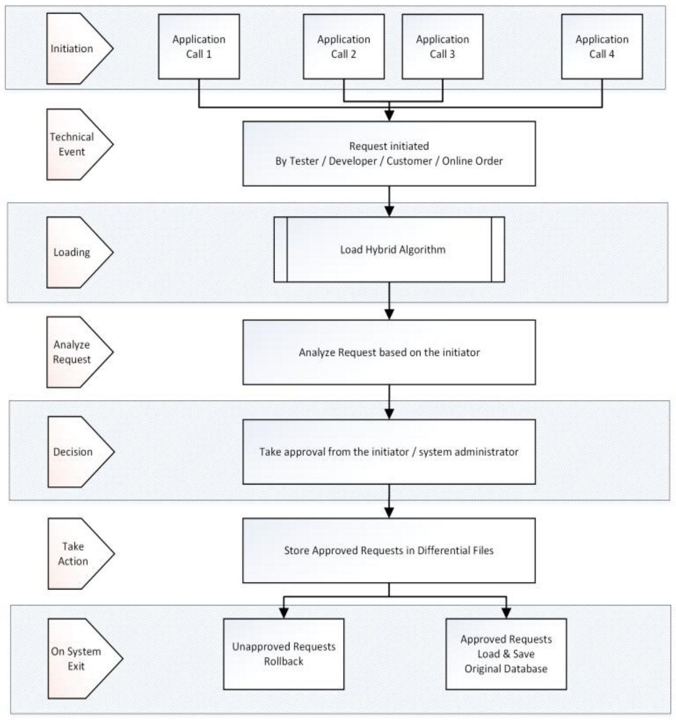

Initiation
Application Call 1
Application Call 2
Application Call 3
Application Call 4
Technical Event
Request initiated By Tester / Developer / Customer / Online Order
Loading
Load Hybrid Algorithm
Analyze Request
Analyze Request based on the initiator
Decision
Take approval from the initiator / system administrator
Take Action
Store Approved Requests in Differential Files
On System Exit
Unapproved Requests Rollback
Approved Requests Load & Save Original Database

Fig. 1. Proposed Methodology.

IV. Hybrid Database Approach

It is evident from recent studies that mode of testing practices are changing due to the software preferences, as well as concerns regarding quality. A new way of to test software applications have been proposed but limited information and significance is available to prove its efficiency. As illustrated in the study by MM Baig [20, 21, 22], database approach is one of the effective ways that provides new features to current testing methods in order to evaluate the performance and risks of the software by building effective test cases. It is also examined that by using hybrid technology, the upgraded features enable developers to assess improved quality of the application while it reduces amount of critical bugs in the application. Moreover, the proposed methods are also effective in terms of providing early information regarding issues that might affect performance of software while deviate its results from the expected outcomes [23].

An algorithm to execute hybrid testing method is proposed which is said to be more efficient and reliable as compared to traditional and hypothetical database testing method. In the process, when a request is generated by the initiator the algorithm analyzes it and based on the evaluation different states are maintained prior loading it into the original database [24, 25]. Treatment of initiated request(s) is as follows:

1) Request initiated by end user / customer / online order: Requests generated by Customer(s) and Online Order(s) are by default approved and original database are updated as per traditional way on reaching timestamps defined for the system.

2) Request initiated by Developer(s): Requests generated by developer(s) are treated through hypothetical database testing method in which a new database state would be generated, known as hypothetical database while it is ensured that the originality of the previously generated database is intact. This is due to the association established between the original database and differential table subject to approval from developer(s) side. Changes in the schema made by the developer can easily be implemented on the primary database at day end after stoppage of daily transactions.

3) Request initiated by Tester(s): When a tester executes a test case(s), the results will be displayed in the grid and will be viewed to tester only. If required these can be saved in differential files related to testing for future correspondence else they will be rolled back when the tester exits the system. This would be done in order to facilitate different anomalies that could be performed through differential file on hypothetical database states. The traditional approaches were using these anomalies on original databases which are inappropriate as its originality would be affected due to which tester face problems in analyzing actual requirements.

By referring to "Fig. 2", it can be viewed that tester implements first test case which is being analyzed by the hybrid algorithm which contains an instance of the primary database state. When the test case is successfully completed at this level, the updated results will be stored in differential file containing differential table ensuring that the originality of the primary database is not intact. When the second test case is run, same procedure will be followed. After execution of entire test suite if the desired results need to be stored in the original database with tester's login for future referencing it is possible only after approval. All the unapproved request(s) will be rolled back automatically [26, 27, 28].

In "Fig. 3" below, state transition diagram is illustrated which is based on different states. From the illustration, it can be viewed a transition will occur on the fulfillment of requirements by the hybrid database instead of original or hypothetically generated database. From the analysis of the following method, it is suggested that hypothetical rollback can be performed to any state rather than executing the action on the original database [29]. However, the time complexity of the roll back is quite efficient as it quickly reaches to the destination state as compared to traditional ways. With the fast and efficient approach, processing time is minimized, while costing and budgeting of the whole method is also reduced. Based on the proposed plan, the algorithm of testing includes number of test cases while test case generates number of differential tables in differential files. In the next step, software tester would be required some time to prepare more test cases as the previously generated cases are not appropriate by current database states in the hypothetical chain [30, 31].

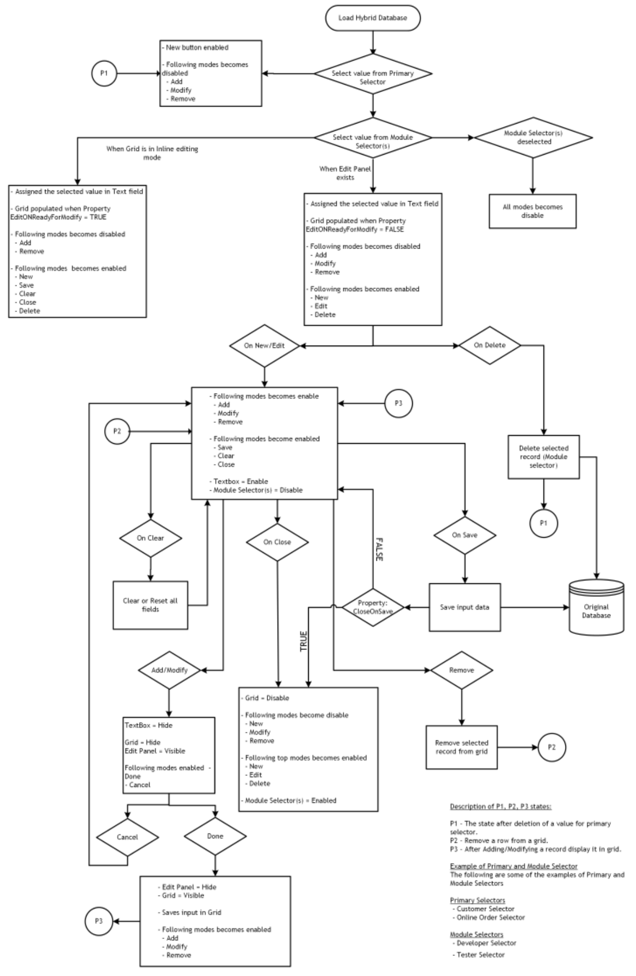

Load Hybrid Database
Select value from Primary Selector
- New button enabled
- Following modes becomes disabled
- Add
- Modify
- Remove
P1
Select value from Module Selector(s)
Module Selector(s) deselected
All modes becomes disable
When Grid is in Inline editing mode
- Assigned the selected value in Text field
- Grid populated when Property EditONReadyForModify = TRUE
- Following modes becomes disabled
- Add
- Remove
- Following modes becomes enabled
- New
- Save
- Clear
- Close
- Delete
When Edit Panel exists
- Assigned the selected value in Text field
- Grid populated when Property EditONReadyForModify = FALSE
- Following modes becomes disabled
- Add
- Modify
- Remove
- Following modes becomes enabled
- New
- Edit
- Delete
On New/Edit
On Delete
- Following modes becomes enable
- Add
- Modify
- Remove
- Following modes become enabled
- Save
- Clear
- Close
- Textbox = Enable
- Module Selector(s) = Disable
P2
P3
Delete selected record (Module selector)
P1
On Clear
On Close
On Save
FALSE
Clear or Reset all fields
Property: CloseOnSave
Save input data
Original Database
TRUE
Add/Modify
Remove
TextBox = Hide
Grid = Hide
Edit Panel = Visible
Following modes enabled - Done
- Cancel
- Grid = Disable
- Following modes become disable
- New
- Modify
- Remove
- Following top modes becomes enabled
- New
- Edit
- Delete
- Module Selector(s) = Enabled
Remove selected record from grid
P2
Cancel
Done
- Edit Panel = Hide
- Grid = Visible
- Saves input in Grid
- Following modes becomes enabled
- Add
- Modify
- Remove
P3
Description of P1, P2, P3 states:
P1 - The state after deletion of a value for primary selector.
P2 - Remove a row from a grid.
P3 - After Adding/Modifying a record display it in grid.
Example of Primary and Module Selector
The following are some of the examples of Primary and Module Selectors
Primary Selectors
- Customer Selector
- Online Order Selector
Module Selectors
- Developer Selector
- Tester Selector

Fig. 2. Hybrid Database Approach.

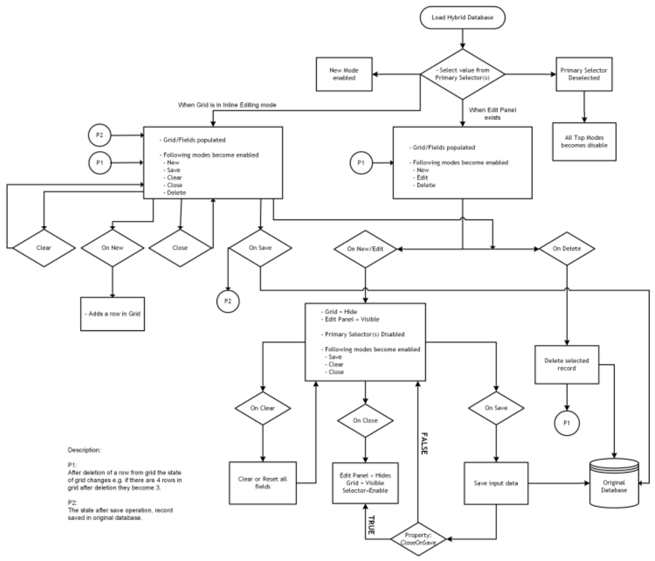
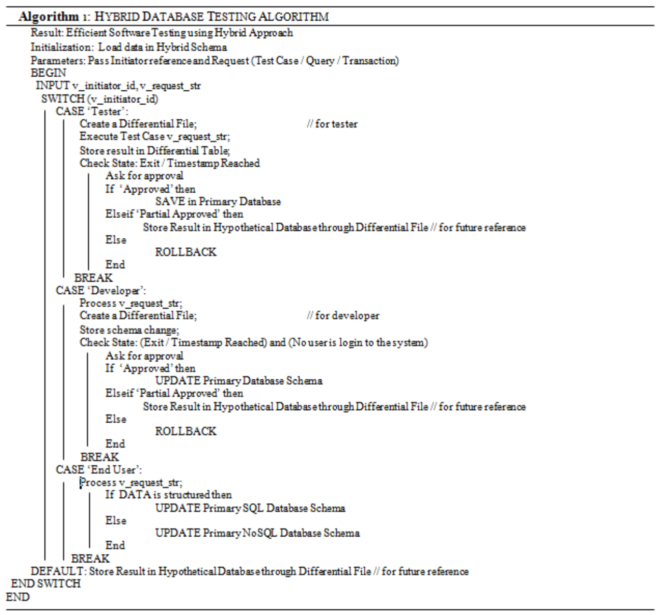

Load Hybrid Database
- Select value from Primary Selector(s)
New Mode enabled
Primary Selector Deselected
All Top Modes becomes disable
When Grid is in Inline Editing mode
When Edit Panel exists
P2
P1
- Grid/Fields populated
- Following modes become enabled
- New
- Save
- Clear
- Close
- Delete
- Grid/Fields populated
- Following modes become enabled
- New
- Edit
- Delete
Clear
On New
Close
On Save
On New/Edit
On Delete
- Adds a row in Grid
P2
- Grid = Hide
- Edit Panel = Visible
- Primary Selector(s) Disabled
- Following modes become enabled
- Save
- Clear
- Close
Delete selected record
P1
On Clear
On Close
On Save
FALSE
Clear or Reset all fields
Edit Panel = Hides Grid = Visible Selector=Enable
Save input data
Original Database
TRUE
Property: CloseOnSave
Description:
P1:
After deletion of a row from grid the state of grid changes e.g. if there are 4 rows in grid after deletion they become 3.
P2:
The state after save operation, record saved in original database.

Fig. 3. State Transition Diagram.

State transition illustrated above completely elaborates the mechanism that a tester would be following in order to achieve desired results. In the following situation, another concept would be proposed in order to fix the state while new generated databases would comply with the established standards. The concept of database rollback will also be used in the following situation in order to let generated databases fulfill conditions of developed test cases. The main purpose for using the option in current testing methodology is that it increases efficiency of the database generation while the preparation time of current state is improved [32, 33].

A. *Hybrid Database Testing Algorithm*

HYBRID DATABASE TESTING ALGORITHM[1] to propose hybrid plan is detailed in "Fig. 4" beneath.



```
Algorithm 1: HYBRID DATABASE TESTING ALGORITHM
   Result: Efficient Software Testing using Hybrid Approach
   Initialization:  Load data in Hybrid Schema
   Parameters: Pass Initiator reference and Request (Test Case / Query / Transaction)
   BEGIN
    INPUT v_initiator_id, v_request_str
     SWITCH (v_initiator_id)
       CASE 'Tester':
           Create a Differential File;                    // for tester
           Execute Test Case v_request_str;
           Store result in Differential Table;
           Check State: Exit / Timestamp Reached
               Ask for approval
               If  'Approved' then
                       SAVE in Primary Database
               Elseif 'Partial Approved' then
                     Store Result in Hypothetical Database through Differential File // for future reference
               Else
                       ROLLBACK
               End
         BREAK
       CASE 'Developer':
           Process v_request_str;
           Create a Differential File;                    // for developer
           Store schema change;
           Check State: (Exit / Timestamp Reached) and (No user is login to the system)
               Ask for approval
               If  'Approved' then
                       UPDATE Primary Database Schema
               Elseif 'Partial Approved' then
                     Store Result in Hypothetical Database through Differential File // for future reference
               Else
                       ROLLBACK
               End
          BREAK
       CASE 'End User':
           Process v_request_str;
               If  DATA is structured then
                       UPDATE Primary SQL Database Schema
               Else
                       UPDATE Primary NoSQL Database Schema
               End
         BREAK
   DEFAULT: Store Result in Hypothetical Database through Differential File // for future reference
  END SWITCH
 END
```

Fig. 4. Hybrid Database Approach Algorithm.

B. *Roadmap towards Hybrid Database Testing*

The strategy to propose hybrid database testing plan is elaborated in Table I below.

C. *Limitations of the Proposed Study*

Different challenges are associated with database testing due to nature of databases which is complex both structural-wise and magnitude-wise. Testers skill set ascends a handful of challenges related to designing of test cases and the tactics to execute them with proper exploration. Core challenges were related to database schema structure, cleansing / synchronization / reliability of quality data and under-testing / incomplete testing of colossal database. However, efforts were made to overcome these issues in the proposed plan so that more effective method could be developed.

TABLE I. HYBRID DATABASE TESTING ROADMAP

	Description
OBJECTIVE	To perform database testing in order to uncover incomplete schema, malfunctioned functionalities, data corruption, deadlocks, data mapping issues and exceptions.
TESTING CRITERIA	Testing of all key database schema tables, methods, processes, sequences, functions, indices, views, cursors, triggers and stored procedures.
PRE-REQUISITES	Testing requirements are well communicated / well documented to testers. Test cases covering all aspects of database testing are designed. Success criteria have been established prior testing phase. Test environment is setup and freeze with latest database schema.
EXCEPTIONAL CONTEMPLATION	Testing conducted with real time data on actual environment. Automatic invocation of stored procedures and processes. For large databases DBMS Development Environment is required to populate data directly into the database from backend in order to monitor its frontend adaptation. For small sized databases limited records are generated to test non-acceptable events / triggers / exceptions.
TESTING TECHNIQUE	Selection of testing technique / strategy is based on the fact it must support the testing of all key use-case scenarios and complete business flows i.e. main features.
REQUIRED TOOLS	➢ Test Management Tools ➢ Test Script Automation Tool ➢ SQL Query Analyzer ➢ Test Data Generator ➢ Bug Tracking tool ➢ Backup and Recovery tools
PROCEDURAL STEPS	Testers will work according to the database testing checklist and guidelines to inspect the database ensuring proper data for correct reasons is inserted and stored in the database.
POST-REQUISITES	Execute each test case separately but sequential using valid and invalid data should reveal expected results for valid data and timely error messages to refrain insertion of invalid data in the database.

From the above algorithm, proposed strategy and limitations it is deduced that existing methods are inefficient and provides limited access to basic software testing which is why appropriate results are not achieved. By using hybrid software testing database approach, software developers and testers would gain more assistance in testing phase while there will be the exceptions for increased bugs and incidents. Moreover, it is considered as efficient while performance is also adequate because of less time consumption, cost-effective and limited resource utilization. This algorithm is not only convenient but also gives easy access to acknowledge basic requirements that must be present in developed software [34].

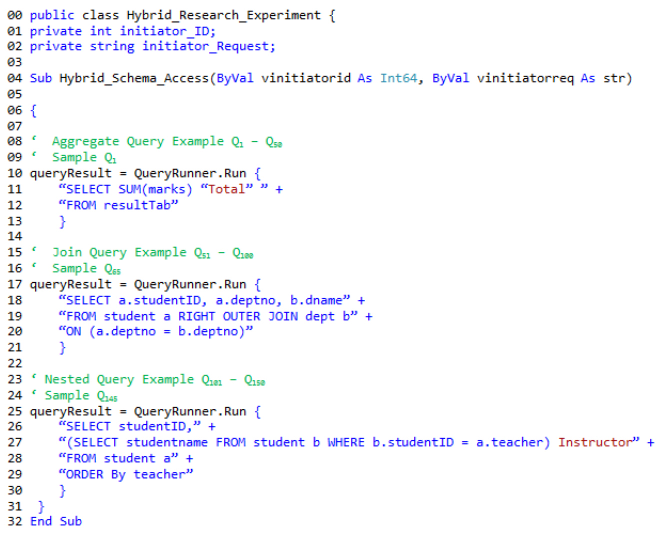
V. Results with Discussion

To check the efficacy of the algorithm it multiple queries and test case(s) were designed and run on the schemas separately in order to monitor their performance. Results of both queries and test case(s) are discussed below.

A. *Query Processing Time*

Multiple queries as illustrated in "Fig. 5"were designed as per the following types: Aggregate Queries (Q_1 – Q_{50}), Join Queries (Q_{51} – Q_{100}) and Nested Queries (Q_{101} – Q_{150}) where "$Q_{\#}$" is used to represent "Query $_{\text{number}}$". These queries were run on a sample database of following size(s): 400 MB, 4 GB, 40 GB and 400 GB. Due to classified database architecture and complex confidential queries only general queries related to student database are being shared in "Fig. 5"for assistance.

To analyze the processing time in seconds these queries were first executed through the traditional database testing approach, secondly through hypothetical database testing approach and lastly through hybrid database approach. According to the results shown in Table II below, it can be concluded that the processing time of the queries illustrated in "Fig. 6", "Fig. 7" and "Fig. 8" has greatly reduced approximately 70% with the use of hybrid database approach as opposed to accessing the data directly.



```
public class Hybrid_Research_Experiment {
private int initiator_ID;
private string initiator_Request;

Sub Hybrid_Schema_Access(ByVal vinitiatorid As Int64, ByVal vinitiatorreq As str)

{

‘  Aggregate Query Example Q1 – Q50
‘  Sample Q1
queryResult = QueryRunner.Run {
    “SELECT SUM(marks) “Total” ” +
    “FROM resultTab”
    }

‘  Join Query Example Q51 – Q100
‘  Sample Q65
queryResult = QueryRunner.Run {
    “SELECT a.studentID, a.deptno, b.dname” +
    “FROM student a RIGHT OUTER JOIN dept b” +
    “ON (a.deptno = b.deptno)”
    }

‘ Nested Query Example Q101 – Q150
‘ Sample Q145
queryResult = QueryRunner.Run {
    “SELECT studentID,” +
    “(SELECT studentname FROM student b WHERE b.studentID = a.teacher) Instructor” +
    “FROM student a” +
    “ORDER By teacher”
    }
 }
End Sub
```

Fig. 5. Sample Query Architecture.

TABLE II. Query Processing Time recorded on Different Schemas

Query Processing Time				
Query Types	***Database Size***	**Processing Time Recorded**		
		Direct Schema	***Hypothetical Schema***	***Hybrid Schema***
Aggregate Queries (Q_1 – Q_{50})	400 MB	≈ 300 s	≈ 180 s	≈ 60 s
	4 GB	≈ 720 s	≈ 420 s	≈ 90 s
	40 GB	≈ 900 s	≈ 720 s	≈ 120 s
	400 GB	≈ 1320 s	≈ 1140 s	≈ 300 s
Join Queries (Q_{51} – Q_{100})	400 MB	≈ 420 s	≈ 300 s	≈ 120 s
	4 GB	≈ 900 s	≈ 600 s	≈ 240 s
	40 GB	≈ 1080 s	≈ 840 s	≈ 300 s
	400 GB	≈ 1500 s	≈ 1200 s	≈ 480 s
Nested Queries (Q_{101} – Q_{150})	400 MB	≈ 480 s	≈ 240 s	≈ 180 s
	4 GB	≈ 1020 s	≈ 720 s	≈ 360 s
	40 GB	≈ 1200 s	≈ 900 s	≈ 540 s
	400 GB	≈ 1620 s	≈ 1200 s	≈ 600 s

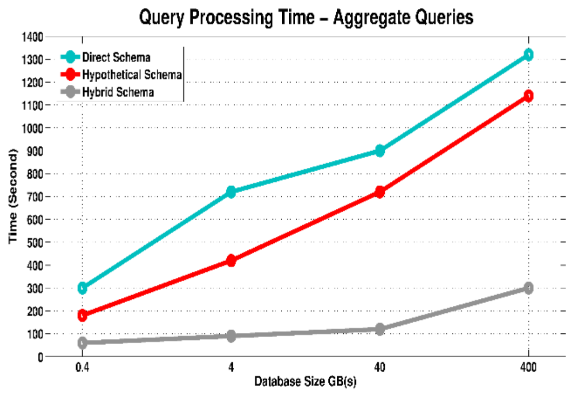


Fig. 6. Aggregate Queries Processing Time.

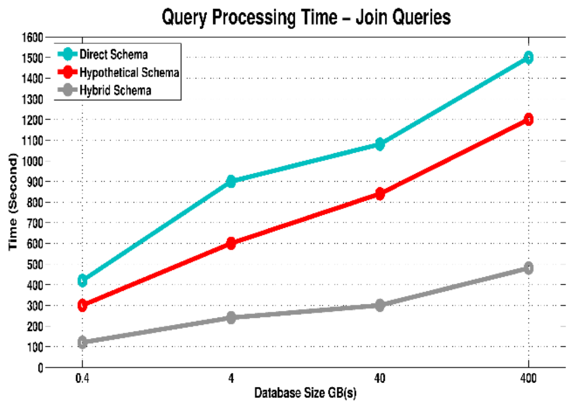


Fig. 7. Join Queries Processing Time.

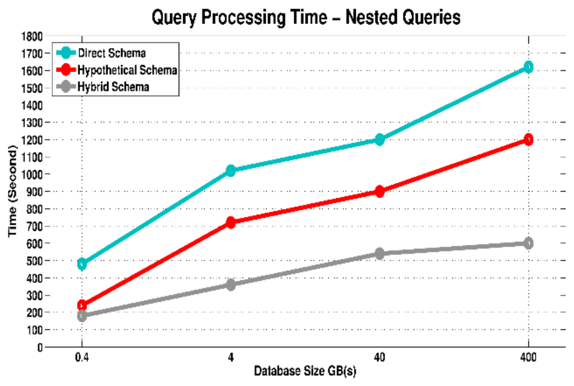
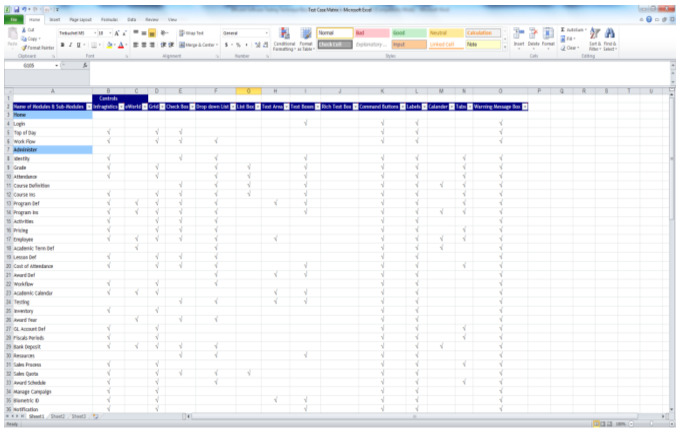


Fig. 8. Nested Queries Processing Time.

B. *Test Case(s) Execution Time*

Test Case(s) Matrix as per "Fig. 9" was designed and executed on the database schema designed hypothetically and hybrid in addition to direct database schema. To check the functional aspects of database "Functional Test Case(s): TC_1 – TC_{50}" and for non-functional aspects "Non-Functional Test Case(s): TC_{51} – TC_{100}" were designed where "$TC_{\#}$" is used to represent "Test Case $_{\text{number}}$". These test cases were run on the same sample database of size(s): 400 MB, 4 GB, 40 GB and 400 GB used for query execution.



	Controls													
Name of Modules & Sub-Modules	Infragistics	eWorld	Grid	Check Box	Drop down List	List Box	Text Area	Text Boxes	Rich Text Box	Command Buttons	Labels	Calander	Tabs	Warning Message Box
Home														
Login								√		√	√			√
Top of Day	√		√	√						√	√			√
Work Flow	√		√	√	√					√	√			√
Administer														
Identity	√			√	√			√		√	√		√	√
Grade	√		√		√	√		√		√	√		√	√
Attendance	√		√		√	√		√		√	√		√	√
Course Definition				√	√	√		√		√	√	√	√	√
Course Ins	√		√	√	√	√		√		√	√		√	√
Program Def	√	√	√	√	√		√	√		√	√		√	√
Program Ins	√	√	√	√	√			√		√	√	√	√	√
Activities	√		√	√	√					√	√			√
Pricing	√		√	√	√					√	√		√	√
Employee	√	√	√	√	√		√			√	√	√	√	√
Academic Term Def		√			√					√	√	√		√
Lesson Def	√		√	√	√					√	√			√
Cost of Attendance	√		√	√	√			√		√	√		√	√
Award Def					√		√	√		√	√			√
Workflow	√		√		√					√	√			√
Academic Calendar	√	√	√				√	√		√	√			√
Testing				√	√		√	√		√	√			√
Inventory	√		√							√	√			√
Award Year		√		√	√					√	√			√
GL Account Def	√		√							√	√		√	√
Fiscals Periods	√		√							√	√		√	√
Bank Deposit	√	√	√	√	√					√	√	√		√
Resources				√	√			√		√	√			√
Sales Process	√		√							√	√		√	√
Sales Quota	√		√	√	√	√				√	√			√
Award Schedule	√		√		√					√	√		√	√
Manage Campaign	√		√							√	√			√
Biometric ID	√		√				√	√		√	√			√
Notification	√		√					√		√	√			√

Fig. 9. Test Case Matrix.

To analyze the execution time in minutes these test case(s) were first executed through the traditional database testing approach, secondly through hypothetical database testing approach and lastly through hybrid database approach. According to the results illustrated in Table III below, it can be concluded that the overall execution time of test case(s) as shown in "Fig. 10" and "Fig. 11" almost reduced to 60% of the actual processing time required to execute the test case(s) directly on the primary database.

TABLE III. TEST CASE(S) EXECUTION TIME RECORDED ON DIFFERENT SCHEMAS

Test Case(s) Execution Time				
Test Suites	***Database Size***	**Execution Time Recorded**		
		Direct Schema	***Hypothetical Schema***	***Hybrid Schema***
Functional Test Cases (TC_1 – TC_{50})	400 MB	≈ 15min	≈ 11min	≈ 5min
	4 GB	≈ 20min	≈ 16min	≈ 8min
	40 GB	≈ 33min	≈ 25min	≈ 18min
	400 GB	≈ 45min	≈ 31min	≈ 23min
Non-Functional Test Cases (TC_{51} – TC_{100})	400 MB	≈ 18min	≈ 10min	≈ 4min
	4 GB	≈ 22min	≈ 11min	≈ 7min
	40 GB	≈ 38min	≈ 27min	≈ 15min
	400 GB	≈ 56min	≈ 35min	≈ 21min

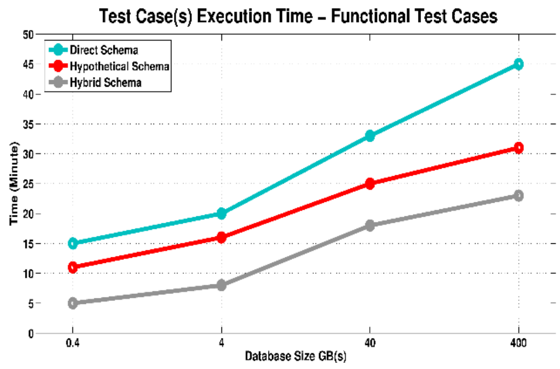


Fig. 10. Functional Test Case(s) Execution Time.

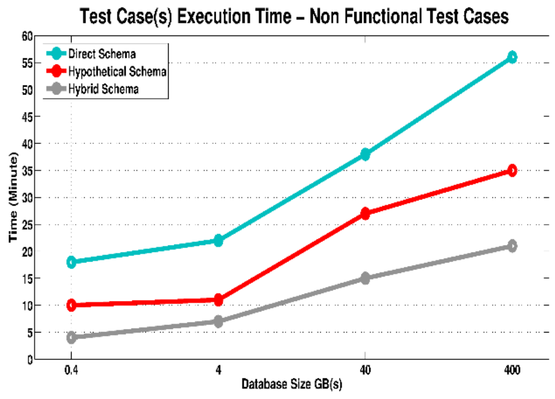


Fig. 11. Non - Functional Test Case(s) Execution Time.

VI. Testing Goals Achieved

Testing goals which were achieved are listed as:

1) Normalization rules: No data was repetitive in the database and all columns were logically connected.

2) Data types: As the systems are taking data in dynamic nature, deciding the type of data columns is very crucial. In some scenarios, you can expect some extremely weird inputs which are important. Make sure the data is going into the right columns and cells. This thing is taken care very seriously as a mismatch of the columns can cause huge issues in the stability of the database.

3) Retrieval of data and number of joins: Usually, developers avoid adding multiple joins or making query complex mainly because of the system's speed and its response time.

4) Data endpoints: It always checks the data populating the database from the system v/s data generating by the database for the database (triggers or metadata).

5) Usage of the flag: For the system to work properly, mainly columns worked as a flag for a different thing. e.g. 0=admin user, 1= Developer, 2 = Tester etc. This is also tested seriously to control the access level permissions.

6) In some big systems: we also test the write speed. Sometimes, the data coming to the database is huge, and the database is not keeping up the phase. So the threshold was also tested. Mainly it happens in ecommerce website where user actions and activities are also recorded against their profile.

7) Database security and password encryption issues.

8) Online testing of software application in parallel with order processing or daily transactional operations without the use of separate test environment was possible with add-on functionality of roll back.

VII. Conclusion and Future Work

During software development, it becomes quite difficult for the developers and testers to identify the bugs and required test case(s) that are needed to be executed to achieve actual results. Hybrid Database testing approach was formulated and tested with execution of queries and test case(s) on sample database(s) of sufficient sizes reflecting the achievement of testing goals in an efficient manner. Using the proposed hybrid plan, businesses would improve in terms of using effective software testing methods without creating separate environments that would cause them to invest little while productive results are achieved. It is also acknowledged that complexity and time-consuming activities in recognizing errors in the software testing phase can be easily managed due to the flexibility it provides in searching and updating records in the primary database without keeping it intact during the whole course of time. Assessments of requests generated from different initiators can also be processed simultaneously with segregation as per requirement.

Moreover, in future, it is anticipated and expected to witness software testing tools being developed with build-in features of hybrid state transition between databases with integration of quantum technology and machine learning techniques. In software testing that would include different processes in order to provide accurate and unambiguous results in even lesser time. As the new technology is becoming famous in IT industry, it is expected to see more transformations in the field of software testing due to which traditional approaches would be diminished while integration of hybrid technologies would be seen.

Acknowledgement

NED University of Engineering & Technology is highly acknowledged for providing research support required to carry out this research.

References

[1] Jamil, Muhammad Abid, Muhammad Arif, Normi Sham Awang Abubakar, and Akhlaq Ahmad. "Software Testing Techniques: A Literature Review." In 2016 6th International Conference on Information and Communication Technology for The Muslim World (ICT4M), pp. 177-182. IEEE, 2016.

[2] Bajaj, Kamini Simi. "Hybrid Test Automation Framework for managing Test Data." International Journal of Pure and Applied Mathematics 118, no. 9 (2018): 265-276.

[3] Muşlu, Kıvanç, Yuriy Brun, and Alexandra Meliou. "Data debugging with continuous testing." In Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp. 631-634. ACM, 2013.

[4] Barr, Earl T., Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. "The oracle problem in software testing: A survey." IEEE transactions on software engineering 41, no. 5 (2015): 507-525.

[5] Arnicans, Guntis, and Vineta Arnicane. "Opportunities to Improve Software Testing Processes on the Basis of Multi-Agent Modeling." In Databases and Information Systems V: Selected Papers from the Eighth International Baltic Conference, DB&IS 2008, vol. 187, pp. 143-156. IOS Press, 2009.

[6] Felderer, Michael, Matthias Büchler, Martin Johns, Achim D. Brucker, Ruth Breu, and Alexander Pretschner. "Security testing: A survey." In Advances in Computers, vol. 101, pp. 1-51. Elsevier, 2016.

[7] Takanen, A., Demott, J. D., Miller, C., & Kettunen, A. (2018). Fuzzing for software security testing and quality assurance. Artech House.

[8] Hogan, R. (2018). A practical guide to database design. Chapman and Hall/CRC.

[9] Felderer, M., Russo, B., & Auer, F. (2019). On Testing of Data-Intensive Software Systems. arXiv preprint arXiv:1903.09413.

[10] R. V. Binder. Testing Object-Oriented Systems Models, Pat- terns, and Tools. Addison Wesley Longman, Inc., Reading, MA, 2000.

[11] S. Berner, R. Weber, and R. Keller. Observations and lessons learned from automated testing. In Proc. 27th Int. Conf. on Sw. Eng., pages 571–579. ACM, 2005.

[12] Bertolino, A. (2007, May). Software testing research: Achievements, challenges, dreams. In 2007 Future of Software Engineering (pp. 85-103). IEEE Computer Society.

[13] Belinfante, Axel, Lars Frantzen, and Christian Schallhart. "14 tools for test case generation." *Model-Based Testing of Reactive Systems*. Springer, Berlin, Heidelberg, 2005. 391-438.

[14] G. Bernot, M. C. Gaudel, and B. Marre. Database testing based on formal specifications: a theory and a tool. Softw. Eng. J., 6(6):387–405, 1991.

[15] A. Bertolino and E. Marchetti. Software testing (chapt.5). In P. Bourque and R. Dupuis, editors, Guide to the Soft- ware Engineering Body of Knowledge SWEBOK, 2004 Version, pages 5–1–5–16. IEEE Computer Society, 2004. http://www.swebok.org.

[16] Bertolino, Antonia, Eda Marchetti, and Henry Muccini. "Introducing a reasonably complete and coherent approach for model-based testing." Electronic Notes in Theoretical Computer Science 116 (2005): 85-97.

[17] Bertolino, Antonia, and Andrea Polini. "The audition framework for testing web services interoperability." In 31st EUROMICRO Conference on Software Engineering and Advanced Applications, pp. 134-142. IEEE, 2005.

[18] A. Bertolino, A. Polini, P. Inverardi, and H. Muccini. To- wards anti-model-based testing. In Proc. DSN 2004 (Ext. abstract), pages 124–125, 2004.

[19] Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., & Grünbacher, P. (Eds.). (2006). Value-based software engineering. Springer Science & Business Media.

[20] M. M. Baig, A. Shaheen and A. A. Khan, “Efficient Execution Plan for Hypothetical Database Testing”, International Journal of Latest Trends in Computing(IJLTC), 2011 (Peer reviewed).

[21] M. M. Baig and A. A. Khan, “Efficient Testing of Database Applications”, IJCSNS International Journal of Computer Science and Network Security, Vol.9 No. 4, 2009. (Peer reviewed).

[22] M. M. Baig, and A. A. Khan, “Database Testing Application for Modifying Tuples Hypothetically”, NED Journal of Research, 2012. (HEC recognised).

[23] Aleem, H., Anwar, S., Shariff, I., & Abdul Aziz, S. (2014). Six sigma application: an order management system. *IOSR Journal of Humanities and Social Science*, *19*(1), 95-100.

[24] Hao, D., Zhang, L., & Mei, H. (2016). Test-case prioritization: achievements and challenges. Frontiers of Computer Science, 10(5), 769-777.

[25] Ramler, Rudolf, Stefan Biffl, and Paul Grünbacher. "Value-based management of software testing." *Value-based software engineering*. Springer, Berlin, Heidelberg, 2006. 225-244.

[26] Li, C., & Gu, J. (2019). An integration approach of hybrid databases based on SQL in cloud computing environment. Software: Practice and Experience, 49(3), 401-422.

[27] Ahmad, J., ul Hassan, A., Naqv, T., & Mubeen, T. (2019). A Review on Software Testing and its Methodology. i-Manager's Journal on Software Engineering, 13(3), 32.

[28] Vyawahare, H. R., Karde, P. P., & Thakare, V. M. (2019). Hybrid Database Model For Efficient Performance. Procedia Computer Science, 152, 172-178.

[29] H. R. Vyawahare, P. P. Karde and V. M. Thakare (2018), “A Hybrid Database Approach Using Graph and Relational Database”, International Conference on Research in Intelligent and Computing in Engineering (RICE), San Salvador,doi: 10.1109/RICE.2018.8509057 1-4.

[30] Wood, L. Y. N. N., Elsethagen, T. O. D. D., Schram, M. A. L. A. C. H. I., & Stephan, E. R. I. C. (2017, November). Conditions database for the Belle II experiment. In J. Phys. Conf. Ser.(Vol. 898, p. 042060).

[31] Lewis, W. E. (2017). Software testing and continuous quality improvement. Auerbach publications.

[32] Kassab, M., DeFranco, J. F., & Laplante, P. A. (2017). Software testing: The state of the practice. IEEE Software, 34(5), 46-52.

[33] James, B. E., & Asagba, P. O. (2017). Hybrid database system for big data storage and management. International Journal of Computer Science, Engineering and Applications (IJCSEA), 7(3/4), 15-27.

[34] Jonsson, P. (2019). Automated Testing of Database Schema Migrations.