Chapter 1

Introduction

1.1 Introduction

Data is an important asset for companies and organizations. Some of these data are worth lacs and crores. The organizations need to take great care at controlling access to such data, from the perspective of both internal users, within the organization, and external users, outside the organization. The organization reputation and client's beliefs lies with the confidentiality of these data. Thus, the development of Database Management Systems (DBMSs) with high-assurance security is always a hot & spicy research topic. Even though DBMSs provide access control mechanisms, these mechanisms alone are not enough to guarantee data security.

Each database (DB) user is authorized to do particular transactions by performing a sequence of queries. These queries perform operations (select, insert, update, delete) on various attributes of different tables in RDBMS. However the sequence of queries, attributes & tables authorized to User1 are not authorized to User2. User2 may be authorized to perform a different sequence and types of queries. For example, consider that a database user/application is authorized to access data related to HR tables and only non financial attributes of tables as EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, PHONE_NUMBER, HIRE_DATE, JOB_ID, etc. but suddenly that user/application submits a SQL command to the DBMS that accesses the records from the Finance Tables or access financial attributes as SALARY, LAST_PAY, GROSSPAY etc. of tables. Such anomalous access pattern of the SQL command may be the result of an SQL Injection vulnerability or privilege abuse by an authorized user.

Even the database administrators (DBAs) be monitored, and responded to if deemed malicious. This is a difficult problem to address since the policies that specify a response action need to be created for the DBAs who are, in turn, responsible for managing the same policies. A Case study for the same is carried out for Oracle based applications databases of Indian Railways and the solutions adopted. *Oracle Vault* tool is used in which audit logs are maintained and administered by Oracle vault user and DBA roles are also performed by DBA user with the persuasion of one or more DBA users. Thus instead of one user, a combined persuasion of 2-3 users is required for access. But the fact is that there is no automated solution for catching an Unauthorized Transaction (UT) and responding to it with an Unauthorized Transaction Response (UTR) for Database security purposes.

Here, we have tried to propose an automated approach through Decision Tables created with the help of TQLC (Transaction Queries Log Crawler), Authorized Query Indicator Array (AQIA), Transaction Query Sequence Analyzer Arrays (TQSAA), Query Weight Analysis Algorithm and Quad Phase Verification Techniques.

1.2 Organization of the Thesis

This thesis is divided into eight chapters. A brief overview of all the chapters is as follows:-

- Chapter 2, discusses the Literature review and related work done in the field.
- Chapter 3, consists of the methodology proposed in our model.
- Chapter 4, describes the Quad-Phase Verifier in detail.
- Chapter 5, focuses on Decision Tables and their usage.
- In Chapter 6, an Oracle-J2EE based implementation of the proposed model is presented.
- In Chapter 7, the experimental results evaluation and analysis is presented.
- Chapter 8 concludes the methodology and future work scope.

Chapter 2

Literature Survey

In 2005 Marco and Henrique [1] proposed a Database Malicious Transaction Detector (DBMTD). DBMTD mechanism is a log based mechanism for the detection of malicious transactions in DBMS. The detection model proposed in [1] is used to detect the malicious transaction after the transaction has been committed. Lee et al.[2] propose an approach for detecting illegitimate database accesses by finger-printing every transaction, mainly by summarizing SQL statements into compact regular expression finger prints. Mathew et al. [3] propose a data-centric approach for solving the AD problem in a DBMS. They model users' access patterns by profiling the used data points. [4] signifies that Anomaly Detection mechanisms are essential to detect anomalies in data accesses by users. Such anomalies may be indicative of insider attacks or compromised database user accounts. [5,6] proposes a methodology for discovering user behaviour from web log data. The concept of Log Data Usage has been derived from here. [7] covers important aspects of Oracle based auditing; from basic configuration to advanced techniques which helped us in designing Log Crawler.[8] proposes a Role Based Access Control (RBAC) methodology which helps us in designing Parsing techniques for different arrays design..

To the best of our knowledge, no work demonstrating use of Decision Tables with the help of TQLC (Transaction Queries Log Crawler), Authorized Query Indicator Arrays (AQIA), Transaction Query Sequence Analyzer Arrays (TQSAA), Query Weight Analysis and Quad Phase Verification Techniques for prevention of UT and responding with UTR for DBMS exists.

Chapter 3

Proposed Methodology

Our methodology as shown in Fig 1 starts with the use of TQLC (Transaction Queries Log Crawler). TQLC is a utility (can be implemented using PL/SQL procedures or Java Code) which scans through the Database AUDIT logs and is responsible for creating valid User Transaction Profiles i.e. User Authorized Transactions. All available enterprise databases come bundled with some inbuilt audit mechanisms. Oracle database has certain Audit tables in System schema which keeps the track of executed queries for particular database users. TQLC collects last 6-10 transactions performed by a User and analyze the queries involved in those transactions. The authorized transactions will always have same set of queries in same sequence. TQLC looks for such transactions and declare them as Authorized Transactions (AT). Such ATs are identified user wise and an AT store is maintained. Each AT in AT store is taken one by one and each query of an AT is parsed in specific ways to generate AQIA and TQSAA.

An example demonstrating the above discussed steps is presented for better understanding of concept. Suppose an Oracle RDBMS instance having an Employee centered schema with Relations (Tables) as EMPLOYEES, DEPARTMENTS, JOB_HISTORY, JOBS, etc. The trusted users for the Database are User1, User2 & User3. The Database Audit Tables are configured to store the Audit log records of all SELECT, INSERT, UPDATE and DELETE commands executed over any trusted user. TQLC reads the Audit Log Table records, recognize ATs and fill AT store. The sample Audit data for a user extracted from Audit log tables is shown as Table 1:

Operation	**SQL Text**	**Object**	**User**
SELECT	Select ……. From Employees	EMPLOYEES	User1

SELECT	Select ……. From JOB_History	JOB_HISTORY	User1
INSERT	Insert into JOB_History () values (….)	JOB_HISTORY	User1
INSERT	Insert into Employees () values (….)	EMPLOYEES	User1
SELECT	Select ……. From Employees	EMPLOYEES	User1
SELECT	Select ……. From Jobs	JOBS	User1

Table 1: Data from Audit Tables

The AT store entries populated with the help of above specified data extracted is shown as Table 2:

Operation	**Referred Attributes**	**Objects**	**Query Serial in Transaction**	**User**
SELECT	EmployeeID, FirstName, JobID…	Employees, Job	1	User1
SELECT	StartDate, JobID	JobHistory	2	User1
INSERT	Employee, Startdate…	JobHistory	3	User1
INSERT	EmployeeID, FirstName	Employees	4	User1
SELECT	EmployeeID, FirstName	Employees	5	User1
SELECT	JobID, JobTitle	Jobs	6	User1

Table 2: AT Store entries

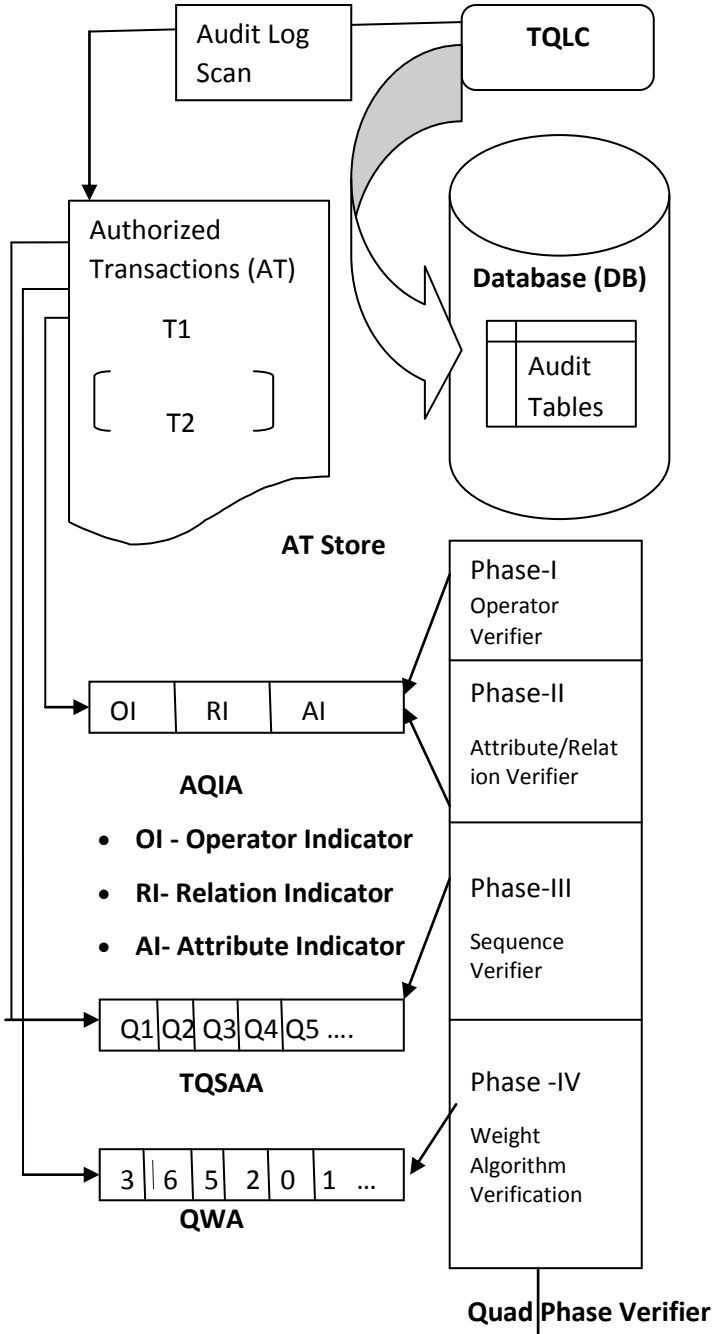
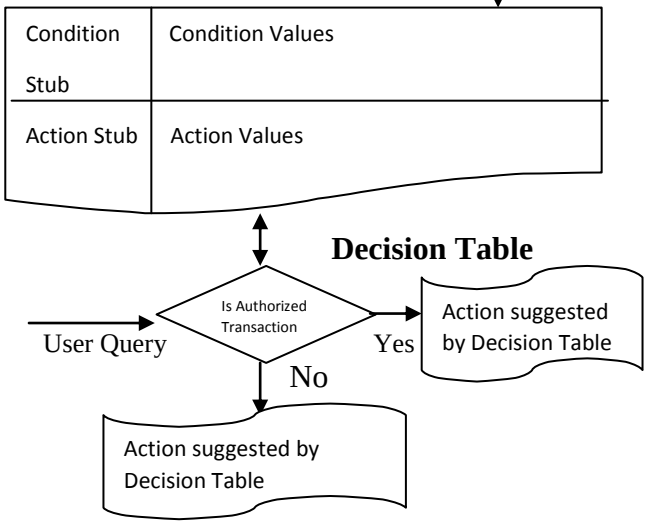


Fig 1. Quad Phase Verifier



Each entry in AT store for a Transaction is parsed to generate AQIA and TQSAA. AQIA (Authorized Query Indicator Array) is a two dimensional array representation of a Transaction.

3.1 AQIA (Authorized Query Indicator Array)

AQIA is an array of following arrays:-

1. OI (Operator Indicator)
2. RI (Relation Indicator)
3. AI (Attribute Indicator)

Suppose an AT taken from AT store consists of 6 queries in a particular sequence as shown in Table 2. The value of 2 dimensional array AQIA is shown in tabular form. Each AT store entry is represented in AQIA table e.g. First entry of AT store in Table 2 is:

Select -- EmployeeID, FirstName, JobID -- Employee, Job -- 1-- User1

The four DML (Data Manipulation Language) operators of SQL are – SELECT, INSERT, UPDATE, DELETE. For OI representation in AQIA table, these operators are symbolized as 0,1,2,3 respectively. Thus OI value for the sample query taken from AT is 0.

Suppose the total number of Relations (tables) in the schema is 4. The sample query taken uses two relations EMPLOYEES & JOBS. For RI representation, first of all a bit stream equal to total number of tables is initialized with value 0 as

0000

Now the bit values of those tables which are referred should be set ON. If EMPLOYEES & JOBS are the table numbers 2 and 4 in schema then bit representation now becomes

0101

Thus RI value for the sample query taken is 0101. The AI (Attribute Indicator) will indicate the attributes of relations referred in the query. In the sample query taken, there are two relations (tables) referred i.e. Employee (10 attributes) & Job (5 attributes). For AI representation, first of all a bit stream equal to total number of attributes in the table is initialized with value 0 as

0000000000 | 00000

Now the bit values of those attributes which are referred should be set ON. If EmployeeID, FirstName are the attribute number 1, 2 in EMPLOYEES table and JOB_ID is the attribute number 1 in Job table then bit representation now becomes

1100000000 | 10000

On applying the similar pattern to all the AT queries for one AT, the AQIA table obtained is shown as Table 3:-

OI	RI	AI
0	0101	1100000000|10000
0	0010	01010
1	0010	11000
1	0100	1100000000
0	0100	1100000000
0	0001	1100

Table 3: AQIA

Random Weights has already been defined for each operation type on each database schema table e.g. as shown in Table 4

Operation	Table	Weight
SELECT	EMPLOYEE	24
INSERT	EMPLOYEE	43
UPDATE	EMPLOYEE	54
DELETE	EMPLOYEE	20

Table 4: Weights Table

Each query in an AT is assigned a random weight during AT identification phase. After this phase each transaction can also be viewed as a strict sequence of weights.

3.2 TQSAA (Transaction Query Sequence Analyzer Array)

TQSAA (Transaction Query Sequence Analyzer Array) stores the sequence of queries in ATs in the form of weights. Thus a TQSAA can look like

24	56	71	43	32	53

TQSAA

Where all numeric values are the weights of queries at serial number 1 to 6.

An algorithm has also been proposed based on query weights to keep a check over UTs and identify ATs. This algorithm is presented in next section.

3.3 QWA (Query Weight Analysis) Algorithm

In this algorithm, there is a Hashed Query Weight Counter, Hash functions & Weighted Queries. Hashed Query Weight Counter is initialized with all zero bits. Each query in an AT has already been assigned a random weight during AT identification phase. A particular number of Hash functions are applied one by one to each query weight of AT. The corresponding hashed values are obtained and the obtained hashed value'th bit in Hashed Query Weight Counter is incremented. After applying all query weights, the counter reaches a particular value. This is called as Transaction Weight.

Suppose the counter is initialized as

0	0	0	0	0

The Hash functions used are:

H1(x) = x % 5

H2(x) = (2x + 5) % 5

The query weights for the selected queries in an AT taken above for example are

24, 56, 71, 43, 32, 53

Applying 24 to H1(x)→ 4

Applying 24 to H2(x)→ 3

Hence after applying first query weight (24), the counter value becomes

0	0	0	1	1

Applying 56 to H1(x)→ 1

Applying 56 to H2(x)→ 2

The counter value after applying second query weight (56) the counter value becomes

0	1	2	1	1

Applying 71 to H1(x)→ 1

Applying 71 to H2(x)→ 2

The counter value after applying second query weight (71) the counter value becomes

0	2	3	1	1

Applying 43 to H1(x)→ 3

Applying 43 to H2(x)→ 1

The counter value after applying second query weight (43) the counter value becomes

0	3	3	2	1

Applying 32 to H1(x)→ 2

Applying 32 to H2(x)→ 4

The counter value after applying second query weight (32) the counter value becomes

0	3	4	2	2

Applying 53 to H1(x)→ 3

Applying 53 to H2(x)→ 1

The counter value after applying second query weight (53) the counter value becomes

0	4	4	3	2

The number of hash functions & the number of bits in counter are kept at optimal level to achieve best output. Thus the transaction weight for AT-1 becomes as above.

Chapter 4

QUAD PHASE VERIFIER

Quad-Phase Verifier is a vital component of the proposed model. This component is responsible for accepting the real time transaction from a trusted User, check the authorization of submitted transaction against the specified parameters and respond to it as mentioned in the Decision Tables.

An organization can design a Decision Table suitable to their policies & standards after discussions with Database owner, Security experts, Domain Analyzers, Technical experts, etc. This Decision Table will pave the way for Quad Phase Verifier usage and respond to UTs by suitable UTRs. A detailed description of Decision Tables will be covered in next section.

Quad-Phase Verifier, as the name suggests, comprises of four verification phases. These phases verify the submitted transaction by a user against the AT profile of that user and declare the transaction as UT or AT.

As mentioned in the Introduction section, our model is also equipped with the feature of "Customization in Implementation" i.e. during implementation of proposed model; an organization is not bound to use all the verification phases of Quad-Phase Verifier. The organizations can select to activate only those phases of Quad-Phase Verifier which they think are more suitable and worthy for their organizational needs. The Quad-Phase verifier will then suppress the unselected phases and continue to work on selected phases only.

The four phases of Quad-Phase Verifier are:-

1. Phase-I Operator Verifier
2. Phase-II Relations/Attributes Verifier
3. Phase-III Sequence Verifier

4. Phase-IV Weight Algorithm Verifier

Suppose a Trusted User submits a real time Transaction. Suppose the real time submitted transactions' query are

1. SELECT FIRST_NAME from EMPLOYEES;
2. INSERT into JOB_HISTORY values ('1750',sysdate,'J1');
3. UPDATE EMPLOYEES set PHONE_NUMBER='111111111';
4. SELECT EMPLOYEE_ID from JOB_HISTORY;
5. DELETE from EMPLOYEES where EMP_ID='001';
6. SELECT JOB_ID,JOB_TITLE from JOBS;

The queries from this transaction are parsed and needs to be verified to declare the transaction as AT/UT.

4.1 Phase I - Operator Verifier

Phase-I is termed as the Operator Verifier phase. This phase verifies the submitted query in terms of Operators' correctness. This can be achieved by using the AQIA's OI array. For the example specified above, the OI array value for AT is

0
0
1
1
0
0

0
1
2
0
3
0

OI for AT **OI for Submitted Transaction**

Now the OI array for the real time submitted transactions' query is also computed. Hence OI Array computed for this transaction is as shown in table 2. Thus OI array value for real time submitted transactions' query doesn't match with OI Array value of AT. Hence Phase-I will declare this submitted transaction as UT. However, if both OI arrays would have same values then it could have been declared as AT.

4.2 Phase II – Relations/Attributes Verifier

Phase-II is termed as the Relations/Attributes Verifier phase. This phase verifies the submitted query in terms of Relations' (Tables) & Attributes' correctness. This can be achieved by using the AQIA's RI & AI array. For the example specified above, the RI array & AI array values for AT are

RI
0101
0010
0010
0100
0100
0001

AI
1100000000|10000
01010
11000
1100000000
1100000000
1100

Now the RI & AI array for the real time submitted transactions' query is also computed Hence RI & AI Arrays computed for this transaction are

0100
0010
0100

0100000000
11100
0000000001

0010
0100
0001

RI

1000000000
0000000000
1100

AI

Thus RI & AI array value for real time submitted transactions' query doesn't match with RI & AI Array value of AT. Hence Phase-II will declare this submitted transaction as UT. However, if both RI & AI arrays would have same values then it could have been declared as AT.

4.3 Phase III – Sequence Verifier

Phase-III is termed as the Sequence Verifier phase. This phase verifies the submitted query in terms of Query Sequence correctness. This can be achieved by using the TQSAA array. For the example specified above, the TQSAA array value for AT is

24	56	71	43	32	53

Now the TQSAA array for the real time submitted transactions' query is also computed. Hence TQSAA Array value computed for this transaction is

12	29	25	43	31	53

Thus TQSAA array value for real time submitted transactions' query doesn't match with TQSAA Array value of AT. Hence Phase-III will declare this submitted transaction as UT. However, if both TQSAA arrays would have same values then it could have been declared as AT.

4.4 Phase IV – Weight Algorithm Verifier

Phase-IV is termed as the Weight Algorithm Verifier phase. This phase verifies the submitted query though a Query Weight based algorithm. This can be achieved by using the QWA array. For the example specified above, the QWA array values computed above for AT are

0	4	4	3	2

This is ultimately the hashed value representation of query weights. Now the Query Weight Analysis Algorithm is applied to the submitted transaction queries to declare a transaction as UT/AT.

4.4.1 Query Weight Analysis Algorithm

QUERY_WEIGHT_ANALYSIS_ALGORITHM ***(ATWeight [], SubTWeight [], Hashfunctions [])***

```
1 Load ATWeight []
2   For 0 to SubTWeight.length
3     For 0 to Hashfunctions.length
4       hashedval <-Apply Hashfunction [index] to SubTWeight [index]
5       ATWeight [hashedval] = ATWeight [hashedval]-1
6     End For;
7   End For;
8   For 0 to ATWeight.length
9     IF ATweight [index]!=0
10      Declare Transaction is UT
11      Exit
13    ENDIF
14    Declare Transaction is AT
15    End For
16    End
```

Suppose the user enter correct sequence of commands then how QUERY_WEIGHT_ANALYSIS_ALGORITHM () algorithm with two hash functions will identify it is shown below.

ATWeight Loaded

0	4	4	3	2

Use 12: 0 0 1 0 1

AT Weight

0	4	3	3	1

Use 29: 0 0 0 1 1

AT Weight

0	4	3	2	0

Use 25: 2 0 0 0 0

AT Weight

-2	4	3	2	0

Use 43: 0 1 0 1 0

AT Weight

-2	3	3	1	0

Use 31: 0 1 1 0 0

AT Weight

-2	2	2	1	0

Use 53: 0 1 0 1 0

AT Weight

-2	1	2	0	0

At the end of Algorithm, if all the values of ATWeight array become zero, then it signifies that the submitted transaction is AT. But if, the ATWeight array is non zero, then the transaction is UT.

Chapter 5

Decision Table

5.1 Background

A Decision table is basically a four quadrant structure

Conditions	Condition Alternatives
Actions	Action Entries

Each decision corresponds to a variable, relation or predicate whose possible values are listed among the condition alternatives. Each action is a procedure or operation to perform, and the entries specify whether (or in what order) the action is to be performed for the set of condition Many decision tables include in their condition alternatives the don't care symbol, a hyphen. Using don't cares can simplify decision tables, especially when a given condition has little influence on the actions to be performed alternatives the entry corresponds to. Decision Table works like an instructor in the proposed model. This Decision Table works like an Instruction Sheet to guide our model how to deal the real time submitted transaction queries.

5.2 Decision Table based Model

As discussed above, the Quad-Phase Verifier consists of four phases that can be applied together, individually or in any combination to declare a submitted transaction as AT/UT. As mentioned in the Introduction section, our model is equipped with the feature of "Customization in Implementation" i.e. during implementation of proposed model, an organization is not bound to use all the verification phases of Quad-Phase Verifier. The organizations can select any combination of those phases of Quad-Phase Verifier which they think are more suitable and worthy for their organizational needs. The Quad-Phase verifier will then suppress the unselected phases and continue to work on selected phases only.

Our model proposes use of two decision tables-

1. **Decision Table 1** demonstrated as Fig2 indicates which array values to be computed for real time submitted transaction queries & compared with AT Arrays values based on the selected combination of phases of Quad-Phase verifier.
2. **Decision Table 2** demonstrated as Fig3 indicates what response to be generated (UTR or some other) based on the result of comparison of various array values of real time submitted transaction with AT arrays values.

The use of Decision tables is the basis of high performance of this model. In the absence of these decision tables, all the array values for real time submitted transaction queries would have been computed, and then according to the selected phases the required array values would be compared with AT array values and a customized message for different types of results to be generated as response by the system. All these activities would have been completed dynamically (at run time).

However, with the use of decision tables, first we have static guidance available that for a selected combination of phases, which array values to be computed and compared. Thus no need to compute all array values. Second, we have static guidance available that what response action to be taken for what phase and what comparison result. The use of static decision tables instead of Dynamic computation proves greatly time and cost effective in nature.

Decision Tables 1 & 2 are shown on next page. Decision Table 1 has four conditions about selection of Verifier Phases. There are five actions specified in the table. Each combination of condition values (True or False) result in zero or more actions from

the action list e.g. If condition values are FFFF i.e. no phase is selected then no action will be performed. If condition values are FTTT i.e. Phase I, II & III are selected and Phase IV not selected then action 1, 2, 3 & 4 will be performed and so on.

The output of Decision Table 1 is fed as input to the Decision Table 2. The output of Decision Table 1 will be a five bit value which indicates the Selected actions of Decision Table 1 e.g. If condition values for Decision Table 1 are FTTT i.e. Phase I, II & III are selected and Phase IV not selected then action 1, 2, 3 & 4 will be performed and the result obtained from Actions values will be like a bit stream as shown

* * * * -

This bit stream has first four bits as * and fifth bit as '-'. The '-' indicates "don't care" and the '*' indicates that this bit needs to be considered. This bit stream is the input to Decision Table 2.

Decision Table 2 has five conditions about selection of bits in the bit stream obtained from Decision Table 1. There are twelve actions specified in the table. Each combination of condition values (True or False) result in one or more actions from the action list e.g. If the bit stream obtained from decision table 1 is

---**

		Rules															
Conditions	Is Phase IV selected?	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T
	Is Phase III selected?	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T
	Is Phase II selected?	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T
	Is Phase I selected?	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T
Actions	Compute OI Array & Match with AT Store entry's OI Array		*		*		*		*		*		*		*		*
	Compute RI Array & Match with AT Store entry's RI Array			*	*			*	*			*	*			*	*
	Compute AI Array & Match with AT Store entry's AI Array			*	*			*	*			*	*			*	*
	Compute TQSAA Array & Match with AT Store entry's TQSAA Array					*	*	*	*					*	*	*	*
	Apply QWA Algorithm and Match final output array with zero.									*	*	*	*	*	*	*	*

Fig 2. Decision Table 1

		Rules									
		*----		-**--					....		*****
Conditions	Is Bit 1 selected?	T	F						....		T
	Is Bit 2 selected?			F	F	T	T		....		T
	Is Bit 3 selected?			F	T	F	T		....		T
	Is Bit 4 selected?								....		T
	Is Bit 5 selected?								....		T
Actions	Declare Transaction as AT	*					*		....		*
	Declare Transaction as UT		*	*	*	*			....		
	Declare OI phase as successful	*							....		*
	Declare OI phase as Failure		*						....		
	Declare AI phase as successful					*	*		....		*
	Declare AI phase as Failure			*	*				....		
	Declare RI phase as successful				*		*		....		*
	Declare RI phase as Failure			*		*			....		
	Declare TQSAA phase as successful								....		*
	Declare TQSAA phase as Failure								....		
	Declare QWA phase as successful								....		*
	Declare QWA phase as Failure								....		

Fig 3. Decision Table 2

It means bit 2 and 3 are * (to be considered) and bit 1, 4, 5 are – (don't care). Referring the Decision Table 2 for the bit stream mentioned, the values of * bit values are to be considered. Bit 2 and Bit 3 are * and there are four possible combinations of bit2 & bit 3 i.e. FF, FT, TF, TT. These are shown in Condition values of Decision Table 2 for bit stream -**--. Depending on the value of these bits, the particular actions from the Action list can be selected e.g. If the bit values are TT for bit stream -**--, then it means that RI & AI arrays have been matched perfectly for the Real time submitted transaction with that of AT, hence the action taken selected from the Action List of Decision Table 2 are

- Declare Transaction as AT
- Declare AI Phase as successful.
- Declare RI Phase as successful

Thus at the end of Decision Table 2, it can be stated whether Real time submitted transaction is AT/UT based on success of which phases verification.

Chapter 6

Oracle-J2EE Implementation

6.1 Enabling Database Audit setup

We have implemented the proposed model using Oracle DBMS, integrated with a J2EE based application. During the implementation, our audience has been given the choice of selecting the Quad-Phase Verifier phases.

Auditing is a default feature of the Oracle RDBMS. Auditing is disabled by default, but can be enabled by setting the AUDIT_TRAIL static parameter.

The system table which keeps the Audit records is “DBA_AUDIT_TRAIL”.

To enable auditing and direct audit records to the database audit trail, the commands executed are:

```
SQL> ALTER SYSTEM SET audit_trail=db SCOPE=SPFILE;


System altered.


SQL> SHUTDOWN
Database closed.
Database dismounted.
ORACLE instance shut down.
SQL> STARTUP
ORACLE instance started.


Total System Global Area 289406976 bytes
```

```
Fixed Size                  1248600 bytes
Variable Size              71303848 bytes
Database Buffers          213909504 bytes
Redo Buffers                2945024 bytes
Database mounted.
Database opened.
SQL>
```

Suppose there are three users User1, User2, User3 created in Oracle. Now we need to Audit all DMLs (SELECT, INSERT, UPDATE, DELETE) executed by these users. To enable this, the commands executed are:

```
CONNECT sys/password AS SYSDBA


SQL> AUDIT SELECT TABLE, UPDATE TABLE, INSERT TABLE, DELETE TABLE BY
User1 BY ACCESS;
SQL> AUDIT SELECT TABLE, UPDATE TABLE, INSERT TABLE, DELETE TABLE BY
User2 BY ACCESS;
SQL> AUDIT SELECT TABLE, UPDATE TABLE, INSERT TABLE, DELETE TABLE BY
User3 BY ACCESS;
```

Now all the DML activities performed by Users User1, User2 and User3 will be audited and recorded in table DBA_AUDIT_TRAIL.

6.2 TQLC Implementation

TQLC (Transaction Query Log Crawler) utility is implemented using SQL queries executed through a JDBC connection from the J2EE Application. These queries perform a SELECT over DBA_AUDIT_TRAIL tables for the specified user and analyze them over different transaction ids.

```
CONNECT sys/password AS SYSDBA

SQL> SELECT OPERATIONTYPE, SQLTEXT, OBJECT, TRANSACTIONID FROM
DBA_AUDIT_TRAIL WHERE USERNAME='USER1' ORDER BY TRANSACTIONID;
```

```
SQL> SELECT OPERATIONTYPE, SQLTEXT, OBJECT, TRANSACTIONID FROM
DBA_AUDIT_TRAIL WHERE USERNAME=’USER2’ ORDER BY TRANSACTIONID;

SQL> SELECT OPERATIONTYPE, SQLTEXT, OBJECT, TRANSACTIONID FROM
DBA_AUDIT_TRAIL WHERE USERNAME=’USER3’ ORDER BY TRANSACTIONID;
```

6.3 AQIA Implementation

AT store entries are created by the output analysis of these queries. For each AT store entry, the TQIA [OI, RI, AI] & TQSAA needs to be evaluated. OI array is computed by using OPERATIONTYPE attribute of AT store entry. For RI & AI, the total number of tables in the schema and total number of attributes in that table needs to be known. Following query is executed for this:

```
CONNECT sys/password AS SYSDBA

SQL> SELECT table_name FROM dba_tables where owner=’USER1’;

SQL> SELECT table_name FROM dba_tables where owner=’USER2’;

SQL> SELECT table_name FROM dba_tables where owner=’USER3’;

For each table_name obtained, the query executed is:

SQL> SELECT count (column_name) FROM USER_TAB_COLUMNS
whereTABLE_NAME=?
```

A database master table is created with records inserted for each operation type for each table in schema assigned with a random weight e.g.

SELECT	EMPLOYEE	35
INSERT	JOBS	54

Each AT store entry refers this table to create TQSAA array and QWA weights Array. The initial setup activities are completed now.

For evaluation of the proposed model and its implementation, a user specified number of random transactions is generated automatically through different combinations of Operators & Tables for different users. The user selects which phases of Quad-Phase verifier to be used. The decision table1 is referred accordingly and corresponding Arrays values are computed for each random

generated transaction and matched with array values of AT computed during setup. The output of decision table 1 is referred to decision table 2 & final action is performed.

Decision tables are referred twice for each transaction. For optimal performance, Decision tables are made as Static Java Classes. These are loaded in memory once on start up and will be referred directly from there for every instance.

Chapter 7

Experimental Evaluation

The efficiency of our model can be characterized by following measures:

- **Decision Table Reference Time** (Time taken to refer decision table 1 + decision table 2 and generating final response) depicted in Fig 4.
- **False Negatives** (number of UTs identified as AT) depicted in Fig 5, 6, 7, 8
- **Coverage** (number of UTs detected * 100 / number of UTs submitted) depicted in Fig 9

For experimental evaluation, 1000 random dummy transactions are generated automatically and above mentioned parameters are calculated and a graphical comparison and evaluation is done.

7.1 Decision Table Reference Time

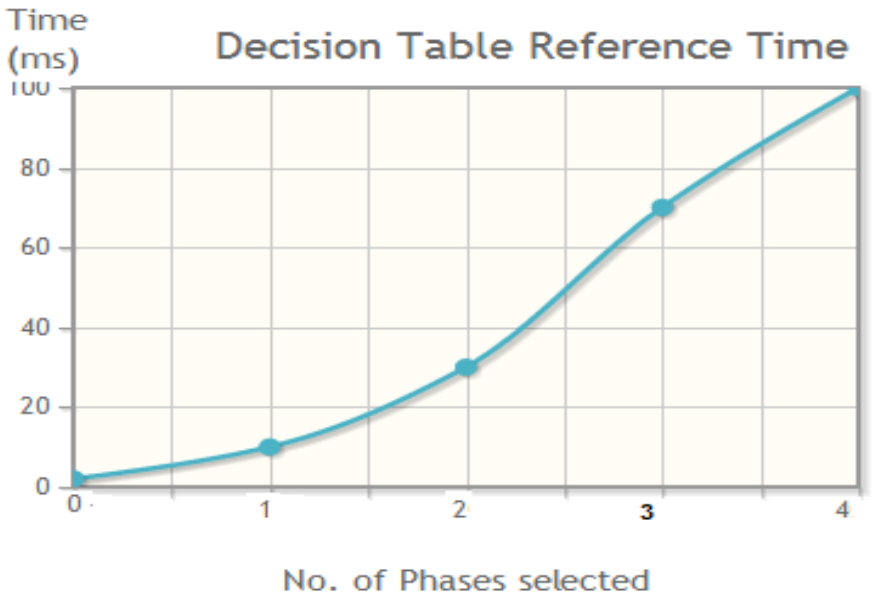


Fig 4 Decision Table Reference Time Vs No. of Phases

7.2 False Negatives

7.2.1 False Negatives % Vs Single Phases selection

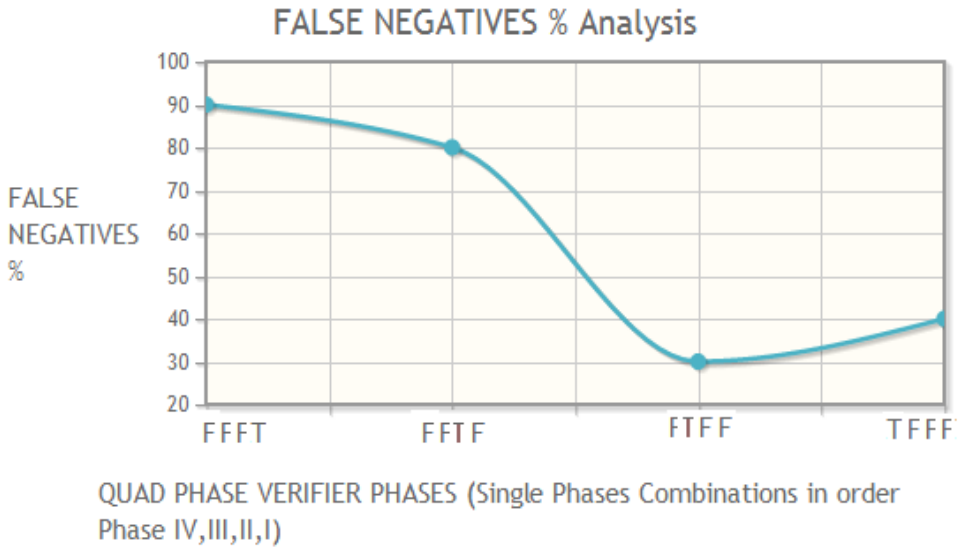


Fig 5 False Negatives % Vs Single Phases selection

7.2.2 False Negatives % Vs Double Phases selection

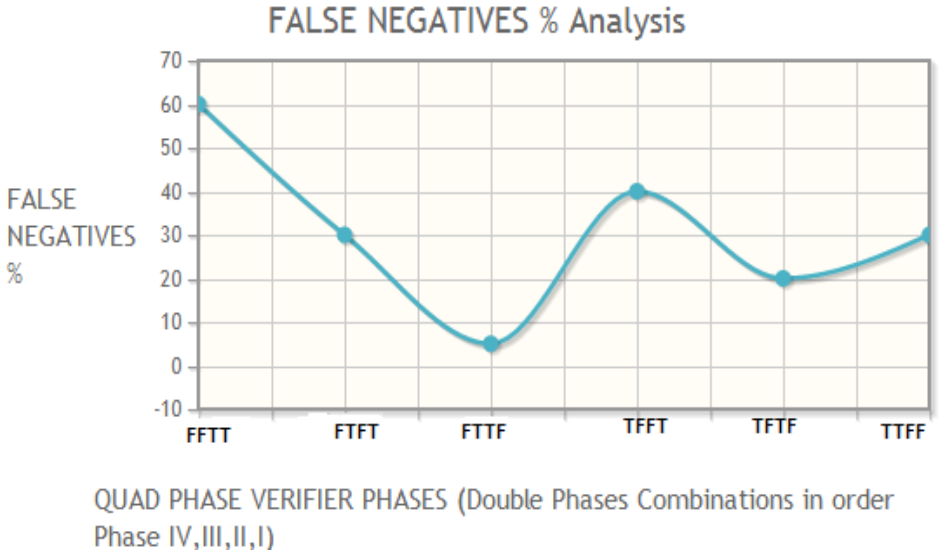


Fig 6 False Negatives % Vs Double Phases selection

7.2.3 False Negatives % Vs Triple Phases selection

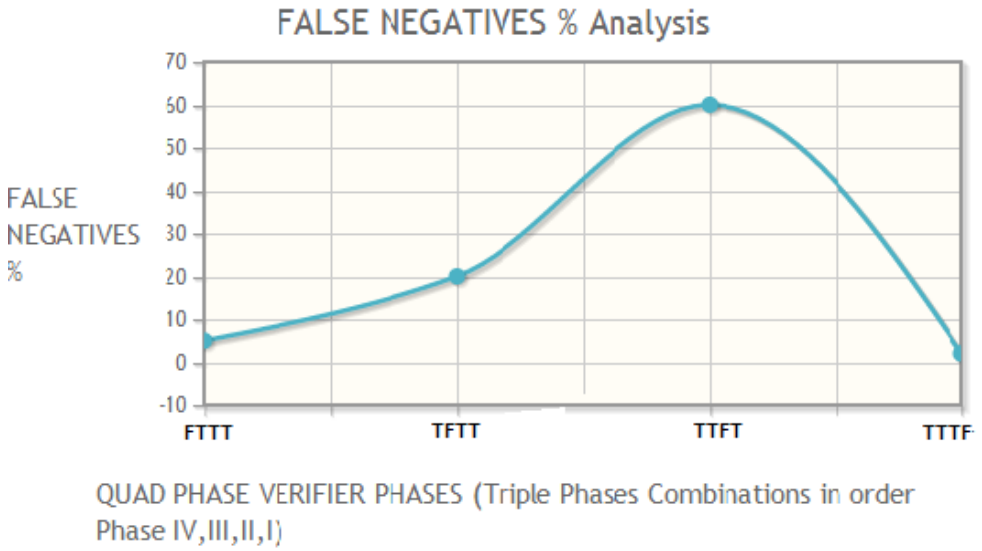


Fig 7 False Negatives % Vs Triple Phases selection

7.2.4 False Negatives % Vs Quad Phases selection

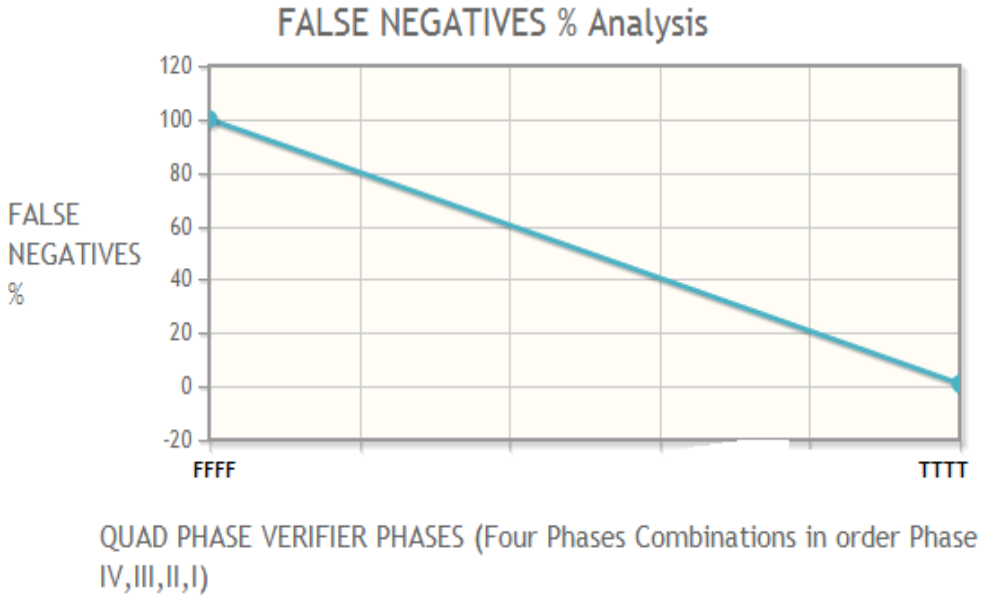


Fig 8 False Negatives % Vs Quad Phases selection

7.3 Coverage

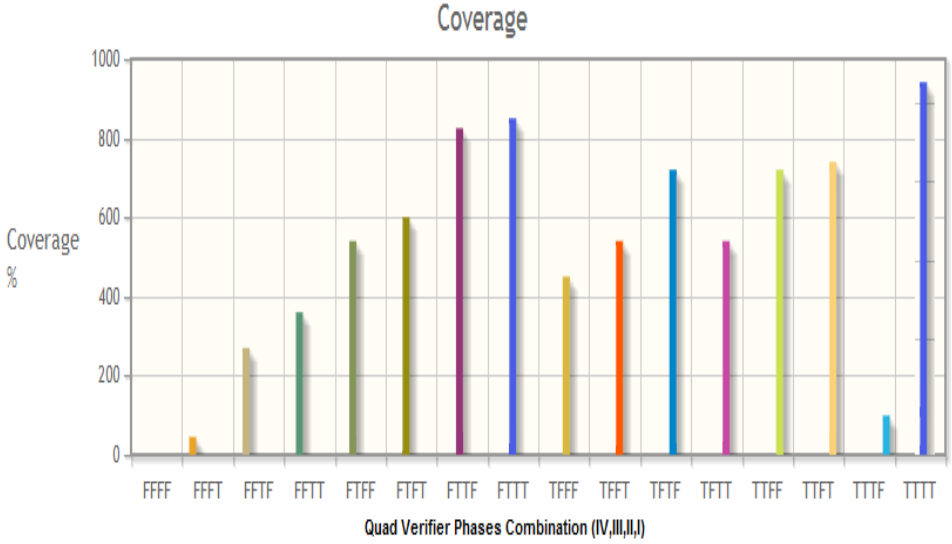


Fig 9 Coverage Vs Quad Phases phase selection

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, we have demonstrated the design and implementation of a Decision Table based model for automatic prevention of Unauthorized Database Transactions by trusted Database users. The model is implemented using J2EE & Oracle RDBMS. We have developed an application interface for the audience to interact and evaluate our system. Based
on the experimental evaluation done in the previous section, it has been concluded that the Decision Table Reference Time is directly proportional to the number of phases. The results show that 99% of the UTs can be detected and responded if best combination of Quad Phase Verifier Phases is selected. Also the False Negatives (no. of UTs identified as ATs) can be kept close to 0.2% if all the four phases of Verifier are used. False Negatives can be kept between 3 to 8% by selected combinations of 2 to 3 phases also. Thus, organizations implementing the solution can implement with those combinations also thereby reducing the Decision Table Reference Time & computation involved.

8.2 Scope for future work

As future work, extension of this model to handle DDL statements is planned. Another complementary area for future research is to enable this model for handling multiple joins, sub-queries, decode etc enable queries by assigning suitable weights for every component.

References

[1] Marco Vieira and Henrique Madeira, "Detection of Malicious Transactions in DBMS", IEEE Proceedings- 11th Pacific Rim International Symposium on Dependable Computing, Dec 12-14, 2005.

[2] S. Y. Lee, W. L. Low, and P. Y. Wong. Learning fingerprints for a database intrusion detection system. In Proceedings of the 7th European Symposium on Research in Computer Security, ESORICS '02, pages 264–280, London, UK, UK, 2002. Springer-Verlag.

[3] S. Mathew, M. Petropoulos, H. Q. Ngo, and S. Upadhyaya. A data-centric approach to insider attack detection in database systems. In Proceedings of the 13th international conference on Recent advances in intrusion detection, RAID'10, pages 382–401, Berlin, Heidelberg, 2010. Springer-Verlag.

[4] E. Bertino. Data Protection from Insider Threats. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012

[5] Navin Kumar Tyagi and A.K. Solanki, "Prediction of Users Behavior through Correlation Rules", (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 2, No. 9, 2011.

[6] L.K.J. Grace, V. Maheshwari, D. Nagamalai "Analysis of web logs and web user in web mining" International journal of Network Security & its Applications(IJNSA),Vol.3,No1.January(2011).

[7] Mike Dean, "All About Oracle Auditing – A White Paper", February 2013

[8] Ashish Kamra, Evimaria Terzi, Elisa Bertino, "Detecting anomalous access patterns in relational databases" The VLDB Journal Springer-Verlag 2007.

Publications from the Work

A research paper titled as "Decision Table based Model & its implementation for Automatic prevention of Unauthorized Database Transactions by Trusted Database Users**"** has been accepted and published in IJESIT (International Journal of Engineering Science and Innovative Technology) ISSN: 2319-5967 Journal Vol.3 Issue 3 May 2014 with paper-id IJESIT1909201403_1773.

APPENDIX-I Source Code

1. QueryCrawler.java

```
package files;
import java.sql.ResultSet;
import java.util.ArrayList;
 public class QueryCrawler {

public ArrayList qaia(ArrayList arr,String username)
{
//ArrayList qaia=new ArrayList();
ArrayList retarr=new ArrayList();
DBConnection dbcon=new DBConnection();
try{
String query="";
dbcon.openConnection();
for(int j=0; j<6;j++)
{
ArrayList qaia=new ArrayList();
query="select upper(column_name) from all_tab_columns where
owner='"+username+"' and
table_name='"+((CommonBean)arr.get(j)).getField3().toString()+"'"
+ " order by column_id";
System.out.println(query);
ResultSet rs1=dbcon.select(query);
int i=0;
while(rs1.next())
{
i++;
qaia.add(rs1.getString(1)!=null?rs1.getString(1):"");
```

```
}
int[] result=new int[i];
String querytxt=((CommonBean)arr.get(j)).getField2().toString().toUpperCase();
String attributetxt="";
String[] attributearray=null;
if(querytxt.startsWith("SELECT"))
{
attributetxt=(querytxt.substring(6,querytxt.indexOf("FROM"))).trim();
attributearray=attributetxt.split(",");
}
else if(querytxt.startsWith("INSERT"))
{
attributetxt=(querytxt.substring(querytxt.indexOf("(")+1,querytxt.indexOf(")"))).trim(
);
attributearray=attributetxt.split(",");
}
else if(querytxt.startsWith("UPDATE"))
{
attributetxt=(querytxt.substring(querytxt.indexOf("SET")+4,querytxt.indexOf("WHE
RE"))).trim();
attributearray=attributetxt.split(",");
}
System.out.println("attribute array length is :"+attributearray.length);
for(int l=0;l<attributearray.length;l++)
{
System.out.println("------"+attributearray[l]);
for(int k=0;k<qaia.size();k++)
{
result[k]+=(attributearray[l]).equals(qaia.get(k).toString())?1:0;
//System.out.println("k loop is :"+k);
}
}
retarr.add(result);
```

```
}
//          System.out.println("size of retarr is :"+retarr.size());
}
catch(Exception e)
{
e.printStackTrace();
}
finally
{
dbcon.closeConnection();
}
return retarr;
}
```

public ArrayList qria(ArrayList arr,String username)

```
{
ArrayList qria=new ArrayList();
ArrayList retarr=new ArrayList();
DBConnection dbcon=new DBConnection();
try{
String query="";
dbcon.openConnection();
query="select table_name from all_tables  where owner='"+username+"' order by
rownum";
System.out.println(query);
ResultSet rs1=dbcon.select(query);
int i=0;
while(rs1.next())
{
i++;
qria.add(rs1.getString(1)!=null?rs1.getString(1):"");
}
```

```
for(int j=0; j<6;j++)
{
int[] result=new int[i];
for(int k=0;k<qria.size();k++)
{
result[k]+=(((CommonBean)arr.get(j)).getField3().toString()).equals(qria.get(k).toStri
ng())?1:0;
}
retarr.add(result);
}
}
catch(Exception e)
{
e.printStackTrace();
}
finally
{
dbcon.closeConnection();
}
return retarr;
}
```

public ArrayList qoia(ArrayList arr)

```
{
ArrayList qoia=new ArrayList();
DBConnection dbcon=new DBConnection();
try{
String query="";
dbcon.openConnection();
for(int i=0; i<6;i++)
{
```

```
query="SELECT
DECODE('"+((CommonBean)arr.get(i)).getField1()+"','SELECT','1','INSERT','2','UP
DATE','3','DELETE','4','0') from dual";
System.out.println(query);
ResultSet rs1=dbcon.select(query);
if(rs1.next())
{
qoia.add(rs1.getString(1)!=null?rs1.getString(1):"");
}
}
//transactionlist.add(queries);
}
catch(Exception e)
{
e.printStackTrace();
}
finally
{
dbcon.closeConnection();
}
return qoia;
}
```

public ArrayList startCrawler(String username)

```
{
String msg="";
DBConnection dbcon=new DBConnection();
ArrayList transactionlist=new ArrayList();
ArrayList queries=new ArrayList();
try{
String query="";
if("USER1".equals(username))
{
```

```
DBConnectionUser1 dbconuser1=new DBConnectionUser1();
dbconuser1.openConnection();
query="SELECT JOB_ID,JOB_TITLE FROM JOBS";
dbconuser1.select(query);
dbconuser1.commit();
query=" SELECT
EMPLOYEE_ID,FIRST_NAME,LAST_NAME,EMAIL,PHONE_NUMBER,HIRE_
DATE,JOB_ID FROM EMPLOYEES";
dbconuser1.select(query);
dbconuser1.commit();
query=" INSERT INTO
EMPLOYEES(EMPLOYEE_ID,FIRST_NAME,LAST_NAME,EMAIL,PHONE_N
UMBER,HIRE_DATE,JOB_ID) VALUES
('1750','Rohit','Jain','xyz@gmail.com','111111',sysdate,'J1')";
dbconuser1.insert(query);
dbconuser1.commit();
query="INSERT INTO JOB_HISTORY(EMPLOYEE_ID,START_DATE,JOB_ID)
VALUES ('1750',sysdate,'J1')";
dbconuser1.insert(query);
dbconuser1.commit();
query="SELECT EMPLOYEE_ID,START_DATE,JOB_ID FROM
JOB_HISTORY";
dbconuser1.select(query);
dbconuser1.commit();
query=" SELECT
EMPLOYEE_ID,FIRST_NAME,LAST_NAME,EMAIL,PHONE_NUMBER,HIRE_
DATE,JOB_ID FROM EMPLOYEES";
dbconuser1.select(query);
dbconuser1.commit();
dbconuser1.closeConnection();
}
else if("USER2".equals(username))
{
```

```
DBConnectionUser2 dbconuser2=new DBConnectionUser2();
dbconuser2.openConnection();
query="SELECT
EMPLOYEE_ID,FIRST_NAME,LAST_NAME,EMAIL,PHONE_NUMBER,HIRE_
DATE,JOB_ID FROM EMPLOYEES";
dbconuser2.select(query);
query=" SELECT
DEPARTMENT_ID,DEPARTMENT_NAME,MANAGER_ID,LOCATION_ID
FROM  DEPARTMENTS";
dbconuser2.select(query);
query=" UPDATE EMPLOYEES SET
SALARY='50000',MANAGER_ID='46',DEPARTMENT_ID='D1' WHERE
EMPLOYEE_ID='1750'";
dbconuser2.update(query);
query="SELECT
EMPLOYEE_ID,FIRST_NAME,LAST_NAME,EMAIL,PHONE_NUMBER,HIRE_
DATE,JOB_ID FROM EMPLOYEES";
dbconuser2.select(query);
query="UPDATE JOB_HISTORY SET DEPARTMENT_ID='D1' WHERE
EMPLOYEE_ID='1750'";
dbconuser2.update(query);
query=" SELECT EMPLOYEE_ID,START_DATE,JOB_ID,DEPARTMENT_ID
FROM JOB_HISTORY";
dbconuser2.select(query);
dbconuser2.commit();
dbconuser2.closeConnection();
}
else
{
DBConnectionUser3 dbconuser3=new DBConnectionUser3();
dbconuser3.openConnection();
```

```
query="SELECT
DEPARTMENT_ID,DEPARTMENT_NAME,MANAGER_ID,LOCATION_ID
FROM  DEPARTMENTS";
dbconuser3.select(query);
query=" SELECT
EMPLOYEE_ID,FIRST_NAME,LAST_NAME,EMAIL,PHONE_NUMBER,HIRE_
DATE,JOB_ID,SALARY,MANAGER_ID, department_id FROM EMPLOYEES";
dbconuser3.select(query);
query="SELECT JOB_ID,JOB_TITLE, min_salary, max_salary FROM JOBS";
dbconuser3.select(query);
query="UPDate jobs set min_salary='20000', max_salary='50000' where job_id='J1'";
dbconuser3.update(query);
query="SELECT JOB_ID,JOB_TITLE, min_salary, max_salary FROM JOBS";
dbconuser3.select(query);
query=" SELECT EMPLOYEE_ID,START_DATE, end_date, job_id, department_id
FROM JOB_HISTORY";
dbconuser3.select(query);
dbconuser3.commit();
dbconuser3.closeConnection();
}
dbcon.openConnection();
CommonBean bn=null;
/*query="SELECT distinct transactionid,session_id FROM
DBA_COMMON_AUDIT_TRAIL where  db_user='"+username+"'  ORDER BY
session_id desc";
System.out.println(query);
ResultSet rs=dbcon.select(query);
int i=0;
ArrayList arr=new ArrayList();
while(rs.next() && i<5)
{
arr.add(rs.getString(1)!=null?rs.getString(1):"");
i++;
```

```
}
for(int j=0;j<arr.size();j++){*/
query="SELECT  max(statement_type),max(sql_text), max(object_name) FROM
DBA_COMMON_AUDIT_TRAIL where object_schema='"+username+"' and"
+ " statement_type in ('SELECT','UPDATE','INSERT','DELETE')  group by
extended_timestamp,SCN ORDER BY extended_timestamp desc,SCN DESC";
System.out.println(query);
ResultSet rs1=dbcon.select(query);
int i=0;
while(rs1.next() && i<30)
{
bn=new CommonBean();
bn.setField1(rs1.getString(1)!=null?rs1.getString(1):"");
bn.setField2(rs1.getString(2)!=null?rs1.getString(2):"");
bn.setField3(rs1.getString(3)!=null?rs1.getString(3):"");
queries.add(bn);
i++;
}
//transactionlist.add(queries);
}
catch(Exception e)
{
e.printStackTrace();
}
finally
{
dbcon.closeConnection();
}
return queries;
}
}
```

2. BusinessLogic.java

```
package files;


import java.sql.ResultSet;
import java.util.ArrayList;
public class BusinessLogic {
public ArrayList graphcoveragehashfunction(int hashfunctionno)
{
String msg="";
String tid="1";
ArrayList arr=new ArrayList();
DBConnection db=null;
try{
db=new DBConnection();
db.openConnection();
String selquery="";
for(int i=4;i<10;i++)
{
selquery="select  floor(  ((select count(*) from mstdummytransaction)-(select
count(*) from mstfalsenegatives where nvl(falsenegatives,'Y')='Y' "
+ "and counterbits="+i+" and hashfunctionno="+hashfunctionno+"))/(select count(*)
from mstdummytransaction) *100) from dual";
System.out.println(selquery);
ResultSet rs=db.select(selquery);
String query="";
if(rs.next())
{
arr.add(rs.getString(1)!=null?rs.getString(1):"0");
}
}
msg="Success";
```

```
}
catch(Exception e)
{
e.printStackTrace();
}
finally
{
db.closeConnection();
}
return arr;
}
```

public ArrayList grapharrhashfunction(int hashfunctionno)

```
{
String msg="";
String tid="1";
ArrayList arr=new ArrayList();
DBConnection db=null;
try{
db=new DBConnection();
db.openConnection();
String selquery="";
for(int i=4;i<10;i++)
{
selquery="  select   floor( (select count(*) from mstfalsenegatives where
nvl(falsenegatives,'Y')='Y' and counterbits="+i+" and
hashfunctionno="+hashfunctionno+")/"
+ "   (select count(*) from mstdummytransaction) *100 ) from dual";
System.out.println(selquery);
ResultSet rs=db.select(selquery);
String query="";
if(rs.next())
{
```

```
arr.add(rs.getString(1)!=null?rs.getString(1):"0");
}
}
msg="Success";
}
catch(Exception e)
{
e.printStackTrace();
}
finally
{
db.closeConnection();
}
return arr;
}
```

public ArrayList grapharrbit(int bits)

```
{
String msg="";
String tid="1";
ArrayList arr=new ArrayList();
DBConnection db=null;
try{
db=new DBConnection();
db.openConnection();
String selquery="";
for(int i=1;i<10;i++)
{
selquery="  select   floor( (select count(*) from mstfalsenegatives where
nvl(falsenegatives,'Y')='Y' and counterbits="+bits+" and hashfunctionno="+i+")/"
+ "   (select count(*) from mstdummytransaction) *100 ) from dual";
System.out.println(selquery);
ResultSet rs=db.select(selquery);
```

```
String query="";
if(rs.next())
{
arr.add(rs.getString(1)!=null?rs.getString(1):"0");
}
}
msg="Success";
}
catch(Exception e)
{
e.printStackTrace();
}
finally
{
db.closeConnection();
}
return arr;
}
```

public String updatefalsenegatives()

```
{
String msg="";
String tid="1";
ArrayList arr=new ArrayList();
DBConnection db=null;
try{
db=new DBConnection();
db.openConnection();
String selquery=" select b.transactionid from  msttransaction a, (select
transactionid,QUERYID,QUERYID1,QUERYID2,QUERYID3,QUERYID4,QUERY
ID5"
+ " from mstdummytransaction where transactionid in   (select distinct
DUMMYTRANSACTIONID from mstfalsenegatives)) b"
```

```
+ "   where a.queryid=b.queryid and a.queryid1=b.queryid1 and
a.queryid2=b.queryid2 and a.queryid3=b.queryid3"
+ "  and a.queryid4=b.queryid4 and a.queryid5=b.queryid5";
System.out.println(selquery);
ResultSet rs=db.select(selquery);
String query="";
while(rs.next())
{
arr.add(rs.getString(1)!=null?rs.getString(1):"");
}
for(int i=0;i<arr.size();i++)
{
query="update MSTFALSENEGATIVES set FALSENEGATIVES='N' "
+ " where DUMMYtransactionid="'+arr.get(i).toString()+"'";
System.out.println(query);
db.update(query);
}
db.commit();
msg="Success";
}
catch(Exception e)
{
e.printStackTrace();
}
finally
{
db.closeConnection();
}
return msg;
}
```

public ArrayList getOriginalTransactionAllhashfunctionscounterArray()

```
{
```

```
ArrayList resultarr=new ArrayList();
DBConnection db=null;
ArrayList arr=new ArrayList();
CommonBean bn=null;
try{
db=new DBConnection();
db.openConnection();
String selquery="select a.queryid,(select operatorid||' '||tablename from mstquery
where queryid=a.queryid )," 
+ "(select weight from mstquery where queryid=a.queryid ),  "
+ "a.queryid1,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid1 )," 
+ "(select weight from mstquery where queryid=a.queryid1 )," 
+ "a.queryid2,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid2 )," 
+ "(select weight from mstquery where queryid=a.queryid2 )," 
+ "a.queryid3,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid3 )," 
+ "(select weight from mstquery where queryid=a.queryid3 )," 
+ "a.queryid4,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid4 )," 
+ "(select weight from mstquery where queryid=a.queryid4 )," 
+ "a.queryid5,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid5 )," 
+ "(select weight from mstquery where queryid=a.queryid5 ) from msttransaction a ";
System.out.println(selquery);
ResultSet rs=db.select(selquery);
if(rs.next())
{
bn=new CommonBean();
bn.setField1(rs.getString(1)!=null?rs.getString(1):"");
bn.setField2(rs.getString(2)!=null?rs.getString(2):"");
bn.setField3(rs.getString(3)!=null?rs.getString(3):"");
```

```
bn.setField4(rs.getString(4)!=null?rs.getString(4):"");
bn.setField5(rs.getString(5)!=null?rs.getString(5):"");
bn.setField6(rs.getString(6)!=null?rs.getString(6):"");
bn.setField7(rs.getString(7)!=null?rs.getString(7):"");
bn.setField8(rs.getString(8)!=null?rs.getString(8):"");
bn.setField9(rs.getString(9)!=null?rs.getString(9):"");
bn.setField10(rs.getString(10)!=null?rs.getString(10):"");
bn.setField11(rs.getString(11)!=null?rs.getString(11):"");
bn.setField12(rs.getString(12)!=null?rs.getString(12):"");
bn.setField13(rs.getString(13)!=null?rs.getString(13):"");
bn.setField14(rs.getString(14)!=null?rs.getString(14):"");
bn.setField15(rs.getString(15)!=null?rs.getString(15):"");
bn.setField16(rs.getString(16)!=null?rs.getString(16):"");
bn.setField17(rs.getString(17)!=null?rs.getString(17):"");
bn.setField18(rs.getString(18)!=null?rs.getString(18):"");
arr.add(bn);
}
//for counter bit 4
ArrayList hshfunccounter4=new ArrayList();
selquery="select hashfunction from msthashfunctions where countervalue=4 and
rownum<=9 ";
System.out.println(selquery);
sultSet rshashfunction4=db.select(selquery);
while(rshashfunction4.next())
{
hshfunccounter4.add(rshashfunction4.getString(1)!=null?rshashfunction4.getString(1)
:"");
}
//for counter bit 5
ArrayList hshfunccounter5=new ArrayList();
selquery="select hashfunction from msthashfunctions where countervalue=5 and
rownum<=9 ";
System.out.println(selquery);
```

```
ResultSet rshashfunction5=db.select(selquery);
while(rshashfunction5.next())
{
hshfunccounter5.add(rshashfunction5.getString(1)!=null?rshashfunction5.getString(1)
:"");
}
//for counter bit 6
ArrayList hshfunccounter6=new ArrayList();
selquery="select hashfunction from msthashfunctions where countervalue=6 and
rownum<=9 ";
System.out.println(selquery);
ResultSet rshashfunction6=db.select(selquery);
while(rshashfunction6.next())
{
hshfunccounter6.add(rshashfunction6.getString(1)!=null?rshashfunction6.getString(1)
:"");
}
//for counter bit 7
ArrayList hshfunccounter7=new ArrayList();
selquery="select hashfunction from msthashfunctions where countervalue=7 and
rownum<=9 ";
System.out.println(selquery);
ResultSet rshashfunction7=db.select(selquery);
while(rshashfunction7.next())
{
hshfunccounter7.add(rshashfunction7.getString(1)!=null?rshashfunction7.getString(1)
:"");
}
//for counter bit 8
ArrayList hshfunccounter8=new ArrayList();
selquery="select hashfunction from msthashfunctions where countervalue=8 and
rownum<=9 ";
System.out.println(selquery);
```

```
ResultSet rshashfunction8=db.select(selquery);
while(rshashfunction8.next())
{
hshfunccounter8.add(rshashfunction8.getString(1)!=null?rshashfunction8.getString(1)
:"");
}
//for counter bit 9
ArrayList hshfunccounter9=new ArrayList();
selquery="select hashfunction from msthashfunctions where countervalue=9 and
rownum<=9 ";
System.out.println(selquery);
ResultSet rshashfunction9=db.select(selquery);
while(rshashfunction9.next())
{
hshfunccounter9.add(rshashfunction9.getString(1)!=null?rshashfunction9.getString(1)
:"");
}
// For finding weighted hashed actual array of 4 bit counter for hash function number
1  to 9
ArrayList counterpositionafterhashingfor4bit=new ArrayList();
ArrayList counterpositionafterhashingfor5bit=new ArrayList();
ArrayList counterpositionafterhashingfor6bit=new ArrayList();
ArrayList counterpositionafterhashingfor7bit=new ArrayList();
ArrayList counterpositionafterhashingfor8bit=new ArrayList();
ArrayList counterpositionafterhashingfor9bit=new ArrayList();
String result="";
for(int i=0;i<arr.size();i=i+1) {
//dummycurrhashedvals="";
bn=(CommonBean)arr.get(i);
//for 4 bit counter
int array[]=new int[4];
for(int j=0;j<hshfunccounter4.size();j=j+1) {
String hashfunction=hshfunccounter4.get(j).toString().replace("x",bn.getField3());
```

```
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue=db.select(selquery);
if(rshashvalue.next())
{
result=rshashvalue.getString(1)!=null?rshashvalue.getString(1):"";
}
array[Integer.parseInt(result)]=array[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter4.get(j).toString().replace("x",bn.getField6());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue1=db.select(selquery);
if(rshashvalue1.next())
{
result=rshashvalue1.getString(1)!=null?rshashvalue1.getString(1):"";
}
array[Integer.parseInt(result)]=array[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter4.get(j).toString().replace("x",bn.getField9());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue2=db.select(selquery);
if(rshashvalue2.next())
{
result=rshashvalue2.getString(1)!=null?rshashvalue2.getString(1):"";
}
array[Integer.parseInt(result)]=array[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter4.get(j).toString().replace("x",bn.getField12());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue3=db.select(selquery);
if(rshashvalue3.next())
{
result=rshashvalue3.getString(1)!=null?rshashvalue3.getString(1):"";
```

```
}
array[Integer.parseInt(result)]=array[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter4.get(j).toString().replace("x",bn.getField15());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue4=db.select(selquery);
if(rshashvalue4.next())
{
result=rshashvalue4.getString(1)!=null?rshashvalue4.getString(1):"";
}
array[Integer.parseInt(result)]=array[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter4.get(j).toString().replace("x",bn.getField18());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue5=db.select(selquery);
if(rshashvalue5.next())
{
result=rshashvalue5.getString(1)!=null?rshashvalue5.getString(1):"";
}
array[Integer.parseInt(result)]=array[Integer.parseInt(result)]+1;
int ar[]=new int[4];
for(int m=0;m<4;m++)
{
ar[m]=array[m];
}
counterpositionafterhashingfor4bit.add(ar);
}
// For 5 bit counter
int array1[]=new int[5];
for(int j=0;j<hshfunccounter5.size();j=j+1) {
String hashfunction=hshfunccounter5.get(j).toString().replace("x",bn.getField3());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
```

```
ResultSet rshashvalue5_0=db.select(selquery);
if(rshashvalue5_0.next())
{
result=rshashvalue5_0.getString(1)!=null?rshashvalue5_0.getString(1):"";
}
array1[Integer.parseInt(result)]=array1[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter5.get(j).toString().replace("x",bn.getField6());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue5_1=db.select(selquery);
if(rshashvalue5_1.next())
{
result=rshashvalue5_1.getString(1)!=null?rshashvalue5_1.getString(1):"";
}
array1[Integer.parseInt(result)]=array1[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter5.get(j).toString().replace("x",bn.getField9());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue5_2=db.select(selquery);
if(rshashvalue5_2.next())
{
result=rshashvalue5_2.getString(1)!=null?rshashvalue5_2.getString(1):"";
}
array1[Integer.parseInt(result)]=array1[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter5.get(j).toString().replace("x",bn.getField12());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue5_3=db.select(selquery);
if(rshashvalue5_3.next())
{
result=rshashvalue5_3.getString(1)!=null?rshashvalue5_3.getString(1):"";
}
array1[Integer.parseInt(result)]=array1[Integer.parseInt(result)]+1;
```

```
hashfunction=hshfunccounter5.get(j).toString().replace("x",bn.getField15());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue5_4=db.select(selquery);
if(rshahvalue5_4.next())
{
result=rshashvalue5_4.getString(1)!=null?rshashvalue5_4.getString(1):"";
}
array1[Integer.parseInt(result)]=array1[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter5.get(j).toString().replace("x",bn.getField18());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue5_5=db.select(selquery);
if(rshashvalue5_5.next())
{
result=rshashvalue5_5.getString(1)!=null?rshashvalue5_5.getString(1):"";
}
array1[Integer.parseInt(result)]=array1[Integer.parseInt(result)]+1;
int ar[]=new int[5];
for(int m=0;m<5;m++)
{
ar[m]=array1[m];
}
counterpositionafterhashingfor5bit.add(ar);
}
// For 6 bit counter
int array2[]=new int[6];
for(int j=0;j<hshfunccounter6.size();j=j+1) {
String hashfunction=hshfunccounter6.get(j).toString().replace("x",bn.getField3());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue6_0=db.select(selquery);
if(rshashvalue6_0.next())
```

```
{
result=rshashvalue6_0.getString(1)!=null?rshashvalue6_0.getString(1):"";
}
array2[Integer.parseInt(result)]=array2[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter6.get(j).toString().replace("x",bn.getField6());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue6_1=db.select(selquery);
if(rshashvalue6_1.next())
{
result=rshashvalue6_1.getString(1)!=null?rshashvalue6_1.getString(1):"";
}
array2[Integer.parseInt(result)]=array2[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter6.get(j).toString().replace("x",bn.getField9());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue6_2=db.select(selquery);
if(rshashvalue6_2.next())
{
result=rshashvalue6_2.getString(1)!=null?rshashvalue6_2.getString(1):"";
}
array2[Integer.parseInt(result)]=array2[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter6.get(j).toString().replace("x",bn.getField12());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue6_3=db.select(selquery);
if(rshashvalue6_3.next())
{
result=rshashvalue6_3.getString(1)!=null?rshashvalue6_3.getString(1):"";
}
array2[Integer.parseInt(result)]=array2[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter6.get(j).toString().replace("x",bn.getField15());
selquery="select "+hashfunction+" from dual";
```

```
System.out.println(selquery);
ResultSet rshashvalue6_4=db.select(selquery);
if(rshashvalue6_4.next())
{
result=rshashvalue6_4.getString(1)!=null?rshashvalue6_4.getString(1):"";
}
array2[Integer.parseInt(result)]=array2[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter6.get(j).toString().replace("x",bn.getField18());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue6_5=db.select(selquery);
if(rshashvalue6_5.next())
{
result=rshashvalue6_5.getString(1)!=null?rshashvalue6_5.getString(1):"";
}
array2[Integer.parseInt(result)]=array2[Integer.parseInt(result)]+1;
int ar[]=new int[6];
for(int m=0;m<6;m++)
{
ar[m]=array2[m];
}
counterpositionafterhashingfor6bit.add(ar);
}
// For 7 bit counter
int array3[]=new int[7];
for(int j=0;j<hshfunccounter7.size();j=j+1) {
String hashfunction=hshfunccounter7.get(j).toString().replace("x",bn.getField3());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue7_0=db.select(selquery);
if(rshashvalue7_0.next())
{
result=rshashvalue7_0.getString(1)!=null?rshashvalue7_0.getString(1):"";
```

```
}
array3[Integer.parseInt(result)]=array3[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter7.get(j).toString().replace("x",bn.getField6());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue7_1=db.select(selquery);
if(rshashvalue7_1.next())
{
result=rshashvalue7_1.getString(1)!=null?rshashvalue7_1.getString(1):"";
}
array3[Integer.parseInt(result)]=array3[Integer.parseInt(result)]+1;
hashfunction=hshfunccounter7.get(j).toString().replace("x",bn.getField9());
selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rshashvalue7_2=db.select(selquery);
}
```

public String generatedummytransactions(String transno)

```
{
String msg="";
String tid="1";
DBConnection db=null;
try{
db=new DBConnection();
db.openConnection();
String query="";
query="insert into
mstdummytransaction(TRANSACTIONID,QUERYID,QUERYID1,QUERYID2,QU
ERYID3,QUERYID4,QUERYID5) "
+ "select rownum,floor(dbms_random.value(1,36)),floor(dbms_random.value(1,36)),"
floor(dbms_random.value(1,36)),floor(dbms_random.value(1,36)),floor(dbms_rando
m.value(1,36)),floor(dbms_random.value(1,36)) "
+ "from sys.access$ where rownum<="+transno;
```

```
System.out.println(query);
db.insert(query);
db.commit();
msg="Success";
}
catch(Exception e)
{
e.printStackTrace();
}
finally
{
db.closeConnection();
}
return msg;
}
```

public String gethashedvalues(String weight,String hashfunction)

```
{
DBConnection db=null;
String result="";
try{
db=new DBConnection();
db.openConnection();
hashfunction=hashfunction.replace("x",weight );
String selquery="select "+hashfunction+" from dual";
System.out.println(selquery);
ResultSet rs=db.select(selquery);
if(rs.next())
{
result=rs.getString(1)!=null?rs.getString(1):"";
}
}
catch(Exception e)
```

```
{
e.printStackTrace();
}
finally
{
db.closeConnection();
}
return result;
}
```

public ArrayList getQueryweights(String transactionid)

```
{
DBConnection db=null;
ArrayList arr=new ArrayList();
CommonBean bn=null;
try{
db=new DBConnection();
db.openConnection();
String selquery="select a.queryid,(select operatorid||' '||tablename from mstquery
where queryid=a.queryid )," 
+ "(select weight from mstquery where queryid=a.queryid ),  "
+ "a.queryid1,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid1 )," 
+ "(select weight from mstquery where queryid=a.queryid1 )," 
+ "a.queryid2,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid2 )," 
+ "(select weight from mstquery where queryid=a.queryid2 )," 
+ "a.queryid3,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid3 )," 
+ "(select weight from mstquery where queryid=a.queryid3 )," 
+ "a.queryid4,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid4 )," 
+ "(select weight from mstquery where queryid=a.queryid4 )," 
```

```
+ "a.queryid5,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid5 )," 
+ "(select weight from mstquery where queryid=a.queryid5 ) from msttransaction a
where a.transactionid='"+transactionid+"'";
System.out.println(selquery);
ResultSet rs=db.select(selquery);
if(rs.next())
{
arr.add(rs.getString(1)!=null?rs.getString(1):"");
arr.add(rs.getString(2)!=null?rs.getString(2):"");
arr.add(rs.getString(3)!=null?rs.getString(3):"");
arr.add(rs.getString(4)!=null?rs.getString(4):"");
arr.add(rs.getString(5)!=null?rs.getString(5):"");
arr.add(rs.getString(6)!=null?rs.getString(6):"");
arr.add(rs.getString(7)!=null?rs.getString(7):"");
arr.add(rs.getString(8)!=null?rs.getString(8):"");
arr.add(rs.getString(9)!=null?rs.getString(9):"");
arr.add(rs.getString(10)!=null?rs.getString(10):"");
arr.add(rs.getString(11)!=null?rs.getString(11):"");
arr.add(rs.getString(12)!=null?rs.getString(12):"");
arr.add(rs.getString(13)!=null?rs.getString(13):"");
arr.add(rs.getString(14)!=null?rs.getString(14):"");
arr.add(rs.getString(15)!=null?rs.getString(15):"");
arr.add(rs.getString(16)!=null?rs.getString(16):"");
arr.add(rs.getString(17)!=null?rs.getString(17):"");
arr.add(rs.getString(18)!=null?rs.getString(18):"");
}
}
catch(Exception e)
{
e.printStackTrace();
}
finally
```

```
{
db.closeConnection();
}
return arr;
}
```

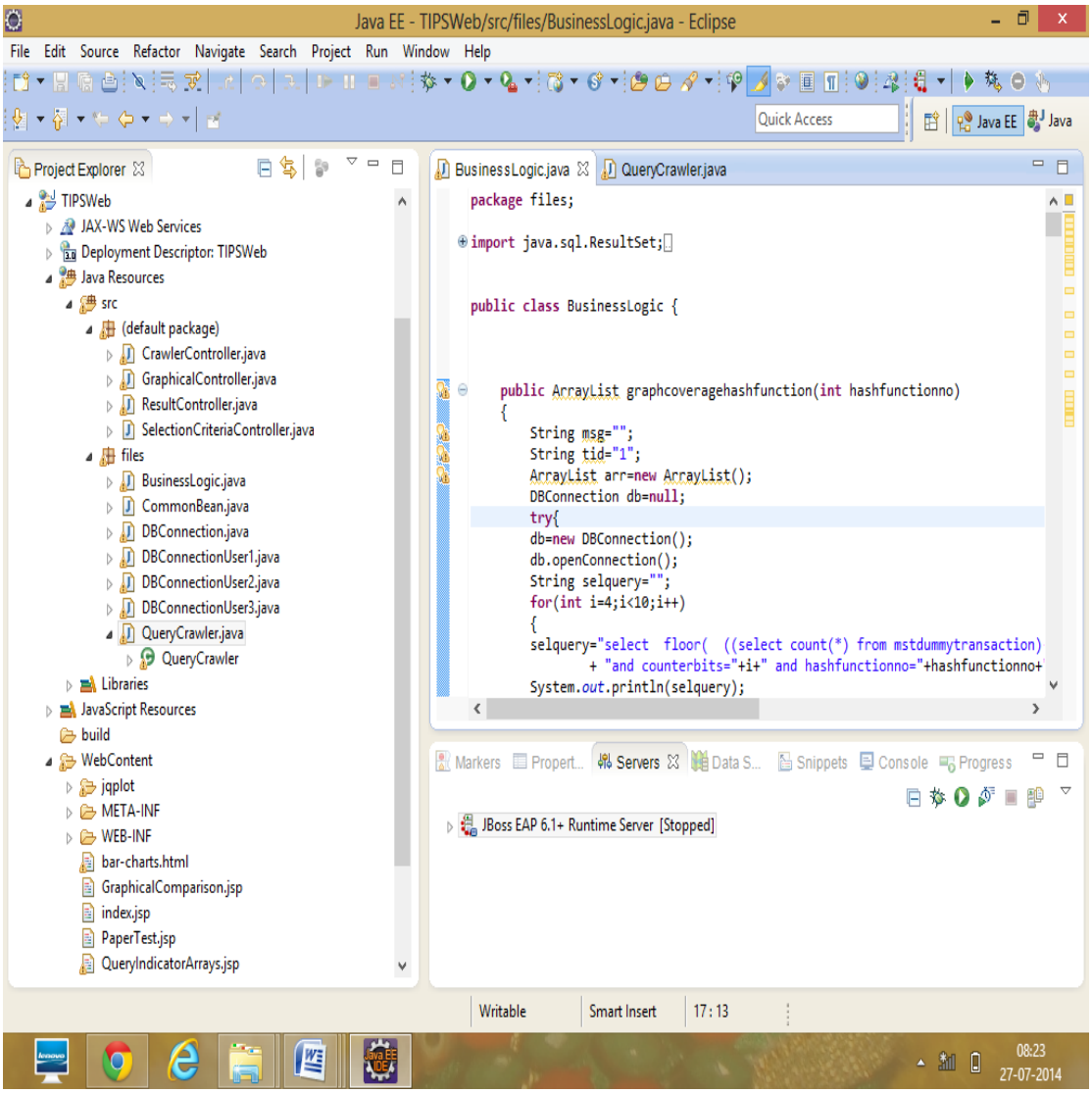
public ArrayList getDummyQueryweights(String transactionid)

```
{
DBConnection db=null;
ArrayList arr=new ArrayList();
CommonBean bn=null;
try{
db=new DBConnection();
db.openConnection();
String selquery="select a.queryid,(select operatorid||' '||tablename from mstquery
where queryid=a.queryid )," 
+ "(select weight from mstquery where queryid=a.queryid ),  "
+ "a.queryid1,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid1 )," 
+ "(select weight from mstquery where queryid=a.queryid1 )," 
+ "a.queryid2,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid2 )," 
+ "(select weight from mstquery where queryid=a.queryid2 )," 
+ "a.queryid3,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid3 )," 
+ "(select weight from mstquery where queryid=a.queryid3 )," 
+ "a.queryid4,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid4 )," 
+ "(select weight from mstquery where queryid=a.queryid4 )," 
+ "a.queryid5,(select operatorid||' '||tablename from mstquery where
queryid=a.queryid5 )," 
+ "(select weight from mstquery where queryid=a.queryid5 ) from
mstdummytransaction a where a.transactionid="'+transactionid+"'";
```

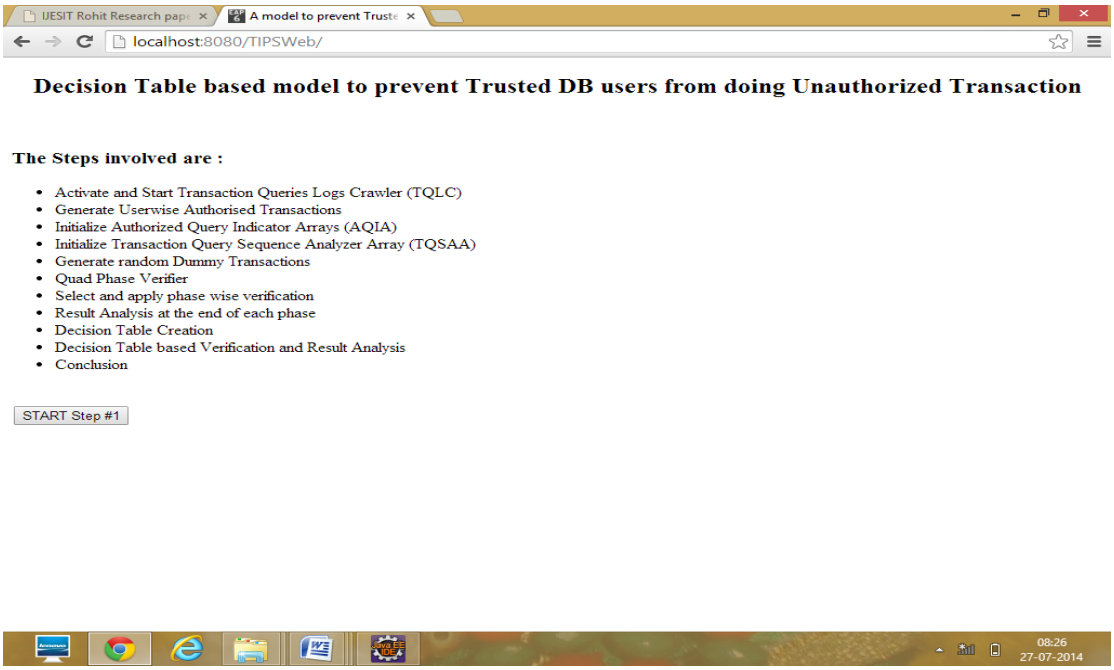
```
System.out.println(selquery);
ResultSet rs=db.select(selquery);
if(rs.next())
{
arr.add(rs.getString(1)!=null?rs.getString(1):"");
arr.add(rs.getString(2)!=null?rs.getString(2):"");
arr.add(rs.getString(3)!=null?rs.getString(3):"");
arr.add(rs.getString(4)!=null?rs.getString(4):"");
arr.add(rs.getString(5)!=null?rs.getString(5):"");
arr.add(rs.getString(6)!=null?rs.getString(6):"");
arr.add(rs.getString(7)!=null?rs.getString(7):"");
arr.add(rs.getString(8)!=null?rs.getString(8):"");
arr.add(rs.getString(9)!=null?rs.getString(9):"");
arr.add(rs.getString(10)!=null?rs.getString(10):"");
arr.add(rs.getString(11)!=null?rs.getString(11):"");
arr.add(rs.getString(12)!=null?rs.getString(12):"");
arr.add(rs.getString(13)!=null?rs.getString(13):"");
arr.add(rs.getString(14)!=null?rs.getString(14):"");
arr.add(rs.getString(15)!=null?rs.getString(15):"");
arr.add(rs.getString(16)!=null?rs.getString(16):"");
arr.add(rs.getString(17)!=null?rs.getString(17):"");
arr.add(rs.getString(18)!=null?rs.getString(18):"");
}}
catch(Exception e)
{
e.printStackTrace();
}
finally
{
db.closeConnection();
}
return arr;
                                        }
```

APPENDIX-II Screen Shots of Application

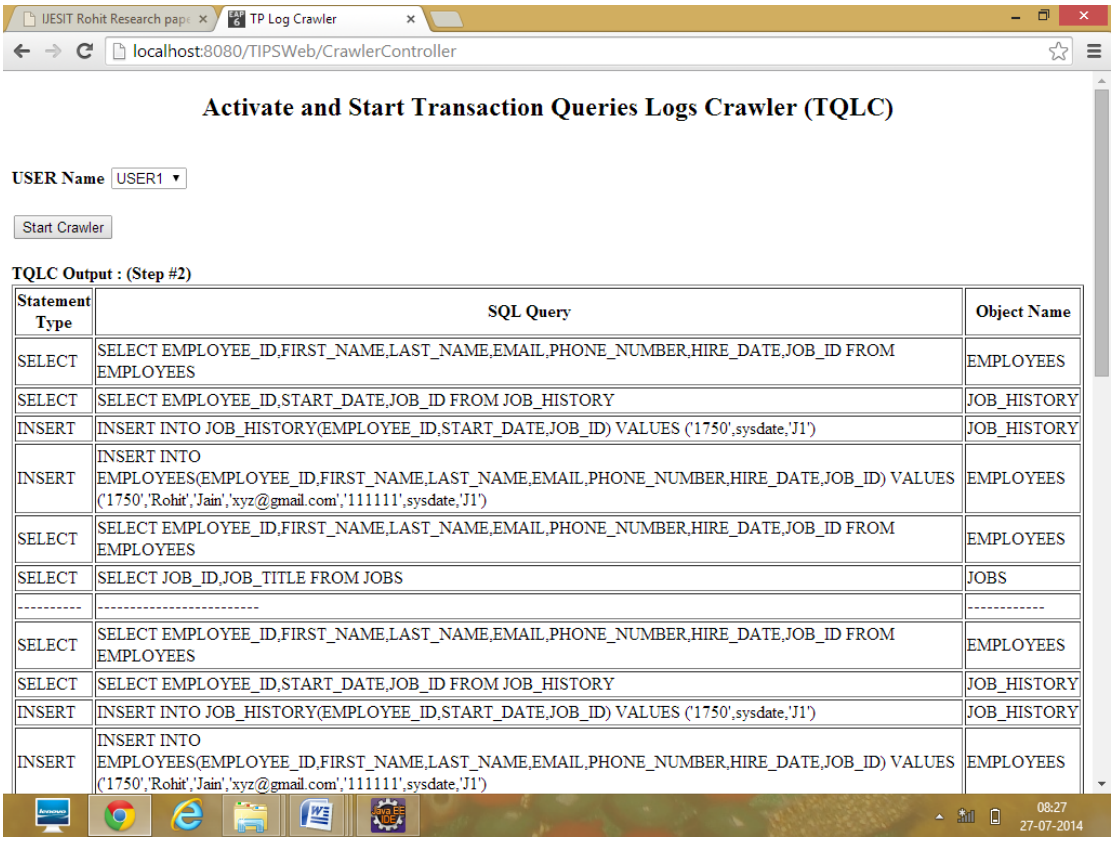
1. Application Directory Structure



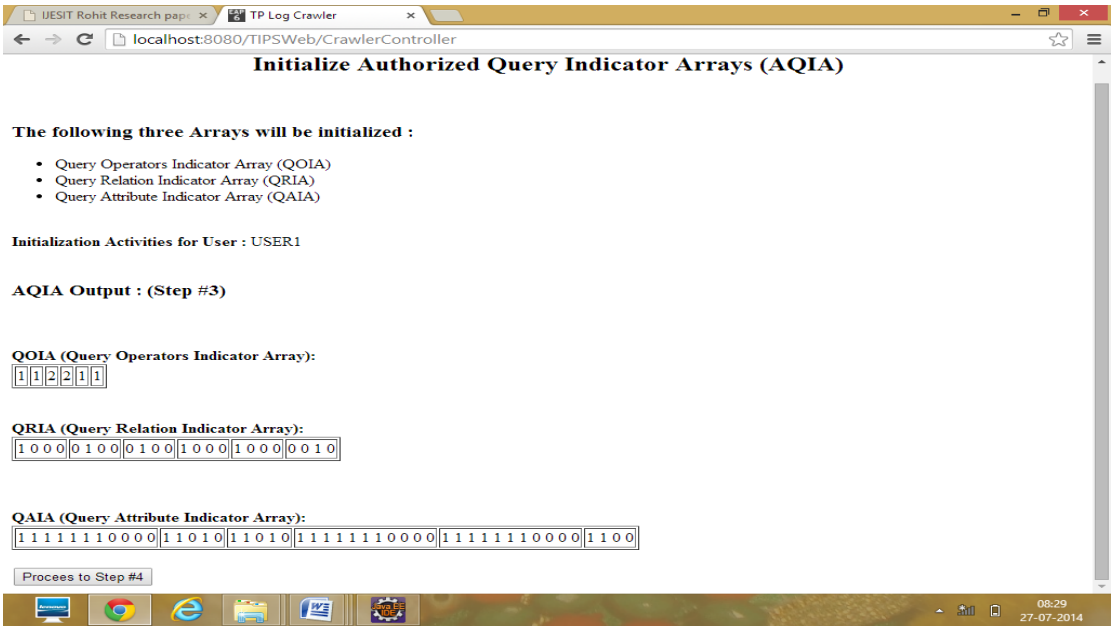
2. Index Page



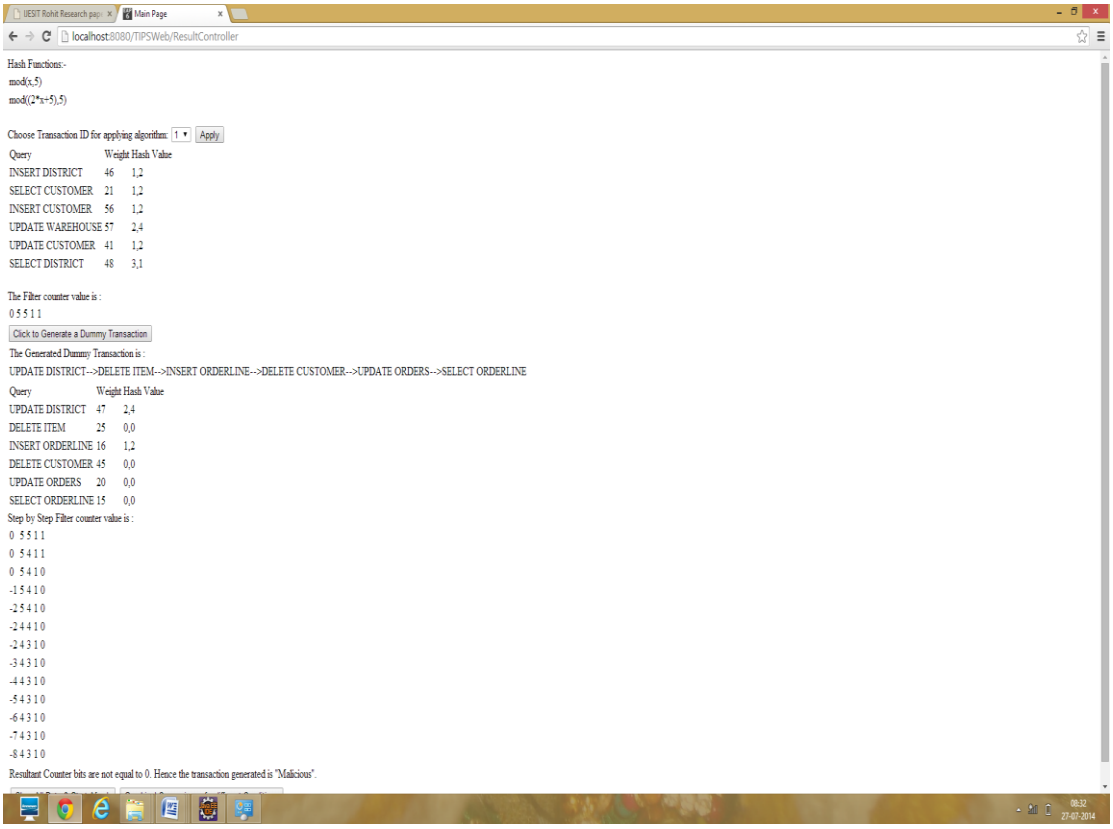
3. Activation & Initialization of TQLC



4. Initialization of AQIA



5. QWA Implementation



6. Graphical DashBoard