Efficient Testing of Database Applications

Mirza Mahmood Baig[†] and Ansar Ahmad Khan[††]

NED University of Engineering & Technology, Karachi Pakistan

Summary
This concept is quite recent provides a technique for more efficient testing of database applications. In the testing technique the original database remains intact throughout the experiment. When tester tests the test suites insert, delete, and modify updates are physically stored in another file or database called "Differential Table. The joint operation on Original database and differential table by sql query language together forms another database called, "Hypothetical database state". In this paper, we have designed a computer aided formula for test suites which make "decision", that the test requirement be executed hypothetically or traditionally. We have also designed pseudo code for hypothetical inserting and deleting tuple in the database without changing the originality of the database.

Key words:
Software Testing, Hypothetical database, Differential Table, Parent Table, traditional database

1. Introduction

The concept of hypothetical relation and differential file was first explored by Michael Stonebraker as described in [1, 2, and 3]. The implementation of hypothetical relation supports "what-if database" which is implemented using differential file. As per description in [4, 5, 6], a new concept in the field of hypothetical database namely "Independent Update Views" popularly known as IUV for database has been presented. As describe in [7] introduces terminology and basic concept to extend IUVs for version management. It forms a version hierarchy and called it version tree like B tree and represents each node (or child) is an IUV change version of its parent.

The advantage of hypothetical database testing is that it is cost effective and time reducing strategy and the additional benefit of this proposed technique is that the original database remains intact throughout the process. As compared to traditional database testing which is also cost effective and time reducing approach but numbers of observations prove that the cost and time increase when at any stage forced reset is used. It takes enough time to reset desired database state as compared to hypothetical database testing (HDT) as hypothetical rollback takes lesser time to reach desired database state. Another main disadvantage of traditional database testing is that original database no longer maintains its originality after executing test suites, whereas in the proposed technique it remains intact.

In this paper we have presented the concept of hypothetical data base in software testing, which is quite a new concept in the field of software testing. The proposed technique consists of: we have developed a new Computer aided formula for hypothetical database testing which checks the test suites requirements whether test suites executed hypothetically or traditionally, designed pseudo code of proposed technique have been presented. An example system, results achieved and conclusions are presented.

2. Computer Aided Formula

We have developed a formula for hypothetical database testing by using test suites requirements. When tester executes test suites requirements one by one, the formula automatically decides, whether the test suites say t_i, where i=1, 2, 3,…, k is executed hypothetically or traditionally. In other words the computer aided formula (or Mathematical function) which tells us when to update-in-place and when to rollback according to the environment of input requirements of test suites. For designing the formula below using the concept of "Functional Analysis" a well known branch of Mathematics, it is claimed that the said formula of this type has not been used in current state of art and may be regarded a novelty in the field of software testing.

The formula is:

$$\mathbf{f(n,m_i,r_{i+1})=2r_{i+1}–(n+m_i)} \qquad \text{(I)}$$

There are three main cases which we will elaborate here. First we will discuss the functionality of above function or formula. In this function 'n' refer to the number of records or tuples present in our base table or Parent table which is retrieved from original database, 'm_i' refers to number of records or tuples present in updated databases state (or current database state). It may be hypothetical, chain hypothetical (i.e. sequence of hypothetical state) or traditional database state, and 'r_{i+1}' refer to the test suites requirements which will be executed one after the other. Now, we deal each of the cases as follows:

Manuscript received April 5, 2009
Manuscript revised April 20, 2009

Case 1 for n = m_i
If **d(n,m_i) = 0**, it means both the database states have equal number of records and attributes. It means there is no difference between the two states. So in this scenario our current state is updated database state which is 'm_i' than execute test case update in place. Keeping in mind the situation remember that we are always in current state whenever trying to execute test suites. The advantage of update in place is that we have saved time and cost. On the contrary if we execute test in our base table or parent table first we have to rollback to PT which will give required more time as compared to update in place.

Case 2 for n > m_i,
Number of records in base table or Parent table is greater than the current state. In this scenario we have to elaborate it into three different sub cases as given below:

Subcase 1 **if r_{i+1} < m_i,** applying computer aid formula (A) to show that the distance between test requirements and current database state is less than zero which shows 'r_{i+1}' is closer to 'm_i',(i.e. current state). Now we define mathematically

$d(r_{i+1}, m_i) < 0$ or $\| r_{i+1,} m_i \|$ **< 0 ,** update in place
(i.e. current state satisfies the requirement of test suite)

This subcase condition shows that the current state is the most suitable to execute test suites requirements because a test suites requirement is closer to 'm_i'. Keeping in mind the situation remember that we are always in current state whenever trying to execute test suites. It is possible that both states will fulfill the requirements of test suites. Now numbers of observations prove that if we execute test suite from the current state, this will produce the same result as we execute from parent state, but in this case it is rather convenient to use current state as compared to PT, because it gives us the saving of time and cost.

Subcase 2 **if r_{i+1} > n**, applying computer aid formula to show that the distance between test requirements and parent database state is greater than zero which show 'r_{i+1}' is closer to 'n',(i.e. current state does not satisfy requirements of test suite and implies that we should rollback hypothetically).

$d(r_{i+1}, n) > 0$ or $\| r_{i+1,} n \| > 0$,
Roll Back Hypothetically

In this subcase condition shows that the current state is not suitable to execute test suites requirements because a test suites requirement is closer to 'n'. So rollback hypothetically to parent state or base table and execute test suites. In the scenario the current state is not fulfilling the requirements of test suites or simply test case requirement is not included in the current state but it is present in **parent state**. If we execute test suites in the current state it will take **much time** for preparation. So, hypothetical roll back is convenient to use parent table because it takes lesser time as compared to current state.

Subcase 3 **if mi < r_{i+1} < n**, after applying computer aided formula we have faced two conditions. One condition shows that a test suites requirement is near to the current state which is 'm_i' so execute test suite update in place. Second condition shows that the test suites requirement is closer to parent state which is 'n' so first rollback hypothetically to the PT and after that execute test suite, the same as symbolically defined as follows:

(i) if $f(n,m_i,r_{i+1}) < 0$, test execute update in place
$d(r_{i+1}, n) > d(r_{i+1}, m_i)$ or
$\| r_{i+1,} n \| > \| r_{i+1,} m_i \|$

(ii) if $f(n,m_i,r_{i+1}) > 0$, test execute Roll back hypothetically
$d(r_{i+1}, n) < d(r_{i+1}, m_i)$ or
$\| r_{i+1,} n \| < \| r_{i+1,} m_i \|$

The subcase 3 is the combination of subcase 1 and subcase 2

Case 3 for n < m_i
Number of records in base table or Parent table is less than the current database state. To remedy the said complexity we have distributed the said case into three different sub cases as given below:

Subcase 1 **if r_{i+1} > m_i,** after applying computer aided formula 'A' we see that the distance between test requirements and current database state is greater than zero. According to the condition of said case (i.e. case 3), it clearly shows that 'r_{i+1}' is closer to 'm_i', symbolically defined as:

$d(r_{i+1}, m_i) > 0$ or $\| r_{i+1,} m_i \| > 0$, (i.e. update in place)

This sub case condition shows that current state is the most suitable to execute test suites requirements because a test suites requirement is closer to 'm_i'. Keeping in mind the situation we are always in current state whenever trying to execute test suites.

Subcase 2 **if r_{i+1} < n,** after applying computer aided formula 'A' we see that the distance between test requirements and parent database state is less than zero, i.e.'r_{i+1}' is closer to 'n', formally defined as:

$d(r_{i+1}, n) < 0$ or $\| r_{i+1,} n \| < 0$, roll back hypothetically

This sub case condition shows that current state is not suitable to execute test suites requirements because a test suites requirement is closer to 'n'. So, rollback hypothetically to parent state (or base table) and execute test suites requirement. In the scenario current state is does not fulfill the requirement of test suites or simply test case requirement is not included in current state but of course, included in parent state. If we execute test suites in current state it will take much time for preparation. So, hypothetical roll back is convenient to use parent table because it takes lesser time as compared to current state.

Subcase 3 **if $m_i < r_{i+1} < n$,** after applying computer aided formula we have faced two conditions. One condition shows that test suite requirement is near to current state which is 'm_i' so execute test suite update in place. Second condition shows that the test suites requirement is closer to parent state which is 'n' so first rollback hypothetically to the PT and execute test suite. Formally we define it as follows:

(i) if $f(n,m_i,r_{i+1}) > 0$, test suite execute update in place
$d(r_{i+1}, n) > d(r_{i+1}, m_i)$ or
$\| r_{i+1,} n \| > \| r_{i+1,} m_i \|$

(ii) if $f(n,m_i,r_{i+1}) < 0$, test suite execute rollback hypothetically
$d(r_{i+1}, n) < d(r_{i+1}, m_i)$
or $\| r_{i+1,} n \| < \| r_{i+1,} m_i \|$

The above discussion can be summarized as: "This subcase 3 is the combination of subcase 1 and subcase 2".

3. Pseudo Code of Proposed Technique

SetCurrentState (String StateName)

a. If a state exists in Hsmetadata with the name "State name"
(i) Set the 'Current state' column of the actual current state to false (or '0').
(ii) Set the 'Current state' column of the state called 'State name' to true (or '1').

CreateNewHypState (String Parentstatename, String newHypstatename)

a. Get the names of the tables of type TABLE through getTables() of the DatabaseMetaData object.
b. Using a while loop, iterate through each table name stored in the resultset obtained in step 2a and performed the following action in each iteration for the current table name.
(i) Get the name of the table from the metadata result set.
(ii) Obtained the maxIndex from the SeqMetaData table for the corresponding table name acquired in step i.
(iii) Set index = maxIndex +1
(iv) Obtained a unique Differential name for the current table name using the unique index.
(v) Call CreateDT( ) by passing the tableName and the unique DT name.
(vi) Execute an insert statement to insert a row containing table name, state name, and DT name in the Metadata table.
c. Set the 'Current state' column of the actual current state to false (or '0').
d. Execute a sql insert statement to insert a row in the HS Metadata table. (Parentstate = parent state name. Current state = '1', HSname = new Hypstate.)

CreateDT (String tableName, String dtName)

a. Using the getColumns() of DatabaseMetaData object, retrieve all the columns of the given tableName.
b. Iterate using a while loop for each column in the obtained resultset and perform the following actions in each iteration.
(i) Retrieve column name and its type from the resultset and append this information to the columnNames string to dynamically create the column field of the CREATE TABLE statement.
c. Add Action field to the columnName to capture action in DT
d. Complete the CREATE TABLE statement by appending missing information.
e. Create a statement object and call its executeUpdate () to run the create table Query constructed in step d.

DeleteHypState (String stateName)

a. Select ParentState, current state, StateName from HSMetaData where StateName is equal to the given stateName.
b. If the row found in the HSMetaDataTable then performed the following actions
i. Execute an Update Statement to set CurrentState =true where StateName is equal to the retrieved ParentState in step f.
ii. Delete the HSMetaData table row with StateName = given state name by calling

DeleteMetaDataRow(). The function will also drop all the corresponding DTs.
iii. Select ParentState, StateName from HSMetaData where ParentState is equal to the given stateName. This is to search child HS.
iv. If child HS exists then call deleteHypState() recursively by giving the child HS as an argument.
v. Set global currentState to given stateName.

DeleteMetaDataRow(String stateName)

a. Execute a delete SQL statement to delete a row from the HSMetaData table with StateName equal to given stateName
b. Drop all the corresponding Differential tables by calling dropHsDTs();

DropHsDTs(String stateName)

a. Select all DT Name from the DTMetaData table with the HSName equal to given stateName.
b. Iterate using a while loop through the DTNames resultSet and drop each table using the SQL drop statement.

RollBackCurrentState ()

a. Execute a select SQL statement to fetch the list of DTables from the DT metadata table.
b. Execute a drop sql statement for each of the DTtable name obtained in a.

3.1 Hypothetical Inserting tuple in differential Table

If we have to insert a tuple in Differential Table (DT), then the following precautions are to be considered.

- Check first the desired input tuple if it does not exist in parent table (PT)
- Check the desired input tuple if it does not exist in DT

In other words the desired inserted tuple should not be seen in PT as well as DT.

If the above precautions are satisfied by tester requirements, then insert a desired input tuple in DT with action column 'i', and applying IUV query which hypothetically updates the PT with the given inserted tuple.

In general, if we have to insert a tuple in PT through DT, we have been facing following eight conditions as given in the form of Table or Matrix below:

Insert tuple	Exists in DT with 'i'	Exists in DT with 'd'	Exists in DT with 'm'	Does not exist in 'DT'
Exists in PT	Malformed Hypothetical table	Delete Tuple from DT	Delete tuple from DT.	Already exist tuple does not insert in DT
Does not exist in PT	Error tuple already exists in DT	Malformed Hypothetical table	Malformed Hypothetical table	Insert into DT with action = 'i'

Table 1

The above tabular form clearly indicates the results of eight conditions (where condition means the location of rows and columns) which will be elaborated one by one. In these conditions only one result is normal, some are abnormal, and some required preparation. In location first row and first column which is condition 1 shows result '**Malformed Hypothetical state**' in this condition inserted tuple is already present in Parent table as well as in Differential table with action 'i',so this is abnormal request. In condition 2 first row second column in this condition inserted tuple is already present in PT as well as in DT with action'd', so this condition requires some preparation. After preparing, **delete tuple** from DT and the request is valid or normal. In condition 3 first row third column, inserted tuple is already present in PT as well as in DT with action'm', applying same preparation as in condition 2. In condition 4 first row fourth column, the inserted tuple is already present in PT but not in DT, showing error request because already existing tuple does not insert in DT. In condition 5 second row first column, the inserted tuple does not exist in PT but exists in DT with action 'i', showing error request, because tuple exist in DT with action 'i', after fire a IUV query 5.3.8 which is join PT and DT tuples the required request is present in PT. In condition 6 second row second column inserted tuple does not exist in PT but exists in DT with action'd' so abnormal request is sent by tester. Because we intend **to** insert tuple in DT but the said tuple already exists in DT with action delete. In condition 7 second row third column, the inserted tuple does not exist in PT but exists in DT with action 'm' so abnormal request is sent by tester because modified tuple must be present in PT. In condition 8 second row fourth column, inserted tuple does not exist in PT as well as in DT, so this is legal condition and as such insert tuple in DT with action 'i'.

3.2 Hypothetical deleting tuple in differential Table

If we have to delete a tuple in PT via DT, then the following precautions are to be considered.

- The desired deleted tuple must be present in PT
- The desired deleted tuple must not exist in DT.

In other words the desired input deleted tuple must be seen in PT and could not be seen in DT

If the above precautions are satisfied by tester requirements, retrieved the desired input tuple from PT and insert this tuple into DT with action column‘d’.

In general, if we have to delete a tuple in PT through DT, we have been facing following eight conditions as given in the form of Table or Matrix below:

Delete tuple	Exists in DT with ‘i’	Exists in DT with ‘d’	Exists in DT with ‘m’	Does not exist in ‘PT’
Exists in PT	Malformed Hypothetical table	Error Tuple does not exist	Delete tuple from DT. Find tuple in PT with same tuple id as delete tuple. Insert this tuple into DT with action column ‘d’	Insert tuple into DT with action = ‘d’
Does not exist in PT	Delete the tuple from DT	Malformed Hypothetical state	Malformed Hypothetical table	Error Tuple does not exist.

Table 2

The above tabular form clearly indicates the results of eight conditions (where condition means the location of rows and columns) which will be elaborated one by one. In these conditions only one result is normal, some are abnormal, and the rest require preparation.

In location first row and first column i.e., condition 1, shows the result ‘**Malformed Hypothetical state**’. In this condition deleted tuple exists in Parent table but the same tuple exist in Differential table with action ‘i’ also. According to table 2, this is abnormal request. In condition 2 first row second column, deleted tuple is present in PT which is right but the same tuple exists in DT with action‘d’. According to table 2, there is error (i.e. tuple does not exist). In condition 3 first row third column, deleted tuple is present in PT as well as in DT with action‘m’, According to table2, this condition requires preparation. First delete tuple from DT and find tuple in PT with same tuple ‘id’ as deleted tuple. Finally insert deleted tuple into DT with action column‘d’. In condition 4 first row fourth column, the deleted tuple exists in PT but does not exist in DT, According to table2, it is purely legal condition i.e. insert deleted tuple in to DT with action column‘d’. In condition 5 second row first column, the deleted tuple does not exist in PT but exists in DT with action ‘i’, According to table 2, request sent by tester shows error. To remove the abnormality of the request, i.e. delete the deleted tuple from DT. In condition 6 second row second column deleted tuple does not exist in PT but exists in DT with action ‘d’, According to table 2 it implies an abnormal request sent by tester ‘**Malformed Hypothetical state**’. In condition 7 second row third column, the deleted tuple does not exist in PT but exists in DT with action ‘m’ which constitutes the same as condition 6. In condition 8 second row fourth column, the deleted tuple does not exist in PT as well as in DT, According to table 2, illegal request is sent by tester (i.e. tuple does not exist).

4. Application on Example System

Example defined: The following example system has been defined and then said technique applied to generate different test-cases, on hypothetically i.e. insert, and delete the records or tuples.

According to Hypothetical database the original table remains intact throughout the experiment. We call the original database refer to Parent table (PT). In PT we cannot have any modification physically. Whatever we change in the PT is stored in a separate table which we call a differential table (DT). As we know DT is physically storage table, which is used to update. Let us suppose PT Staff is Parent Table and DT_Staff is Differential table. Applying the proposed hypothetical database testing strategy (HDTS) to given problem to update records or tuples. (i.e. inserting the tuple, deleting the tuple, and modifying the tuple by using SQL query).

Given PT with following attributes, Staff number (staffNo), First Name (fName), Last Name (LName), Position, Sex, Date of Birth (DOB), Salary, branch number (BNo). The PT retrieved from given database. The difference between PT and DT are, the attributes of DT same as PT except one attribute namely ‘Action’ column which is in DT only. In other words the number of attributes in PT is one less than DT. For example PT has ‘n’ attributes the corresponding DT has ‘n+1’ attributes.

PT Staff							
Staff No	**fName**	**lName**	**Position**	**sex**	**DOB**	**salary**	**BNo**
SG14	David	Ford	Manager	M	3/24/1958	18000	B003
SG16	Alan	Brown	Assistant	M	5/25/1957	8549	B003
SG37	Ann	Beech	Assistant	F	10/11/1960	12360	B003
SG44	Anne	Jones	Assistant			8343	B003
SG46	Arif	James	Assistant	M	5/25/1997	9100	C003
SG5	Susan	Brand	Manager	F	3/6/1940	25956	B003
SL21	John	White	Manager	M	1/10/1945	32445	B005
SL41	julie	Lee	Assistant	F	6/13/1965	9270	B005

Table 3 (Before update)
This is called parent table (PT) or current state.

DT_Staff								
StaffNo	**fName**	**lName**	**Position**	**sex**	**DOB**	**salary**	**BNo**	**Action**

Table 4 (initial Differential table)

Test suite 1

Query: INSERT INTO DT_Staff
VALUES ('SG46', 'ARIF', 'JAMES', 'Assistant', 'M', '5/25/1997', 9100, 'C003',’Insert’);

Explanation:

Step 1: According to test case 1, check all the necessary conditions given in 3.1 and table 1 pseudo code for inserting tuple into PT through DT. According to pseudo code said query fulfills the requirement of condition 8. As the tuple does not exist in PT as well as in DT is purely a legal condition.

Table 5

DT_Staff								
Staff No	**fName**	**lName**	**Position**	**sex**	**DOB**	**salary**	**BNo**	**Action**
SG46	ARIF	JAMES	Assistant	M	5/25/1997	9100	C003	Insert

Step 2: Insert tuple into DT_Staff with required attributes and applying join query given in 4.2 (Query for Established new Hypothetical State) Differential Table after step 2

For completing the procedure of test 1, As a result the current state changes from table 3 to table 6.

New hypothetical State table

Now we execute next test suite requirement say test case 2.

Test suite 2

Table 6

PT Staff							
Staff No	**fName**	**lName**	**Position**	**sex**	**DOB**	**salary**	**BNo**
SA9	Mary	Howe	Assistant	F	2/19/1970	9270	B007
SG14	David	Ford	Manager	M	3/24/1958	18000	B003
SG16	Alan	Brown	Assistant	M	5/25/1957	8549	B003
SG37	Ann	Beech	Assistant	F	10/11/1960	12360	B003
SG44	Anne	Jones	Assistant			8343	B003
SG46	Arif	James	Assistant	M	5/25/1997	9100	C003
SG5	Susan	Brand	Manager	F	3/6/1940	25956	B003
SL21	John	White	Manager	M	1/10/1945	32445	B005
SL41	julie	Lee	Assistant	F	6/13/1965	9270	B005

Query: DELETE FROM Staff where StaffNo=‘SA9’;
INSERT INTO DT_Staff
VALUES ('SA9', 'MARY', 'Howe', 'Assistant', 'F', '2/19/1970', 9270, 'B007','Delete’);

Explanation:

Step 1: According to test suite 2, check all the necessary conditions given in 3.2 and table 2 pseudo code for deleting tuple from PT through DT. In this case current state is table 6, which is PT. According to said query clearly shows that the given tuple exist in PT but not exists in DT table 5. This is purely a legal condition of delete tuple from PT through DT.

Step 2: Retrieved the desired tuple from PT table 6 and insert into DT_staff table 5 with action field‘d’. Differential Table after step 2, as shown table 7.

Table 7

DT_Staff								
Staff No	**fName**	**lName**	**Position**	**sex**	**DOB**	**salary**	**BNo**	**Action**
SA9	Mary	Howe	Assistant	F	2/19/1970	9270	B007	delete
SG46	ARIF	JAMES	Assistant	M	5/25/1997	9100	C003	Insert

Table 8 (New hypothetical State table)

For completing the procedure of test 2, As a result the current state changes from table 6 to table 8.

4.1 Analysis the application program results

The result of the application program clearly verifies that in the example system we have considered only those test suites, which follow the legal conditions of insert, and

delete records (or tuples). All these test suites requirements are successfully executed from proposed application program. The example system clearly shows that when the tester executes test suites requirements one by one, the originality of the parent table (PT) cannot be disappeared throughout the testing process.

In most of the cases test suites requirements do not fulfill the valid conditions of insert/delete records. In other words test suites requirements do not follow the legal conditions. According to illegal condition of test suites requirements, the application program throws an exception or error. Some time we face "Bad request"/ "Malformed" condition sent by tester.

Some time we need to roll back any hypothetical state as per test suite requirements sent by the tester. We further accelerate this application program by some precautions.

- First analyze the all test suites requirements before executing the application program.
- Separate those test suites which have similar requirements in form of group

4.2 Join Query for established new hypothetical state

getIUV(TableName, DTName)

Following query will be dynamically created on the basis of metadata of the given parent and differential table:

Select tb.Column 1, tb.Column 2, . . .from tableName as tb
Difference
Select tb.Column 1, tb.Column 2,… from tableName as tb, DTName as DT where tb.Column 1 = DT.column 1 and tb.Column 2= DT.Column2 … (check all unique columns)
Unoin
Select DT>Column 1, DT>Column 2, … from DTName as DT where action =! Delete

5. Conclusion

The application program of proposed HDT has easily tested the test suites requirements consisting of insert, and delete.

If the tester sends request, to insert the record into database, then application program automatically checks the existence of the record in Parent Table (PT) as well as in Differential Table (DT) to avoid any duplication of pre-existed records. If the record exists, then generate exception to the user with detail of the error. If the record does not exist then application program inserts record successfully.

If the tester sends request, to delete the record into database, then application program automatically checks the given record in Parent Table (PT) as well as in Differential Table (DT) to avoid duplication. If the record is not fulfilled the legal condition then generate exception to the user with detail of the error. If the record fulfills the legal condition, then application program deletes record successfully.

The most significant feature of the Hypothetical Database testing (HDT) is that the originality of the database remains intact throughout the experiments. In case of traditional database testing the originality of the database disappears. The advantage of the HDTS is that, it is useful for most business application, and other important applications, where original database is required to be intact

Acknowledgements

We acknowledged the all kind of support by NED University of Engineering & Technology and specially the Vice Chancellor as chairman of Advanced Studies Research Board who always personally supported the research project being undertaken.

References

[1] Michael Stone Braker, "Hypothetical DataBases as Views", Proceedings of the 1981 ACM SIGMOD international conference on Management of data, 1981.
[2] Stone braker, M. and Keller, K., "Embedding Expert Knowledge and Hypothetical Databases into a Database System", Proc. 1980 ACM-SIGMOD Conference on Management of Data. Santa Monoca, Ca, May 1980-Sep. 1980.
[3] Woodfill, J., and Stonebraker, M., "An Implementation of hypothetical relational", Proceedings of the ninth International very large database Conference, Florence, Italy.
[4] R. Agrawal and D.J. Dewitt, "Updating Hypothetical Databases", Information processing Letter vol. 16 pp. 145-146. 1983.
[5] Richard G. Ramirez, Uday R. Kulkarni, Kathleen A. Moser. "The Cost of Retrievals in What-If Databases. Decision and Information Systems", Arizona State University, IEEE, 1991.
[6] Uday R. Kulkarni, Richard G. Ramirez. "Independently Updated Views", IEEE Transaction on Knowledge and Data Engineering, vol. 9, September 1997.
[7] Kathleen A. Moser, Uday R. Kulkarni, Richard G. Ramirez. "Scenario Management in Organizational DSS", IEEE, 1994.