An Oracle's Mutating Table Visual Detector Using Table's Trigger Dependency Method

Darlis Herumurti, Irfan Subakti and Dimas Kaharudin I.R.
Department of Informatics, Faculty of Information Technology
Institute Technology of Sepuluh Nopember Surabaya (ITS)
Kampus ITS, Keputih, Sukolilo, Surabaya, Indonesia
darlis@its-sby.edu, yifana@gmail.com, dimas@cs.its.ac.id

Abstract

A visualization is a popular method for the problem modeling. By using visualization processing, a problem is easier to comprehend, analyze and solve. In an active database (e.g., Oracle), the occurences of the mutating table is usual. A mutating table is a table which is currently being modified by an update, delete, or insert statement. When a trigger tries to refer a table which is in state of flux (i.e., being changed), it is considered as *mutating*. And it raises an error since Oracle should not return data which has not yet reached its final state. This problem is difficult to trace manually (i.e, by reading the source code of trigger) and it found frequently only at the runtime.

This paper describes a method for mutating table detection by a visualization (i.e, visualizing trigger dependency on the table) and then tracing it. Visualization processing is done by parsing the body of trigger, and furthermore it can be shown in a diagram. A dependency trigger shows the events (insert, update, delete) and the actions from the table. Here we proposed two mechanism to overcome the mutating table: (1) Chain Reaction Verification Algorithm which is used for comparing the previous *Action trigger* with the next *event trigger;* and (2) *Vertex to Circuit Expansion* (*VtCX*) Algorithm as an expansion principal from *vertices* to *circuit*. This expansion is done by substituting a vertex in path of one *single circuit* with another *single circuit*.

Finally, base on our experiments, trigger can be visualized into visual diagram. Together with graph theory, this diagram detects a mutating table easier , better than reading the source code directly.

Keywords: mutating table, trigger, active database, graph theory, diagram.

1. Introduction

Database becomes an essential component of daylife activities. It's fair enough to say that databases play a critical role in most areas where computer is involved within, such as business, electronic commerce, engineering, medicine, law, education and many more. A database system with rule processing capabilities provides a useful platform for large and efficient knowledge-base and expert systems. Database systems with production rules are referred to an active database systems. Rules that specify actions which are automatically triggered by certain events have been considered as an important enhancements to a database system for quite periods. These rules can be automatically triggered by the occurence of events (e.g., a database updating) or a certain time being reached, and can initiate certain actions that have been specified in the rule declaration if certain conditions are met. Many commercial packages already have some of the functionality provided by the active databases in the form of triggers. Triggers are a part of the SQL-99 standard, nowdays.

Types of an Oracle trigger are: Row Triggers and Statement Triggers, BEFORE and AFTER Triggers, INSTEAD OF Triggers, Triggers on System Events and User Events.

There's the trigger restrictions on mutating table. A mutating table is a table which is being modified by an UPDATE, DELETE, or INSERT statements, or a table which might be updated by the effects of a DELETE CASCADE constraint.

The session which is issued the triggering statement cannot be queried or modified a mutating table. This restriction prevents a trigger from obtaining an inconsistent set of data.

This restriction applies to all triggers which use the FOR EACH ROW clause. Views being modified in INSTEAD OF triggers are not considered mutating.

When a trigger encounters a mutating table, a runtime error will be occured, the effects of the trigger body and triggering statement are rolled back, and control is returned to the user or the application.

In the simple case of a trigger, a developer can be easy to see the triggers created and the dependencies among them. But in the complex systems, the excessive use of triggers can be result in the complex interdependencies. It can be difficult to maintain in a large application, especially if a mutating table is occurred which involves more than two tables. Searching for which triggers that cause a mutating table can be also difficult.

2. Trigger and Mutating Table

A database trigger is a stored subprogram associated with a database table, view, or event. A trigger can be called once, when some events are occurred; or many times, for each row affected by an `INSERT`, `UPDATE`, or `DELETE` statements. The trigger can be called after the event, for recording or for taking some follow-up actions. Alsor, the trigger can be called before the event to prevent erroneous operations or to fix a new data so that it conforms to the business rules. Syntax to create a trigger is:

```
CREATE [OR REPLACE] TRIGGER <trigger_name>
     {BEFORE|AFTER}  {INSERT|DELETE|UPDATE}
ON <table_name>
     [REFERENCING  [NEW  AS  <new_row_name>]
[OLD AS
        <old_row_name>]]
          [FOR      EACH      ROW      [WHEN
(<trigger_condition>)]]
    <trigger_body>
```

A trigger has three basic parts:
A. Triggering Event or Statement
B. Trigger Restriction
C. Trigger Action

A. Triggering Event or Statement

It's the SQL statement, database event, or user event that causes a trigger to fire. A triggering event can be one or more of the following:

- An INSERT, UPDATE or DELETE statements on a specific table (or view, in some cases)
- A CREATE, ALTER or DROP statements on any schema objects
- A database startup or instance shutdown
- A specific error message or any error messages
- A user logon or logoff

B. Trigger Restriction

It specifies a Boolean expression that must be *true* for the trigger for firing. The trigger action will not run if the trigger restriction is evaluated to *false* or *unknown*.

C. Trigger Action

It's the procedure (PL/SQL block, Java program, or C callout) which is containing the SQL statements and the code to be run when the following events occur:

- A triggering statement is issued.
- The trigger restriction evaluates to *true*.

Just like the stored procedures, a trigger action can be:

- Contains SQL, PL/SQL or Java statements
- Define PL/SQL language constructs such as variables, constants, cursors and exceptions
- Define Java language constructs
- Call stored procedures

If the triggers are row triggers, the statements in a trigger action have access to column values of the row being processed by the trigger. The correlation names provide an access to the old and new values for each column. In the following we describe the trigger action in details.

Row Triggers

A row trigger is fired each time the table is affected by the triggering statement. For example, if an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement. If a triggering statement affects no rows, a row trigger will not run.

Row triggers are useful if the code in the trigger action depends on data provided by the triggering statement or rows that are affected.

Statement Triggers

A statement trigger is fired once on behalf of the triggering statement, regardless of the number of rows in the table that the triggering statement is affected, even if no rows are affected. For example, if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once.

Statement triggers are useful if the code in the trigger action does not depend on the data provided by the triggering statement or the rows affected.

BEFORE and AFTER Triggers

BEFORE and AFTER triggers fired by DML statements can be defined only on the tables, not on the views. However, triggers on the base tables of a view are fired if an INSERT, UPDATE or DELETE statements are issued against the view. BEFORE and AFTER triggers fired by DDL statements can be defined only on the database or a schema, not on particular tables.

BEFORE triggers run the trigger action prior the triggering statement is running. AFTER triggers run the trigger action after the triggering statement is running.

A *mutating* table is a table which is being modified by an UPDATE, DELETE or INSERT statements, or a table that might be updated by the effects of a DELETE CASCADE constraints. When a trigger tries to refer a table which is in state of flux (i.e., being changed), it is considered as *mutating*. And it raises an error since Oracle should not return data which has not yet reached its final state. A mutating table is applied to all triggers which use the FOR EACH ROW clause (row trigger).

3. Graph Theory

Definitions of graphs vary in the style and the substance, according to the level of abstraction which is appropriate to a particular approach or application. For the sake of perspective, the following definitions present the same substance in two different styles [Wik06]:

First, a classic definition that covers most of the essential ideas in a very short space:

A graph G consists of a finite nonempty set V = V(G) of p points together with a prescribed set X of q unordered pairs of distinct points of V. Each pair x = {u, v} of points in X is a line of G, and x is said to join u and v. We write x = uv and say that u and v are adjacent points (sometimes denoted u adj v); point u and line x are incident with each other, as are v and x. If two distinct lines x and y are incident with a common point, then they are adjacent lines. A graph with p points and q lines is called a (p, q) graph. The (1, 0) graph is trivial. ([Har69] p. 9)

Next, a style of definition that is preferred in some approaches and applications:

A graph or undirected graph G is an ordered triple G:=(V, E, f) subject to the following conditions:

- V is a set, whose elements are variously referred to as nodes, points, or vertices.
- E is a set, whose elements are known as edges or lines.
- f is a function that maps each element of E to an unordered pair of distinct vertices in V, referred to as the ends, endpoints, or end vertices of the edg*e.*

V (and hence E) are usually taken to be finite sets, and many of the well-known results are not true (or are rather different) for infinite graphs because many of the arguments fail in the infinite case.

A digraph or a directed graph G is an ordered pair G:=(V, A) subject to the following conditions:

- V is a set, whose elements are variously referred to as nodes, points, or vertices.
- A is a set of ordered pairs of vertices, called arcs, arrows, or directed edges. An edge e = (x, y) is said to be directed from x to y, where x is the tail of e and y is the head of e.

Alternatively, a digraph or a directed graph may be defined as an ordered triple G:=(V, E, f) subject to the following conditions:

- V is a set, whose elements are variously referred to as nodes, points or vertices.
- E is a set, whose elements are known as arcs, arrows or directed edges.
- f is a function that maps each element in E to an ordered pair of vertices in V.

There are also some mixed types of graphs with undirected and directed edges

A weighted graph is a graph in which each branch is given a numerical weight. A weighted graph is therefore a special type of labeled graph in which the labels are numbers (which are usually taken to be positive).

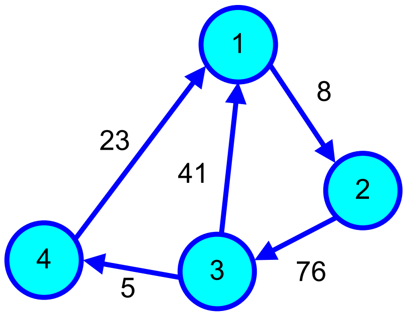


Figure 1. Directed Weighted Graph

A *path* joining two vertices X and Y of a digraph is a sequence of distinct points (vertices) and directed edges. A path starting and ending at one vertex P is called a *loop at* P. For example, in the digraph

A graph is called connected if there is a path connecting any two distinct vertices. It is called *disconnected* otherwise

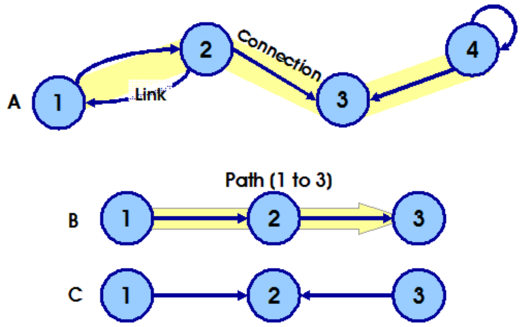


Figure 2. Path and Connection in a Graph

A *chain* is a sequence $x_1, x_2, \ldots, x_n$ such that $(x_1,x_2),(x_2,x_3), \ldots,(x_{n-1}, x_n)$ are graph edges of the graph and the are distinct. A closed path on a graph is called a graph *cycle or circuit*.

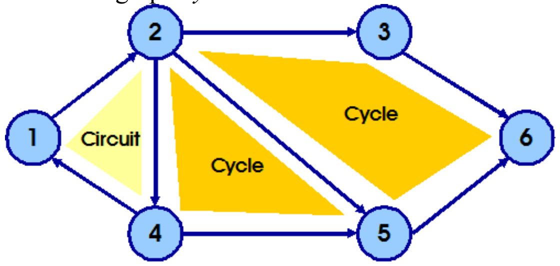


Figure 3. Cycle and Circuit

Chain Reaction: A series of events in which each induces or influences the next.

Depth First Search (DFS): is an uninformed search which is working by expanding the first appeared child node of the search tree and thus going deeper and so forth until a goal node is found, or until it encounters a node which has not a children. The search process, then, performs backtracking, returning the most recent nodes which have not finished to explore yet [Cor01].

4. Trigger Visualization Design (Dependency Trigger Diagram)

According to [Rac05], a diagram design should be fulfilled the following requirements:

A diagram should has a compatibility with current existing Physical Data Model (PDM) The symbols in the PDM diagram still can be used in this diagram.

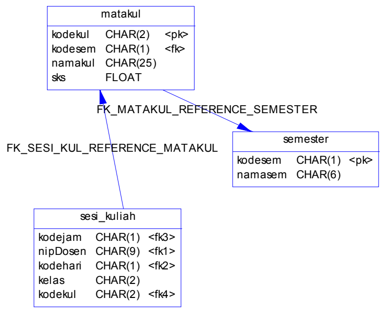


Figure 4. PDM Diagram from Power Designer 9

A diagram should be easy to read and to understand.

A diagram should be simple and easy to implement in an application.

Trigger visualization has priority on event and action, without detail information about event and action. This means that there is no information about what column which accessed by the trigger.

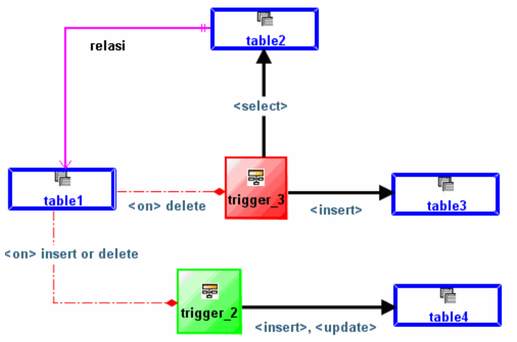


Figure 5. Design of trigger visualization diagram

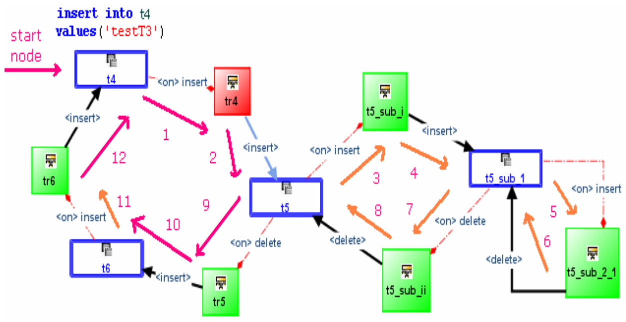
A red vertex (trigger_3) is row trigger and a green vertex (trigger_2) is statement trigger. Trigger action is symbolized by edge/arrow which comes from a trigger to a table which is affected an action trigger.

5. The Causes of Mutating Table

The cause of mutating table in Oracle is defined into two causes. The first one is a mutating table caused by chain reaction between triggers, while the latter is a mutating table caused by chain reaction between relations of cascade delete. The details of them are in the following.

A. Mutating Table Case I (Chain Reaction Trigger-Trigger)

The occurrence of such kind is caused by chain reaction, i.e., it comes up from events and actions in the series of triggers without the involving of cascade delete constraints. It means that this mutating table case can be occurred in a table without relation.



```
insert into t4
values('testT3')
```

Figure 6. Mutating table caused by trigger chain reaction

B. Mutating Table Case II (Cascade Delete – Constraint)

The occurrence of this kind is caused by events and actions which are added by the role of cascade delete constraints.

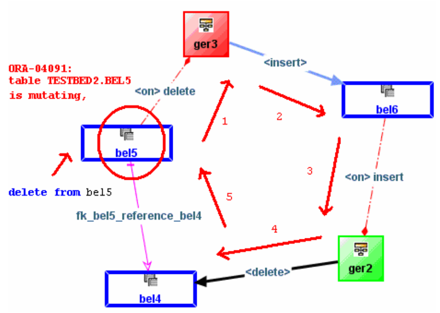


Figure 7. An example of mutating table Case II

From the two examples above, *mutating table* cases are occurred if and only if the first vertices trigger is joined in the *path* of the *circuit* which is *trigger row level.* Otherwise, a recursion is possible to be occurred.

6. Proposed Method

Here, we proposed two mechanisms to overcome the mutating table problem as described earlier, in the following:

1. Chain Reaction Verification Algorithm

In order to make a path particularly produces a mutating table case, every *Action trigger* must be appropriate with the required *event* to trigger the next trigger. Otherwise, the chain reaction will not be occurred in that circular path. Without chain reaction, the mutating table will also not be occurred. To verify this problem, a *Chain Reaction Verificator* (*CRV*) algorithm is used. The main idea of this algorithm is comparing the previous *Action trigger* with the next *event trigger*.

2. *Vertex to Circuit Expansion* (*VtCX*) Algorithm

A *path* which is caused a *mutating table* is called a *circular path*. It is the *path* which has the same start and destination *vertices.* The common method for finding a circular path is using the *Brute Force* algorithm based on the recursion processing and seeking using Depth First Search (DFS). This method is not really effective because the algorithms produce all circular paths, whereas the required paths are only the single/multi circuit circular paths [Kam03].

To maximize the process of seeking paths, *Brute Force* algorithm should be optimized. The optimization is based on the expansion principal from *vertices* to *circuit*. This expansion is done by substituting a vertex in path of one *single circuit* with another *single circuit*. This substitution can be occurred once or more depend on requirement. The new algorithm refers to *Vertex to Circuit Expansion* (*VtCX*) algorithm.

The examples of the using of *VtCX* and *CRV* algorithm to detect *mutating table* are shown bellow:

(1) Find all *circular paths* which are considered as *single circuit* in *tbl_c* table. The result are *circuit* 1→2→3→4→11→12 and *circuit* 13→14. Then, *circuit* 13→14 is discarded because it is impossible to create *mutating table* on *tbl_c* table.

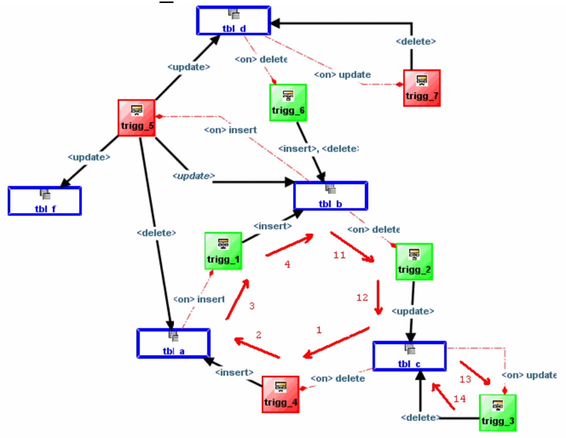


Figure 8. Two single circuits belong to *tbl_c*

(2) *CRV* algorithm, here, finds out a failure of *chain reaction.* It works in *tbl_b* table (showed by red ellipses, *Action insert* can not execute *trigger* with *event delete*).

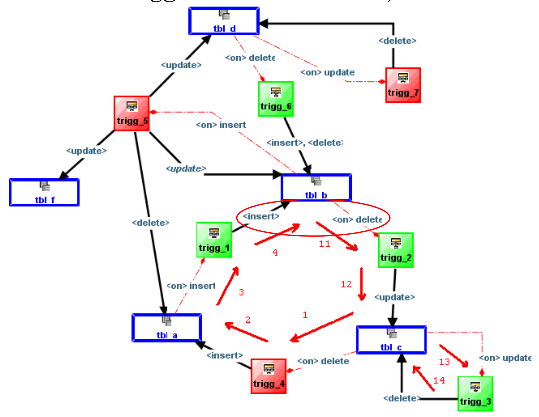
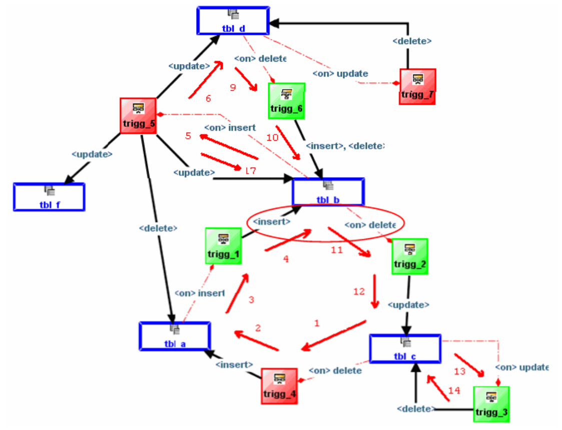


Figure 9. *tbl_b* is failed to continue *chain reaction*

(3) We knew that *tbl_b* table is failed to continue *chain reaction*, then *tbl_b* is being expanded to produce a new *single circuit* which has a start *vertice* on *tbl_b* table. The valid *single*

circuits belong to *tbl_b* are: 5→17 and 5→6→9→10.



Picture 10. Two *single circuits* as the expansion result belong to *tbl_b*

(4) Check the expansion result *circuit* (*single circuit* 5→17 and 5→6→9→10) using *CRV* algorithm. It 's clear that *circuit* 5→17 is failed (*Action update* on *edge* 17 is failed to *trigger edge* 11 by *event delete*). This circuit is no longer to be expanded anymore. Apparently, *circuit* 5→6→9→10 is also failed on *tbl_d* table. In turn, *Action update* of *trigg_5* don't proceed the *event delete* of *trigg_6.*

(5) Because *tbl_d* is still possible to be expanded, then it will be expanded. *Single circuit* (i.e., the result of that expansion) is *single circuit* 8→7

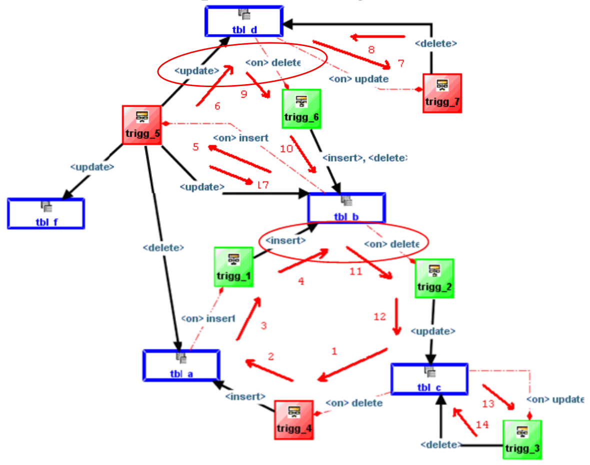


Figure 11. A *single circuit*, i.e., the result of the expansion of *tbl_d*

(6) In turn, the result of the expansion proceed the *event delete* of *trigg_6*, while previously, *event delete* can't be accessed directly by *Action update* of *trigg_5.*

(7) For the rest, all of *Action Events* are succeed to reach the start table, thus, *tbl_c* table. Here, *tbl_c* is *mutating*.

(8) The *mutating route* is 1→2→3→4→ (5→6→ (7→8→) 9→) 10→11→12→13→14. The path written within brackets means the result of the expansion.

If the vertices of mutating route are illustrated in tree, it will be shown as the fig. 12 bellow:

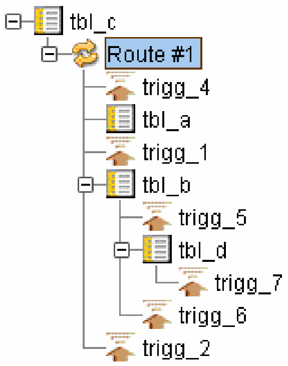

```
tbl_c
  Route #1
    trigg_4
    tbl_a
    trigg_1
    tbl_b
      trigg_5
      tbl_d
        trigg_7
      trigg_6
    trigg_2
```

Figure 12. A tree of *mutating route*

From the tree above, it shown that the expansion processing occur in vertex of *tbl_b* and vertixe of *tbl_d*

7. Test and Evaluation

The first test is based on the structure of **Trigger Dependency Diagram** as shown in fig. 6. The result of detection using *CRV* and *VtCX algorithm* shows that *t4 table* will be *mutating* if it's affected by an *insert* operation.

The following is a tree which is figuring the route:

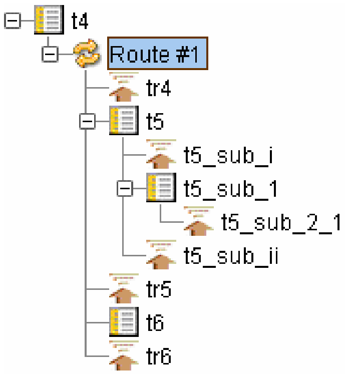

```
t4
  Route #1
    tr4
    t5
      t5_sub_i
      t5_sub_1
        t5_sub_2_1
      t5_sub_ii
    tr5
    t6
    tr6
```

Figure 13. A *tree* of *Mutating Route*

This detection is proved by running *DML insert* on *t4 table.* The reason why we using *insert* is because *insert* is the first *Event* which causes *chain reaction* in that structure. The error message from such circumstance is:

```
ORA-04091: table TESTBED.T4 is mutating,
trigger/function may not see it
```

Proving the mutating table detection

The second test is running on the structure of academic information system (Academic IS) database. The following is *PDM* Diagram of IS that is made in *Power Designer 9.0*.

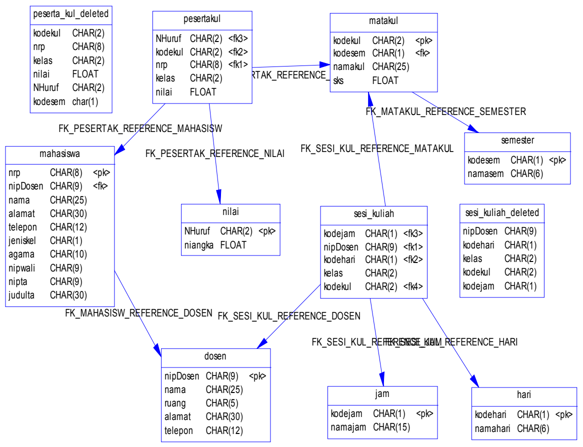


Figure 14. *PDM* Diagram of Academic IS

All relations in database structure above are *cascade delete* relations, except a relation between table of *nilai* with table of *pesertakul,* and a relation between table of *dosen* with table of *mahasiwa*

The trigger used in this test is *hist_trigg*. This *trigger* is made for writing the history of data deleted from *pesertakul* table, into *peserta_kul_deleted* table. Table of *peserta_kul_deleted* has the same column with *pesertakul* table, plus one extra column to write what semester which the record in *pesertakul* table come from. This extra column is filled by *hist_trigg trigger,* by running query on *matakul* table and *semester* table. Source code of the *trigger* is:

```
CREATE OR REPLACE TRIGGER
"TESTBED3"."hist_trigg"
  after delete on pesertakul
  for each row
declare
    recordDeleted pesertakul%rowtype;
    kodesem varchar(1);
begin
    select semester.kodesem
    into kodesem
    from matakul, semester
where :old.kodekul = matakul.kodekul and
matakul.kodesem = semester.kodesem;

    insert into
peserta_kul_deleted(kodekul,nrp,kelas,nilai
,nhuruf,kodesem)
values(:old.kodekul,:old.nrp,:old.kelas,:ol
d.nilai,:old.nhuruf,kodesem);

end hist_trigg;
```

The Trigger Dependency Diagram is shown bellow:

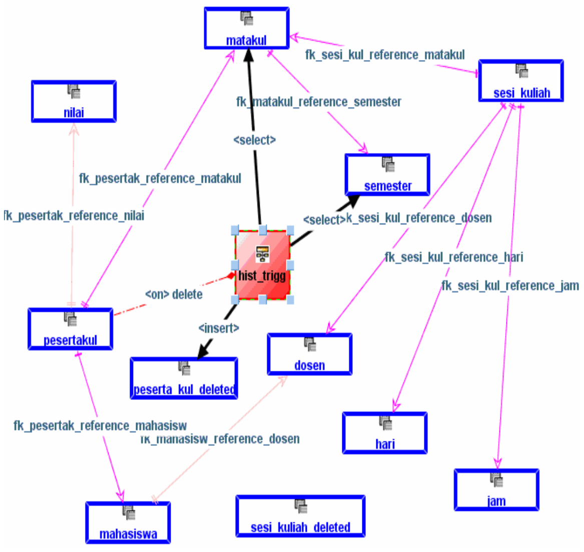

matakul
fk_sesi_kul_reference_matakul
sesi_kuliah
nilai
fk_matakul_reference_semester
<select>
fk_pesertak_reference_matakul
semester
fk_pesertak_reference_nilai
<select>
fk_sesi_kul_reference_dosen
hist_trigg
fk_sesi_kul_reference_hari
fk_sesi_kul_reference_jam
<on> delete
<insert>
pesertakul
dosen
peserta_kul_deleted
hari
fk_pesertak_reference_mahasisw
fk_mahasisw_reference_dosen
jam
mahasiswa
sesi_kuliah_deleted

Figure 15. *Trigger Dependency Diagram* of academic IS

The detection result shows that *matakul* and *semester* table is *mutating.* The route is shown in the fig. 16 and fig. 17 as following:

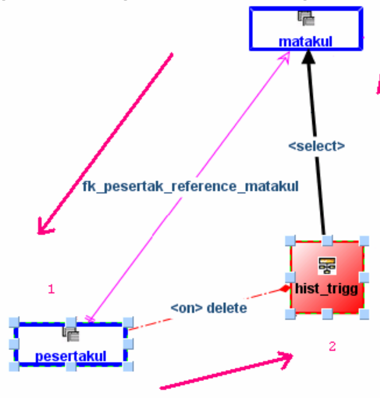


Figure 16. *MutatingRoute* of *Matakul* table

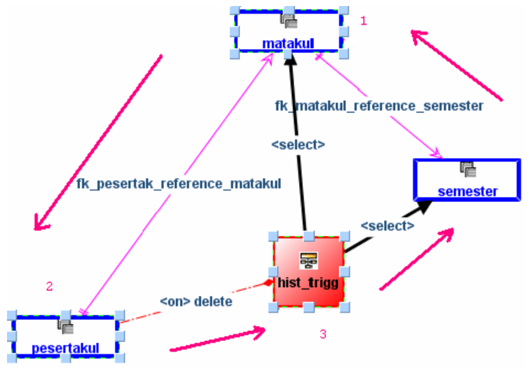


Figure 17. *Mutating Route* of *Semester* table

Mutating detection of *matakul* table will be proved by running *DML delete* on that table. The message obtained so far, is shown bellow:

```
ORA-04091:   table   TESTBED3.MATAKUL   is
mutating, trigger/function may not see it
```

Thus, a mutating table detection of *matakul* table is proved.

The next process is to prove the second detection, i.e., a mutating table of *semester* table. Just like the previous mutating table, statement of *delete* is used to make chain reaction in this table. The error message is shown as bellow:

```
ORA-04091:   table   TESTBED3.MATAKUL   is
mutating, trigger/function may not see it
```

Apparently this message is not an error message which is related to the mutating of *semester* table, but it is related to the mutating of *matakul* table. It's because the mutating of *semester* table occurred after the mutating of matakul table.

By reading the source code from the trigger body, it is known that *matakul* table is accessed before *semester* table. The trigger execution is stopped at *matakul* table, so that the mutating of *semester* table doesn't occur yet.

.

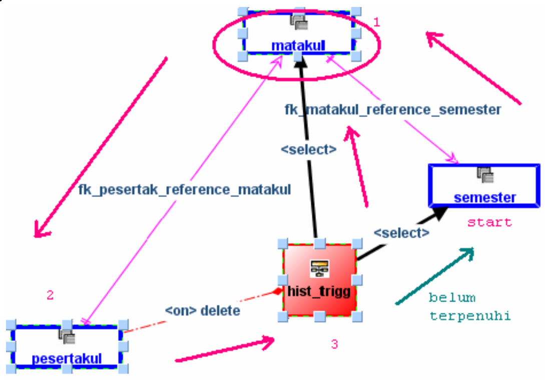


Figure 18. Action Select isn't being execution yet

It doesn't mean that the detection is wrong but this proves that the detection is only a warning. This means that the detection becomes valid if only if the conditions are fulfilled. In this case, not all of the conditions are fulfilled, so that the detection becomes invalid.

8. Conclusion

We now conclude our research as following:

- Instead of using the PDM diagram, the using of Trigger Dependency Diagram can make Database Administrators and developers easier to analyze and arrange triggers and relations into a database structure, especially to understand the logical rule of triggers in a database.
- The using of Trigger Dependency Diagram can make the detection mutating table becomes easier, better than reading the source code directly.

9. References

[Cor01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Introduction to Algorithms*, Second Edition, MIT Press and McGraw-Hill, 2001, ISBN 0262032937, Section 22.3: Depth-first search, pp.540–549, http://en.wikipedia.org/wiki/Depth_first_search

[Har69] F. Harary, *Graph Theory*, Addison-Wesley Publishing, 1969.

[Kam03] Amir Kamil, *Graph Algorithms*, UC Berkeley, California, 2003, www.cs.berkeley.edu/~kamil/sp03/041403.pdf

[Rac05] Ruli Dyah Rachmawati, *Case Tool Pembangkit Script Trigger Untuk DBMS Oracle*, Teknik Informatika-ITS, Surabaya, 2005.

[Rod05] Jean-Paul Rodrigue, *Graph Theory: Definition and Properties*, Hofstra University, 2005, http://people.hofstra.edu/geotrans/eng/ch2en/meth2en/ch2m1en.html

[Wik06] wikipedia.org, *Chain Reaction*, wikipedia.org, 2006, http://en.wikipedia.org/wiki/Chain_reaction