Database Management Systems

Oracle Triggers

D^B^G M

Trigger syntax

```
CREATE TRIGGER TriggerName
Mode Event {OR Event }
ON TargetTable
[[ REFERENCING ReferenceName]
FOR EACH ROW
[WHEN Predicate]]
PL/SQL Block
```

- The action is
  - a sequence of SQL instructions
  - a PL/SQL block
- *No* transactional and DDL instructions

Execution algorithm

1. Before statement triggers are executed
2. For each tuple in *TargetTable* affected by the triggering statement
   a) Before row triggers are executed
   b) The triggering statement is executed
      + integrity constraints are checked on tuples
   c) After row triggers are executed
3. Integrity constraints on tables are checked
4. After statement triggers are executed

Trigger semantics

- The execution order for triggers with the same event, mode and granularity is not specified
  - it is a source of non determinism
- When an error occurs
  - rollback of all operations performed by the triggers
  - rollback of the triggering statement in the triggering transaction

Non termination

- Trigger execution may activate other triggers
  - Cascaded trigger activation may lead to non termination of trigger execution
- A maximum length for the cascading trigger execution may be set
  - default = 32 triggers
- If the maximum is exceeded
  - an execution error is returned

Mutating tables

- A *mutating table* is the table modified by the statement (i.e., event) triggering the trigger
- The mutating table
  - *cannot* be accessed in row level triggers
  - may *only* be accessed in statement triggers
- Limited access on mutating tables only characterizes Oracle applications
  - accessing mutating tables is *always* allowed in SQL3

Database Management Systems

Guidelines in writing triggers in Oracle

Guidelines in writing triggers in Oracle

- Execution Mode INSTEAD OF is allowed in Oracle but it should be avoided
- Usage of before triggers in Oracle to be compliant with the standard
  - Modifications of the NEW variable in tuples affected by the triggering statement are allowed in before triggers
  - Other databases modifications apart those reported in the previous point are not allowed on before triggers
  - Before triggers cannot trigger other triggers

Trigger Design

Trigger design

- The design of a single trigger is usually simple
  - Identify
    - execution semantics
    - event
    - condition (optional)
    - action

Trigger design

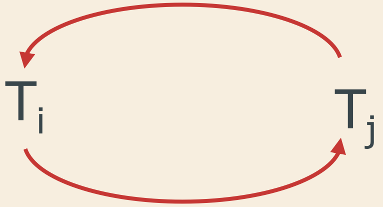
- Understanding *mutual* interactions among triggers is more complex
  - The action of one trigger may be the event of a different trigger
    - Cascaded execution
- If mutual triggering occurs
  - Infinite execution is possible

Trigger execution properties

- Termination
  - For an arbitrary database state and user transaction, trigger execution *terminates* in a final state (also after an abort)
- Confluence
  - For an arbitrary database state and user transaction, trigger execution *terminates in a unique final state*, independently of the execution order of triggers
- Termination is the most important property
- Confluence is enforced by deterministic trigger execution

Guaranteeing termination

- Termination is guaranteed at run time by aborting trigger execution after a given cascading length
- Termination may be verified at design time by means of the triggering graph
  - a node for each trigger
  - a directed edge $T_i \rightarrow T_j$ if trigger T_i is performing an action triggering trigger T_j
- A cycle in the graph shows potential non terminating executions



Example

- Trigger managing salary amounts
  - When a given average salary value is exceeded, a salary reduction is automatically enforced
- The following table is given

  Employee (<u>Emp#</u>, Ename, …, Salary)
- Event
  - Update of the Salary attribute in Employee
  - Insert into Employee
    - Will write only trigger for update

Example

- Trigger managing salary amounts
  - When a given average salary value is exceeded, a salary reduction is automatically enforced
- The following table is given

  Employee (Emp#, Ename, …, Salary)
- Execution semantics
  - After the modification events
  - Separate execution for each update instruction
- No condition for execution

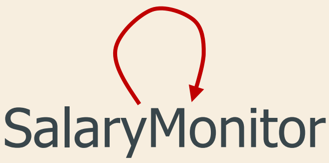
Example

```
CREATE TRIGGER SalaryMonitor
AFTER UPDATE OF Salary ON Employee
FOR EACH STATEMENT
BEGIN
  update Employee
  set Salary = Salary * K
  where 2500 < (select AVG (Salary) from Employee);
END;
```

The value of K may be

K = 0.9 ⟶ execution terminates

K = 1.1 ⟶ infinite execution



Trigger applications

- Internal applications
  - maintenance of complex integrity constraints
  - replication management
  - materialized view maintenance
- Business Rules
  - Incorporate into the DBMS application knowledge
    - E.g., reorder rule
- Alerters
  - widely used for notification

Triggers for constraint management

- Triggers are exploited to enforce complex integrity constraints
- Design procedure
  1. Write the constraint as a SQL predicate
     - It provides a condition for the trigger execution
  2. Identify the events which may violate the constraint
     - i.e. the condition
  3. Define the constraint management technique in the action

Design example (1)

- The following tables are given
  - Supplier S (S#, SName, …)
  - Part P (P#, PName, …)
  - Supply SP (S#, P#, Qty)
- Constraint to be enforced
  - A part may be supplied by at most 10 different suppliers

Design example (1)

- Constraint predicate

```
select P#
from SP
group by P#
having count(*) > 10
```

  - set of parts violating the constraint
- Events
  - insert on SP
  - update of P# on SP
- Action
  - reject the violating transaction

Design example (1)

- Execution semantics
  - *after* the modification
  - *statement level*
    - to capture the effect of the entire modification
    - (Oracle) to allow access to the mutating table
- (Oracle) No condition
  - The condition cannot be specified in the WHEN clause
  - It is checked in the trigger body
- Design for Oracle trigger semantics

Design example (1)

```
CREATE TRIGGER TooManySuppliers
AFTER UPDATE OF P# OR INSERT ON SP
DECLARE
 N number;
BEGIN
 select count(*) into N
 from SP
 where P# IN (select P# from SP
                   group by P#
                   having count(*) > 10);
 if (N <> 0) then
   raise_application_error (xxx, ‘constraint violated’);
 end if;
END;
```

Design example (2)

- The following tables are given
  - Supplier S (S#, SName, …)
  - Part P (P#, PName, …)
  - Supply SP (S#, P#, Qty)
- Constraint to be enforced
  - The quantity of a product supply cannot be larger than 1000. If it is larger, trim it to 1000.
- Check constraints do not allow compensating actions
  - Implement with a trigger

Design example (2)

- Constraint predicate
  - Qty > 1000
  - It is also the trigger condition
- Events
  - insert on SP
  - update of Qty on SP
- Action
  - Qty = 1000

Design example (2)

- Execution semantics
  - *before* the modification takes place
    - its effect can be changed before the constraint is checked
  - *row level*
    - each tuple is modified separately

Design example (2)

```
CREATE TRIGGER ExcessiveQty
BEFORE UPDATE OF Qty OR INSERT ON SP
FOR EACH ROW
WHEN (NEW.Qty > 1000)
BEGIN
 :NEW.Qty := 1000;
END;
```