Triggers and Active Databases
- Lab 6 -

Original slides by Lewis, Bernstein and Kifer.
Trigger Overview

- Element of the database schema
- General form:
  \[
  \text{ON } \text{<event>} \text{ IF } \text{<condition>} \text{ THEN } \text{<action>}
  \]
  - event: request to execute database operation
  - condition: predicate evaluated on database state
  - action: execution of procedure that might involve database updates
- Example:
  \[
  \text{ON updating maximum\_course\_enrollment} \\
  \text{IF number\_registered} \ > \ \text{new\_max\_enrollment\_limit} \\
  \text{THEN deregister students using LIFO policy}
  \]
Trigger Details

- **Activation** - occurrence of the event
- **Consideration** - the point, after activation, when condition is evaluated
  - Immediate or deferred (when the transaction requests to commit)
  - Condition might refer to both the state before and the state after event occurs
Trigger Details

- **Execution** - point at which action occurs
  - With deferred consideration, execution is also deferred
  - With immediate consideration, execution can occur immediately after consideration or it can be deferred
    - If execution is immediate, execution can occur before, after, or instead of triggering event
    - Before triggers adapt naturally to maintaining integrity constraints: violation results in rejection of event
Granularity

- Row-level granularity: change of a single row is an event (a single UPDATE statement might result in multiple events)
- Statement-level granularity: events are statements (a single UPDATE statement that changes multiple rows is a single event)
Trigger Details

- **Multiple Triggers**
  - How should multiple triggers activated by a single event be handled?
    - Evaluate one condition at a time and if true immediately execute action or
    - Evaluate all conditions, then execute actions
  - The execution of an action can affect the truth of a subsequently evaluated condition so the choice is significant
Triggers in SQL:1999

- **Events:** INSERT, DELETE, or UPDATE statements or changes to individual rows caused by these statements

- **Condition:** anything allowed in a WHERE clause

- **Action:** an individual SQL statement or a program written in the language of Procedural Stored Modules (PSM) (which can contain embedded SQL statements)
Triggers in SQL:1999

- **Consideration:** Immediate
  - Condition can refer to both the state of the affected row or table before and after the event occurs

- **Execution:** Immediate - can be before or after the execution of triggering event
  - Action of before trigger cannot modify the database

- **Granularity:** both row-level and statement-level granularity
Trigger Syntax

CREATE TRIGGER trigger_name
  { BEFORE | AFTER }
  { INSERT | DELETE | UPDATE } [ OF columns ]
ON table
  [ REFERENCING
    [ OLD AS var_to_old_row ]
    [ NEW AS var_to_new_row ]
    [ OLD TABLE AS var_to_old_table ]
    [ NEW TABLE AS var_to_new_table ] ]
  [ FOR EACH { ROW | STATEMENT } ]
  [ WHEN (condition) ]
  statements
Before Trigger Example (row granularity)

CREATE TRIGGER Max_EnrollCheck

BEFORE INSERT ON Transcript

REFERENCING NEW AS N -- row to be added

FOR EACH ROW

WHEN

((SELECT COUNT (T.StudId) FROM Transcript T
  WHERE T.CrsCode = N.CrsCode
  AND T.Semester = N.Semester)

>=

(SELECT C.MaxEnroll FROM Course C
  WHERE C.CrsCode = N.CrsCode ))

ROLLBACK
After Trigger Example (row granularity)

CREATE TRIGGER LimitSalaryRaise  
AFTER UPDATE OF Salary ON Employee  
REFERENCING OLD AS O  
NEW AS N  
FOR EACH ROW  
WHEN (N.Salary - O.Salary > 0.05 * O.Salary)  
UPDATE Employee -- action  
SET Salary = 1.05 * O.Salary  
WHERE Id = O.Id

Note: The action itself is a triggering event (but in this case a chain reaction is not possible – Why?)
After Trigger Example (statement granularity)

CREATE TRIGGER RecordNewAverage
AFTER UPDATE OF Salary ON Employee
FOR EACH STATEMENT
  INSERT INTO Log
  VALUES (CURRENT_DATE,
          SELECT AVG (Salary)
                 FROM Employee)
create or replace trigger check_salary_SALGRADE
before update or delete on SALGRADE
for each row
    when (new.MINSAL > old.MINSAL or new.MAXSAL < old.MAXSAL)
    -- only restricting a salary range can cause a constraint violation
    declare
        job_emps number (3):=0;
    begin
        if deleting then
            -- Does there still exist an employee having the deleted job ?
            select count ( *)into job_emps from EMP
            where JOB =:old.JOB;
            if job_emps !=0 then
                raise_application_error (-20240,'There still exist employees');
            end if ;
        end if ;
        if updating then
            -- Are there employees whose salary does not lie within the modified salary range ?
            select count ( *)into job_emps from EMP
            where JOB =:new.JOB and SAL not between :new.MINSAL and :new.MAXSAL;
            if job_emps !=0 then -- restore old salary ranges
                :new.MINSAL :=:old.MINSAL;
                :new.MAXSAL :=:old.MAXSAL;
            end if ;
        end if ;
    end ;

*http://sirius.cs.ucdavis.edu/teaching/sqltutorial/

Lab 6
Triggers
Database Management Systems
CMPUT 391
Sample trigger* (Oracle’s syntax)

create or replace trigger check_salary_SALGRADE
before update or delete on SALGRADE
for each row
  when (new.MINSAL > old.MINSAL or
       new.MAXSAL < old.MAXSAL
-- only restricting a salary range can cause
-- a constraint violation
declare
  job_emps number (3):=0;
...

*http://sirius.cs.ucdavis.edu/teaching/sqltutorial/
Sample trigger* (Oracle’s syntax)

... begin
  if deleting then  -- Does there still exist an employee having the deleted job?
    select count(*) into job_emps from EMP
    where JOB =:old.JOB;
    if job_emps !=0 then
      raise_application_error (-20240,’There still exist employees’);
    end if ;
  end if ;
  if updating then  -- employees whose salary not within the modified salary range?
    select count(*) into job_emps from EMP
    where JOB =:new.JOB and
      SAL not between :new.MINSAL and :new.MAXSAL;
    if job_emps !=0 then  -- restore old salary ranges
      :new.MINSAL :=:old.MINSAL;
      :new.MAXSAL :=:old.MAXSAL;
    end if ;
  end if ;
end ;

*http://sirius.cs.ucdavis.edu/teaching/sqltutorial/
What does this one* do? (Try it)

CREATE TABLE T4 (a INTEGER, b CHAR(10));
CREATE TABLE T5 (c CHAR(10), d INTEGER);

CREATE TRIGGER trig1
AFTER INSERT ON T4
REFERENCING NEW AS newRow
FOR EACH ROW
    WHEN (newRow.a <= 10)
    BEGIN
        INSERT INTO T5 VALUES(:newRow.b, :newRow.a);
    END trig1;
.
run;

-- REMOVE THE TABLES AND DROP THE TRIGGER ONCE YOU’RE DONE !!

*http://www-db.stanford.edu/~ullman/fcdb/oracle/or-triggers.html
To keep in mind …

- A trigger definition cannot be changed, it can only be re-created using the replace clause.
- The command ‘drop trigger <trigger_name>’ deletes a trigger.
- After a trigger definition has been compiled, it is automatically enabled. The command ‘alter trigger <trigger_name> disable | enable’ is used to deactivate/activate a trigger.
To keep in mind …

- If a trigger is specified within the SQL*Plus shell, the definition **must end** with a point “.” in the last line. Issuing the command ‘run’ causes SQL*Plus to compile this trigger definition.

- Trigger syntax in Oracle differs from the syntax in standard SQL!

- **Triggers can be quite tricky!** More often than not they are not really needed (e.g., many constraints can be enforced at create table time).
A “tricky” example

Consider these two simple tables:

CREATE TABLE T1 (  
    sid INTEGER, -- student id  
    cid CHAR(10), --course id  
    gr INTEGER, -- grade student sid obtained when taking course cid  
    PRIMARY KEY (sid,cid)  
);

CREATE TABLE T2 (  
    cid CHAR(10), -- course cid  
    avggr NUMBER, -- average of grade over all students who took course cid  
    PRIMARY KEY (cid)  
);
Any problem here?

CREATE OR REPLACE TRIGGER trigger_lab2
AFTER INSERT OR DELETE OR UPDATE ON T1
FOR EACH ROW
BEGIN
    UPDATE T2
    SET avggr = (SELECT AVG(gr)
                  FROM T1
                  WHERE cid=:new.cid)
    WHERE cid = :new.cid;
END trigger_lab2;

MUTATING TABLE ISSUE!
The solution is wrong …

➢ One would be much better off simply creating a VIEW based on T1!
Trigger Resources

Quite nice “straight to the point” tutorials can be found at:

- http://www-db.stanford.edu/~ullman/fcdb/oracle/or-triggers.html
- http://sirius.cs.ucdavis.edu/teaching/sqltutorial
Exercise

- Create 2 tables:
  - \( T1(sID, cID, gr) \) and \( T2(cID, sGr, nGr) \)

- Create 3 triggers:
  - When a tuple is inserted in \( T1 \)
    - if the grade is not between 1 and 9, reject the insert
    - if the courseID is not in \( T2 \), add it to \( T2 \)
  - When a tuple is inserted/updated/deleted in \( T1 \), update
    the sum (\( sGr \)) and number of grades (\( nGr \)) in \( T2 \)

- If the inserted tuple is rejected due to grade, what happens to courseID? Could you make the same triggers before/after, row/statement level? If yes, which choice is better? If not, why? Could you create less than 3 triggers (with the same effect)?