COSC344
Database Theory and Applications

Lecture 11 Triggers

Learning Objectives of This Lecture

- You should
  - understand the difference between a trigger and a PL/SQL program
  - understand what triggers can be used for
  - understand how a trigger works
  - distinguish between row-level and statement-level triggers
  - be able to use triggers to maintain the values for derived attributes
  - understand mutating table and constraining table
- Source
  - Lecture note,
  - Oracle documentation

What Is a Trigger

A trigger is a PL/SQL **stored subprogram** associated with a table, and is **automatically invoked** by the DBMS in response to specified changes to the database.

- Triggers are commonly used to
  - Enforcement of complex business rules
    - e.g. whenever a sales transaction is greater than $50,000, the salesperson must be personally congratulated.
  - Enforcement of some types of referential integrity
    - e.g. Oracle does not support ON UPDATE CASCADE
  - Auditing purpose (creating audit log)
    - Who did what to my table? when?
  - Automatic generation the values for derived attributes
  - Creation of replica tables and backup files

How does a trigger work

- Follows the *Event-Condition-Action* model
  - *triggering event*: the statement that causes the trigger to execute
    - Triggering statement: INSERT, UPDATE, or DELETE
    - Triggering timing: when the trigger is fired
      - BEFORE - Fire before the triggering SQL statement is executed
      - AFTER - Fire after the triggering SQL statement is executed
  - *triggering condition: determines whether the action should be executed*
    - Condition is optional
  - *action: a block of PL/SQL statements to be executed*

How to define a Trigger

Trigger Syntax

```
CREATE [OR REPLACE] TRIGGER name
  {BEFORE | AFTER | INSTEAD OF}
  {DELETE
   | INSERT
   | UPDATE [OF column [,column] ...]}
   [OR
  {DELETE
   | INSERT
   | UPDATE [OF column [,column]
              ...]}] ...
  ON {TABLE | VIEW} tablename
  [ [REFERENCING {OLD [AS] old
                 |NEW [AS] new} ...]
  FOR EACH {ROW | STATEMENT}
  [WHEN (condition)] ]
  PL/SQL block
```

Types of Triggers

- Row-level triggers
  - Execute once for each row affected by the triggering event

- Statement-level triggers
  - Execute only once even multiple rows are affected by the triggering event.

Correlation Values - NEW and OLD

- OLD.<attribute name>
  - The value of the attribute before a change from an UPDATE statement or before a DELETE statement. This value is NULL for INSERT statements.
- NEW.<attribute name>
  - The value of the attribute after an UPDATE statement or after an INSERT statement. This value is NULL for DELETE statements.
- Can be aliased
  - NEW AS newname
  - OLD AS oldname
- In the trigger body, NEW and OLD must be preceded by a colon (":"), but in the WHEN clause, they do not have a preceding colon!

How to define a Trigger

Trigger Syntax

```
CREATE [OR REPLACE] TRIGGER name
  {BEFORE | AFTER | INSTEAD OF}
  {DELETE
   | INSERT
   | UPDATE [OF column [,column] ...]}
   [OR
  {DELETE
   | INSERT
   | UPDATE [OF column [,column]
          ...]}] ...
  ON {TABLE | VIEW} tablename
  [ [REFERENCING {OLD [AS] old
                 |NEW [AS] new} ...]
  FOR EACH {ROW | STATEMENT}
  [WHEN (condition)] ]
  PL/SQL block
```

Row-level Trigger Example

Deleted customer records must be moved to a customer history table.

```
CREATE OR REPLACE TRIGGER after_delete_customer
AFTER DELETE ON customers
FOR EACH ROW
BEGIN
    INSERT INTO cust_history(cno, cname, address)
    VALUES (:old.cnum,:old.cname,:old.city);
END;
/
```

Statement-level Trigger Example

Customer tables can only be modified between 8am to 6pm.

```
CREATE OR REPLACE TRIGGER modify_customer
BEFORE DELETE OR UPDATE OR INSERT ON customers
BEGIN
  If to_char(sysdate,’hh24’)<’08’ OR
     to_char(sysdate, ’hh24’)>’18’  THEN
  RAISE_APPLICATION_ERROR (-20001, ’Data can not be
       modified at this time’);
  END IF;
END;
/
```

Derived Value Trigger Example

The department relation has a derived attribute `total_salary` to store the total salary paid to the employees in each department.

```
CREATE OR REPLACE TRIGGER modify_salary
AFTER UPDATE OF salary ON employee
FOR EACH ROW
BEGIN
 UPDATE department
 SET total_salary = total_salary + :NEW.salary
                                 - :OLD.salary
 WHERE dnumber = :OLD.dno;
END;
/
```

What is wrong with this trigger definition?

Trigger Attributes

- Three Boolean trigger attributes that allow us to determine what DML activity has caused the trigger to execute
  - INSERTING
    - True if the trigger is fired due to an INSERT operation
  - UPDATING
    - True if the trigger is fired due to an UPDATE operation
  - DELETING
    - True if the trigger is fired due to an DELETE operation

- Trigger attributes can be used in both row and statement triggers

Derived Value Trigger Example (revisit)

The department relation has a derived attribute `total_salary` to store the total salary paid to the employees in each department.

```
CREATE OR REPLACE TRIGGER modify_salary
AFTER INSERT OR UPDATE OR DELETE OF salary ON employee
FOR EACH ROW
BEGIN
    IF INSERTING THEN
      UPDATE department
      SET total_salary = total_salary + :NEW.salary
      WHERE dnumber = :NEW.dno;
    ELSIF UPDATING THEN
      UPDATE department
      SET total_salary = total_salary + :NEW.salary - :OLD.salary
      WHERE dnumber = :OLD.dno;
    ELSE -- deleting
      UPDATE department
      SET total_salary = total_salary - :OLD.salary
      WHERE dnumber = :OLD.dno;
    END IF;
END;
/
```

Triggers Can Call Procedures

```
CREATE OR REPLACE PROCEDURE sumit
AS
    sal_sum employee.salary%TYPE;
BEGIN
    SELECT SUM(salary)
    INTO sal_sum
    FROM employee;
    UPDATE department
    SET total_salary = sal_sum;
END;
/

CREATE OR REPLACE TRIGGER modify_salary
AFTER UPDATE OF salary ON employee
BEGIN
  sumit();
END;
/
```

Exceptions

```
CREATE OR REPLACE TRIGGER modify_salary
BEFORE UPDATE OF salary ON employee
FOR EACH ROW
DECLARE
  too_much EXCEPTION;
BEGIN
  IF :NEW.salary>99000 THEN
    RAISE too_much;
  END IF;
EXCEPTION
  WHEN too_much THEN
    RAISE_APPLICATION_ERROR (-20001,
      'Cannot pay that much');
END;
/
```

Oracle’s Execution Model

- Execute all BEFORE statement triggers that apply to the SQL statement.
- Loop for each row affected by the SQL statement
  - Execute all BEFORE row triggers that apply.
  - Change the row. Perform integrity constraint checking.
  - Execute all AFTER row triggers that apply.
- Complete integrity constraint checking.
- Execute all AFTER statement triggers that apply to the SQL statement.

Referential Integrity Example

- Oracle does not support
  - ON UPDATE CASCADE
  - ON UPDATE SET NULL
  - ON UPDATE SET DEFAULT

```
FOREIGN KEY  (dno)    REFERENCES    department(dnumber))
               ON DELETE  CASCADE     ON UPDATE   CASCADE,
```

✗

```
CREATE OR REPLACE TRIGGER update_dno
AFTER UPDATE OF dnumber ON department
FOR EACH ROW
BEGIN
   UPDATE employee
   SET dno = :new.dnumber
   WHERE dno =:old.dnumber;
END;
/
```

Will this work?

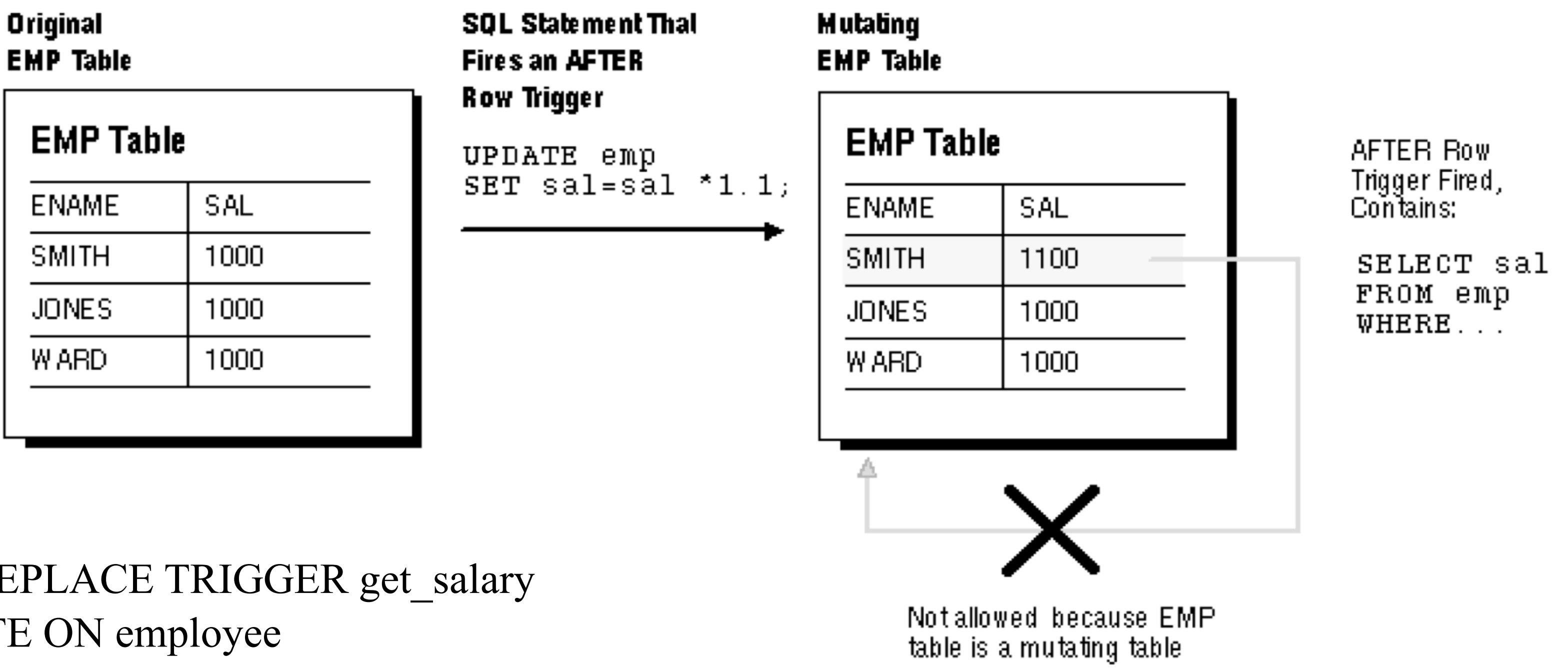
Mutating/Constraining Table Error (1)

- A mutating table is a table that is
  - currently being modified by an UPDATE, DELETE, or INSERT statement,
  - or a table that might need to be updated by the effects of a declarative DELETE CASCADE referential integrity constraint.

- Mutating error occurs when

  - the SQL statements of a trigger read from (query) or modify a mutating table of the triggering statement.

Mutating/Constraining Error (2)



```
CREATE OR REPLACE TRIGGER get_salary
AFTER UPDATE ON employee
FOR EACH ROW
DECLARE
  total_salary   NUMBER;
BEGIN
   SELECT sum(salary) INTO total_salary FROM employee
   WHERE dno = :old.dno;
   DBMS_OUTPUT.PUT_LINE(total_salary);
END;
/
```

Mutating/Constraining Table Error (4)

- An mutating table is a table that is
  - currently being modified by an UPDATE, DELETE, or INSERT statement,
  - or a table that might need to be updated by the effects of a declarative DELETE CASCADE referential integrity constraint.
- A constraining table is a table that a triggering statement might need to read
  - either directly, for a SQL statement,
  - or indirectly, for a declarative referential integrity constraint.
- Restriction on constraining table

  - The statements of a trigger cannot change the PRIMARY, FOREIGN, or UNIQUE KEY columns of a constraining table of the triggering

Mutating/Constraining Error (5)

```
create table P ( p1 number PRIMARY KEY);
create table F ( f1 number  references P (p1) on
delete cascade);
```

```
create trigger pf
after update on P  for each row
begin
 if (:new.p1 != :old.p1) then
        update f
           set f1 = :new.p1
        where f1 = :old.p1;
   end if;
end;  /
```

```
insert into p values (1);
insert into p values (2);
insert into p values (3);
insert into f values (1);
insert into f values (2);
insert into f values (3);
```

```
update p set p1 = p1+1;
SQL> select * from p;
         2
         3
         4
SQL> select * from f;
         4
         4
         4
```

Other Points

- Do not create recursive triggers
- SHOW ERRORS;
- SHOW ERRORS TRIGGER *name*;
- DROP TRIGGER *name*;
- ALTER TRIGGER *name* ENABLE;
- ALTER TRIGGER *name* DISABLE;
- ALTER TABLE *name* ENABLE ALL TRIGGERS;
- ALTER TABLE *name* DISABLE ALL TRIGGERS;