Part I

Active Databases
Chapter 2

Syntax and Semantics of Active Databases

2.1. Given the relational database schema

\[
\text{Employee}(\text{Name, Salary, Department})
\]

\[
\text{Department}(\text{Dept-No, Manager})
\]

define the following active rules in Starburst, Oracle, and DB2:

a. A rule that, whenever a department is deleted from the database, sets to null the value of the Department attribute for those tuples in relation Employee having the number of the deleted department.

b. A rule that, whenever a department is deleted from the database, deletes all employees in the deleted department.

c. A rule that, whenever the salary of an employee exceeds the salary of its manager, sets the salary of the employee to the salary of the manager.

d. A rule that, whenever salaries are updated, if the total of the updated salaries exceeds their total before the updates, then gives all the employees of the 'Research' department a 5% salary cut.

Complete this exercise by writing the same triggers in Chimera on the following object-oriented schema (which is equivalent to the previous relational schema):

\[
\text{create object class Employee}
\]

\[
\text{attributes Name: string,}
\]

\[
\text{Salary: integer,}
\]

\[
\text{Department: Dept}
\]

\[
\text{end;}
\]

\[
\text{create object class Dept}
\]

\[
\text{attributes Manager: Employee}
\]

\[
\text{end;}
\]

Answer:
2. SYNTAX AND SEMANTICS OF ACTIVE DATABASES

• In Starburst:
  a. **CREATE RULE** SetNull **ON** Department
     **WHEN DELETED**
     **THEN** UPDATE Employee
     SET Department = Null
     **WHERE** Department IN (SELECT DeptNo
     **FROM DELETED**)  
  b. **CREATE RULE** CascDel **ON** Department
     **WHEN DELETED**
     **THEN** DELETE FROM Employee
     **WHERE** Department IN (SELECT DeptNo
     **FROM DELETED**)  
  c. **CREATE RULE** CheckSal **ON** Employee
     **WHEN INSERTED, UPDATED** Salary
     **THEN** UPDATE Employee X
     SET Salary = (SELECT Salary
     **FROM** Employee, Department
     **WHERE** X.Department = DeptNo AND
     Name = Manager)
     **WHERE** Salary > (SELECT Salary
     **FROM** Employee, Department
     **WHERE** X.Department = DeptNo AND
     Name = Manager)  
  d. **CREATE RULE** CutSal **ON** Employee
     **WHEN UPDATED** Salary
     **IF** (SELECT SUM(Salary) **FROM** NewUpdated) >
     (SELECT SUM(Salary) **FROM** OldUpdated)
     **THEN** UPDATE Employee
     SET Salary = 0.95 * Salary
     **WHERE** Department = 'Research'

• In Oracle:
  a. **CREATE TRIGGER** SetNull
     **AFTER DELETE** **ON** Department
     **REFERENCING OLD AS** DelDept
     **FOR EACH ROW**
     UPDATE Employee
     SET Department = Null
     **WHERE** Department = DelDept.DeptNo  
  b. **CREATE TRIGGER** CascDel
     **AFTER DELETE** **ON** Department
     **REFERENCING OLD AS** DelDept
     **FOR EACH ROW**
     DELETE FROM Employee
     **WHERE** Department = DelDept.DeptNo
c. **CREATE TRIGGER** CheckSal
   **AFTER INSERT, UPDATE OF** Salary **ON** Employee
   **FOR EACH ROW**
   
   IF NEW.Salary > (SELECT Salary FROM Employee, Department
   WHERE NEW.Department = DeptNo AND
   Name = Manager)
   THEN
   UPDATE Employee
   SET Salary = (SELECT Salary FROM Employee, Department
   WHERE NEW.Department = DeptNo AND
   Name = Manager)
   WHERE Name = NEW.Name
   END;
   
   d. This trigger cannot easily be encoded into Oracle because the trigger language does not support statement-level references to the old state. In order to give the required semantics, we first collect all the old and new values into suitable tables by means of a row-level, after trigger; then we compare them by means of a statement-level, after trigger.

   **CREATE TABLE** OldTable
   (Salary: Integer);

   **CREATE TABLE** NewTable
   (Salary: Integer);

   **CREATE TRIGGER** CutSal1
   **AFTER UPDATE OF** Salary **ON** Employee
   **FOR EACH ROW**
   THEN
   INSERT INTO OldTable VALUES (OLD.Salary);
   INSERT INTO NewTable VALUES (NEW.Salary)
   END;

   **CREATE TRIGGER** CutSal2
   **AFTER UPDATE OF** Salary **ON** Employee
   IF (SELECT SUM(Salary) FROM NewTable) >
   (SELECT SUM(Salary) FROM OldTable)
   THEN
   UPDATE Employee
   SET Salary = 0.95 * Salary
   WHERE Department = 'Research'
   END;

   • In DB2:
   a. **CREATE TRIGGER** SetNull
      **AFTER DELETE ON** Department
      **REFERENCING OLD_TABLE AS** DelDept
2. SYNTAX AND SEMANTICS OF ACTIVE DATABASES

FOR EACH STATEMENT
UPDATE Employee
SET Department = Null
WHERE Department IN (SELECT DeptNo
                     FROM DelDept);

b. CREATE TRIGGER CascDel
   AFTER DELETE ON Department
   REFERENCING OLD_TABLE AS DelDept
   FOR EACH STATEMENT
   DELETE FROM Employee
   WHERE Department IN (SELECT DeptNo
                         FROM DelDept);

c. CREATE TRIGGER CheckSal1
   AFTER INSERT ON Employee
   REFERENCING NEW AS InsEmp
   FOR EACH ROW
   WHEN InsEmp.Salary > (SELECT Salary
                         FROM Employee. Department
                         WHERE InsEmp.Department = DeptNo
                         AND Name = Manager)
   UPDATE Employee X
   SET Salary = (SELECT Salary
                 FROM X. Department
                 WHERE InsEmp.Department = DeptNo
                 AND Name = Manager);

CREATE TRIGGER CheckSal2
AFTER UPDATE OF Salary ON Employee
REFERENCING NEW AS UpdEmp
FOR EACH ROW
WHEN UpdEmp.Salary > (SELECT Salary
                      FROM Employee. Department
                      WHERE UpdEmp.Department = DeptNo
                      AND Name = Manager)
UPDATE Employee X
SET Salary = (SELECT Salary
              FROM X. Department
              WHERE UpdEmp.Department = DeptNo
              AND Name = Manager);

d. CREATE TRIGGER CutSal
   AFTER UPDATE OF Salary ON Employee
   REFERENCING NEW_TABLE AS NEmp. OLD_TABLE AS OEmp
   FOR EACH STATEMENT
   WHEN (SELECT SUM(Salary) FROM NEmp) >
\[
\begin{align*}
&\text{SELECT SUM(\text{Salary}) FROM \text{OEmp)}} \\
&\text{UPDATE Employee} \\
&\text{SET Salary = 0.95 * Salary} \\
&\text{WHERE Department = 'Research'}
\end{align*}
\]

- In Chimera:
  a. define trigger SetNull for Dept
events delete
condition Employee(E). Dept(D). occurred(delete.D).
E.Department = D
action modify(Employee.Department,E.null)
end;
  b. define trigger CascDel for Dept
events delete
condition Employee(E). Dept(D). occurred(delete.D).
E.Department = D
action delete(Employee.E)
end;
  c. define trigger CheckSal for Employee
events create, modify(\text{Salary})
condition Employee(E). E.Salary > E.Department.Manager.Salary
action modify(Employee.Salary,E.E.Department.Manager.Salary)
end;
  d. define trigger CutSal for Employee
events modify(\text{Salary})
condition sum(X.Salary where Employee(X)) > 
\text{sum(old}(X.Salary) where Employee(X))
action select(E where Employee(E). E.department = 'Research').
\text{modify}(Employee.Salary,E.E.Salary*0.95)
end;

2.2. Referring to the relational schema above, define in Starburst or Chimera a deferred trigger \( R_1 \) that, whenever an employee who is a manager is deleted, also deletes all employees in the department managed by the deleted employee, along with the department itself.

Then define another deferred trigger \( R_2 \) that, whenever salaries are updated, checks the average of the updated salaries; if it exceeds 50,000, then it deletes all employees whose salary was updated and now exceeds 80,000.

Consider next a database state containing six employees: Jane, Mary, Bill, Jim, Sam, and Sue, with the following management structure:

- Jane manages Mary and Jim
- Mary manages Bill
- Jim manages Sam and Sue