CS240A Homework 3

Author: Sneha Shankar
UID: 404946026

Q3) 2.1) Given the relational database schema

Employee (Name, Salary, Department)
Department (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.

Starburst

```
CREATE RULE DelDept ON Department
WHEN DELETED
        THEN UPDATE Employee
        SET Department = NULL
        WHERE Department IN (SELECT Dept-No FROM DELETED)
```

Oracle

```
CREATE TRIGGER DelDept
AFTER DELETE ON Department
FOR EACH ROW
        BEGIN
                UPDATE Employee
                SET Department = NULL WHERE Department = OLD.Dept-No
        END;
```

DB2

```
CREATE TRIGGER DelDept
AFTER DELETE ON Department
FOR EACH ROW
UPDATE Employee
SET Department = NULL WHERE Department = OLD.Dept-No
```

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

<u>Starburst</u>

```
CREATE RULE DelEmp ON DEPARTMENT
WHEN DELETED
        THEN DELETE FROM Employee WHERE Department IN (SELECT Dept-No FROM DELETED)
```

<u>Oracle</u>

```
CREATE TRIGGER DelEmp
AFTER DELETE ON Department
FOR EACH ROW
        BEGIN
                DELETE FROM Employee WHERE Department = OLD.Dept-No
        END;
```

<u>DB2</u>

```
CREATE TRIGGER DelEmp
AFTER DELETE ON Department
FOR EACH ROW
DELETE FROM Employee WHERE Department = OLD.Dept-No
```

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.

<u>Starburst</u>

```
CREATE RULE SetSal ON Employee
WHEN INSERTED, UPDATED (Salary)
THEN UPDATE Employee E1
SET Salary = (SELECT E2.Salary FROM Employee E2, Department D2 WHERE E2.Name = D2.Manager
AND E1. Department = D2.Dept-No)
WHERE Salary > (SELECT E2.Salary FROM Employee E2, Department D2 WHERE E2.Name =
D2.Manager AND E1. Department = D2.Dept-No)
```

<u>Oracle</u>

```
CREATE TRIGGER SetSal
AFTER INSERT OR UPDATE OF Salary ON Employee E
FOR EACH ROW
DECLARE NUMBER Sal
        BEGIN
                SELECT M.Salary FROM Employee M, Department D WHERE D.Manager = M.Name
                AND D.Dept-No = E.Department INTO Sal
                UPDATE Employee
                SET Salary = Sal
                WHERE Salary > Sal
        END;
```

__DB2__

```
CREATE TRIGGER SetSal
AFTER INSERT OR UPDATE OF Salary ON Employee E
FOR EACH ROW
        BEGIN
                UPDATE Employee
                SET Salary = (SELECT M.Salary FROM Employee M, Department D WHERE
                D.Manager = M.Name AND D.Dept-No = E.Department)
                WHERE Salary > (SELECT M.Salary FROM Employee M, Department D WHERE
                D.Manager = M.Name AND D.Dept-No = E.Department)
        END;
```

Q4) Consider the (Oracle) Reorder Rule of Example 2.3 of the ADS book. To stop the reordering when there is already a pending order, we can use an active rule on PendingOrders, instead of the PL/SQL code currently used in Example 2.3. Please write such a DB2 rule and also revise the Reorder rule accordingly (and to conform to the syntax of DB2 triggers). You are not required to test your triggers.

```
CREATE TRIGGER StopReorder
BEFORE INSERT ON PendingOrders
FOR EACH ROW
WHEN (EXISTS (SELECT * FROM PendingOrders WHERE Part = New.Part))
        SIGNAL SQLSTATE '70005' ('Reorder Quantity already exists')


CREATE TRIGGER Reorder
AFTER UPDATE OF PartOnHand ON Inventory
FOR EACH ROW
WHEN (NEW.PartOnHand < NEW.ReorderPoint)
        BEGIN
                INSERT INTO PendingOrders
                VALUES (NEW.Part, NEW.ReorderQuantity, SYSDATE)
        END;
```

Q5) Revise your previous rule on PendingOrders so that when there is already a pending order for that part you only add in an order for 1/2 of the requested OrderQuantity instead of the requested quantity. You are not required to test your triggers.

```
CREATE TRIGGER ModReorderQuantity
BEFORE INSERT ON PendingOrders
FOR EACH ROW
UPDATE NEW
SET ReorderQuantity = 0.5 * NEW.ReorderQuantity
WHERE EXISTS (SELECT * FROM PendingOrders WHERE Part = NEW.Part)
```

```
CREATE TRIGGER Reorder
AFTER UPDATE OF PartOnHand ON Inventory
FOR EACH ROW
WHEN (NEW.PartOnHand < NEW.ReorderPoint)
        BEGIN
                INSERT INTO PendingOrders
                VALUES (NEW.Part, NEW.ReorderQuantity, SYSDATE)
        END;
```

Q6) Refer the cp2.pt notes (slides 33-36) and the table Part(Partno, Supplier, Cost), and let us change its FK declaration to "ON DELETE SET NULL." Please explain what happen when the statement " DELETE FROM Distributor WHERE State ='CA'" is executed.

Solution: In this case, the ForeignKey declaration is changed to "ON DELETE SET NULL". Now, when we execute the statement DELETE FROM Distributor WHERE State='CA', the following will happen: If there is a corresponding entry in the Part table which has the same Supplier as that of the Distributor of CA state (i.e. Jones), then that Supplier in Part table will want to become NULL because of the FK declaration. However, we have a before update trigger defined on Part which will throw an exception if it finds the Supplier value is trying to become NULL. When this exception is thrown, this will force a roll-back, not allowing the Supplier to become NULL and thereby not allowing the deletion to happen in the Distributor table. This is because if the deletion happens then the FK constraint should be satisfied by making the Supplier NULL, however, that is not possible because of the trigger.
On the other hand, if there is no corresponding entry in the Part table for a particular distributor then that row will be safely deleted from the Distributor table as there is no corresponding reference in the Part table for the same. There will be no exception raised for this case.