1. Consider the following relational schema:

Reader(RDNR, Surname, Firstname, City, Birthdate)
Book(ISBN, Title, Author, NoPages, PubYear, PublisherName)
Publisher(PublisherName, PublisherCity)
Category(CategoryName, BelongsTo)
Copy(ISBN, CopyNumber, Shelf, Position)
Loan(ReaderNr, ISBN, Copy, ReturnDate)
BookCategory(ISBN, CategoryName)

Enforce that a reader can only borrow up to 20 books. Give a solution using Trigger.

Solution:

```
CREATE TRIGGER MAX_NO_BOOK

BEFORE INSERT ON Loan

REFERENCING NEW ROW AS newl

FOR EACH ROW

WHEN ( (SELECT COUNT (*) FROM Loan

WHERE Loan.readerId = newl.readerId) >= 20)

BEGIN

raise_exception (`Illegal Insert - too many books per reader');

END;
```

2. Enforce AC→B for every insertion in the relation R (A, B, C, <u>D</u>, E) with an SQL trigger.

Solution:

```
CREATE TRIGGER fd_enforcer_insert

BEFORE INSERT on R

FOR EACH ROW

DECLARE counter INT

BEGIN

SELECT COUNT(*) INTO counter

FROM R

WHERE R.A = NEW.A AND R.C = NEW.C AND R.B <> NEW.B;

IF (counter >0 )

THEN raise_exception('AC->B on R was violated');

END;
```

A	B	C	D	E
a1	b1	c1	d1	e1
a2	b2	c2	d2	e2

TABLE BEFORE INSERT

a1	b3	c1	d3	e3

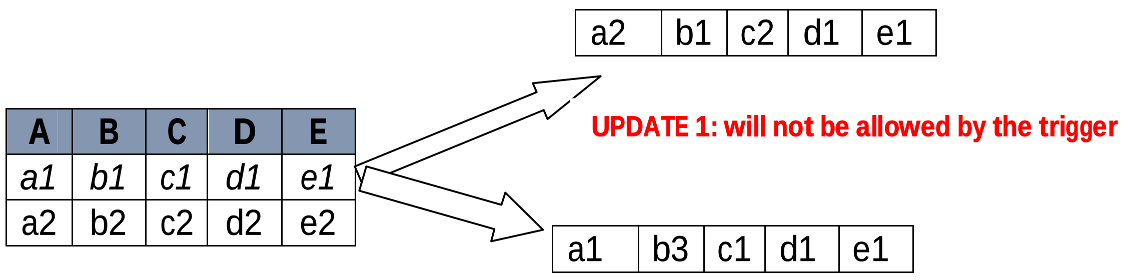
NEW ROW TO BE INSERTED

This insertion will not be allowed by the trigger

3. Enforce AC→B for every update in the relation R (A, B, C, <u>D</u>, E) with an SQL trigger.

Solution:

```
CREATE TRIGGER fd_enforcer_update
BEFORE UPDATE on R
FOR EACH ROW
DECLARE counter INT
BEGIN
  SELECT COUNT(*) INTO counter
  FROM R
  WHERE R.A = NEW.A AND R.C = NEW.C AND R.B <> NEW.B AND
       NOT (R.A = OLD.A AND R.B = OLD.B AND R.C =   OLD.C
AND R.D = OLD.D AND R.E = OLD.E);
  IF (counter > 0)
       THEN raise_exception('AB->C on R was violated');
END;
```



A	B	C	D	E
a1	*b1*	*c1*	*d1*	*e1*
a2	b2	c2	d2	e2

a2	b1	c2	d1	e1

UPDATE 1: will not be allowed by the trigger

a1	b3	c1	d1	e1

TABLE BEFORE UPDATE OF FIRST ROW

UPDATE 2: will be allowed by the trigger

4. Consider the simple relation Employee(ID, salary) storing the employee Ids and salaries, where ID is a key. Consider the following two triggers over this relation:

Trigger T1:

```
CREATE TRIGGER T1
AFTER INSERT ON Employee
REFERENCING NEW ROW as New_Emp
FOR EACH ROW
UPDATE Employee
SET salary = 1.1 * (SELECT max(salary)
                    FROM Employee)
WHERE ID = New_Emp.ID;
```

Trigger T2:

```
CREATE TRIGGER T2
AFTER INSERT ON Employee
REFERENCING NEW TABLE AS New_Emp
FOR EACH STATEMENT
UPDATE Employee
SET salary = 1.1 * (SELECT max(salary)
                    FROM Employee)
WHERE ID IN (SELECT ID FROM New_Emp);
```

Assume that relation Employee has no tuple initially. Suppose that we had inserted the following four rows into the Employee table as the result of a single SQL statement:

1	1000
2	2000
3	3000
4	4000

(i) Show the final database state after trigger execution if only trigger T1 is defined.

Solution:

1	1100
2	2200
3	3300
4	4400

(ii) Show the final database state after trigger execution if only trigger T2 is defined.

Solution:

1	4400
2	4400
3	4400
4	4400