Assertions

Assertions Example 1

- Consider the following schema:

  Emp(eid: integer, ename: string, age: integer, salary: real)

  Works(eid: integer, did: integer, time: integer)

  Dept(did: integer, budget: real, managerid: integer)

- Assertion which will ensure that all managers are more than 30 years old.

```
CREATE ASSERTION managerAge CHECK (
              (SELECT age
              FROM Emp, Dept
              WHERE eid = managerid) > 30)
```

Assertions Example 2

- Consider the following schema for a university:

  Student(rollno, name, address, CPI)

  Campus(location, rank)

  Apply(rollno, location, date, programme, decision)

- Assertion to satisfy the constraint: A student with CPI > 8.0 can only apply to campuses with rank < 3.

```
CREATE ASSERTION canapply CHECK (
    NOT EXISTS ( SELECT *
                 FROM Student, Apply, Campus
                 WHERE Student.rollno = Apply.rollno
                 AND Apply.location = Campus.location
                 AND Student.CPI > 8.0 AND Campus.rank >= 3))
```

Triggers

TRIGGERS Example 1

- Example 3: consider the following schema for a university:

  Student(rollno, name, address, CPI)

  Campus(location, rank)

  Apply(rollno, location, date, programme, decision)

- Write trigger for the rule: if a student with CPI > 8 applies for any programme in Jabalpur campus, then he/she is accepted for registration (i.e., decision is set to ‘Y’).

TRIGGERS Example 1

- If a student with CPI > 8 applies for any program in Jabalpur campus, then he/she is accepted for registration (i.e., decision is set to ‘Y’).

```
CREATE TRIGGER acceptjbpcampus
        AFTER INSERT ON Apply
        FOR EACH ROW
         BEGIN
            WHEN (NEW.location = ‘Jabalpur’ AND
                  (SELECT CPI FROM Student WHERE rollno = NEW.rollno) > 8)
            UPDATE Apply
                  SET decision = ‘Y’
                  WHERE rollno = NEW. Rollno
                    AND location = NEW.location
                    AND date = NEW. date
         END
```

TRIGGERS – How to Implement

- Syntax

```
DELIMITER $$

CREATE TRIGGER trigger_name
trigger_time
trigger_event ON tbl_name FOR EACH ROW
trigger_body

DELIMITER;
```

A number of SQL commands separated by a semi-colon (;) are required to create the full trigger code, therefore delimiter must be changed to something else - such as $$.

Set the delimiter back to a semi-colon

TRIGGERS (3)

trigger_name	Name of the trigger.
trigger_time	Trigger action time. It can be BEFORE or AFTER to indicate that the trigger activates before or after each row to be modified.
trigger_event	Indicates the kind of statement that activates the trigger. It can be INSERT, UPDATE or DELETE to indicate that the trigger activates on inserting, updating or deleting a row.
tbl_name	Table to which a trigger is associated.
trigger_body	Statements to be executed when the trigger activates. To execute multiple statements, use the BEGIN ... END compound statement construct.

Note: There cannot be two triggers for a given table that have the same trigger action time and event

TRIGGERS Example 2

Let we have a table ‘customer_time’ to record which customer is inserted at what moment.

```
CREATE TABLE customer_time
    (
    customer_name  VARCHAR(15) NOT NULL
    PRIMARY KEY,
    TIME                    TIMESTAMP  NOT NULL
     );
```

TRIGGERS Example 2

Example 2 … contd …

```
DELIMITER $$
CREATE TRIGGER insert_customer1
AFTER INSERT ON customer
FOR EACH ROW
BEGIN
        INSERT INTO customer_time
        (customer_name , TIME)
        VALUES (NEW.customer_name, TIMESTAMP());
 END$$
DELIMITER ;
```

TRIGGERS Example 3

- Example 2: Let we have a table ‘Account_interest’ to record when the account balance (in account table) is updated:

- CREATE TABLE account_interest

```
      ( account_number  INT(8),
        balance_before   INT(8),
        balance_after      INT (8));
```

TRIGGERS Example 3

Example 3 … contd …

```
DELIMITER $$
CREATE TRIGGER update_interest
 AFTER UPDATE ON account
 FOR EACH ROW
 BEGIN
   INSERT INTO account_interest
(account_number, balance_before, balance_after) VALUES
(NEW.account_number, OLD.balance, NEW.balance);
END $$
DELIMITER ;
```

TRIGGERS Example 3

- Example 3 … contd …
- Now run update query

  update *account*
  set *balance = balance* ∗ 1.1
  where *balance* > 500

- This will update balance in account table and make an entry in Account_interest table