This tutorial uses the relational schema in T07.sql.

1. Suppose that no employee can be both an engineer and a manager. Create two triggers to enforce this constraint on the Engineers and Managers tables. The triggers should run before insert or update and prevent changes (i.e., no insertion and no update) when the condition is not met (i.e., when an employee is about to be both engineer and manager).

Solution: The solution given here is on Managers. The solution on Engineers is CREATE OR REPLACE FUNCTION not_manager() RETURNS TRIGGER AS \$\$ DECLARE 4 count NUMERIC; 5 6 SELECT COUNT(*) INTO count 7 FROM Managers WHERE NEW.eid = Managers.eid; /* Engineers.eid */ 8 9 10 IF count > 0 THEN 11 RETURN NULL; 12 ELSE RETURN NEW; 13 END IF; 14 15 END; 16 \$\$ LANGUAGE plpgsql; CREATE TRIGGER non_manager BEFORE INSERT OR UPDATE ON Engineers FOR EACH ROW EXECUTE FUNCTION not_manager();

2. Suppose that we pay every engineers working on a project \$100 per hour worked. Since every project has a budget, the total number of hours worked by every engineer multiplied by 100 cannot exceed the project budget. Create a trigger to enforce this constraint such that when an insert or update is performed on Works table that violates this constraint, the number of hours worked by the engineer is set to the maximum allowable for that project.

```
Solution:
   CREATE OR REPLACE FUNCTION check_budget()
2
   RETURNS TRIGGER AS $$
3
   DECLARE
4
     hrs INTEGER;
5
     bgt INTEGER;
6
     rst INTEGER;
7
     SELECT COALESCE (SUM (hours), 0) INTO hrs
8
9
            /* COALESCE is used to handle NULL values */
10
     FROM Works
11
     WHERE pid = NEW.pid
12
       AND eid <> NEW.eid; /* for update */
13
14
     SELECT pbudget INTO bgt
15
     FROM
            Projects
16
     WHERE pid = NEW.pid;
17
18
     rst := (bgt - hrs*100)/100;
19
     IF NEW.hours > rst THEN
20
       RETURN (NEW.pid, NEW.eid, NEW.wid, rst);
21
     ELSE
22
       RETURN NEW;
23
     END IF;
24
   END:
25
   $$ LANGUAGE plpgsql;
   CREATE TRIGGER budget_check
2
   BEFORE INSERT OR UPDATE ON Works
  FOR EACH ROW EXECUTE FUNCTION check_budget();
```

3. As each work how has a type, we have an additional constraint that for a given work, the amount of time spent on the work cannot exceed the maximum hours for that particular type of work. Create a trigger to restrict Works table such that the hours worked cannot exceed the maximum hours for the given type. Whenever we want to insert or update such that the hours worked exceed the maximum hours, we set the hours worked to the maximum hours.

```
Solution:
   CREATE OR REPLACE FUNCTION max_hour_work()
2
   RETURNS TRIGGER AS $$
   DECLARE
4
     maximal INTEGER; /* cannot be NUMERIC */
5
   BEGIN
6
     SELECT max_hours INTO maximal
     FROM
            WorkType
8
     WHERE WorkType.wid = NEW.wid;
9
10
     IF NEW.hours > maximal THEN
11
       RETURN (NEW.pid, NEW.eid, NEW.wid, maximal);
12
13
       RETURN NEW;
     END IF;
14
15
16
   $$ LANGUAGE plpgsql;
  CREATE TRIGGER hours_max
  BEFORE INSERT OR UPDATE ON Works
  FOR EACH ROW EXECUTE FUNCTION max_hour_work();
```

4. Consider a case where we have a default work type. For simplicity, we let wid = 0 to be the default work type. As this is the default, we can neither modify nor delete this work type. Create a trigger to prevent modification or deletion of the default work type. The trigger should raise notice that some users are trying to modify or delete this default work type. Furthermore, the trigger should not raise any notice when some users are trying to modify or delete other type of work.

```
Solution:
  CREATE OR REPLACE FUNCTION default_work()
2
  RETURNS TRIGGER AS $$
3
  BEGIN
    RAISE NOTICE 'some user tried to';
4
5
    RAISE NOTICE 'modify/delete default';
6
    RAISE NOTICE 'work type';
    RETURN NULL;
  END;
  $$ LANGUAGE plpgsql;
  CREATE TRIGGER work_default
  BEFORE UPDATE OR DELETE ON WorkType
 FOR EACH ROW WHEN (OLD.wid = 0) EXECUTE FUNCTION default_work
  ();
```