Consider the following schema, which concerns projects carried out by engineers and managers in various departments in various offices.

```
CREATE TABLE Offices (
        oid INTEGER ,
        address VARCHAR(60),
        PRIMARY KEY (oid)
);

-- eid = eid of department 's manager
CREATE TABLE Departments (
        did INTEGER ,
        dbudget INTEGER NOT NULL,
        oid INTEGER NOT NULL,
        eid INTEGER NOT NULL,
        PRIMARY KEY (did),
        FOREIGN KEY (oid) REFERENCES Offices
);

CREATE TABLE Employees (
        eid INTEGER ,
        did INTEGER NOT NULL,
        PRIMARY KEY (eid),
        FOREIGN KEY (did) REFERENCES Departments
);

CREATE TABLE Engineers (
        eid INTEGER ,
        PRIMARY KEY (eid),
        FOREIGN KEY (eid) REFERENCES Employees
);

CREATE TABLE Managers (
        eid INTEGER ,
        PRIMARY KEY (eid),
        FOREIGN KEY (eid) REFERENCES Employees
);

-- eid = eid of project's supervisor
CREATE TABLE Projects (
        pid INTEGER ,
        pbudget INTEGER NOT NULL,
        eid INTEGER NOT NULL,
        PRIMARY KEY (pid),
        FOREIGN KEY (eid) REFERENCES Managers
);

CREATE TABLE WorkType (
        wid INTEGER ,
```

```
        max_hours INTEGER NOT NULL,
        PRIMARY KEY (wid)
);

CREATE TABLE Works (
        pid INTEGER ,
        eid INTEGER ,
        wid INTEGER ,
        hours INTEGER NOT NULL,
        PRIMARY KEY (pid,eid),
        FOREIGN KEY (eid) REFERENCES Engineers ,
        FOREIGN KEY (pid) REFERENCES Projects ,
        FOREIGN KEY (wid) REFERENCES WorkType
                ON DELETE CASCADE
);
```

1. Suppose that no employee can be both an engineer and a manager. Create two TRIGGERs to enforce this constraint on the Manager and Engineers tables, respectively. The TRIGGERs should run before INSERT or UPDATE and prevent changes (insertion or update) when the condition is not met.

Solution for the trigger on Manager: (the solution for Engineer is similar)

```
CREATE OR REPLACE FUNCTION not_mngr()
RETURNS TRIGGER AS
$$
DECLARE count NUMERIC;
BEGIN
        SELECT COUNT(*) INTO count
        FROM Managers
        WHERE NEW.eid = Managers.eid; -- Engineers.eid
        IF count > 0 THEN
                RETURN NULL;
        ELSE
                RETURN NEW;
        END IF;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER non_mngr
BEFORE INSERT OR UPDATE
ON Engineers
FOR EACH ROW
EXECUTE FUNCTION not_mngr();
```

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()
RETURNS TRIGGER AS
$$
DECLARE hrs INTEGER;
        bdgt INTEGER;
        rest INTEGER;
BEGIN
        SELECT COALESCE(SUM(hours), 0) INTO hrs
                -- COALESCE is used to handle NULL values
        FROM Works
        WHERE pid=NEW.pid AND
                eid <> NEW.eid; -- for update
        SELECT pbudget INTO bdgt
        FROM Projects
        WHERE pid = NEW.pid;
        rest := (bdgt - hrs*100)/100;
        IF NEW.hours > rest THEN
                RETURN (NEW.pid, NEW.eid, NEW.wid, rest);
        ELSE
                RETURN NEW;
        END IF;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER budget_check
BEFORE INSERT OR UPDATE
ON Works
FOR EACH ROW
EXECUTE FUNCTION check_budget();
```

3. As each work now 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 work type. Create a TRIGGER to restrict Works table such that the hours worked cannot exceed 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()
RETURNS TRIGGER AS
$$
DECLARE maximal INTEGER; -- cannot be NUMERIC
BEGIN
        SELECT max_hours INTO maximal
        FROM WorkType
        WHERE WorkType.wid = NEW.wid;
        IF NEW.hours > maximal THEN
                RETURN (NEW.pid, NEW.eid, NEW.wid, maximal);
        ELSE
                RETURN NEW;
        END IF;
END;
$$
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 def_work()
RETURNS TRIGGER AS
$$
BEGIN
        RAISE NOTICE 'some user tried to';
        RAISE NOTICE 'modify/delete default';
        RAISE NOTICE 'work type';
        RETURN NULL;
END;
$$
LANGUAGE plpgsql;

CREATE TRIGGER work_def
BEFORE UPDATE OR DELETE
ON WorkType
FOR EACH ROW
WHEN (OLD.wid = 0)
EXECUTE FUNCTION def_work();
```