TP3 — Triggers and integrity constraints

Advanced Databases (RSD/AI)

Triggers allow actions to be executed automatically when events occur on the data (`INSERT`, `UPDATE`, `DELETE`). They offer many advantages, including:

1. **Automatic enforcement of constraints** of any kind, independently of the applications that manipulate the data.
2. **Reduced application maintenance**: a change in a trigger is automatically applied to all the applications using the table, with no recompilation or client-side changes.
3. **Automatic logging** of the operations performed on the tables (history, audit, tracking).
4. **Automatic notification** of changes through the alert or signal mechanisms supported by the DBMS.

These mechanisms make it possible to centralize business rules directly in the database, ensuring better data integrity and consistency.

1 Relational schema

```
EMPLOYEE(ID, EMPLOYEE, QUALIFICATION, SALARY)
PROJECT(ID, PROJECT, MANAGER_ID, START_DATE, END_DATE)
PARTICIPATES(EMPLOYEE_ID, PROJECT_ID, ROLE)
```

Using the database created by the script at https://www.al-moualime.com/bdda/sql/tp3-fr.sql, whose table-definition part is shown below (translated identifiers):

```
-- EMPLOYEE
CREATE TABLE IF NOT EXISTS employee (
    id             did             NOT NULL PRIMARY KEY,
    employee       dname           NOT NULL UNIQUE,   -- UPPERCASE WITHOUT SPACES
    qualification  dqualification  NOT NULL,           -- UPPERCASE WITHOUT SPACES
    salary         dsalary         NOT NULL           -- > 0.00 AND VALUE <= 100000.00
);
```

```
-- PROJECT
CREATE TABLE IF NOT EXISTS project (
    id              did   NOT NULL PRIMARY KEY,
    project         dname NOT NULL UNIQUE,
    manager_id      did   REFERENCES employee(id),
    start_date      ddate NOT NULL,
    end_date        DATE,

    CONSTRAINT chk_project_dates
        CHECK (end_date IS NULL OR end_date >= start_date)
);
```

```
-- PARTICIPATES
CREATE TABLE IF NOT EXISTS participates (
    employee_id  did   NOT NULL,
    project_id   did   NOT NULL,
    role         drole NOT NULL DEFAULT 'MEMBER',  -- MANAGER / MEMBER

    PRIMARY KEY (employee_id, project_id),
    FOREIGN KEY (employee_id) REFERENCES employee(id),
    FOREIGN KEY (project_id)  REFERENCES project(id)
);
```

2 Work to do

Each rule below must be implemented by means of an appropriate trigger (`BEFORE`/`AFTER` `INSERT`, `UPDATE` or `DELETE`).

1. Checking roles and the number of projects per employee

The objective is to automatically enforce business rules when participations are inserted or modified. You must guarantee that:

- An employee's role in a project must be either `MEMBER` or `MANAGER`. Any other value must be rejected.
- An employee can be manager of **at most 2 projects**. Any insertion or update exceeding this limit must be rejected.
- An employee cannot participate as a member in **more than 3 projects**. Any insertion or update violating this rule must be rejected.

2. Synchronizing manager ↔ participation

As soon as an employee is appointed manager of a project, they must automatically appear as a participant in that project. You must therefore set up a mechanism that:

- detects any appointment or change of manager in `PROJECT`;
- checks whether they already appear in `PARTICIPATES`;
- automatically adds them with the `MANAGER` role otherwise.

3. Forbidding deletion of the manager's participation

The manager's participation must never be deletable. You must:

- block any attempt to delete the `PARTICIPATES` row corresponding to the project's manager;
- raise an explicit error (`RAISE EXCEPTION`) when this happens.

4. Enforced role for the manager

The manager's role must be impossible to change. You must prevent:

- a manager from becoming a member;
- their role from being changed to any other value.

The trigger must either block the operation, or automatically correct the role by setting it back to `MANAGER`.