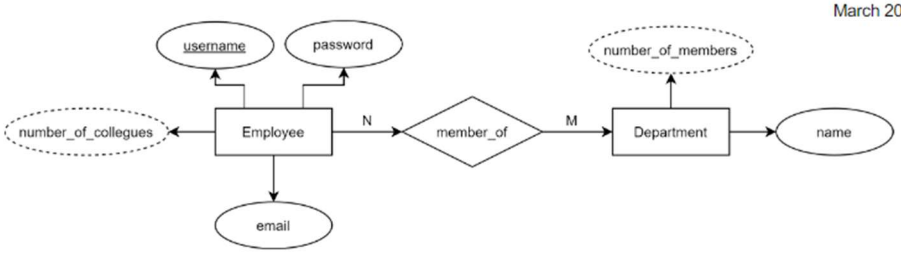
Lektionsøvelse 6

Data Management

March 2020

Exercises



sdu.dk

#sdudk

→ Create the SQL implementation of the above ER diagram
  → All creation of tables and inserts must be inside of a transaction, to ensure that it will only be created if everything works.
    → Remember to ROLLBACK or COMMIT to ensure you are not caught in a nested BEGIN.
    → If caught, restart the Postgres server.
→ As several queries are going to be of the form "select * from employee where email = 'some@email.com'", create the appropriate Index for the table.
→ Create a stored procedure or function that can update the number of members of a department (the derived attribute – see the slides for help)
→ Create a trigger that updates the number of members when a new membership is inserted or deleted.
→ Stretch Goal (Optional): Also create a stored procedure, and triggers, that calculate the number of colleagues an employee has based on the number of colleagues in the departments they are a member of.
  → Hint, if number_of_members is updated, run, then take all the departments an employee is member of and subtract one from the number, and add them together

Opgavebesvarelse

Down below, I have inserted the solution where it should give a clear review over the code which is being executed for this task.

```
BEGIN;

-- create  employees table
CREATE TABLE employees(
   id serial PRIMARY KEY,
   username VARCHAR (50) UNIQUE NOT NULL,
   password VARCHAR (50) NOT NULL,
   email VARCHAR (100) UNIQUE NOT NULL
);

-- Creating index on email
CREATE INDEX email_index ON employees (email);

-- inserting employees
INSERT INTO employees (username, password, email) VALUES ('John', 'myPassW0rd',
'john@acme.com');
INSERT INTO employees (username, password, email) VALUES ('Anne', 'myPassW0rd',
'anne@acme.com');
INSERT INTO employees (username, password, email) VALUES ('Jane', 'myPassW0rd',
'jane@acme.com');

-- creating department
CREATE TABLE departments(
   id serial PRIMARY KEY,
   name VARCHAR (50) UNIQUE NOT NULL,
   number_of_members INTEGER
);
```

```
-- inserting the department
INSERT INTO departments (name) VALUES ('Sales');

-- creating the many to many table in between employee and department
CREATE TABLE department_members(
   employee_id INTEGER NOT NULL REFERENCES employees(id),
   department_id INTEGER NOT NULL REFERENCES departments(id),
   PRIMARY KEY (employee_id, department_id)
);

-- Define procedures
-- update single department procedure
CREATE OR REPLACE PROCEDURE update_department_size(department_number INTEGER)
AS $$
DECLARE
    number_of_department_members integer := 0;
BEGIN
    SELECT COUNT(*) INTO number_of_department_members FROM department_members
WHERE department_id = department_number;
    UPDATE departments SET number_of_members = number_of_department_members
WHERE id = department_number;
END; $$
LANGUAGE plpgsql;

-- update all departments procedure
CREATE OR REPLACE PROCEDURE update_all_department_sizes()
AS $$
DECLARE
    departments CURSOR FOR SELECT DISTINCT(id) AS id FROM departments;
BEGIN
    FOR department in departments LOOP
        CALL update_department_size(department.id);
    END LOOP;
END; $$
LANGUAGE plpgsql;

-- function wrapper to enable running procedure as trigger
CREATE OR REPLACE FUNCTION update_all_department_sizes_trigger()
    RETURNS TRIGGER
AS $$
BEGIN
    CALL update_all_department_sizes();
    RETURN NULL;
END; $$
LANGUAGE plpgsql;

-- Define trigger to update departments(number_of_members)
CREATE TRIGGER update_number_of_members_trigger
    AFTER INSERT OR DELETE ON department_members
    EXECUTE PROCEDURE update_all_department_sizes_trigger();

-- Commit all changes to the databaqse
COMMIT;

-- *******************************************************
-- Testing everything works
-- *******************************************************
select * from departments; -- number of members is 0
```

```
INSERT INTO department_members (employee_id, department_id) VALUES (1,1);
select * from departments; -- number of members is now 1

INSERT INTO department_members (employee_id, department_id) VALUES (2,1);
select * from departments; -- number of members is now 2

INSERT INTO department_members (employee_id, department_id) VALUES (3,1);
select * from departments; -- number of members is now 3

DELETE FROM department_members WHERE department_id = 1;
select * from departments; -- number of members is now 0
```