Data management lecture 6

Concat

```
|| ' ' ||
```

```
concat(<var>, <seperator>, <var>....)
```

```
concat_ws(<seperator>, <var>, <var> ....)
```

```
create or replace function fnMakeFull(firstName varchar, lastName varchar)
returns varchar
as
$$
begin
    if firstName is null and lastName is null then
        return null;
    elsif firstName is null and lastName is not null then
        return lastName;
    elsif firstName is not null and lastName is null then
        return firstName;
    else
        return concat_{ws(' ', firstName, lastName)
    end if;
end;
$$
language plpgsql;
```

```
select * from fnMakeFull('John', 'Doe');
```

Run queries

```
create or replace function fnGetMoviesByYear(yr int)
returns table (
    movie_id int,
    movie_name varchar(60),
    year_released int
)
```

```
as
$$
begin
    return query
        select m.movie_id, m.movie_name, m.year_released from movies m
        where m.year_released = yr;
end
$$
language plpgsql;

select * from fnGetMoviesByYear(2001);
```

Transactions

```
create table if not exists HREmployee (
    emp_id int not null,
    annual_salary numeric not null,
    salary numeric generated always as (annual_salary/2080) stored
);

begin transaction;

    insert into HREmployee(emp_id, annual_salary) values (1, 400000);
    insert into HREmployee(emp_id, annual_salary) values (2, 400000);
    insert into HREmployee(emp_id, annual_salary) values (3, 400000);
    insert into HREmployee(emp_id, annual_salary) values (4, 400000);
    savepoint test_1;


    update HREmployee set annual_salary = 200000 where emp_id=2;
    update HREmployee set annual_salary = 100000 where emp_id=3;
    update HREmployee set annual_salary = 50000 where emp_id=4;
    savepoint test_2;
    rollback to savepoint test_1;
    release savepoint test_2;
    commit



select * from HREmployee;

truncate table HREmployee;
```

Keywords

```
- `truncate` drops data in the table
```

- `commit` commits the data in a transaction to a table
- `savepoint` creates a savepoint so you can go back to the working "branch"
- `rollback` rolls back to the provided savepoint
- `release` deletes safepoint

Triggers

```
create or replace function fn_calculateSalary()
returns trigger as $$
begin
    new.salary := new.hourly_pay * 2080;
    return new;
end;
$$ language plpgsql

create or replace trigger before_hourly_pay_update
    before update on Employee for each row execute function fn_calculateSalary();

update Employee set hourly_pay = 20;
```

Exercise

- 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 PostgreSQL server.

```
begin;

create table if not exists Employee(
    id serial not null primary key ,
    username varchar not null,
    password varchar not null
);

create table if not exists Department(
    id serial not null primary key,
    name varchar not null,
    number_of_members int
);
```

```
create table if not exists Employee_Department(
    employee_id int not null references Employee(id),
    department_id int not null references Department(id),
    primary key (employee_id, department_id)
);



-- inserting employees
INSERT INTO employee (username, password, email) VALUES ('John', 'myPassW0rd', 'johr
INSERT INTO employee (username, password, email) VALUES ('Anne', 'myPassW0rd', 'anne
INSERT INTO employee (username, password, email) VALUES ('Jane', 'myPassW0rd', 'jane

insert into department (name) values ('Sales');

--create index email_index on employee(email);

create or replace procedure UpdateDepartmentMembers(department_number int)
as $$
    declare number_of_department_members integer := 0;
        begin
            select count(*) into number_of_department_members from employee_departme
            update department set number_of_members = number_of_department_members w
        end;
    $$
language plpgsql;

create or replace procedure Update_all_department_sizes()
as $$
    declare departments cursor for select distinct(id) from department;
        begin
            for department in departments loop
                call UpdateDepartmentMembers(department.id);
            end loop;
        end;
    $$
language plpgsql;


create or replace function fnUpdateAllDepartmentSizesTrigger()
returns trigger as $$
    begin
        call Update_all_department_sizes();
        return null;
    end;
$$ language plpgsql;

create or replace trigger NumberOfMembersTrigger
    after insert or delete or update on employee_department
    execute procedure fnUpdateAllDepartmentSizesTrigger();
```

```
commit;



-- [TESTING] --


select * from employee_department;

insert into employee_department (employee_id, department_id) values (1, 1);
select * from department; -- should have 1 member

INSERT INTO employee_department(employee_id, department_id) VALUES (2,1);
select * from department; -- should have 2 members

INSERT INTO employee_department(employee_id, department_id) VALUES (3,1);
select * from department; -- should have 3 members

delete from employee_department where employee_id = 3;
select * from department; -- should have 2 members

insert into department (name) values ('Marketing');
insert into employee_department (employee_id, department_id) values (1, 2);

select * from department; -- should have 1 member

insert into employee_department (employee_id, department_id) values (2, 2);
select * from department; -- should have 2 members
end;
```