DB TUT 07

1. SQL table constraints cannot be used to define conditions which involove multiple tables.
    1. example:

```sql
create assertion manager_works_in_department
check (
    not exists (
        select *
        from Employee e
        join Department d on (d.manager = e.id)
        where e.works_in <> d.id -- <> is the same as !=
    )
)
```

2. write an assertion to ensure that no employee in a department earns more than the manager of their department.

```sql
create assertion employee_manager_salary
check (
    not exists (
        select *
        from Employee e
        join Department d on (e.works_in = d.id)
        where e.salary > (
            select e1.salary
            from Employee e1
            where d.manager = e1.id
        )
    )
)
```

3. What is the SQL command used in PostgreSQL to define a trigger? And what is the SQL command to remove it?
    1. in pspgsql

```sql
CREATE FUNCTION func_name() RETURNS trigger
AS $$
```

```
DECALRE
    ...
BEGIN
    ...
END;
$$ LANGUAGE plpgsql;
```

2. in sql
```sql
    CREATE TRIGGER trigger_name
    BEFORE operation                -- also can be AFTER
    ON table_name FOR EACH ROW
    EXECUTE PROCEDURE func_name();
```

3. remove trigger
```sql
    DROP TRIGGER trigger_name ON table_name
```

4. Triggers can be defined as BEFORE or AFTER. What exactly are they _before_ or _after_?
    1. BEFORE -> constraint checking -> AFTER
5. Give examples of when you might use a trigger BEFORE and AFTER
    1. an insert operation
        1. trigger before an insert
            1. before insert tuples, set a timestamp. for employees table has attribute create_at. we create a trigger, for each row we insert, we set a create_at = now()
        2. trigger after an insert
            1. after insert tuples, for example, we inserted an order, and want to trigger an logger
    2. an update operation
        1. trigegr before
            1. check validation
            2. create timestamp
        2. trigger after
            1. same as insert
    3. delete operation
        1. trigger before

```
            1. check referential integrity constraints
            2. safe to delete
        2. trigger after
            1. same as insert
6. constraint checking
    1. pk constraint on relation R
    ```sql
    create trigger R_pk_check before insert or update on R
    for each row execute procedure R_pk_check();
    --
    create function R_pk_check() returns trigger
    as $$
    begin
        if (new.a is null or new.b is null) then
            raise exception 'partially specified pk for R';
        end if;
        if (TG_OP = 'UPDATE' and old.a=new.a and old.b=new.b) then
            return;
        end if;
    -- not update -> insert
        select * from R where a=new.a and b=new.b
        if (found) then
            raise exception 'Duplicate primary key for R'
        end if;
    end;
    $$ language plpgsql;
```

7. .
8. trigger1 update table 2 when insert into table 1, trigger2 insert into table 1 when update table 2. which will cause infinite loop
9. Define a trigger that ensures that any time a row is inserted or updated in the table, the current user name and time are stamped into the row. The trigger should also ensure that an employee's name is given and that the salary has a positive value.

```
create or replace function emp_stamp() returns trigger
as $$
begin
    -- name is not null
    if new.empname is null
    then
        raise exception 'name cannot be null'
```
```

```
    end if;
    -- salary is pos
    if new.salary is null
    then
        raise exception 'salary cannot null'
    end if;

    if new.salary < 0
    then
        raise exception 'salary cannot less than 0'
    end if;

    -- name stamped
    new.last_user := user();
    -- time stamped
    new.last_date := now();
end;
$$ language plpgsql;
--
create trigger emp_stamp before insert or update on emp
    for each row execute procedure emp_stamp();
```

10. Define triggers which keep the numStudes field in the Course table consistent with the number of enrolment records for that course in the Enrolment table, and also ensure that new enrolment records are rejected if they would cause the quota to be exceeded.

```sql
    -- 1 --
    -- trigger to check number of numStudes consistent after insert
    create or replace ins_stu() returns trigger
    as $$
    begin
        update course
        set numStudies = numStudies + 1
        where code = new.course;
        return new;
    end;
    $$ language plpgsql;
    --
    create trigger ins_stu after insert on enrolment
        for each row execute procedure ins_stu();
    -- 2 --
```

```
-- trigger to check number of numStudes consistend after delete
create or replace del_stu() returns trigger
as $$
begin
    update course
    set numStudes = numStudies - 1
    where code = new.course;
    return new;
end;
$$ language plpgsql;
--
create trigger del_stu after delete on enrolment
    for each row execute procedure del_stu()
-- 3 --
-- trigger to check quota of the course
create or replace function check_quota() returns trigger
as $$
delare
    is_full boolean
begin
    select (numStudies >= quota) into is_full
    from Course where code = new.course
    --
    if is_full then
        raise exception 'class is full';
    end if;
    return new;
end;
$$ language plpgsql;
--
create trigger check_quota before insert on enrolment
    for each row execute procedure check_quota();
```

11. `
12. \
13. user defined aggregate
    1. stype: type of state
    2. initcond: initial value of the state variable
    3. sfuc: state function, takes a state and a value, return new state
    4. finalfunc: takes a state and return final value (opt)

14. define a avg function as aggregate

```
-- stype:
create type StateType as ( sum numeric, count numeric );
-- sfunc:
create function include (s stateType, v numeric) returns statetype
as $$
begin
    if (v is not NULL) then
        s.sum := s.sum + v;
        s.count := s.count + 1;
    end if;
    return s;
end;
$$ language plpgsql;
-- final func
create or replace functionn compute (s StateType) returns numeric
as $$
begin
    if (s.count = 0) then
        return null;
    else
        return s.sum::numeric / s.count;
    end id;
end;
$$ language plpgsql;
--
-- aggregate
create aggregate mean(numric) (
    stype = stateType
    initCond = '(0, 0)',
    sfunc = include,
    finalfunc = compute
);
```