Database Management

2019-2020

Midterm Homework: LinkKariyerMood Database

ANALYSIS

1. LinkedIn is a networking web site for businesses and people. Aim of this web site is to allow people and businesses create networks with each other. LinkedIn lets people to prepare a profile page where they can share their education and work experiences with other people. In LinkedIn people can follow businesses to be informed about changes in businesses or job offers. This website offers great convenience for community to share and know others' experiences.

   Kariyer.net is a web site for employees to search and apply for job offers and for companies to search and hire employees. Kariyer.net is apart from the LinkedIn is only created with the purpose of seeking a job. Kariyer.net lets employees to prepare CV which can be viewed by companies. Kariyer.net lacks the networking feature compared to Linkedn, but it makes it more centre on jobs and companies.

   Moodle is a web site and a server for universities. Unlike the other web sites in this project, Moodle only focuses on education and supplying it. In Moodle universities can have seperate web pages for each of their departments. Each department's page is composed of the course pages given in that department. In these course pages, teachers can share course materials or initialize homework submit areas where users can upload their homework files to the system. Moodle serves as a cloud storage so users of Moodle can upload and keep their files and access them anywhere.

2. A. LinkedIn aims connecting users in their each respectable line of work. Allowing them to build a network with people who are relevant to their career. It aims to be a professional networking site with social overtones.

   Kariyer.net aims to be a bridge between many workplace who are in need of new employees and the people who can place this vacancies. It stores peoples Cvs and lets the business owner s see available candidates.

   Moodle aims to be a learning platform which provides educators, administrators and learners with a secure, easy to access, integrated system to create learning environments.

   B. Main entities of LinkedIn are; Member, Group, Company, Address,University.
   Main entities of Kariyer.net are; Member, Skill, Address, Company, Job Offer, Office and University.
   Main entities of Moodle are; Teacher, Student, Project, File, Faculty, Department, Course and University.

C. Each person has identifyng person number, mail address and non identifying, phone number, firstname, lastname and birthdate.

Each teacher has a branch of research field.

Each student has GPA indicating current point in university and grade implementing which and class the student is in.

Each member has a password to login to LinkKariyerMood.

Each skill has a identifying skill number and non identifying name, category.

Each address has an identifying address number and non identifying country, city, street information.

Each graduage level has an identifier number and non identifyng name.

Each department has an identifier number and non identifyng name.

Each faculty has an identifier number and non identifyng name.

Each course has identifyng course number and non identifying name, course code and credit.

Each project has identifyng project number and non identifyng title, description.

Each file has a name and privacy which indicates wheter file is public or private.

Each organization has identifyng organization number, mail address, name and non identifyng phone number.

Each university can have any number of faculties belonging to it.

Each group has identifyng group number and non identifyng name, description.

Each job offer has identifyng job offer number and non identifyng offer title, description.

Each office has identifyng office number and non identifyng name.

D. Each person must be any combination of teacher, student or member.

Each person can educate in any number of graduate levels.

Each person must live in exactly one address.

Each person can know any number of skills.

Each person can be referenced by any number of teachers.

Each teacher must work in a department.

Each teacher can message any number of students.

Each teacher can upload any number of files.

Each teacher can teach any number of courses.

Each student can upload any number of files.

Each student can work on any number of projects.

Each student can message any number of teachers.

Each student can enroll any number of courses.

Each member can recommend or be recommended by any number of members.

Each member can work for any number of offices.

Each member can follow any number of companies.

Each member can apply to any number of job offers.

Each member can create any number of groups.

Each member can join any number of groups.

Each member can connect with or be connected with any number of members.

Each skill can be known by any number of people.

Each address can be a residential addres of any number of people.

Each address can be a work address of any number of job offers.

Each address can be location of any number of offices.

Each graduate level can educate any number of people.

Each graduate level must be given by a department.

Each department can give any number of graduate levels.

Each department must belong to a faculty.

Each department must be located in a address.

Each department can give any number of courses.

Each faculty can have any number of departments.

Each faculty must belong to a university.

Each course can give any number of projects.

Each course must belong to a department.

Each course can be taught by any number of teachers.

Each course can be enrolled by any number of students.

Each project must be given by a course.

Each project can be worked on any number of students.

Each file can be upladed by a teacher or a student.

Each organization must be either one of the university or company.

Each university can have any number of faculties belonging to it.

Each group must be created by exactly one member.

Each group can have any number of members who joined it.

Each job offer must be given by an office.

Each job offer can be applied by any number of members.

Each job offer must have a work address.

Each company can have any number of offices.

Each company can be followed by any number of members.

Each office must be controlled by a company.

Each offica can give any number of job offers.

Each office must be located in an address.

Each office employs any number of members as employees.

E. For skill entity, skill's category must be either one of Software, Finearts, Science or Sports

For person entity, person must be at least one or more of teacher, student and member.

For organization entity, type attribute must be one of CMP representing comapny or UNI representing university.

For office entity, the organization that office belongs to must be a company.

For faculty entity, the organization that faculty belongs to must be a university.

For student entity, the grade attribute must be one of 1, 2 ,3 or 4 representing which year the student is studying. For gpa attribute, gpa must be a float number in between 0 and 4.

For member entity, the password attribute must be equal or more than 8 digits for security purposes.

For file entity, the owner person of the file must belong to either student or teacher type.

For course entity, credit attribute must be greater than 0 and smaller than 9.

For any entites may have a start and end date attributes, start date must come before than end date.

DESIGN-LOGICAL MODEL

Iteration1

Step1

Skill(skill_id, name, category)

Address(address_id, country, city, street)

Group(group_id, name, description)

Job_offer(offer_id, job_title, description)

Office(office_id, name)

File(file_id, name, privacy)

Course(course_id, name, code, credit)

Project(project_id ,title, description)

Graduate_level(grad_id, name)

Department(dept_id, name)

Faculty(faculty_id, name)

Step2

-

Step3

-

Step4

Job_offer(offer_id, job_title, description, address_id, office_id)

Project(project_id ,title, description, course_id)

File(file_id, name, privacy, person_id)

Course(course_id, name, code, credit, dept_id)

Project(project_id ,title, description, course_id)

Graduate_level(grad_id, name, dept_id)

Department(dept_id, name, grad_id, address_id)

Step 5

-

Step 6

-

Step 7

-

Step 8

Person(person_id, fname, lname, phone, mail, bday)

Teacher(teacher_id, branch)

Student(student_id, gpa, grade)

Member(member_id, password)

Organization(org_id, name, phone, mail, type)

Step 9

-

Iteration 2

Step1

-

Step2

-

Step 3

-

Step 4

Person(person_id, fname, lname, phone, mail, bday,type, address_id)

Teacher(teacher_id, branch, dept_id)

Group(group_id, name, description, member_id)

Office(office_id, name, address_id, org_id)

Faculty(faculty_id, name, org_id)

Step 5

Educates_in(start_date, end_date, gpa, grad_id, person_id)

Messages(from_person_id, to_person_id, title, context,date)

References(teacher_id, referenced_person_id, date, context)

Knows(person_id, skill_id)

Teaches(teacher_id, course_id, semester)

Enrolls(student_id, course_id, semester, grade)

Works_on(student_id, project_id, grade)

Recommends(member_id, recommended_person_id, date, context)

Follows(member_id, org_id)

Works_for(member_id, start_date, end_date, job_title, office_id)

Applies(member_id, status, date, address_id, offer_id)

Joins(member_id, group_id)

Connects(member_id, connected_member_id)

Step 6

-

Step7

-

Step8

-

Step9

-

IMPLEMENTATION-PHYSICAL MODEL

1. CREATION SCRIPT

```
CREATE  TABLE tbl_skill (
skill_id SERIAL PRIMARY KEY,
name TEXT NOT NULL UNIQUE,
category TEXT NOT NULL,
CHECK(category IN ('Software', 'FineArts', 'Science','Sports'))
);
Create TABLE tbl_address(
address_id SERIAL PRIMARY KEY,
country TEXT NOT NULL,
city TEXT NOT NULL,
street TEXT NOT NULL,
UNIQUE(country, city, street)
);
```

```
CREATE TABLE tbl_organization(
org_id SERIAL PRIMARY KEY,
name TEXT NOT NULL UNIQUE,
phone VARCHAR(11) NOT NULL,
mail TEXT NOT NULL UNIQUE,
type VARCHAR(3) NOT NULL,
CHECK( type IN ('CMP','UNI'))
);

CREATE FUNCTION get_type(data_id INT)
RETURNS VARCHAR(3)
AS $$
BEGIN
RETURN (SELECT type FROM tbl_organization WHERE org_id = data_id);
END; $$
LANGUAGE PLPGSQL;

CREATE FUNCTION assert_is_teacher(data_id INT)
RETURNS VARCHAR(5)
AS $$
BEGIN
IF ( SELECT EXISTS (SELECT 1 FROM tbl_teacher WHERE teacher_id = data_id)) THEN
RETURN 'True';
ELSE
RETURN 'False';
END IF;
END; $$
LANGUAGE PLPGSQL;

CREATE FUNCTION assert_is_student(data_id INT)
RETURNS VARCHAR(5)
```

```
AS $$
BEGIN
IF ( SELECT EXISTS (SELECT 1 FROM tbl_student WHERE student_id = data_id)) THEN
RETURN 'True';
ELSE
RETURN 'False';
END IF;
END; $$
LANGUAGE PLPGSQL;

CREATE FUNCTION assert_is_member(data_id INT)
RETURNS VARCHAR(5)
AS $$
BEGIN
IF ( SELECT EXISTS (SELECT 1 FROM tbl_member WHERE member_id = data_id)) THEN
RETURN 'True';
ELSE
RETURN 'False';
END IF;
END; $$
LANGUAGE PLPGSQL;
CREATE TABLE tbl_office(
office_id SERIAL PRIMARY KEY,
name TEXT NOT NULL,
org_id INTEGER NOT NULL REFERENCES tbl_organization(org_id) ON DELETE CASCADE,
address_id INTEGER NOT NULL REFERENCES tbl_address(address_id) ON DELETE CASCADE,
CHECK( 'CMP' = get_type(org_id)),
UNIQUE(org_id ,name, address_id)
);
CREATE TABLE tbl_job_offer(
offer_id SERIAL PRIMARY KEY,
```

```
job_title TEXT NOT NULL,
description TEXT NOT NULL,
address_id INTEGER NOT NULL REFERENCES tbl_address(address_id) ON DELETE CASCADE,
office_id INT NOT NULL REFERENCES tbl_office(office_id) ON DELETE CASCADE,
UNIQUE (job_title, address_id, office_id)
);
CREATE TABLE tbl_faculty(
faculty_id SERIAL PRIMARY KEY,
name TEXT NOT NULL,
org_id INTEGER NOT NULL REFERENCES tbl_organization(org_id) ON DELETE CASCADE,
CHECK( 'UNI' = get_type(org_id)),
UNIQUE (org_id , name)
);
CREATE TABLE tbl_department(
dept_id SERIAL PRIMARY KEY,
name TEXT NOT NULL,
faculty_id INT NOT NULL REFERENCES tbl_faculty(faculty_id) ON DELETE CASCADE,
address_id INTEGER NOT NULL REFERENCES tbl_address(address_id) ON DELETE CASCADE,
UNIQUE (dept_id , name)
);
CREATE TABLE tbl_graduate_level(
grad_id SERIAL PRIMARY KEY,
name TEXT NOT NULL,
dept_id INT NOT NULL REFERENCES tbl_department(dept_id) ON DELETE CASCADE,
UNIQUE (grad_id , name)
);
CREATE TABLE tbl_person(
person_id SERIAL PRIMARY KEY,
fname TEXT NOT NULL,
lname TEXT NOT NULL,
phone VARCHAR(11),
```

```
mail TEXT NOT NULL UNIQUE,
bday DATE NOT NULL,
address_id INTEGER NOT NULL REFERENCES tbl_address(address_id) ON DELETE CASCADE
);
CREATE TABLE tbl_teacher(
teacher_id INT NOT NULL UNIQUE REFERENCES tbl_person(person_id) ON DELETE CASCADE,
branch TEXT,
dept_id INT NOT NULL REFERENCES tbl_department(dept_id) ON DELETE CASCADE
);
CREATE TABLE tbl_student(
student_id INT NOT NULL UNIQUE REFERENCES tbl_person(person_id) ON DELETE CASCADE,
gpa FLOAT,
grade varchar(1) NOT NULL,
CHECK( grade IN ('1', '2', '3', '4')),
CHECK( gpa = null OR  ( 0<= gpa AND gpa <= 4.0) )
);
CREATE TABLE tbl_member(
member_id INT UNIQUE REFERENCES tbl_person(person_id) ON DELETE CASCADE,
password TEXT NOT NULL,
CHECK( char_length(password)>=8 )
);
CREATE TABLE tbl_file(
file_id SERIAL PRIMARY KEY,
name TEXT NOT NULL,
privacy  BOOLEAN DEFAULT TRUE,
person_id INTEGER NOT NULL REFERENCES tbl_person(person_id) ON DELETE CASCADE,
CHECK( 'True' = assert_is_student(person_id) OR 'True' = assert_is_teacher(person_id)),
UNIQUE(person_id, name)
);
CREATE TABLE tbl_course(
course_id SERIAL PRIMARY KEY,
```

```
name TEXT NOT NULL,
code TEXT NOT NULL,
credit INTEGER NOT NULL,
dept_id INT NOT NULL REFERENCES tbl_department(dept_id) ON DELETE CASCADE,
CHECK(0<credit AND credit<9)
);
CREATE TABLE tbl_project(
project_id SERIAL PRIMARY KEY,
title TEXT NOT NULL,
description TEXT NOT NULL,
course_id INTEGER NOT NULL REFERENCES tbl_course(course_id) ON DELETE CASCADE,
UNIQUE(course_id , title)
);
CREATE TABLE tbl_group(
group_id SERIAL PRIMARY KEY,
name TEXT NOT NULL UNIQUE,
description TEXT NOT NULL,
member_id INTEGER NOT NULL REFERENCES tbl_member(member_id) ON DELETE CASCADE
);
CREATE TABLE tbl_educates_in(
start_date DATE NOT NULL,
end_date DATE,
gpa FLOAT,
grad_id INT NOT NULL REFERENCES tbl_graduate_level(grad_id) ON DELETE CASCADE,
person_id INT NOT NULL REFERENCES tbl_person(person_id) ON DELETE CASCADE,
CHECK(gpa = null OR ( 0<= gpa AND gpa <= 4.0)),
CHECK( start_date < end_date ),
UNIQUE(grad_id , person_id)
);
CREATE TABLE tbl_references(
teacher_id INTEGER NOT NULL REFERENCES tbl_teacher(teacher_id) ON DELETE CASCADE,
```

```
referenced_person_id INTEGER NOT NULL REFERENCES tbl_person(person_id) ON DELETE
CASCADE,
date DATE NOT NULL,
context TEXT NOT NULL,
CHECK(teacher_id != referenced_person_id),
UNIQUE(teacher_id ,referenced_person_id, date)
);
CREATE TABLE tbl_knows(
person_id INT REFERENCES tbl_person(person_id) ON DELETE CASCADE,
skill_id INT REFERENCES tbl_skill(skill_id) ON DELETE CASCADE,
UNIQUE(person_id , skill_id)
);

CREATE TABLE tbl_teaches(
teacher_id INTEGER NOT NULL REFERENCES tbl_teacher(teacher_id) ON DELETE CASCADE,
course_id INTEGER NOT NULL REFERENCES tbl_course(course_id)  ON DELETE CASCADE,
semester TEXT NOT NULL,
UNIQUE(teacher_id, course_id, semester)
);
CREATE TABLE tbl_messages(
from_person_id INT NOT NULL REFERENCES tbl_person(person_id),
to_person_id INT NOT NULL REFERENCES tbl_person(person_id),
title TEXT,
context TEXT,
date DATE NOT NULL,
CHECK( ('True' = assert_is_student(to_person_id) OR 'True' =
assert_is_teacher(from_person_id)) OR
( ('True' = assert_is_student(from_person_id) OR 'True' = assert_is_teacher(to_person_id))),
CHECK(from_person_id != tpo_person_id)
);
CREATE TABLE tbl_works_on(
student_id INTEGER NOT NULL REFERENCES tbl_student(student_id)  ON DELETE CASCADE,
```

```
project_id INTEGER NOT NULL REFERENCES tbl_project(project_id)  ON DELETE CASCADE,
grade INT,
semester TEXT NOT NULL,
UNIQUE(student_id, project_id, semester),
CHECK( grade = null OR ( 0<= grade AND grade<= 100))
);

CREATE TABLE tbl_recommends(
member_id INTEGER NOT NULL REFERENCES tbl_member(member_id) ON DELETE CASCADE,
recommended_member_id INTEGER NOT NULL REFERENCES tbl_member(member_id) ON DELETE CASCADE,
date DATE NOT NULL,
context TEXT NOT NULL,
CHECK(member_id != recommended_member_id),
UNIQUE(member_id, recommended_member_id, context )
);

CREATE TABLE tbl_follows(
member_id INTEGER NOT NULL REFERENCES tbl_member(member_id) ON DELETE CASCADE,
org_id INTEGER NOT NULL REFERENCES tbl_organization(org_id) ON DELETE CASCADE,
CHECK( 'CMP' = get_type(org_id)),
UNIQUE(member_id , org_id)
);

CREATE TABLE tbl_enrolls(
student_id INT NOT NULL REFERENCES tbl_student(student_id) ON DELETE CASCADE,
course_id INT NOT NULL REFERENCES tbl_course(course_id) ON DELETE CASCADE,
semester TEXT NOT NULL,
grade FLOAT,
CHECK (grade = null OR ( 0<= grade AND grade <= 4.0)),
```

```
UNIQUE(student_id, course_id ,semester)

);



CREATE TABLE tbl_works_for(

member_id INTEGER NOT NULL REFERENCES tbl_member(member_id) ON DELETE CASCADE,

start_date DATE NOT NULL,

end_date DATE,

job_title TEXT NOT NULL,

office_id INT NOT NULL REFERENCES tbl_office(office_id) ON DELETE CASCADE,

CHECK( start_date < end_date )

);


CREATE TABLE tbl_applies(

member_id INTEGER NOT NULL REFERENCES tbl_member(member_id) ON DELETE CASCADE,

date DATE NOT NULL,

status TEXT DEFAULT 'NOTEXAMINED',

offer_id INT NOT NULL REFERENCES tbl_job_offer(offer_id) ON DELETE CASCADE,

CHECK(status IN ('NOTEXAMINED', 'REJECTED', 'ACCEPTED')),

UNIQUE(offer_id , member_id)

);


CREATE TABLE tbl_joins(

member_id INTEGER NOT NULL REFERENCES tbl_member(member_id) ON DELETE CASCADE,

group_id INTEGER NOT NULL REFERENCES tbl_group(group_id) ON DELETE CASCADE,

UNIQUE(member_id, group_id)

);


CREATE TABLE tbl_connects(
```

```
member_id INTEGER NOT NULL REFERENCES tbl_member(member_id) ON DELETE
CASCADE,

connected_member_id INTEGER NOT NULL REFERENCES tbl_member(member_id) ON
DELETE CASCADE,

CHECK(member_id != connected_member_id),

UNIQUE(member_id,connected_member_id )

);



CREATE OR REPLACE PROCEDURE add_person_student(

        firstname text,

        lastname text,

        phone varchar(11),

        email text,

        bday DATE,

        address_id INT,

        gpa FLOAT,

        grade varchar(1))

LANGUAGE 'plpgsql'


AS $$

BEGIN


  WITH new_student as (

      INSERT INTO public.tbl_person(fname, lname, phone, mail, bday, address_id)

      VALUES ( firstname, lastname, phone, email, bday, address_id) returning person_id )

      INSERT INTO tbl_student(student_id, gpa, grade) SELECT person_id,gpa, grade FROM
new_student ;

   COMMIT;

END;

$$;
```

```
CREATE OR REPLACE PROCEDURE add_person_member(
            firstname text,
            lastname text,
            phone varchar(11),
            email text,
            bday DATE,
            address_id INT,
            password TEXT)
LANGUAGE 'plpgsql'

AS $$
BEGIN

  WITH new_member as (
        INSERT INTO public.tbl_person(fname, lname, phone, mail, bday, address_id)
         VALUES ( firstname, lastname, phone, email, bday, address_id) returning person_id )
         INSERT INTO tbl_member(member_id, password) SELECT person_id ,password FROM
new_member;
   COMMIT;
END;
$$;

CREATE OR REPLACE PROCEDURE add_person_teacher(
            firstname text,
            lastname text,
            phone varchar(11),
            email text,
            bday DATE,
            address_id INT,
            branch TEXT,
            dept_id)
```

```
LANGUAGE 'plpgsql'

AS $$
BEGIN

  WITH new_teacher as (
       INSERT INTO public.tbl_person(fname, lname, phone, mail, bday, address_id)
        VALUES ( firstname, lastname, phone, email, bday, address_id) returning person_id )
        INSERT INTO tbl_teacher(teacher_id, branch, dept_id) SELECT person_id , branch ,
dept_id FROM new_teacher ;

  COMMIT;
END;
$$;
```

2. INSERTION SCRIPT

```
INSERT INTO public.tbl_skill(
            name, category)
           VALUES ( '3D art skills', 'FineArts'),
                  ( 'Quantum Phsyics knowledge', 'Science'),
                  ( 'Python knowledge', 'Software');

INSERT INTO public.tbl_address(
            country, city, street)
           VALUES ( 'Turkey', 'Istanbul', 'Cumhuriyet Street'),
                  ( 'Turkey', 'Bursa', 'Zafer Street'),
                  ( 'Turkey', 'Izmir', 'Sencer Street'),
                  ( 'Turkey', 'Izmir', 'Ataturk Street');

INSERT INTO public.tbl_organization(
```

```
        name, phone, mail, type)
        VALUES ( 'Ege Üniversitesi', 2323881032, 'webadmin@ege.edu.tr', 'UNI'),
                ('9 Eylül Üniversitesi', 2324121212, 'egitim@deu.edu.tr', 'UNI'),
                ('Ege Profil', 2323989898, 'info@egeprofil.com.tr', 'CMP'),
                ('İzmir Yazilim', 2323453190, 'info@izmiryazilim.com.tr', 'CMP');

INSERT INTO public.tbl_office(
        name, org_id, address_id)
        VALUES ('IT', 3,2),
                ('Frontend', 4,1),
                ('R&D', 3,2);

INSERT INTO public.tbl_job_offer(
        job_title, description, address_id, office_id)
        VALUES ( 'Frontend Developer', 'Frontend developer for developing demanding
webpages',1     ,2),
                ('IT technician', 'IT technician to maintain the computer networks and
providing tech support', 2, 1),
                ('R&D Leader', 'R&D Leader to steer a engineer team to develop new
products', 2, 3);

INSERT INTO public.tbl_faculty(
        name, org_id)
        VALUES ( 'Engineering',1),
                ('Science',1),
                ('Fine Arts',2),
                ('Pharmacy',2);

INSERT INTO public.tbl_department(
        name, faculty_id, address_id)
        VALUES ( 'Computer Science'             ,1      ,3),
                ('Biochemistry'                 ,2      ,4),
```

```
                ('Statue'                          ,3        ,3),
                ('Pharmaseutical Technologies'  ,4        ,4);

INSERT INTO public.tbl_graduate_level(
         name, dept_id)
        VALUES ( 'Bachelors Degree'     ,1),
                ('Associate Degree'     ,2),
                ('Masters Degree'       ,3),
                ('Doctoral degree'      ,4);

CALL add_person_student('Deniz',        'Yurekdeler',    '05544413061',
        'dyurekdeler@gmail.com'         ,        '1997-07-03',    1, 3.6, '4')
CALL add_person_student('Cem',                  'Corbacioglu',   '05544413061',
        'cemcorbacioglu@gmail.com',  '1995-01-06',    2, 3.5 , '4')
CALL add_person_student('Aybars',       'Kokce'  ,       '05544413061',
        'akokce@hotmail.com',        '1993-11-09',    3, 1.9, '3' )
CALL add_person_member('Ege',                   'Kubilay',       '05544413061',
        'ekubilay@gmail.com',        '1990-02-05',    3, '12345678')
CALL add_person_member('Melisa',        'Erdem' ,        '05544413061', 'merdem@aol.com',
                '1985-01-02',    2, '12345678')
CALL add_person_member('Hakan',         'Atilgan',       '05544413061' ,
        'hatilgan@hotmail.com',      '1992-11-10',    3,'12345678')
CALL add_person_teacher('Ahmet',        'Atakus',        '05544413061',
        'ahmetatakus@hotmail.com',   '1980-05-07',    3, 'Linguistic', 1)
CALL add_person_teacher('Bilge',        'Usak'   ,       '05544413061',
        'bilgeusak@hotmail.com'         ,        '1994-10-11',    3, 'Object Oriented' ,2)

INSERT INTO public.tbl_course(
         name, code, credit, dep_id)
        VALUES ( 'Algorithms', 'BIL115115', 7, 1),
                ( 'Carving', 'STA223344', 6, 3),
```

```
                ( 'Organic modeling', 'BIO11345', 5, 2);

INSERT INTO public.tbl_project(
            title, description, course_id)
            VALUES ( 'Sorting algorithm project', 'describe each algorithms characteristics', 1),
                ( 'Organical modeling examples project', 'draw 10 organical compunds
model', 3);

INSERT INTO public.tbl_educates_in(
            start_date, end_date, gpa, grad_id, person_id)
            VALUES ('2013-11-07', '2017-07-07', 2.9, 1, 1),
                ('2015-11-05', '2017-07-07', 2.9, 4, 2),
                ('2016-11-03', '2018-08-08', 2.9, 3, 3);

INSERT INTO public.tbl_references(
            teacher_id, referenced_person_id, date, context)
            VALUES (4, 1, '2019-12-12', 'Bright kid'),
                (4, 2, '2020-01-01', 'Best in her class');
INSERT INTO tbl_messages(from_person_id, to_person_id, title, context, date) VALUES
(1,3,'Sınav Saati Hk.', 'Sınav saati ne zaman?','01.01.2020'),
(4,2,'Ders Programı', 'Ders programında Salı günü hangi ders var?');

INSERT INTO public.tbl_knows(
            person_id, skill_id)
            VALUES (1, 3),
                (2, 3),
                (4, 2);

INSERT INTO public.tbl_teaches(
            teacher_id, course_id, semester)
            VALUES (4, 1, 'winter'),
```

```
                (5, 2,'winter'),
                (3, 3, 'spring');

INSERT INTO public.tbl_works_on(
        student_id, project_id, grade, semester)
        VALUES (1, 1, '40' ,'spring'),
                (2, 1, '50', 'spring');

INSERT INTO public.tbl_enrolls(
        student_id, course_id, semester, grade)
        VALUES (1, 1, 'spring', 1.9),
                (2, 1, 'spring', 1.9);
```

3. TRIGGERS SCRIPT

```
CREATE OR REPLACE FUNCTION add_active_student()
  RETURNS trigger AS
$$
BEGIN
IF NEW.end_date = Null  AND assert_is_student = 'False' THEN
      INSERT INTO tbl_student(student_id,gpa,grade)
      VALUES(NEW.person_id,NEW.gpa,'1');

END IF;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER add_active_student_trigger
  AFTER INSERT
  ON tbl_educates_in
  FOR EACH ROW
  EXECUTE PROCEDURE add_active_student();



CREATE OR REPLACE FUNCTION init_employment()
  RETURNS trigger AS
$$
BEGIN
IF NEW.status = 'ACCEPTED' THEN
```

```
    WITH new_worksfor as (
                    SELECT job_title, member_id, office_id  FROM tbl_job_offer , NEW WHERE
offer_id  = NEW.offer_id )
                    INSERT INTO tbl_works_for(member_id,  job_title, office_id, start_date)
                    SELECT member_id,job_title, office_id FROM  new_worksfor, current.date;

END IF;
END;
$$
LANGUAGE 'plpgsql';


CREATE TRIGGER init_employment_trigger
 AFTER INSERT OR UPDATE
 ON tbl_applies
 FOR EACH ROW
 EXECUTE PROCEDURE init_employment();


CREATE OR REPLACE FUNCTION remove_student()
 RETURNS trigger AS
$$
BEGIN
IF NEW.end_date != Null THEN
      DELETE FROM tbl_student WHERE student_id = OLD.student_id;

END IF;
END;
$$
LANGUAGE 'plpgsql';

CREATE TRIGGER remove_student_trigger
 AFTER UPDATE
 ON tbl_educates_in
 FOR EACH ROW
 EXECUTE PROCEDURE remove_student();


CREATE OR REPLACE FUNCTION auto_join_group()
 RETURNS trigger AS
$$
BEGIN

   INSERT INTO tbl_joins(member_id, group_id) VALUES ( NEW.member_id, NEW.group_id);

END;
$$
LANGUAGE 'plpgsql';
```

```
CREATE TRIGGER auto_join_group_trigger
 AFTER INSERT
 ON tbl_group
 FOR EACH ROW
 EXECUTE PROCEDURE auto_join_group();
```

4. SELECTION SCRIPTS

A)

```
INSERT INTO tbl_skill(name,category) VALUES ('Android Developement','Software');
UPDATE tbl_skill SET  name='Android Mobile Development' WHERE skill_id = 4;
DELETE FROM tbl_skill WHERE skill_id = 4;

INSERT INTO tbl_job_offer VALUES (job_title, description, address_id , office_id) VALUES
('Müzik Öğretmeni', 'Ege Üniversitesinde 1. sınıflara müzik eğitimi verecek öğretmen', 1,
1);
UPDATE tbl_job_offer SET job_title='Halk Müziği Öğretmeni', description='Ege
üniversitesinde ortak seçmeli derste halk müziği dersi verecek öğretmen', address_id=1,
office_id=1 WHERE offer_id = 5;
DELETE FROM tbl_job_offer WHERE offer_id = 5;


INSERT INTO tbl_works_for(member_id, start_date,  job_title, office_id) VALUES (2,
'12.12.2016', 'RD Engineer' , 1);
UPDATE tbl_works_for SET , end_date='05.09.2019', job_title='Senior RD Engineer'
    WHERE member_id = 2, job_title = 'RD Engineer', office_id=1;
DELETE FROM tbl_works_for WHERE member_id = 2, job_title = 'Senior RD Engineer',
office_id=1;
```

B)

i)

```
SELECT name, description FROM tbl_group

SELECT branch FROM tbl_teacher

SELECT name FROM tbl_faculty


SELECT fname , lname, bday FROM tbl_person
WHERE bday >
(SELECT bday FROM tbl_personWHERE fname='Ege' AND lname='Kubilay')
```

ii)

```
SELECT name
FROM tbl_course C
```

```
INNER JOIN tbl_enrolls E ON C.course_id = E.course_id
WHERE student_id = 1;

SELECT fname
FROM tbl_person
WHERE address_id =
(SELECT address_id
FROM tbl_address
WHERE Country='Turkey' AND (City='Ankara' OR City='Bursa'))

SELECT gpa, fname, lname
FROM tbl_person
INNER JOIN tbl_student ON person_id = student_id

SELECT country, city ,street, name
FROM tbl_department D
INNER JOIN tbl_address A ON D.address_id = A.address_id

SELECT D.name , C.name
FROM tbl_course C
INNER JOIN tbl_department D ON dep_id = dept_id
```

iii)

```
SELECT grade
FROM tbl_project P
INNER JOIN  tbl_course C ON P.course_id = C.course_id
INNER JOIN tbl_enrolls E ON P.course_id = E.course_id
WHERE student_id =1

SELECT fname, lname , W.grade
FROM tbl_person P
INNER JOIN tbl_student ON student_id = person_id
INNER JOIN tbl_works_on W ON W.student_id = P.person_id

SELECT fname, lname, context
FROM tbl_person

INNER JOIN tbl_references
ON referenced_person_id =person_id
INNER JOIN tbl_student ON student_id = person_id
WHERE gpa>3.0


SELECT fname, lname
FROM tbl_person
WHERE EXISTS(
SELECT * FROM tbl_educates_in EI
```

```
INNER JOIN tbl_person P ON EI.person_id = P.person_id
INNER JOIN tbl_graduate_level G ON EI.grad_id = G.grad_id
WHERE name= 'Associate Degree' )
```

C)

```
SELECT fname,lname
FROM tbl_person
WHERE person_id=
 (SELECT teacher_id FROM tbl_teacher
WHERE dept_id=  (SELECT dept_id
FROM tbl_department WHERE name = 'Computer Science' ))

SELECT fname, lname ,E.grade
FROM tbl_enrolls E
INNER JOIN tbl_student S ON E.student_id = S.student_id
INNER JOIN tbl_person  ON person_id = S.student_id
INNER JOIN tbl_course C on E.course_id = C.course_id
INNER JOIN tbl_teaches T ON T.course_id = C.course_id
WHERE teacher_id=7 AND E.semester='winter 2019-2020' AND E.semester =
T.semester;

SELECT fname, lname FROM tbl_person
WHERE person_id IN
(SELECT student_id FROM tbl_student
UNION
SELECT member_id FROM tbl_member)

SELECT fname, lname, branch, name
FROM tbl_teacher T
INNER JOIN tbl_person ON teacher_id = person_id
INNER JOIN tbl_department D ON T.dept_id = D.dept_id

SELECT C.name
FROM tbl_course C
INNER JOIN tbl_department D ON C.dep_id = D.dept_id
INNER JOIN tbl_teaches T ON T.course_id = C.course_id
WHERE teacher_id = 7 AND semester='winter 2019-2020';
```