2021-2022 FALL SEMESTER

EGE UNIVERSITY
FACULTY of ENGINEERING
COMPUTER ENGINEERING DEPARTMENT
DATABASE MANAGEMENT
2021-2022
LİNKMOOD

Anıl Muslu 05190000045

Enes Aydın 05200001161

Mert Ali Koçak 05190000031

Onur Çılğın 05190000850

İÇİNDEKİLER

INTRODUCTION TO LINK MOOD DATABASE ANALYSIS

LINKEDIN

LinkedIn is a social network that you can use to create a professional network for you. By focusing on career development on LinkedIn, you can find suitable opportunities, events, job and internship announcements and many more news. You can also share your resume so that opportunities can find you.

Linkedin kendinize profesyonel bir ağ oluşturmak için için kullanabileceğiniz bir sosyal ağdır. Linkedinde kariyer gelişimine odaklanarak kendiniz için uygun fırsatlar, etkinlikler, iş ve staj ilanları ve daha birçok habere ulaşabilirsiniz. Ayrıca kendi özgeçmişinizi paylaşarak fırsatların da sizi bulmasını sağlayabilirsiniz. İlgili olduğunuz alanlardan birçok yetkin insanla bağlantı kurup iletişim becerinizi ve profesyonel sosyal ağınızı geliştirebilirsiniz.

MOODLE

Moodle is a website that facilitates digital education by sharing documentation between students and educators. Educators teaching a course can add files such as documents and projects related to that course to the system, and students can access these files. Moodle lets you organize a course by week, so each chapter is labeled with a date instead of a number. Instructors can hide and show episodes at any time. This allows an instructor to turn resources and activities on or off as the course progresses.

Moodle öğrenciler ve eğitimciler arasında dokümentasyon paylaşımı yaparak dijital eğitimi kolaylaştıran bir internet sayfasıdır. Bir dersi veren eğitimciler o dersle alakalı doküman ve proje gibi dosyaları sisteme ekleyebilir ve öğrenciler bu dosyalara ulaşabilir. Moodle, bir kursu haftaya göre düzenlemenizi sağlar, bu durumda her bölüm bir sayı yerine bir tarihle etiketlenir. Eğitmenler istedikleri zaman bölümleri gizleyebilir ve gösterebilir. Bu, bir eğitmenin kurs ilerledikçe kaynakları ve etkinlikleri açmasını veya kapatmasını sağlar.

Aim of Each Application

Linkedin: Linkedin is a social network established for business purposes. It has all the content a social network can contain, for example: sharing posts, making comments, viewing profiles. You can connect with many competent people in your fields of interest and improve your communication skills and professional social networking. To talk about the network system; Since this system is installed for network purposes, you offer a CV-like profile service to your connections. In this way, you can access news such as job and internship postings. There are also company pages for users who are interested in company information.

Moodle: This system can be created by an university, an educational institution. You must specify a user name in the system before your user can login to the page. You can then sign up for the courses provided with a specific key and access the documentation for that course. The aim of Moodle is to facilitate learning, to enable instructors to share and receive feedback more easily with students, to store course content, and to do all this with an easy interface.

LinkMood: Our site is a social network where users can access documents and news about their education and careers. Students can access their instructors and course-related documents, as well as be aware of internship announcements. Instructors, on the other hand, can improve the quality of education by easily sharing documents with their students. At the same time, you can access job postings and company information, and be informed about developments in your fields of interest. In addition, all users can connect with each other and send messages. They can form groups and share their experiences.

Sitemimiz kullanıcıların eğitim ve kariyerleriyle ilgili doküman ve haberlere ulaşabilecekleri bir sosyal ağdır. Öğrenciler eğitmenlerine ve dersle ilgili dokümanlara ulaşabildikleri gibi staj ilanlarından da haberdar olabilir. Eğitmenler ise öğrencileriyle kolaylıkla dokümanları paylaşarak eğitim kalitesini geliştirebilirler. Aynı zamanda iş ilanları ve şirket bilgilerine ulaşabilir, ilgi alanlarınızla ilgili gelişmelerden haberdar olabilirsiniz. Ayrıca tüm kullanıcılar birbirleriyle bağlantı kurup mesaj gönderebilirler. Gruplar oluşturup deneyimlerini paylaşabilirler.

Main Entities of Linkedin

User : It is the part where the members' information is kept.

Company : It is where company information is kept.

Connections : : It is where the connection information is kept.

User_Profile : It is the section where member profile information is kept.

Group : It is the part where the groups of members are kept.

Characteristics of each entity of Linkedin

USER

- user_id
- join_date
- birth_date
- email
- first_name
- last_name
- gender

COMPANY

- organization_id
- organization_name

CONNECTIONS

- connection_id
- connection_user_id
- user_id
- date_connection_made

USER_PROFILE

- profile_id
- user_id
- date_created
- date_last_updated

GROUP

- user_id
- group_id
- date_joined
- date_left
- group_id
- group_name

Relationships exists among the entities of Linkedin

- USER -> CREATE -> COMPANY
- USER -> CAN -> CONNECTION
- USER -> HAS -> USER_PROFILE
- USER -> CREATE -> GROUP

Constraints related to entities, their characteristics and the relationships of Linkedin

- An USER can admin multiple COMPANY.
- A COMPANY belongs to a USER.
- An USER can create CONNECTION.
- A CONNECTION belongs to a USER.
- An USER can follow USER.
- An USER may be followed by more than one USER.
- An USER has a USER_PROFILE.
- An USER can create more than one GROUP.
- A GROUP can be create by only one USER

Main Entities of Moodle

UNIVERSITY : It is the entity that holds information about the universities.
DEPARTMENT : It is the entity that holds information about the department.
COURSE: It is the entity that holds information about the courses.
INSTRUCTORS: It is the entity where the subjects of the courses are held.
STUDENT: It is the entity that holds the characteristics of the student.

Characteristics of each entity of Moodle

COURSE

- course_code
- instructor_id
- course_name

STUDENT

- student_number
- date_of_registration
- date_of_latest_login
- first_name
- last_name
- gender

INSTRUCTOR

- instructor_id
- first_name
- last_name
- gender

UNIVERSITY

- name
- adress

DEPARTMENT

- name
- department_id

Relationships exists among the entities of Moodle

- COURSES -> NEED-> INSTRUCTOR
- UNIVERSİTY-> BELONG -> DEPARTMENT
- DEPARTMENT->HAS->COURSE
- STUDENT -> ENROLL -> COURSE
- INSTRUCTOR-> GİVE-> COURSE

Constraints related to entities, their characteristics and the relationships of Moodle

- You need to be a INSTRUCTOR of a COURSE.
- INSTRUCTOR can be linked to more than one COURSES.
- A STUDENT can enroll to more than one COURSES.
- More than one STUDENT can be enrolled in a COURSE
- A DEPARTMENT can have more than one COURSE
- A COURSE can belong to a department
- A DEPARTMENT can belong to a ONLY UNIVERSITY
- A UNIVERSITY can have more than one DEPARTMENT

Database LinkMood

Our system allows students to apply directly to job postings published by companies, interact and socialize with people working in these companies, as well as choose courses, access the contents of the courses and access the projects given by the trainers. They can make job applications according to their own fields. They can message with people, participate in various communities or events. They can make their course selections over the courses given by their department and communicate with the lecturers. At the same time, trainers can apply for different jobs and choose which courses to teach and which projects to upload. Our system can also be used as an employee only. Or, an account can be opened as a "user" for the sole purpose of using social facilities.

- This system is a system that companies appeal to students.
- The type of user can be student, instructor, employee. There may be more than one of these.
- Users must have a different user name. A user must have the following information: first name, last name, email address, age, gender. An email address that has not been registered before must be used when logging in.
- One user can follow multiple users and send messages. A user can also be followed by more than one person and can also receive messages from more than one person. A message must include; sender name and receiver name and text.
- A user can set up a community of multiple users, regardless of type. this community must have a name and community id. A user can create multiple community pages. A community can only be created by one user. A user can join more than one community. A community can have multiple users. A user can be more than one community admin.
- A user can share multiple posts. A user can comment on the shared post and like the shared post. A comment has: comment content. When a comment is made, the username of the commentator appears. A post can be liked and commented by multiple users. However, a post belongs to a user.

- A user has education information. Education has education id, school name and department name.

- An instructor has a instructor id. A student must necessarily have an advisor instructor.

- The instructor can only work in one department. But that doesn't mean there's only one instructor in the department.

- A faculty member can teach more than one course. These courses have projects. These projects are done by the students. There is a department to which a course dependents on. A department must contain at least one course.

- An employee can work a company. There must be at least one employee in a company. An employee has a role and start date in the company.
- A company can have more than one job posting. A job advertisement belongs to a company. A job advertisement has the following: working type(full time, part time, intern), name, company id. More than one employee can apply for a job advertisement. A employee can apply for more than one job.

Main Entities of LinkMood

USER: It is the entity where user’s records are kept.

COMPANY: It is the entity where company’s records are kept.

JOB: It is the entity where job advertisers records are kept.
UNIVERSITY: It is the entity where universities records are kept.
DEPARTMENT: It is the entity where departments records are kept.
PROJECT: It is the entity where projects records are kept.
COURSE: It is the entity where courses records are kept.
INSTRUCTOR: It is the entity where instructors records are kept.
EMPLOYEE : It is the entity where employee records are kept.
STUDENT: It is the entity where student records are kept.
POST: It is the entity where posts records are kept.
COMMUNITY: It is the entity where group records are kept.
EDUCATION: It is the entity where education records are kept.
EVENT: It is the entity where event records are kept.

Characteristics of each entity of LinkMood

USER

- firstName
- lastName
- User_id
- Email
- Age
- Sex
- Phone

UNIVERSITY

- Name
- Adress

COMPANY

- Company_id
- Company_Name
- Country
- City
- Phonen

JOB

- JobName
- JobType

DEPARTMENT

- Department_id
- DName

PROJECT

- Project_id
- Name
- Deadline

COURSE

- Course_Code
- Name
- Credit

INSTRUCTOR

EMPLOYEE

POST

- Post_id
- Context
- Post_time

STUDENT

- StudentNumber
- Credit_taken

COMMUNİTY

- Community_id
- CommunityName

EDUCATİON

- Education_id
- School_Name
- Department_Name

EVENT

- Max_participant
- Event_time
- Event_id

Relationships exists among the entities of LinkMood

- USER -> LIKE -> POST
- USER -> COMMENT -> POST
- USER -> SHARE -> POST
- USER -> CREATE -> COMMUNITY
- USER -> JOINS -> COMMUNITY
- USER -> FOLLOW -> USER
- USER -> MESSAGE -> USER
- USER -> CONNECTS -> USER
- USER -> HAS -> EDUCATİON
- USER -> CREATE -> EVENT
- USER -> ENROLL -> EVENT
- USER -> ADMINS -> COMPANY
- STUDENT -> ENROLL -> COURSE
- STUDENT -> BELONG_DEPT -> DEPARTMENT
- STUDENT -> TAKES -> PROJECT
- INSTRUCTOR -> WORKS_DEPT -> DEPARTMENT
- INSTRUCTOR -> ADVISOR -> STUDENT
- INSTRUCTOR -> UPLOADS -> PROJECT
- INSTRUCTOR -> GIVE -> COURSE
- COURSE -> HAS -> PROJECT
- DEPARTMENT -> BELONG -> UNIVERSITY
- DEPARTMENT -> HAS -> COURSE
- EMPLOYEE -> APPLY -> JOB
- EMPLOYEE -> WORKS_ON -> COMPANY
- COMPANY -> OFFER -> JOB

Constraints related to entities, their characteristics and the relationships of LinkMood

A USER can LIKE a POST once.

A LIKE belongs to a USER.

A USER can SHARE multiple POSTs.

A POST belongs only to a USER.

A USER can assign multiple COMMENTS to a POST.

It can assign multiple USER COMMENTS to a POST.

A USER can CREATE multiple COMMUNITY.

A COMMUNITY has only one creative USER.

A USER can join more than one COMMUNITY.

A COMMUNITY can have multiple member USERs.

A USER can FOLLOW multiple USERs.

A USER can CONNECT multiple USERs.

A USER can send MESSAGE to multiple USERs.

A USER has EDUCATİON information.

A USER can CREATE multiple events.

A EVENT has only one creative USER.

A USER can ENROLL multiple EVENT.

A USER ADMİNS a COMPANY page.

A EVENT has multiple participant.

A EMPLOYEE can WORKS ON one COMPANY.

A EMPLOYEE can APPLY multiple JOB.

A COMPANY offer multiple JOB.

A STUDENT can TAKE multiple PROJECT.

More than one STUDENT can TAKE a PROJECT.

A STUDENT BELONGS a DEPT.

A STUDENT ENROLL multiple COURSE.

An INSTRUCTOR can WORKS in one DEPT.

More than one INSTRUCTOR can WORK in a DEPT.

An INSTRUCTOR can be an ADVISOR to multiple STUDENTS.

A STUDENT, has only one ADVISOR.

A COURSE has multiple INSTRUCTOR.

A İNSTRUCTOR can UPLOAD multiple PROJECT.

A COURSE can have multiple PROJECTs.

A PROJECT belongs to only one COURSE.

A DEPARTMENT is BELONG only to a UNIVERSITY.

A UNIVERSITY may have at least one or more DEPARTMENTS.

A DEPARTMENT has at least one COURSE.

A COURSE is belongs to only one DEPARTMENT.

Database LinkMood Mapping

1. ITERATION

1) POST (<u>Post_id</u>, Content, Post_time)

   GROUP (<u>Group_id</u>, Group_name)

   EDUCATİON (<u>Education_id</u>, School_name, Department_name)

   EVENT (<u>Event_id</u>, Event_name, Event_time)

COMPANY (Company_id, Company_name, Country, City)

PROJECT (Project_id, Deadline, Name)

COURSE (Course_code, Name, Credit)

DEPARTMENT (Dep_id, Dname)

UNIVERSITY (Name, Adress)

DEPARTMENT (Dep_id, Dname)

2) JOB (Company_id, Job_name, Job_type)

3)x

4)

COURSE (Course_code, Name, Credit, Department_id)

PROJECT (Project_id, Deadline, Name, Course_code)

DEPARTMENT (Dep_id, Dname, uni_name)

5) APPLY (Emp_id, Company_id, Job_name, appdate)

6) COMPANY_PHONE (Company_id, PhoneN)

7)X

8A)

USER (User_id, Phone, Sex, Email, FirstName, LastName, Bdate)

EMPLOYEE (Emp_id)

STUDENT (Student_id, Student_number, credit_taken)

INSTRUCTOR (Instructor_id)

2. ITERATION

1)X

2)X

3)X

4)

POST (Post_id, Content, Post_time, Shared_id)

GROUP (Group_id, Group_name, Creator_id)

EVENT (Event_id, Event_name, Event_time, Creator_id)

EMPLOYEE (Emp_id, Start_date, Role, Company_id)

STUDENT (Student_id, Student_number, credit_taken, Dep_id)

STUDENT (Student_id, Student_number, Advisor_id, Dep_id, credit_taken)

INSTRUCTOR (Instructor_id, Dep_id)

5)x

LİKE (User_id, Post_id)

COMMENT (User_id, Post_id, Text)

CONNECT (Follower, Following)

FOLLOW (Follower, Following)

JOIN_GROUP (User_id, Group_id)

HAS_EDUCATE (User_id, Education_id, Start_date, End_Date)

ENROLL_EVENT (User_id, Event_id)

MESSAGE (Sender, Receiver, Text, Message_time)

ENROLL_COURSE (Course_Code, Student_id)

TAKEPROJECT (Project_id, Student_id)

UPLOAD_PROJECT (Project_id, Ins_id)

GİVE_COURSE (Ins_id, Course_Code)

ADMİNS (User_Id, Company_Id)

6)x

7)x

USER

User_id	Phone	Sex	email	Firstname	Lastname	age

COMPANY

company_id	company_name	country	city

UNIVERSITY

names	address

STUDENT

Student_id	Student_number	Advisor_id	Dep_id	credit_taken

Student_id_ REFERENCES USER.User_id

Advisor_id REFERENCES INSTRUCTOR.İnstructor_id

Dep_id REFERENCES DEPARTMENT.dep_id

DEPARTMENT

dep_id	dname	uni_name

uni_name REFERENCES university.names

POST

post_id	content	post_time	Shared_id

Shared_id REFERENCES USER.User_id

JOB

company_id	Job_name	Job_type

Company_id REFERENCES Company.company_id

PROJECT

project_id	Deadline	Name	Course_code

Course_code REFERENCES COURSE._Course_code

COMPANY_PHONE

company_id	PhoneN

company_id REFERENCES COMPANY._company_id

EMPLOYEE

Emp_id	Start_date	Role	Company_id

company_id REFERENCES COMPANY._company_id

COURSE

Course_code	Name	Credit	Department_id

Department_id REFERENCES DEPARTMENT. dep_id

EDUCATION

Education_id	School_name	Department_name

INSTRUCTOR

Instructor_id

Instructor_id REFERENCES USER.user_id

LIKE

User_id	Post_id

User_id REFERENCES USER.User_id

post_id REFERENCES POST.post_id

COMMENT

User_id	post_id	Text

User_id	REFERENCES	USER.User_id
post_id	REFERENCES	POST.post_id

MESSAGE

Sender	Receiver	Text	Message_time	messagetime

Sender	REFERENCES	USER.User_id
Receiver	REFERENCES	USER.User_id

CONNECT

Connecter	Connecting

Connecter	REFERENCES	USER.User_id
Connecting	REFERENCES	USER.User_id

FOLLOW

Follower	Following

Follower	REFERENCES	USER.User_id
Following	REFERENCES	USER.User_id

JOIN_GROUP

User_id	Comp_id

User_id	REFERENCES	USER.User_id
Comp_id	REFERENCES	COMPANY._company_id

GROUP

Group_id	Group_name	Creator_id

Creator_id REFERENCES USER.User_id

HAS_EDUCATE

User_id	Education_id	Start_date	End_date

User_id REFERENCES USER.User_id

Education_id REFERENCES EDUCATION._Education_id

EVENT

Event_id	Event_name	Event_time	Creator_id

Creator_id REFERENNCES USER.User_id

ENROLL_EVENT

User_id	Event_id

User_id REFERENCES USER.User_id

Event_id REFERENCES EVENT.Event_id

ENROLL_COURSE

Course_code	Student_id

Course_code REFERENCES COURSE._Course_code

Student_id REFERENCES STUDENT.Student_id

TAKE_PROJECT

Project_id	Student_id

Project_id REFERENCES PROJECT.project_id

Student_id REFERENCES STUDENT.Student_id

UPLOAD_PROJECT

Project_id	Student_id

Project_id	REFERENCES	PROJECT.project_id
Student_id	REFERENCES	STUDENT.Student_id

GIVE_COURSE

Ins_id	Course_code

Ins_id	REFERENCES	INSTRUCTOR.ins_id
Course_code	REFERENCES	COURSE._Course_code

ADMINS

User_id	Company_id

User_id	REFERECES	USER.User_id
Comp_id	REFERENCES	COMPANY._company_id

APPLY

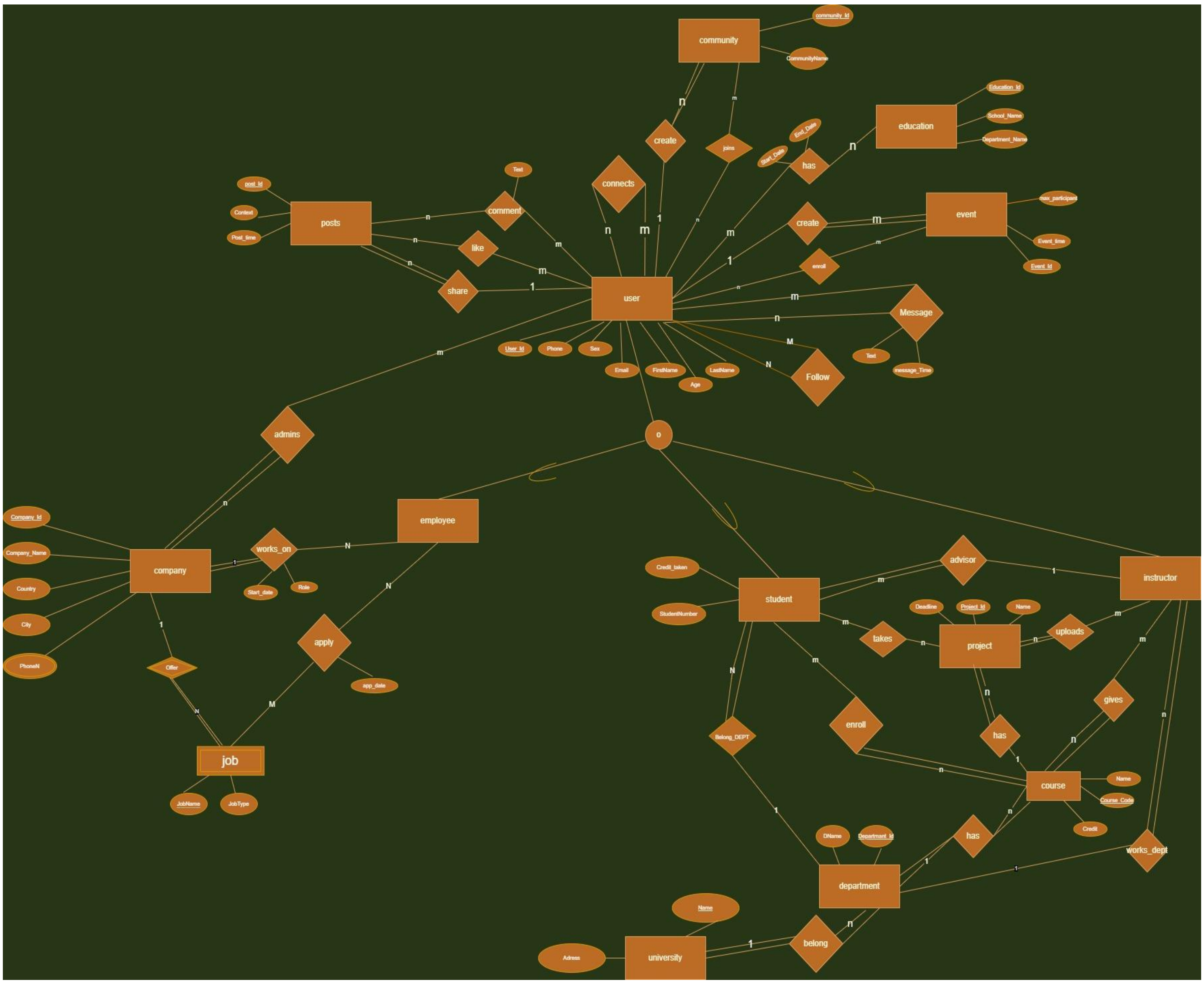
Emp_id	Company_id	Job_name	appdate

Emp_id	REFERENCES	EMPLOYEE.Emp_id

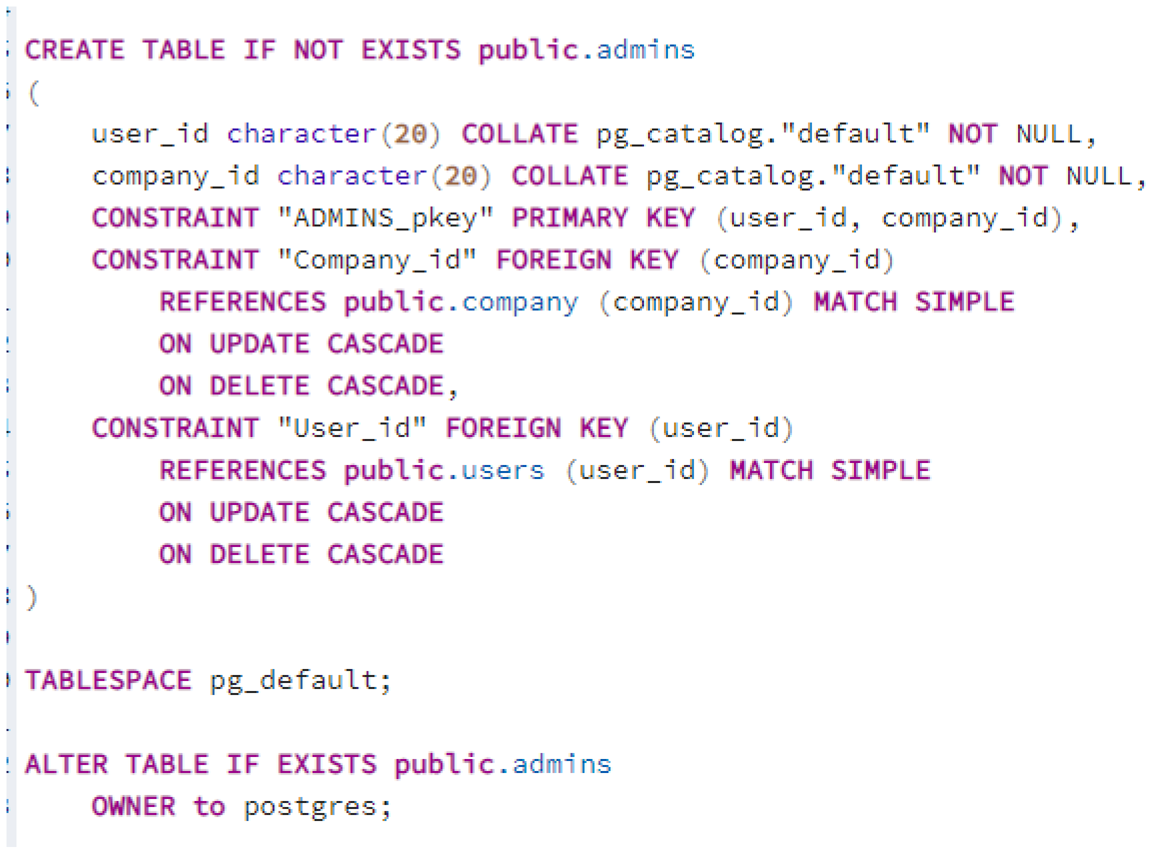
Company_id REFERENCES COMPANY.Company_id

Extended Entity Relationship Model of LinkMood

https://drive.google.com/file/d/1dJYcnG87Ym1WwoQazufWJp8HYs4j-Jl7/view?usp=sharing

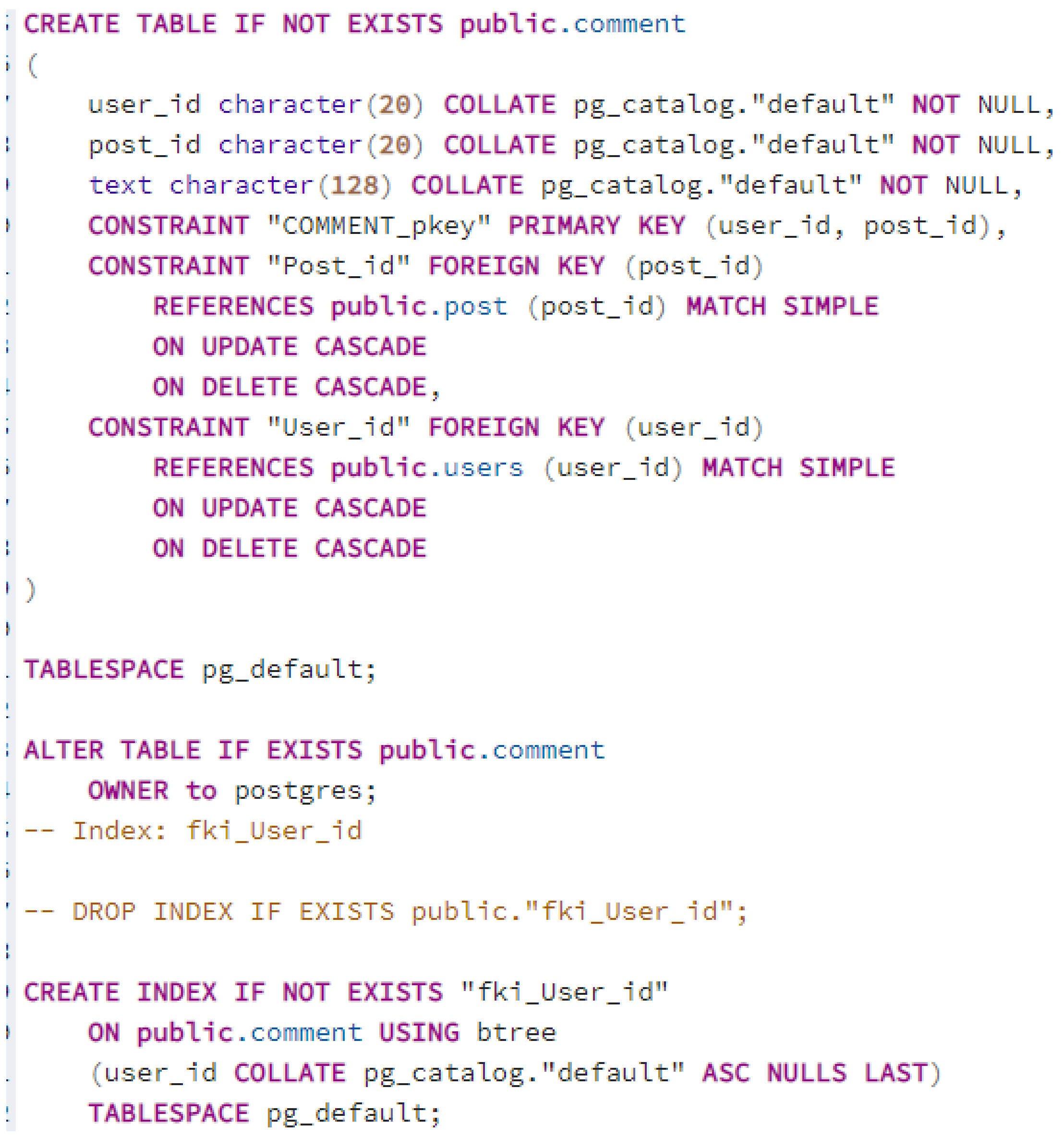


Create Table of LinkMood

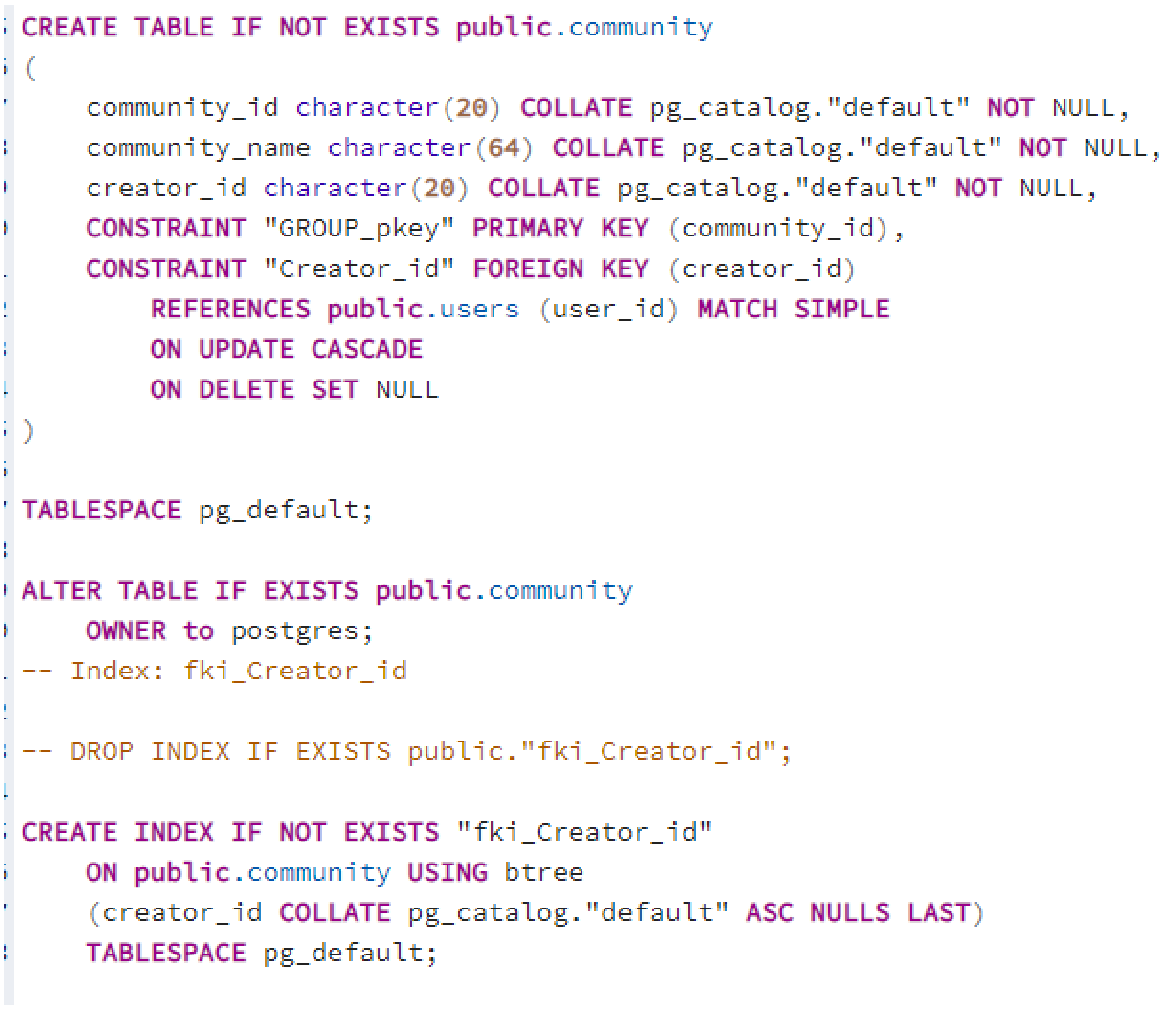


```sql
CREATE TABLE IF NOT EXISTS public.admins
(
    user_id character(20) COLLATE pg_catalog."default" NOT NULL,
    company_id character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "ADMINS_pkey" PRIMARY KEY (user_id, company_id),
    CONSTRAINT "Company_id" FOREIGN KEY (company_id)
        REFERENCES public.company (company_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "User_id" FOREIGN KEY (user_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.admins
    OWNER to postgres;
```

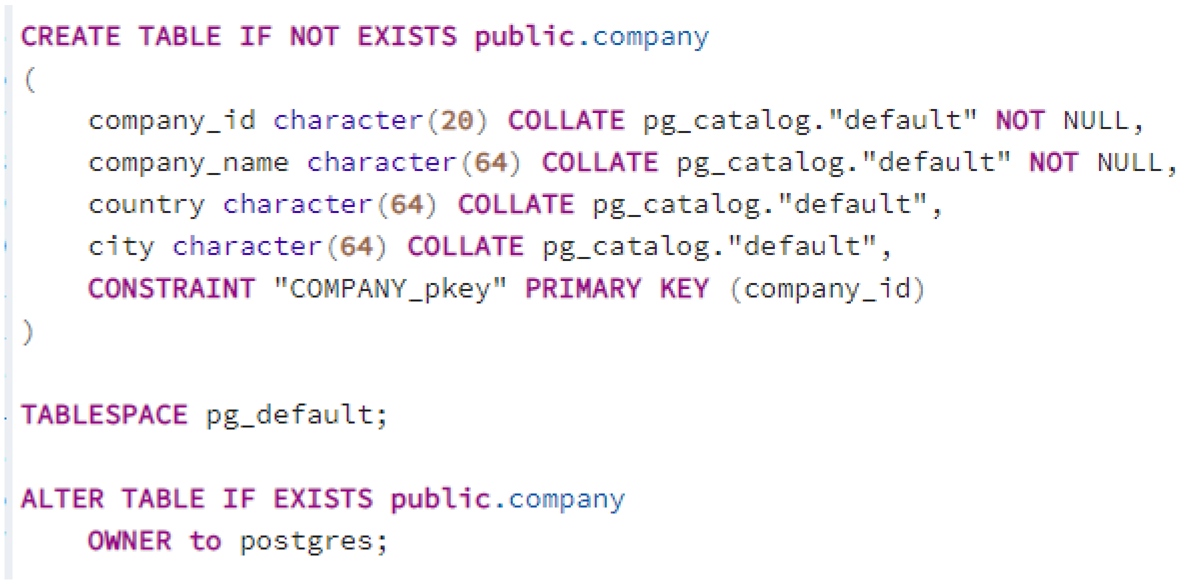
```sql
CREATE TABLE IF NOT EXISTS public.apply
(
    emp_id character(20) COLLATE pg_catalog."default" NOT NULL,
    company_id character(20) COLLATE pg_catalog."default" NOT NULL,
    job_name character(20) COLLATE pg_catalog."default" NOT NULL,
    appdate character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT apply_pkey PRIMARY KEY (emp_id, company_id, job_name),
    CONSTRAINT company FOREIGN KEY (company_id, job_name)
        REFERENCES public.job (company_id, job_name) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID,
    CONSTRAINT emp_id FOREIGN KEY (emp_id)
        REFERENCES public.employee (emp_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.apply
    OWNER to postgres;
-- Index: fki_company

-- DROP INDEX IF EXISTS public.fki_company;

CREATE INDEX IF NOT EXISTS fki_company
    ON public.apply USING btree
    (company_id COLLATE pg_catalog."default" ASC NULLS LAST, job_name COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;

-- Trigger: max_apply_job

-- DROP TRIGGER IF EXISTS max_apply_job ON public.apply;

CREATE TRIGGER max_apply_job
    BEFORE INSERT
    ON public.apply
    FOR EACH ROW
    EXECUTE FUNCTION public.check_numberof_apply();
```



```
CREATE TABLE IF NOT EXISTS public.comment
(
    user_id character(20) COLLATE pg_catalog."default" NOT NULL,
    post_id character(20) COLLATE pg_catalog."default" NOT NULL,
    text character(128) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "COMMENT_pkey" PRIMARY KEY (user_id, post_id),
    CONSTRAINT "Post_id" FOREIGN KEY (post_id)
        REFERENCES public.post (post_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "User_id" FOREIGN KEY (user_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.comment
    OWNER to postgres;
-- Index: fki_User_id

-- DROP INDEX IF EXISTS public."fki_User_id";

CREATE INDEX IF NOT EXISTS "fki_User_id"
    ON public.comment USING btree
    (user_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```



```sql
CREATE TABLE IF NOT EXISTS public.community
(
    community_id character(20) COLLATE pg_catalog."default" NOT NULL,
    community_name character(64) COLLATE pg_catalog."default" NOT NULL,
    creator_id character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "GROUP_pkey" PRIMARY KEY (community_id),
    CONSTRAINT "Creator_id" FOREIGN KEY (creator_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.community
    OWNER to postgres;
-- Index: fki_Creator_id

-- DROP INDEX IF EXISTS public."fki_Creator_id";

CREATE INDEX IF NOT EXISTS "fki_Creator_id"
    ON public.community USING btree
    (creator_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```



```sql
CREATE TABLE IF NOT EXISTS public.company
(
    company_id character(20) COLLATE pg_catalog."default" NOT NULL,
    company_name character(64) COLLATE pg_catalog."default" NOT NULL,
    country character(64) COLLATE pg_catalog."default",
    city character(64) COLLATE pg_catalog."default",
    CONSTRAINT "COMPANY_pkey" PRIMARY KEY (company_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.company
    OWNER to postgres;
```

```
CREATE TABLE IF NOT EXISTS public.company_phone
(
    company_id character(20) COLLATE pg_catalog."default" NOT NULL,
    phonen character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "COMPANY_PHONE_pkey" PRIMARY KEY (company_id, phonen),
    CONSTRAINT "Company_id" FOREIGN KEY (company_id)
        REFERENCES public.company (company_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT company_phone_phonen_check CHECK (char_length(phonen) = 11) NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.company_phone
    OWNER to postgres;
-- Index: fki_Company_id

-- DROP INDEX IF EXISTS public."fki_Company_id";

CREATE INDEX IF NOT EXISTS "fki_Company_id"
    ON public.company_phone USING btree
    (company_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

```
CREATE TABLE IF NOT EXISTS public.connects
(
    connector_id character(20) COLLATE pg_catalog."default" NOT NULL,
    connect_by_id character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "CONNECT_pkey" PRIMARY KEY (connector_id, connect_by_id),
    CONSTRAINT "Connect_by" FOREIGN KEY (connect_by_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "Connector" FOREIGN KEY (connector_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT connects_check CHECK (connector_id <> connect_by_id) NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.connects
    OWNER to postgres;
-- Index: fki_Connect_by

-- DROP INDEX IF EXISTS public."fki_Connect_by";

CREATE INDEX IF NOT EXISTS "fki_Connect_by"
    ON public.connects USING btree
    (connect_by_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
-- Index: fki_Connector

-- DROP INDEX IF EXISTS public."fki_Connector";

CREATE INDEX IF NOT EXISTS "fki_Connector"
    ON public.connects USING btree
    (connector_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

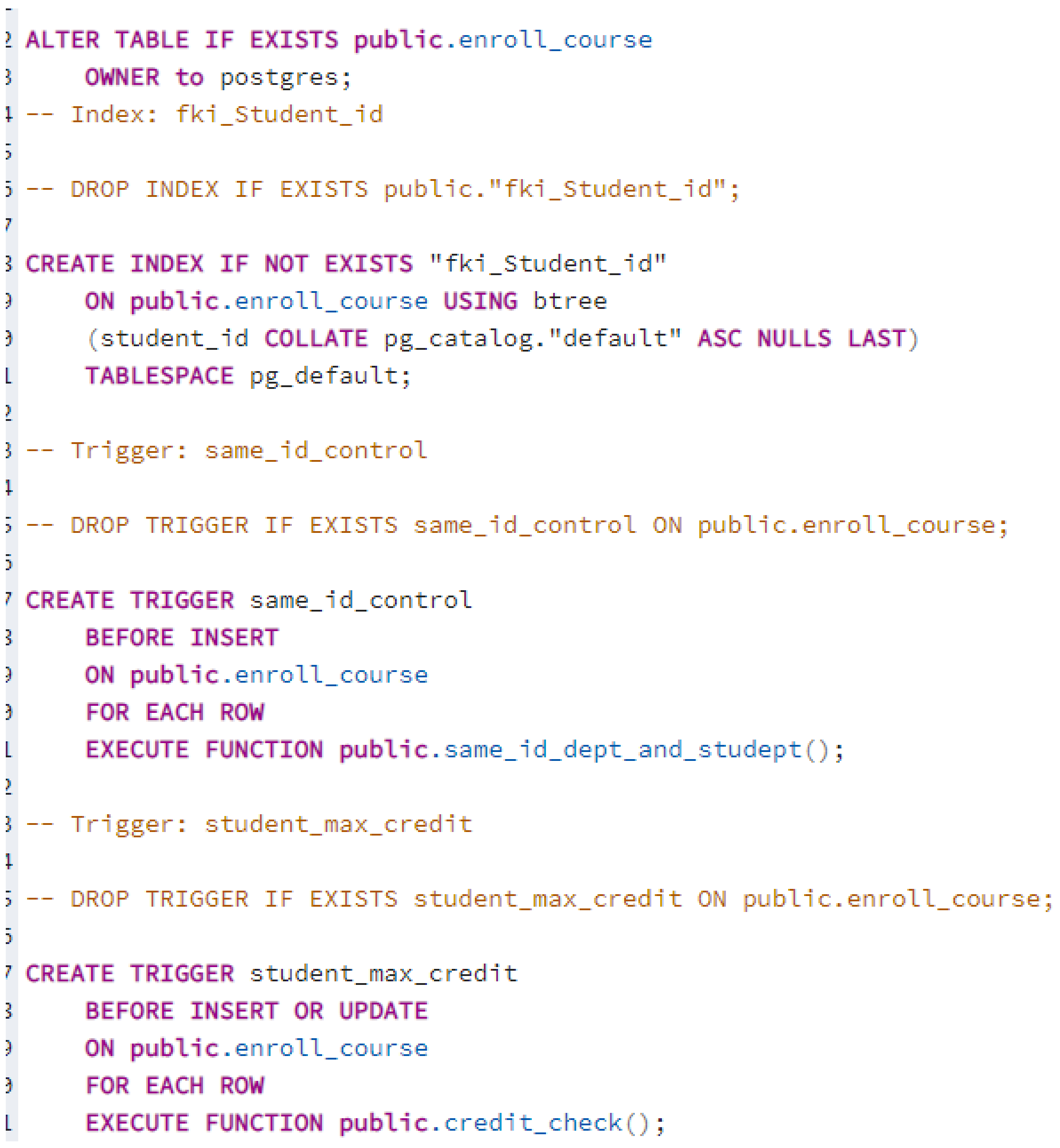
```
CREATE TRIGGER connect_to
    AFTER INSERT
    ON public.connects
    FOR EACH ROW
    EXECUTE FUNCTION public.connect_each_other();

-- Trigger: unconnect_to

-- DROP TRIGGER IF EXISTS unconnect_to ON public.connects;

CREATE TRIGGER unconnect_to
    AFTER DELETE
    ON public.connects
    FOR EACH ROW
    EXECUTE FUNCTION public.unconnect_each_other();
```

```
CREATE TABLE IF NOT EXISTS public.course
(
    course_code character(40) COLLATE pg_catalog."default" NOT NULL,
    names character(64) COLLATE pg_catalog."default" NOT NULL,
    credit integer,
    department_id character(40) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "COURSE_pkey" PRIMARY KEY (course_code),
    CONSTRAINT "Department_id" FOREIGN KEY (department_id)
        REFERENCES public.department (dep_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.course
    OWNER to postgres;
-- Index: fki_Department_id

-- DROP INDEX IF EXISTS public."fki_Department_id";

CREATE INDEX IF NOT EXISTS "fki_Department_id"
    ON public.course USING btree
    (department_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

```
CREATE TABLE IF NOT EXISTS public.department
(
    dep_id character(20) COLLATE pg_catalog."default" NOT NULL,
    dname character(64) COLLATE pg_catalog."default" NOT NULL,
    uni_name character(64) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "DEPARTMENT_pkey" PRIMARY KEY (dep_id),
    CONSTRAINT uni_name FOREIGN KEY (uni_name)
        REFERENCES public.university (names) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.department
    OWNER to postgres;
-- Index: fki_uni_name

-- DROP INDEX IF EXISTS public.fki_uni_name;

CREATE INDEX IF NOT EXISTS fki_uni_name
    ON public.department USING btree
    (uni_name COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

```
CREATE TABLE IF NOT EXISTS public.education
(
    education_id character(20) COLLATE pg_catalog."default" NOT NULL,
    school_name character(64) COLLATE pg_catalog."default",
    department_name character(64) COLLATE pg_catalog."default",
    CONSTRAINT "EDUCATION_pkey" PRIMARY KEY (education_id)
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.education
    OWNER to postgres;
```

```sql
CREATE TABLE IF NOT EXISTS public.employee
(
    emp_id character(20) COLLATE pg_catalog."default" NOT NULL,
    start_date date,
    role character(20) COLLATE pg_catalog."default",
    company_id character(20) COLLATE pg_catalog."default",
    CONSTRAINT "EMPLOYEE_pkey" PRIMARY KEY (emp_id),
    CONSTRAINT "Company_id" FOREIGN KEY (company_id)
        REFERENCES public.company (company_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL,
    CONSTRAINT emp_id FOREIGN KEY (emp_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.employee
    OWNER to postgres;
-- Index: fki_emp_id

-- DROP INDEX IF EXISTS public.fki_emp_id;

CREATE INDEX IF NOT EXISTS fki_emp_id
    ON public.employee USING btree
    (emp_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
-- Index: fki_employe_id

-- DROP INDEX IF EXISTS public.fki_employe_id;

CREATE INDEX IF NOT EXISTS fki_employe_id
    ON public.employee USING btree
    (emp_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```



```
ALTER TABLE IF EXISTS public.enroll_course
    OWNER to postgres;
-- Index: fki_Student_id

-- DROP INDEX IF EXISTS public."fki_Student_id";

CREATE INDEX IF NOT EXISTS "fki_Student_id"
    ON public.enroll_course USING btree
    (student_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;

-- Trigger: same_id_control

-- DROP TRIGGER IF EXISTS same_id_control ON public.enroll_course;

CREATE TRIGGER same_id_control
    BEFORE INSERT
    ON public.enroll_course
    FOR EACH ROW
    EXECUTE FUNCTION public.same_id_dept_and_studept();

-- Trigger: student_max_credit

-- DROP TRIGGER IF EXISTS student_max_credit ON public.enroll_course;

CREATE TRIGGER student_max_credit
    BEFORE INSERT OR UPDATE
    ON public.enroll_course
    FOR EACH ROW
    EXECUTE FUNCTION public.credit_check();
```

```
CREATE TABLE IF NOT EXISTS public.enroll_course
(
    course_code character(20) COLLATE pg_catalog."default" NOT NULL,
    student_id character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "ENROLL_COURSE_pkey" PRIMARY KEY (course_code, student_id),
    CONSTRAINT "Course_code" FOREIGN KEY (course_code)
        REFERENCES public.course (course_code) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "Student_id" FOREIGN KEY (student_id)
        REFERENCES public.student (student_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.enroll_course
    OWNER to postgres;
-- Index: fki_Student_id

-- DROP INDEX IF EXISTS public."fki_Student_id";

CREATE INDEX IF NOT EXISTS "fki_Student_id"
    ON public.enroll_course USING btree
    (student_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

```
CREATE TABLE IF NOT EXISTS public.enroll_event
(
    user_id character(20) COLLATE pg_catalog."default" NOT NULL,
    event_id character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "ENROLL_EVENT_pkey" PRIMARY KEY (user_id, event_id),
    CONSTRAINT "Event_id" FOREIGN KEY (event_id)
        REFERENCES public.events (event_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "User_id" FOREIGN KEY (user_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.enroll_event
    OWNER to postgres;
-- Index: fki_Event_id

-- DROP INDEX IF EXISTS public."fki_Event_id";

CREATE INDEX IF NOT EXISTS "fki_Event_id"
    ON public.enroll_event USING btree
    (event_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;

-- Trigger: join_event

-- DROP TRIGGER IF EXISTS join_event ON public.enroll_event;

CREATE TRIGGER join_event
    BEFORE INSERT
    ON public.enroll_event
    FOR EACH ROW
    EXECUTE FUNCTION public.check_participant();
```

```
CREATE TABLE IF NOT EXISTS public.events
(
    event_id character(20) COLLATE pg_catalog."default" NOT NULL,
    event_name character(64) COLLATE pg_catalog."default" NOT NULL,
    event_time date,
    creator_id character(20) COLLATE pg_catalog."default" NOT NULL,
    maximum_participant integer NOT NULL,
    CONSTRAINT "EVENT_pkey" PRIMARY KEY (event_id),
    CONSTRAINT "Creator_id" FOREIGN KEY (creator_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE NO ACTION,
    CONSTRAINT events_event_time_check CHECK (event_time > CURRENT_DATE) NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.events
    OWNER to postgres;

-- Trigger: creato_join

-- DROP TRIGGER IF EXISTS creato_join ON public.events;

CREATE TRIGGER creato_join
    AFTER INSERT
    ON public.events
    FOR EACH ROW
    EXECUTE FUNCTION public.creator_join_event();
```

```
CREATE TABLE IF NOT EXISTS public.follow
(
    follower character(20) COLLATE pg_catalog."default" NOT NULL,
    follower_by character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT follow_pkey PRIMARY KEY (follower, follower_by),
    CONSTRAINT "Follower" FOREIGN KEY (follower)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "Follower_by" FOREIGN KEY (follower_by)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT follow_check CHECK (follower <> follower_by) NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.follow
    OWNER to postgres;
-- Index: fki_Follower

-- DROP INDEX IF EXISTS public."fki_Follower";

CREATE INDEX IF NOT EXISTS "fki_Follower"
    ON public.follow USING btree
    (follower COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
-- Index: fki_Follower_by

-- DROP INDEX IF EXISTS public."fki_Follower_by";

CREATE INDEX IF NOT EXISTS "fki_Follower_by"
    ON public.follow USING btree
    (follower_by COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

```
CREATE TABLE IF NOT EXISTS public.give_course
(
    ins_id character(20) COLLATE pg_catalog."default" NOT NULL,
    course_code character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "GIVE_COURSE_pkey" PRIMARY KEY (ins_id, course_code),
    CONSTRAINT "Course_code" FOREIGN KEY (course_code)
        REFERENCES public.course (course_code) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "Ins_id" FOREIGN KEY (ins_id)
        REFERENCES public.instructor (instructor_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.give_course
    OWNER to postgres;

-- Trigger: max_give_course

-- DROP TRIGGER IF EXISTS max_give_course ON public.give_course;

CREATE TRIGGER max_give_course
    BEFORE INSERT
    ON public.give_course
    FOR EACH ROW
    EXECUTE FUNCTION public.check_numberof_course();

-- Trigger: same_id_control_ins

-- DROP TRIGGER IF EXISTS same_id_control_ins ON public.give_course;

CREATE TRIGGER same_id_control_ins
    BEFORE INSERT
    ON public.give_course
    FOR EACH ROW
    EXECUTE FUNCTION public.same_id_dept_and_insdept();
```

```sql
CREATE TABLE IF NOT EXISTS public.has_educate
(
    user_id character(20) COLLATE pg_catalog."default" NOT NULL,
    education_id character(20) COLLATE pg_catalog."default" NOT NULL,
    start_date date NOT NULL,
    end_date date,
    CONSTRAINT "HAS_UPDATE_pkey" PRIMARY KEY (user_id, education_id),
    CONSTRAINT "Education_id" FOREIGN KEY (education_id)
        REFERENCES public.education (education_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "User_id" FOREIGN KEY (user_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.has_educate
    OWNER to postgres;
-- Index: fki_Education_id

-- DROP INDEX IF EXISTS public."fki_Education_id";

CREATE INDEX IF NOT EXISTS "fki_Education_id"
    ON public.has_educate USING btree
    (education_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
-- Index: fki_u

-- DROP INDEX IF EXISTS public.fki_u;

CREATE INDEX IF NOT EXISTS fki_u
    ON public.has_educate USING btree
    (user_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

```sql
CREATE TABLE IF NOT EXISTS public.instructor
(
    instructor_id character(20) COLLATE pg_catalog."default" NOT NULL DEFAULT 1,
    dep_id character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "INSTRUCTOR_pkey" PRIMARY KEY (instructor_id),
    CONSTRAINT ins_id FOREIGN KEY (instructor_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.instructor
    OWNER to postgres;
-- Index: fki_ins_id

-- DROP INDEX IF EXISTS public.fki_ins_id;

CREATE INDEX IF NOT EXISTS fki_ins_id
    ON public.instructor USING btree
    (instructor_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

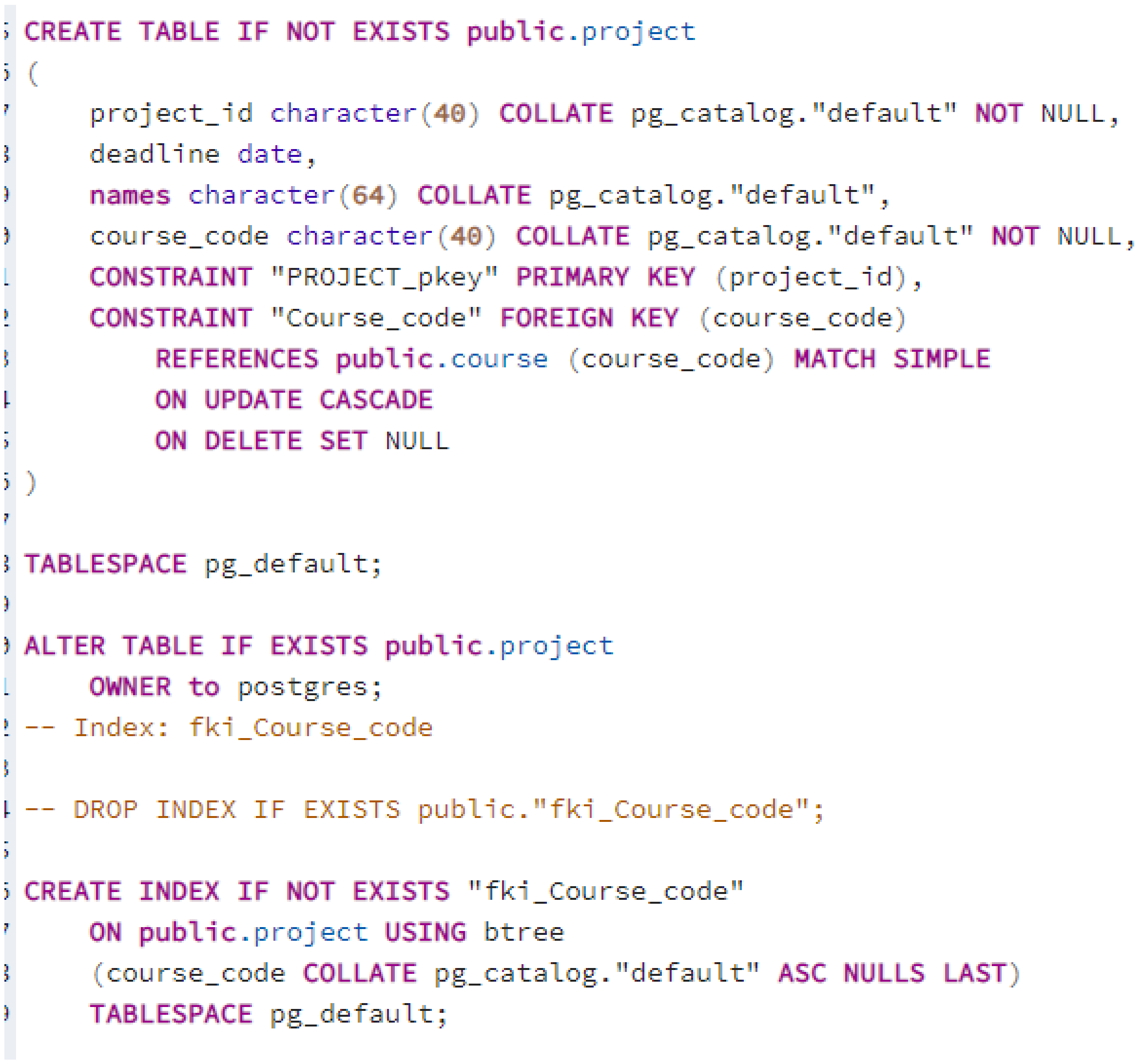
```sql
CREATE TABLE IF NOT EXISTS public.job
(
    company_id character(20) COLLATE pg_catalog."default" NOT NULL,
    job_name character(40) COLLATE pg_catalog."default" NOT NULL,
    job_type character(40) COLLATE pg_catalog."default",
    CONSTRAINT "JOB_pkey" PRIMARY KEY (company_id, job_name),
    CONSTRAINT "Company_id" FOREIGN KEY (company_id)
        REFERENCES public.company (company_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL,
    CONSTRAINT job_job_type_check CHECK (job_type = 'full-time'::bpchar OR job_type = 'part-time'::bpchar OR job_type = 'intern'::bpchar) NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.job
    OWNER to postgres;
-- Index: fki_O

-- DROP INDEX IF EXISTS public."fki_O";

CREATE INDEX IF NOT EXISTS "fki_O"
    ON public.job USING btree
    (company_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

```
CREATE TABLE IF NOT EXISTS public.join_community
(
    user_id character(20) COLLATE pg_catalog."default" NOT NULL,
    community_id character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "JOIN_GROUP_pkey" PRIMARY KEY (user_id, community_id),
    CONSTRAINT "Group_id" FOREIGN KEY (community_id)
        REFERENCES public.community (community_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL,
    CONSTRAINT "User_id" FOREIGN KEY (user_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.join_community
    OWNER to postgres;


CREATE TABLE IF NOT EXISTS public.likes
(
    user_id character(20) COLLATE pg_catalog."default" NOT NULL,
    post_id character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "LIKE_pkey" PRIMARY KEY (user_id, post_id),
    CONSTRAINT "Post_id" FOREIGN KEY (post_id)
        REFERENCES public.post (post_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.likes
    OWNER to postgres;
```

```sql
CREATE TABLE IF NOT EXISTS public.message
(
    sender character(20) COLLATE pg_catalog."default" NOT NULL,
    receiver character(20) COLLATE pg_catalog."default" NOT NULL,
    text character(128) COLLATE pg_catalog."default" NOT NULL,
    message_time date NOT NULL,
    CONSTRAINT "MESSAGE_pkey" PRIMARY KEY (sender, receiver),
    CONSTRAINT "Receiver" FOREIGN KEY (receiver)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "Sender" FOREIGN KEY (sender)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.message
    OWNER to postgres;
-- Index: fki_Receiver
```

```sql
ALTER TABLE IF EXISTS public.message
    OWNER to postgres;
-- Index: fki_Receiver

-- DROP INDEX IF EXISTS public."fki_Receiver";

CREATE INDEX IF NOT EXISTS "fki_Receiver"
    ON public.message USING btree
    (receiver COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
-- Index: fki_Sender

-- DROP INDEX IF EXISTS public."fki_Sender";

CREATE INDEX IF NOT EXISTS "fki_Sender"
    ON public.message USING btree
    (sender COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;

-- Trigger: message_control

-- DROP TRIGGER IF EXISTS message_control ON public.message;

CREATE TRIGGER message_control
    BEFORE INSERT
    ON public.message
    FOR EACH ROW
    EXECUTE FUNCTION public.message_to_connect();
```

```
CREATE TABLE IF NOT EXISTS public.post
(
    post_id character(20) COLLATE pg_catalog."default" NOT NULL,
    content character(128) COLLATE pg_catalog."default" NOT NULL,
    post_time date,
    shared_id character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "POST_pkey" PRIMARY KEY (post_id),
    CONSTRAINT "Shared_id" FOREIGN KEY (shared_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT post_post_time_check CHECK (post_time < CURRENT_DATE) NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.post
    OWNER to postgres;
-- Index: fki_Shared_id

-- DROP INDEX IF EXISTS public."fki_Shared_id";

CREATE INDEX IF NOT EXISTS "fki_Shared_id"
    ON public.post USING btree
    (shared_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```



```
CREATE TABLE IF NOT EXISTS public.project
(
    project_id character(40) COLLATE pg_catalog."default" NOT NULL,
    deadline date,
    names character(64) COLLATE pg_catalog."default",
    course_code character(40) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "PROJECT_pkey" PRIMARY KEY (project_id),
    CONSTRAINT "Course_code" FOREIGN KEY (course_code)
        REFERENCES public.course (course_code) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET NULL
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.project
    OWNER to postgres;
-- Index: fki_Course_code

-- DROP INDEX IF EXISTS public."fki_Course_code";

CREATE INDEX IF NOT EXISTS "fki_Course_code"
    ON public.project USING btree
    (course_code COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

```
CREATE TABLE IF NOT EXISTS public.student
(
    student_id character(20) COLLATE pg_catalog."default" NOT NULL,
    student_number character(64) COLLATE pg_catalog."default" NOT NULL,
    advisor_id character(20) COLLATE pg_catalog."default" NOT NULL,
    dep_id character(20) COLLATE pg_catalog."default" NOT NULL,
    taken_credit integer NOT NULL DEFAULT 0,
    CONSTRAINT "STUDENT_pkey" PRIMARY KEY (student_id),
    CONSTRAINT "Advisor_id" FOREIGN KEY (advisor_id)
        REFERENCES public.instructor (instructor_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE SET DEFAULT,
    CONSTRAINT "Dep_id" FOREIGN KEY (dep_id)
        REFERENCES public.department (dep_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE RESTRICT,
    CONSTRAINT user_id FOREIGN KEY (student_id)
        REFERENCES public.users (user_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
        NOT VALID,
    CONSTRAINT student_student_number_check CHECK (char_length(student_number) > 10) NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.student
    OWNER to postgres;
-- Index: fki_Advisor_id

-- DROP INDEX IF EXISTS public."fki_Advisor_id";

CREATE INDEX IF NOT EXISTS "fki_Advisor_id"
    ON public.student USING btree
    (advisor_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
```

```
CREATE INDEX IF NOT EXISTS "fki_Dep_id"
    ON public.student USING btree
    (dep_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;
-- Index: fki_user_id

-- DROP INDEX IF EXISTS public.fki_user_id;

CREATE INDEX IF NOT EXISTS fki_user_id
    ON public.student USING btree
    (student_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;

-- Trigger: student_number_check

-- DROP TRIGGER IF EXISTS student_number_check ON public.student;

CREATE TRIGGER student_number_check
    BEFORE UPDATE
    ON public.student
    FOR EACH ROW
    EXECUTE FUNCTION public.no_update_number();
```

```
CREATE TABLE IF NOT EXISTS public.takeproject
(
    project_id character(20) COLLATE pg_catalog."default" NOT NULL,
    student_id character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "TAKEPROJECT_pkey" PRIMARY KEY (project_id, student_id),
    CONSTRAINT "Project_id" FOREIGN KEY (project_id)
        REFERENCES public.project (project_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "Student_id" FOREIGN KEY (student_id)
        REFERENCES public.student (student_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.takeproject
    OWNER to postgres;
-- Index: fki_Project_id

-- DROP INDEX IF EXISTS public."fki_Project_id";

CREATE INDEX IF NOT EXISTS "fki_Project_id"
    ON public.takeproject USING btree
    (project_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;

-- Trigger: take_project_trigger

-- DROP TRIGGER IF EXISTS take_project_trigger ON public.takeproject;

CREATE TRIGGER take_project_trigger
    BEFORE INSERT
    ON public.takeproject
    FOR EACH ROW
    EXECUTE FUNCTION public.same_course_project_stu();
```

```
CREATE TABLE IF NOT EXISTS public.takeproject
(
    project_id character(20) COLLATE pg_catalog."default" NOT NULL,
    student_id character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "TAKEPROJECT_pkey" PRIMARY KEY (project_id, student_id),
    CONSTRAINT "Project_id" FOREIGN KEY (project_id)
        REFERENCES public.project (project_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "Student_id" FOREIGN KEY (student_id)
        REFERENCES public.student (student_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.takeproject
    OWNER to postgres;
-- Index: fki_Project_id

-- DROP INDEX IF EXISTS public."fki_Project_id";

CREATE INDEX IF NOT EXISTS "fki_Project_id"
    ON public.takeproject USING btree
    (project_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;

-- Trigger: take_project_trigger

-- DROP TRIGGER IF EXISTS take_project_trigger ON public.takeproject;

CREATE TRIGGER take_project_trigger
    BEFORE INSERT
    ON public.takeproject
    FOR EACH ROW
    EXECUTE FUNCTION public.same_course_project_stu();
```

```
CREATE TABLE IF NOT EXISTS public.upload_project
(
    project_id character(20) COLLATE pg_catalog."default" NOT NULL,
    ins_id character(20) COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT "UPLOAD_PROJECT_pkey" PRIMARY KEY (project_id, ins_id),
    CONSTRAINT "Ins_id" FOREIGN KEY (ins_id)
        REFERENCES public.instructor (instructor_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE,
    CONSTRAINT "Project_id" FOREIGN KEY (project_id)
        REFERENCES public.project (project_id) MATCH SIMPLE
        ON UPDATE CASCADE
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.upload_project
    OWNER to postgres;
-- Index: fki_Ins_id

-- DROP INDEX IF EXISTS public."fki_Ins_id";

CREATE INDEX IF NOT EXISTS "fki_Ins_id"
    ON public.upload_project USING btree
    (ins_id COLLATE pg_catalog."default" ASC NULLS LAST)
    TABLESPACE pg_default;

-- Trigger: upload_project_trigger

-- DROP TRIGGER IF EXISTS upload_project_trigger ON public.upload_project;

CREATE TRIGGER upload_project_trigger
    BEFORE INSERT
    ON public.upload_project
    FOR EACH ROW
    EXECUTE FUNCTION public.same_course_project_ins();
```

```sql
CREATE TABLE IF NOT EXISTS public.users
(
    user_id character(40) COLLATE pg_catalog."default" NOT NULL,
    phone character(12) COLLATE pg_catalog."default",
    sex character(10) COLLATE pg_catalog."default",
    email character(64) COLLATE pg_catalog."default",
    firstname character(30) COLLATE pg_catalog."default" NOT NULL,
    lastname character(30) COLLATE pg_catalog."default" NOT NULL,
    age integer,
    CONSTRAINT "USER_pkey" PRIMARY KEY (user_id),
    CONSTRAINT "EMAILSK" UNIQUE (email),
    CONSTRAINT "PHONESK" UNIQUE (phone),
    CONSTRAINT users_sex_check CHECK (sex = 'F'::bpchar OR sex = 'M'::bpchar) NOT VALID,
    CONSTRAINT users_age_check CHECK (age > 17) NOT VALID,
    CONSTRAINT users_phone_check CHECK (char_length(phone) = 11) NOT VALID
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.users
    OWNER to postgres;
```

Assertion of LinkMood

Instructorlar en fazla 5 ders verebilir.

```
create or replace function check_numberof_course()
returns trigger
language plpgsql
as
$$
declare
   given_course integer;
begin
   select count() into given_course
   from give_course
   where give_course.ins_id=new.ins_id;

   if(given_course<5)then
      return new;
   else
      raise notice'Maksimum 5 ders verebilir.';
      return null;
   end if;

end;
$$

create trigger max_give_course
before insert
on public.give_course
for each row
execute procedure check_numberof_course();
```

Bir employee en fazla 10 tane ilana başvuru yapabilir

```
create or replace function check_numberof_apply()
returns trigger
language plpgsql
as
$$
declare
   apply_job integer;
begin
   select count() into apply_job
   from apply
   where emp_id=new.emp_id ;

   if(apply_job<10)then
      return new;
   else
      raise notice'Maksimum 10 iş ilanına başvurabilirsiniz.';
      return null;
   end if;

end;
$$

create trigger max_apply_job
before insert
on public.apply
for each row
execute procedure check_numberof_apply();
```

Maksimum katılımcı sayısının geçilmemesini control eden assertion

```
create or replace function check_participant()
returns trigger
language plpgsql
as
$$
declare
   participant integer;
declare
   current_participant integer;
```

```
begin

   select maximum_participant  into participant
   from events
   where event_id=new.event_id  ;

   select count(*)  into current_participant
   from enroll_event ee
   where event_id=new.event_id ;



   if(current_participant <participant)then
      return new;
   else
      raise notice'Bu event katitilimci sinirina ulasmistir.';
      return null;
   end if;

end;
$$


create trigger join_event
before insert
on public.enroll_event
for each row
execute procedure check_participant();
```

Triggers of LinkMood

Alınacak krediyi 40la sınırlayan Trigger

```
create or replace function credit_check() -- kredi kontrol fonksiyonu
returns trigger
language plpgsql
as
$$
declare
            stu_credit integer;
declare
            cour_credit integer;


begin

                      select taken_credit into stu_credit
                      from student s
                      where student_id  = new.student_id;

                      select credit into cour_credit
                      from course c
                      where course_code = new.course_code;

           stu_credit := stu_credit + cour_credit ;

           if (stu_credit<=40) then
                      UPDATE public.student
                      SET  taken_credit=stu_credit
                      WHERE student_id=new .student_id ;
                      return new;
           else
                      RAISE NOTICE '40 krediyi asti';
                      return null;
           end if;


end;
$$



create trigger student_max_credit -- kredi kontrol triggeri
before insert or update
on public.enroll_course
for each row
execute procedure credit_check();

drop trigger student_max_credit on public.enroll_course; --kredi kontrol triggerini sil
```

Öğrencinin numarasının güncellenmeyen trigger

```
create or replace function no_update_number()
returns trigger
language plpgsql
as
```

```
$$
declare
	stu_number bpchar(64);
begin
	select student_number into stu_number
	from student s
	where student_id=new.student_id;

	if (new.student_number!=stu_number) then
		raise notice 'Ogrenci numarasi guncellenemez.';
		return null;
	else
		return new;
	end if;
end;
$$

create trigger student_number_check
before update
on public.student
for each row
execute procedure no_update_number();

drop trigger student_number_check  on public.enroll_course
```

--
Takipleşmeyen userların birbirlerine mesaj atma engeli triggerı

```
create or replace function message_to_connect()
returns trigger
language plpgsql
as
$$

declare
	messageto character;
begin

	select c.connect_by_id into messageto
	from connects c
	where connector_id = new.sender and connect_by_id = new.receiver;

	if (messageto is NULL) then
		raise notice 'takiplesmiyorlar';
		return null;
	else
		return new;
	end if;
end;
$$

create trigger message_control
before insert
on public.message
for each row
execute procedure message_to_connect();

drop trigger message_control on public.message ;
```

---
connect olan userın connect by olması triggerı
```
create or replace function connect_each_other()
returns trigger
language plpgsql
as
$$
declare
        temp1 bpchar(20);
begin
        select * into temp1
        from connects c
        where connector_id=new.connect_by_id and connect_by_id=new.connector_id;

        if (temp1 is  NULL) then
                        INSERT INTO public.connects
                        (connector_id, connect_by_id)
                        VALUES(new.connect_by_id , new.connector_id);
                        return new;
        else
                return null;
        end if;
end;
$$

create trigger connect_to
after insert
on public.connects
for each row
execute procedure connect_each_other();

drop trigger connect_to on public.connects
```

---
unconnect trigger olan kullanıcıların unconnect_by olması triggerı
```
create or replace function unconnect_each_other()
returns trigger
language plpgsql
as
$$
declare
        temp1 character;
begin
        select * into temp1
        from connects c
        where connector_id=old.connect_by_id and connect_by_id=old.connector_id;

        if (temp1 is not  NULL) then
                        DELETE FROM public.connects
                        WHERE connector_id=old.connect_by_id  AND connect_by_id=old.connector_id ;
                        return old;
        else
                return null;
        end if;
end;
$$

create trigger unconnect_to
after delete
```

```
on public.connects
for each row
execute procedure unconnect_each_other();
```

Event yaratıcısının direkt evente eklenmesi triggerı

```
create or replace function creator_join_event()
returns trigger
language plpgsql
as
$$
begin
	 INSERT INTO public.enroll_event
	(user_id, event_id)
	VALUES(new.creator_id, new.event_id);
	return new;
end;
$$

create trigger creato_join
after insert
on public.events
for each row
execute procedure creator_join_event ();
```

Öğrencinin sadece kendi departmanından ders seçimi yapabilmesi triggerı

```
create or replace function same_id_dept_and_studept()
returns trigger
language plpgsql
as
$$
declare
	stu_dep bpchar(20);
declare
	course_dep bpchar(20);
begin
	select dep_id into stu_dep
	from student
	where student_id = new.student_id;

	select department_id into course_dep
	from course c
	where course_code=new.course_code;

	if (stu_dep=course_dep) then
		return new;
	else
		raise notice 'ogrenci kendi departmanindan course secebilir';
		return null;
	end if;
end;
$$

create trigger same_id_control
before insert
on public.enroll_course
for each row
execute procedure same_id_dept_and_studept();
```

--
Eğitmenin kendi departmanından ders verebilmesi triggerı
```
create or replace function same_id_dept_and_insdept()
returns trigger
language plpgsql
as
$$
declare
	ins_dep bpchar(20);
declare
	course_dep bpchar(20);
begin
	select dep_id  into ins_dep
	from instructor
	where instructor_id  = new.ins_id;

	select department_id into course_dep
	from course c
	where course_code=new.course_code;

	if (ins_dep=course_dep) then
		return new;
	else
		raise notice 'egitmen kendi departmanindan course verebilir';
		return null;
	end if;
end;
$$

create trigger same_id_control_ins
before insert
on public.give_course
for each row
execute procedure same_id_dept_and_insdept();
```

--

Eğitmen kendi dersinin projesini verir Triggerı

```
create or replace function same_course_project_ins()
returns trigger
language plpgsql
as
$$
declare
	project_course bpchar(20);

begin

	select course_code into project_course
	from project p
	where project_id = new.project_id and course_code in(select course_code
	from give_course gc
	where ins_id=new.ins_id);

	if(project_course is null) then
		raise notice 'egitmen kendi dersinin projesini verebilir';
		return null;
```

```
        else
                return new;
        end if;

end;
$$

create trigger upload_project_trigger
before insert
on public.upload_project
for each row
execute procedure same_course_project_ins();
```

Öğrencinin kendi dersinin projesini almasını sağlayan trigger

```
create or replace function same_course_project_stu()
returns trigger
language plpgsql
as
$$
declare
        project_course bpchar(20);

begin

        select course_code into project_course
        from project p
        where project_id = new.project_id and course_code in(select course_code
        from enroll_course ec
        where student_id=new.student_id);

        if(project_course is null) then
                raise notice 'ogrenci kendi dersinin projesini alabilir';
                return null;
        else
                return new;
        end if;

end;
$$

create trigger take_project_trigger
before insert
on public.takeproject
for each row
execute procedure same_course_project_stu();
```

Check constraints of LinkMood

```
ALTER TABLE IF EXISTS public.users
    ADD CHECK (age>17)
    NOT VALID;

ALTER TABLE IF EXISTS public.post
    ADD CHECK (post_time<CURRENT_DATE)
    NOT VALID;

ALTER TABLE IF EXISTS public.events
    ADD CHECK (event_time>CURRENT_DATE)
    NOT VALID;

ALTER TABLE IF EXISTS public.users
    ADD CHECK (sex='F' or sex='M')
    NOT VALID;

  ALTER TABLE IF EXISTS public.job
    ADD CHECK (job_type='full-time' or job_type='part-time' or job_type='intern')
    NOT VALID;

  ALTER TABLE IF EXISTS public.student
    ADD CHECK (char_length (student_number) > 10 )
    NOT VALID;

  ALTER TABLE IF EXISTS public.users
    ADD CHECK (char_length (phone) = 11)
    NOT VALID;

  ALTER TABLE IF EXISTS public.company_phone
    ADD CHECK (char_length (phonen) = 11)
    NOT VALID;

   ALTER TABLE IF EXISTS public.connects
    ADD CHECK (connector_id!=connect_by_id)
    NOT VALID;

   ALTER TABLE IF EXISTS public.follow
    ADD CHECK (follower!=follower_by)
    NOT VALID;
```

Insert, Delete and Update statements of LinkMood

```
D INSERT INTO public.users --user tablosune kullanıcı ekledik.
(user_id, phone, sex, email, firstname, lastname, age)
VALUES('11', '5353959815', 'M', '11@gmail.com', 'MURAT', 'ÜNALIR', 40);

UPDATE public.users – yaşını güncelledik
SET  age=41
WHERE user_id='11';

DELETE FROM public.users
WHERE user_id='11';


INSERT INTO public.company
(company_id, company_name, country, city)
VALUES('c100', 'database aş', 'TR', 'izmir');


UPDATE public.company -- database aş firmasını izmirden istanbula taşıdık.
SET   city='İSTABUL'
WHERE company_id='c100';

DELETE FROM public.company
WHERE company_id='c100';

INSERT INTO public.enroll_course --30 idli ögrenci co7 kodlu dersi alıyor
(course_code, student_id)
VALUES('30', 'co7');


DELETE FROM public.enroll_course --33 id li öğrencinin co13 code lu dersi almasını sildik.
WHERE course_code='co13' AND student_id='33';
```

Queries of LinkMood

1.Select statements using one table

```
select * --YAŞI 25TEN BÜYÜK OLAN KULLANICILAR
from users u
where U.age>25;


select *--ÜLKESİ TR OLAN ŞİRKETLER
from company c
where c.country ='TR';

select * --MAX KATILIMCISI 200 VE 200 DEN FAZLA OLAN ETKİNLİKLER
from events e
where e.maximum_participant>=200;
```

2.Select statements using two tables

```
--20 KREDİDEN FAZLA ALAN ÖĞRENCİLERİN BİLGİLERİ
select u.firstname,u.lastname, s.student_number
from student s ,users u
where u.user_id=s.student_id and s.taken_credit>20

--GRUP YÖNETİCİSİ OLAN KADINLAR
select public.users.firstname ,public.users.lastname,public.community.community_name
from public.users,public.community
where public.users.sex='F' and public.community.creator_id=public.users.user_id

--FULL TİME İŞ İLANI VEREN ŞİRKETLER
select public.company.company_name,public.job.job_name,public.job.job_type
from public.company,public.job
where public.job.job_type='full-time' and public.job.company_id =public.company.company_id

-- full STACK DEVELOPER ARAYAN ŞİRKETLER
select public.company.company_name,public.job.job_name
from public.company,public.job
where public.job.job_name ='FULL STACK DEVELOPER' and
public.job.company_id=public.company.company_id
```

3. Select statements using minimum three tables

```
--ŞİRKET SAYFASI YÖNETEN KULLANICILARIN İSİM VE SOY İSİMLERİ
select public.users.firstname ,public.users.lastname,public.company.company_name
from public.users,public.company,public.admins
where public.admins.user_id=public.users.user_id and
public.admins.company_id=public.company.company_id;


--Hem event hemde community yaratıcısı olanlar
select public.users.firstname ,public.users.lastname
,public.events.event_name,public.community.community_name
from public.users ,public.events,public.community
where public.events.creator_id=public.users.user_id and public.community.creator_id
=public.users.user_id


--advisor olan eğitmenler
select distinct u.firstname,u.lastname
from student s , instructor i ,users u
where s.advisor_id =i.instructor_id and i.instructor_id =u.user_id
```

4. Original select statements

```
-- BİLMUH HOCALARININ KATILDIĞI ETKİNLİKLER

select u.firstname ,u.lastname ,e.event_name
from department d ,instructor i ,users u ,enroll_event ee ,events e
where d.dep_id =i.dep_id and u.user_id =i.instructor_id and ee.user_id =u.user_id and d.dname='BİLMUH'
and ee.event_id =e.event_id

-- CALCULUS DERSİ ALANLARIN KATILDIĞI TOPLULUKLAR

select distinct u.firstname ,u.lastname,c.names,c2.community_name
from course c ,student s  ,users u ,enroll_course ec ,join_community jc  ,community c2
where  ec.student_id =s.student_id and s.student_id =u.user_id and ec.course_code=c.course_code and
jc.user_id =u.user_id and jc.community_id = c2.community_id
and c.names ='CALCULUS'

--- VESTELE BAŞVURANLARIN İSİM SOYİSİMLERİ

select u.firstname ,u.lastname
from job j ,apply a  ,users u ,company c
where a.job_name =j.job_name and a.emp_id = u.user_id
and a.company_id =j.company_id and c.company_id =a.company_id and c.company_name ='VESTEL'

---- ADVİSORI MURAT ÜNALIR OLAN ÖĞRENCİLERİN NUMARASI

select s.student_number
from student s ,instructor i ,users u ,users u2
where i.instructor_id =s.advisor_id and u.user_id =i.instructor_id and u.firstname ='MURAT' and
u.lastname='ÜNALIR'
and u2.user_id =s.student_id
```

---Ankara'daki şirketlerin sayfalarını yöneten kullanıcının mail adresi ve yönettiği şirket bilgisi

```
select public.users.email,public.company.company_name
from public.admins,public.company,public.users
where public.company.city='ANKARA' and public.admins.company_id=public.company.company_id and
public.admins.user_id=public.users.user_id
```