2019-2020 FALL SEMESTER

EGE UNIVERSITY
FACULTY of ENGINEERING
COMPUTER ENGINEERING DEPARTMENT
DATABASE MANAGEMENT
2019-2020
LİNKARİYERMOOD

Aytuğ Sevği 05160000539

Beyza Sığınmış 05160000697

Ceren Erdoğan 05160000581

İsmail Gündüz 05170000101

İÇİNDEKİLER

INTRODUCTION TO LINK KARIYER MOOD DATABASE ANALYSIS

KARIYER.NET

Basically, Kariyer.net is a web page that can be used to communicate with members who have uploaded their job search processes to the internet.

LINKEDIN

Linkedin is a web page that has been called a professional business network environment and has carried the network business to the internet world. With this page, we can speed up communication with the people we want to network with.

MOODLE

Moodle provides the service of viewing and publishing the required documentation of the students and the administrators.

Aim of Each Application

Kariyer.net: Kariyer.net is a system in which companies enter their information and become members and share job advertisements to users. From the point of view of the job seeker, kariyer.net: is a system where it can view the sectors it is looking for, upload resumes and receive messages from companies. Apart from the main characters, kariyer.net is a web page that can search, edit a profile and have its own profile page.
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. To talk about the network system; Since this system is installed for network purposes, you offer a CV-like profile service to your connections. Linkedin lets you share a job posting just like Kariyer.net. There are also company pages for users who are interested in company information.
Moodle: Moodle is an open source course management system. 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. This system allows you to upload information to the course, so that you can load assignments, projects; You can communicate with other users through the forums that have been created.
LinkKariyerMood: Our system is an application in which students are in contact with companies for both internship and work, can access company pages and apply directly to job advertisements published by companies. In addition, the content of courses registered in the university; It provides a system in which the students can view the projects of the courses and upload documents to the grades of the exams. From a instructor's point of view, our system offers an application where he can give lessons in the courses opened by the department where he works, and share projects on this course page. For the Company, our application ensures that: it publishes and receives applications; to be able to view the notes, projects, courses taken in the profile information of the students.

Main Entities of Linkedin

MEMBER: It is the part where the members' information is kept.
ORGANIZATIONS: It is where organizational information is kept.
CONNECTIONS: It is where the connection information is kept.
CV: It is the part where a member's cv information is kept.
MEMBER_PROFILE: It is the section where member profile information is kept.
ADRESSES: It is the part where members' address information is kept
GROUP: It is the part where the groups of members are kept.

Characteristics of each entity of Linkedin

MEMBER

- member_id
- address_id
- current_organizatin_id
- date_joined
- date_of_birth
- email_address
- email_password
- first_name
- last_name
- gender

ORGANIZATON

- organization_id
- organization_name

CONNECTION

- connection_id
- connection_member_id
- member_id
- date_connection_made

PEOPLE_BEING_FOLLOWED

- member_id
- member_being_followed_id
- date_started_following
- date_stopped_following

RECOMMENDATION

- member_recommending_id
- member_being_recommended_id
- date_of_recommendation

CV

- cv_id
- member_id
- date_created
- date_updated

MEMBER_PROFILE

- profile_id
- member_id
- date_created
- date_last_updated

MEMBER_GROUP

- member_id
- group_id
- date_joined
- date_left

GROUP

- group_id

- created_by_member_id
- group_name
- group_description
- started_date
- ended_date

ADDRESSES

- adresses_id
- line_1
- line_2
- city
- state_country_province
- zip
- country

Relationships exists among the entities of Linkedin

- MEMBER -> CREATE -> ORGANIZATION
- MEMBER -> CAN -> CONNECTION
- MEMBER -> HAS -> CV
- MEMBER -> HAS -> MEMBER_PROFILE
- MEMBER -> CREATE -> GROUP
- MEMBER-> HAS -> ADDRESSES

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

- A MEMBER can create multiple ORGANIZATION.
- An ORGANIZATION belongs to a MEMBER.
- A MEMBER must create at least one CONNECTION.
- A CONNECTION belongs to a MEMBER.
- A MEMBER must follow at least one MEMBER.
- A MEMBER may be followed by more than one MEMBER.
- A MEMBER must leave RECOMMENDATION to at least one MEMBER.
- A MEMBER can receive RECOMMENDATION by more than one MEMBER.
- A MEMBER must have at least one CV.
- A CV belongs to a MEMBER.
- A MEMBER definitely has MEMBER_PROFIL.
- A MEMBER can have more than one ADDRESS.
- A MEMBER can create more than one GROUP.
- A GROUP u can be created by more than one MEMBER.

Main Entities of Kariyer.net

USER_TYPE: The type of user information is the entity that holds the information.
USER_ACCOUNT: An entity that holds information about the user's account.
USER_LOG: An entity that holds the time the user logs in.
COMPANY: It is the entity that holds the characteristics of a company.
COMPANY_IMAGE: An entity that holds images that a company can have.
BUSINESS_STREAM: A company's workflow is the entity that holds its name.
EDUCATION_DETAIL: An entity that holds a seeker's training information.
EXPERIENCE_DETAIL: It is the entity that holds the experiences of a seeker.
SEEKER_PROFILE: An entity that holds a seeker's profile information.
SEEKER_SKILL_SET: An entity that holds information about a seeker's ability.
SKILL_SET: An entity that holds the name of a seeker's ability.
JOB_POST_ACTIVITY: An entity that holds the referenced dates of a job submission.
JOB_POST: An entity that holds information about a job post.
JOB_POST_SKILL_SET: It is an entity that holds information about the skills that are required to be possessed in a work post.
JOB_TYPE: An entity that provides information about the type of job posting.
JOB_LOCATION: An entity that holds the address information of a job shipment.

Characteristics of each entity of Kariyer.net

USER_TYPE
- id
- user_type_name

USER_ACCOUNT
- id
- user_type_id
- email
- password
- birth
- gender
- sms_notification
- registration_date

USER_LOG
- user_account_id
- last_login_date
- last_job_apply_date

BUSINESS_STREAM
- id
- business_stream_name

COMPANY
- id
- company_name
- profile_description
- business_stream_id
- company_website_url

COMPANY_IMAGE
- id
- company_id
- company_image

EDUCATION_DETAIL
- user_account_id
- major
- institute_university_name
- starting_date
- finish_date
- gpa

SEEKER_PROFILE
- user_account_id
- first_name
- last_name
- current_salary

SEEKER_SKILL_SET
- user_account_id
- skill_set_id
- skill_level

EXPERIENCE_DETAIL
- user_account_id
- is_current_jıb
- start_date
- end_date
- job_tittle
- company_name
- job_location_city
- job_location_state
- job_location_country
- description

SKILL_SET
- id
- skill_set_name

JOB_POST_ACTIVITY
- user_account_id
- job_post_id
- apply_date

JOB_POST
- id
- posted_by_id
- job_type_id
- company_id
- create_date
- company_name
- job_description
- job_location_id

JOB_TYPE
- id
- job_type

JOB_POST_SKILL_SET
- skill_set_id
- job_post_id
- skill_level

JOB_LOCATION
- id
- city
- state
- country
- zip

Relationships exists among the entities of Kariyer.net

- USER_ACCOUNT ->HAVE _-> USER_TYPE
- USER_ACCOUNT -> HAVE -> USER_LOG
- COMPANY -> HAS -> BUSINESS_STREAM
- COMPANY -> HAS -> COMPANY_IMAGE
- SEEKER_PROFILE -> NEED -> EDUCATION_DETAIL
- SEEKER_PROFILE-> HAS -> SEEKER_SKILL_SET
- SEEKER_PROFILE-> HAS -> EXPERIENCE_DETAIL
- SEEKER_SKILL_SET-> HAS -> SKILL_SET
- JOB_POST-> HAVE -> JOB_POST_ACTIVITY
- JOB_POST-> HAVE -> JOB_POST_SKILL_SET
- JOB_POST-> CAN -> JOB_TYPE
- JOB_POST -> AIM -> JOB_LOCATION
- COMPANY-> CAN_PUBLISH -> JOB_POST
- USER_ACCOUNT-> REACH -> JOB_POST
- USER_ACCOUNT-> CAN_BE -> SEEKER_PROFILE

Constraints related to entities, their characteristics and the relationships of Kariyer.net

- A COMPANY can have more than one COMPANY_IMAGE.
- A COMPANY_IMAGE must belong to a COMPANY.
- A BUSINESS_STREAM can be linked to more than one COMPANY.
- A COMPANY must have a BUSINEES_STREAM.
- An EDUCATION_DETAIL belongs to a SEEKER_PROFILE.
- A SEEKER_PROFILE can have more than one EDUCATION_DETAIL.
- An EXPERIENCE_DETAIL must belong to a SEEKER_PROFILE.
- A SEEKER_PROFILE can have multiple EXPERIENCE_DETAILs.
- A SEEKER PROFILE can have more than one SEEKER_SKILL_SET.
- It must be a SEEKER_PROFILE to which a SEEKER_SKILL_SET belongs.
- A SKILL_SET can belong to more than one SEEKER_SKILL_SET.
- If there is a SEEKER_SKILL_SET, it must be SKILL_SET.
- A JOB_POST can have more than one JOB_POST_ACTIVITY.
- A JOB_POST_ACTIVITY must belong to a JOB_POST.
- A JOB_POST can have more than one JOB_POST_SKILL_SET.
- The existence of a JOB_POST_SKILL_SET does not depend on JOB_POST.
- A JOB_LOCATION can exist in more than one JOB_POST.
- A JOB_POST must have a JOB_LOCATION.
- A JOB_POST must have a JOB_TYPE.
- A JOB_TYPE can be linked to more than one JOB_POST.

Main Entities of Moodle

COURSES: It is the entity that holds information about the courses.
SUBJECTS: It is the entity that holds information about the courses.
COURSE_AUTHORS_AND_TUTORS : It is the entity where the subjects of the courses are held.
STUDENT_COURSE_ENROLMENT: It is the entity that establishes the connection with the course to which the student will enroll and contains the password information.
STUDENT_TESTS_TAKEN: It is the entity that holds the information of the tests presented to the students.
STUDENT: It is the entity that holds the characteristics of the student.

Characteristics of each entity of Moodle

COURSES
- course_id
- author_id
- subject_id
- course_name
- course_description
- others

STUDENT
- student_id
- date_of_registration
- date_of_latest_login
- login_name
- passwords
- personal_name
- last_name
- others

COURSE_AUTHORS_AND_TUTORS
- author_id
- login_name
- password
- personal_name
- last_name
- gender
- addres_line
- others

STUDENT_COURSE_ENROLMENT
- registration_id
- course_id
- date_of_enrollment
- date_of_completion

SUBJECTS
- subject_id
- subject_name

STUDENT_TESTS_TAKEN
- registration_id
- test_results
- date_test_taken

Relationships exists among the entities of Moodle

- COURSES -> HAVE -> SUBJECTS
- COURSES -> NEED-> COURSE_AUTHORS_AND_TUTORS
- COURSES -> HAVE -> STUDENT_COURSE_ENROLMENT
- STUDENTS -> ENROLL -> STUDENT_COURSE_ENROLMENT
- STUDENT_TESTS_TAKEN> BELONG -> STUDENT_COURSE_ENROLMENT

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

- A SUBJECT can belong to more than one COURSE.
- A COURSES does not have to be a SUBJECT.
- You need to be a COURSE_AUTHOR_AND_TUTORS of a COURSE.
- A COURSE_AUTHOR_AND_TUTORS can be linked to more than one COURSES.
- A COURSES must have a STUDENT_COURSE_ENROLLMENT.
- A STUDENT_COURSE_ENROLMENT can belong to more than one COURSES.
- A STUDENT_COURSE_ENROLMENT has to have a STUDENT.
- A STUDENT can have more than one STUDENT_COURSE_ENROLLMENT.
- A STUDENT_COURSE_ENROLLMENT contain multiple STUDENT_TESTS_TAKEN.
- A STUDENT_TEST_TAKEN belongs to a STUDENT_COURSE_ENROLLMENT.

Database Link Kariyer Mood

Our system is an application in which students are in contact with companies for both internship and work, can access company pages and apply directly to job advertisements published by companies. In addition, the content of courses registered in the university; It provides a system in which the students can view the projects of the courses and upload documents to the grades of the exams. From a instructor's point of view, our system offers an application where he can give lessons in the courses opened by the department where he works, and share projects on this course page. For the Company, our application ensures that: it publishes and receives applications; to be able to view the notes, projects, courses taken in the profile information of the students.

- This system is a system that companies appeal to students.
- There must be a user to log into the system.
- 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, date of birth, 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 together with message time and content.
- A user can set up a community of multiple users, regardless of type. this community must have a name and a type (for example: student community, company page, university page). 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. Likewise, a community can have more than one user manager. 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, comment time. 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. Just like the user, a community page has the ability to share posts.
- A student can be both licence and graduate students. A student who has graduated from a license can be a master or doctoral student. A student may have graduated from more than one department. A department is related to a university. The university where the student graduated can be more than one. A graduate student can only enroll in graduate courses and an undergraduate student can only enroll in undergraduate courses either.
- An instructor has a user name and instructor type. This instructor type must be a research assistant or faculty member. A student must necessarily have an advisor instructor. If this student is studying at more than one university, he / she will have more than one advisor.
- 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. But research assistants only take part in undergraduate courses. 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 must work in at least one company. There must be at least one employee in a company. An employee has a role and working type 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 information and announcement date. In order to apply for a job, it is necessary to have the date of the day applied, the name of the advertisement and the information of the company to which the advertisement belongs. More than one student can apply for a job advertisement. A student can apply for more than one job. Only students can apply for a job advertisement.

Main Entities of LinkKariyerMood

USER: It is the entity where user's records are kept.
COMPANY: It is the entity where company's records are kept.
JOB_ADVERT: It is the entity where job advertisers records are kept.
LICENCE_STUDENT: It is the entity where licence students records are kept.
GRAD_STUDENT: It is the entity where grad student 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.
GRAD_COURSE: It is the entity where grad courses records are kept.
INSTRUCTOR: It is the entity where instructors records are kept.
POST: It is the entity where posts records are kept.
COMMUNITY: It is the entity where communities records are kept.
SKILL: It is the entity where user's skills are kept.

Characteristics of each entity of LinkKariyerMood

USER
- Username
- Fname
- Lname
- Email
- Bdate
- Sex
- Phones
- Languages
- isEmployee
- isStudent
- isInstructor
- City
- Country
- Creatated_date

UNIVERSITY
- Uni_id
- Uni_name
- Web
- Country
- City
- State
- Zip

COMPANY
- Company_id
- Company_name
- Sector
- Web
- Country
- City
- State
- Zip

JOB_ADVERT
- Advert_name
- Comp_id
- Advert_time
- Content

LICENCE_STUDENT
- Username

GRAD_STUDENT
- Username
- Grad_Student_Type

UNIVERSITY
- Uni_id
- Uni_name
- Web
- Country
- City
- State
- Zip

DEPARTMENT
- Uni_id
- Dept_id
- Dept_name

PROJECT
- Course_id
- Project_name
- Project_content
- Due_date

COURSE
- Course_id
- Course_name
- Course_type
- Faculty_member
- Dept_id

GRAD_COURSE
- Course_id
- Grad_course_type

INSTRUCTOR
- Username
- Instructor_type
- Dept_id

POST
- Post_id
- Post_content
- Poster_user
- Image_url
- Poster_comm
- Post_time
- isEdited

COMMUNITY
- Comm_id
- Comm_name
- Comm_type
- Creater_user

SKILL
- Skill_id
- Skill_name

Relationships exists among the entities of LinkKariyerMood

- USER -> LIKE -> POST
- USER -> COMMENT -> POST
- USER -> CREATE -> POST
- USER -> CREATE -> COMMUNITY
- USER -> JOIN -> COMMUNITY
- USER -> FOLLOW -> USER
- USER -> MESSAGE -> USER
- USER -> HAS -> SKILL
- USER -> ENDORSE -> USER_SKILL
- COMMUNITY -> CREATE -> POST
- USER -> WORKSFOR -> COMPANY
- USER -> ENROLL -> LICENSECOURSE
- USER -> ENROLL -> GRADCOURSE
- USER -> BELONG -> DEPARTMENT
- USER -> DO -> PROJECT
- USER -> APPLICATES -> JOB_ADVERT
- INSTRUCTOR -> WORKSAT -> COMPANY
- INSTRUCTOR -> ADVISE -> STUDENT
- INSTRUCTOR -> ASISTS -> LICENSECOURSE
- INSTRUCTOR -> GIVE -> COURSE
- COURSE -> HAS -> PROJECT
- DEPARTMENT -> BELONG -> UNIVERSITY
- DEPARTMENT -> HAS -> COURSE

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

A USER can LIKE a POST once.

A LIKE belongs to a USER.

A USER can throw 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 COMMUNITY can create multiple POSTs.

A POST belongs only to a COMMUNITY.

A USER can join more than one COMMUNITY.

A COMMUNITY can have multiple member USERs.

A USER can be the ADMINS of multiple COMMUNITY..

A USER can FOLLOW multiple USERs.

A USER can send MESSAGE to multiple USERs.

A USER can have one or more SKILLs.

A USER can ENDORSE another USER’s SKILL.

A USER can WORK FOR in more than one COMPANY.

A COMPANY may contain more than one USER.

An EMPLOYEE has a ROLE in the WORKS COMPANY.

LICENCE STUDENT can enroll in more than one LICENCE COURSE.

More than one LICENCE STUDENT can enroll in a LICENCE COURSE.

GRAD STUDENT can enroll in more than one GRAD COURSE.

More than one GRAD STUDENT can enroll in a GRAD COURSE.

A LICENCE STUDENT or GRAD STUDENT may be connected to at least one or more DEPARTMENTS.

A STUDENT USER can do multiple PROJECT.

More than one STUDENT USER can make a PROJECT.

USER, a STUDENT, can apply to multiple JOB ADVERTs.

More than one STUDENT USER may apply to a JOB ADVERT.

An INSTRUCTOR can work in more than one COMPANY.

More than one INSTRUCTOR can work in a COMPANY.

An INSTRUCTOR can be an ADVISOR to multiple STUDENTS.

A STUDENT, has only one ADVISOR for each department he/she studies at.

As a RESEARCH ASSISTANT, the INSTRUCTOR can be the ASSISTANT of more than one LICENCE COURSE.

A LICENCE COURSE can have RESEARCH ASISTANT, which is more than one INTRUCTOR.

The INSTRUCTOR which is a FACULTY MEMBER can give more than one COURSE.

A COURSE has only the INSTRUCTOR, which is a FACULTY MEMBER.

A COURSE can have multiple PROJECTs.

A PROJECT belongs to only one COURSE.

A DEPARTMENT is connected only to a UNIVERSITY.

A UNIVERSITY may have at least one or more DEPARTMENTS.

There are at least one or more COURSE connected to a DEPARTMENT.

A COURSE is connected to only one DEPARTMENT.

Database Link Kariyer Mood Mapping

1.ITERATION

1.)STEP

COMPANY(company_id, company_name, sector, web, country, city, state, zip)
UNIVERSITY(uni_id, uni_name, web, country, city, state, zip)
DEPARTMENT(dep_id,dep_name)
POST(post_id,text,img_url, post_time, isEdited)
SKILL(skill_id, skill_name)
COMMUNITY(comm_id, comm_name, comm_type)

2.)STEP

JOB_ADVERT(ad_name, comp_id, date, content)
PROJECT(project_name, course_id, project_content)

3.)STEP We don't have any one to one relation.

4.)STEP

POST(...., comm_id)
DEPARTMENT(...., uni_id)

5.)STEP We don't have regular many to many relation.

6.)STEP

COMP_PHONES(comp_id, phone)

7.)STEPWe don't have any N-ary relation.

8.)STEP

d)
USER(user name, fname, lname, email, birth_date, sex, isEmployee, isStudent, isInstructor, city, country, createddate)
c)
COURSE(course_id, course_name, course_type)

2.ITERATION

4.)STEP

COURSE(....,dept_id)
POST(...., poster_user)
COMMUNITY(...., creator_user)

5.)STEP

LIKE(username, post_id)
COMMENT(comment_id, username, post_id, content, comment_time)
COMM_MEMBER(username, comm_id, isAdmin)

FOLLOW(follower, following)
MESSAGE(message_id, sender, receiver, content, messagetime)
USER_SKILL(username, skill_id)
WORKS_FOR(username, comp_id, role, working_type, start_date, end_date)
JOB_APP(username, advert_name, comp_id, app_date)
STUDENT_PROJECT(username, project_name, course_id, grade)
STUDENT_DEPT(username, dept_id, student_no, start_year, grade, gpa, grade_year)
ADVISOR(username, dept_id, ins_user)

6.)STEP

USER_PHONE(username, phone)
USER_LANG(username, language)

7.)STEP

USER_ENDORSE_SKILL(username, skill_id, endorsed_by)

8.)STEP

b)
LICENSE_STUDENT(username)
GRAD_STUDENT(username)
c)
GRAD_COURSE(course_id, grad_course_type)
c)
INSTRUCTOR(username, instructor_type)

3.ITERATION

4.)STEP

COURSE(.....,fac_mem_id)

5.)STEP

LICENCE_ENROLL(username, course_id, midterm, final, makeup, lab)
GRAD_ENROLL(username, course_id, midterm, final, makeup, lab)
ASISTANTSHIP(ins_id, course_id)

8.)STEP

c)
GRAD_STUDENT(....., grad_student_type)

USER

user_name	fname	lname	email	birth_date	sex	isEmployee	isInstructor	isStudent	city	country	createdDate

COMPANY

company_id	company_name	sector	web	country	city	state	zip

UNIVERSITY

uni_id	uni_name	web	country	city	state	zip

DEPARTMENT

dep_id	dep_name	uni_id

POST

post_id	text	img_url	post_time	isEdited	comm_id	poster_user

SKILL

skill_id	skill_name

COMMUNITY

comm_id	comm_name	comm_type	creater_user

JOB_ADVERT

ad_name	comp_id	date	content

PROJECT

project_name	course_id	project_content

COMP_PHONES

comp_id	phone

COURSE

course_id	course_name	course_type	dept_id	fac_mem

LIKE

username	post_id

COMMENT

comment_id	username	post_id	content	comment_time

COMM_MEMBER

username	comm_id	isAdmin

FOLLOW

follower	following

MESSAGE

message_id	sender	receiver	content	messagetime

USER_SKILL

username	skill_id

WORKS_FOR

username	comp_id	role	working_type	start_date	end_date

JOB_APP

username	advert_name	comp_id	app_date

STUDENT_PROJECT

username	project_name	course_id	grade

STUDENT_DEPT

username	dept_id	student_no	start_year	grade	gpa	grade_year

ADVISOR

username	dept_id	ins_user

USER_PHONE

username	phone

USER_LANG

username	language

USER_ENDORSE_SKILL

username	skill_id	endorsed_by

LICENSE_STUDENT

username

GRAD_STUDENT

username	grad_student_type

GRAD_COURSE

course_id	grad_course_type

INSTRUCTOR

username	instructor_type

LICENSE_ENROLL

username	course_id	midterm	final	makeup	lab

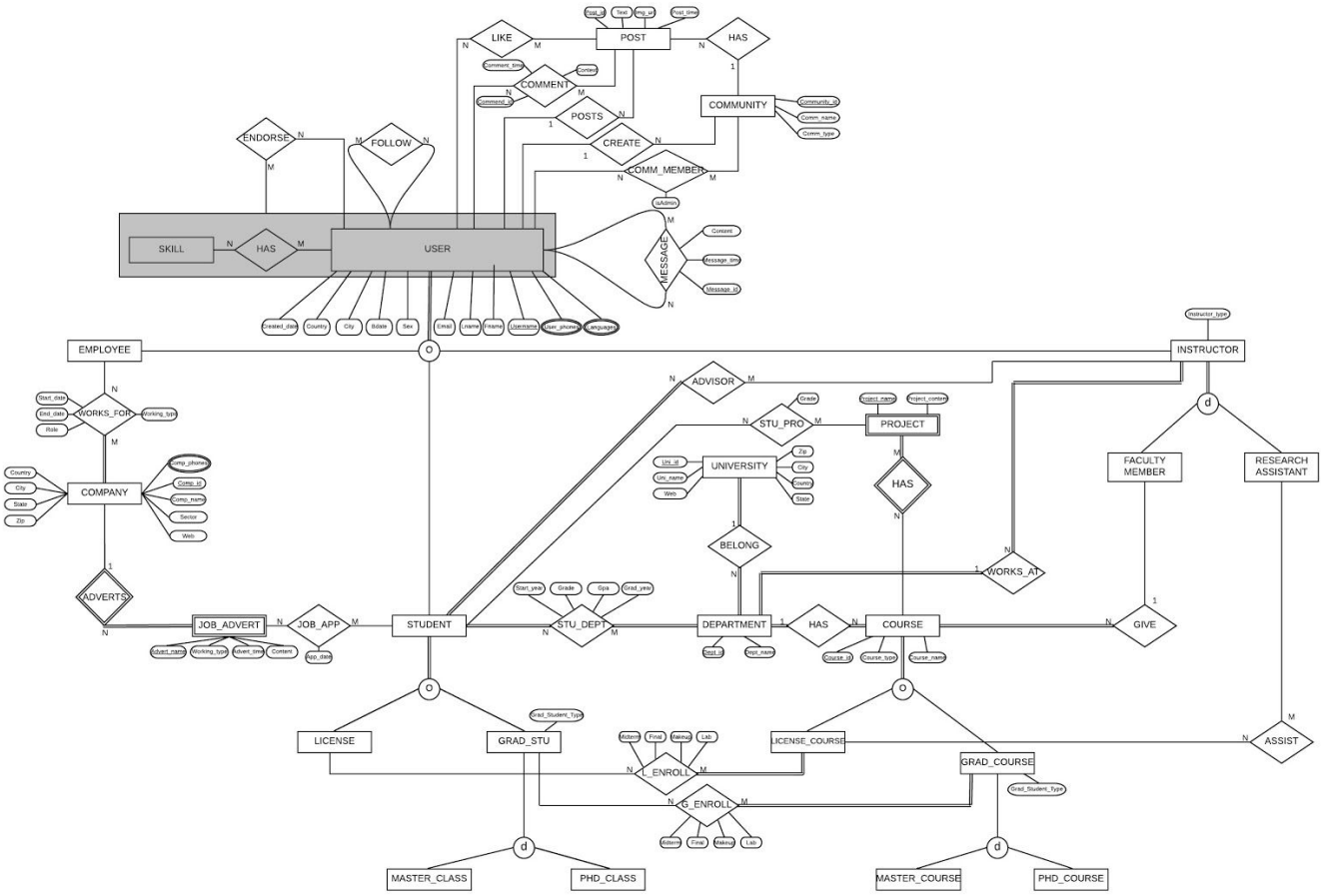
GRAD_ENROLL

username	course_id	midterm	final	makeup	lab

ASISTANTSHIP

ins_id	course_id

Extended Entity Relationship Model of LinkKariyerMood



Create Table of LinkKariyerMood

```
-- Schema linkkariyermood
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `linkkariyermood` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci ;
USE `linkkariyermood` ;

-- -----------------------------------------------------
-- Table `linkkariyermood`.`user`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`user` (
  `username` VARCHAR(16) NOT NULL,
  `fname` VARCHAR(45) NOT NULL,
  `lname` VARCHAR(45) NOT NULL,
  `email` VARCHAR(100) NOT NULL,
  `bdate` DATE NOT NULL,
  `city` VARCHAR(45) NOT NULL,
  `country` VARCHAR(45) NOT NULL,
  `sex` CHAR(1) NOT NULL,
  `isEmployee` (4) NOT NULL,
  `isStudent` (4) NOT NULL,
  `isInstructor` (4) NOT NULL,
  `created_date`  NULL DEFAULT NULL,
  PRIMARY KEY (`username`),
  UNIQUE INDEX `username_UNIQUE` (`username` ASC) VISIBLE,
  UNIQUE INDEX `email_UNIQUE` (`email` ASC) VISIBLE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

-- -----------------------------------------------------
-- Table `linkkariyermood`.`advisor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`advisor` (
  `student_user` VARCHAR(16) NOT NULL,
  `ins_user` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`student_user`, `ins_user`),
  INDEX `ins_user` (`ins_user` ASC) VISIBLE,
  CONSTRAINT `advisor_ibfk_1`
    FOREIGN KEY (`student_user`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----------------------------------------------------
-- Table `linkkariyermood`.`university`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`university` (
  `uni_id` (11) NOT NULL,
  `uni_name` VARCHAR(45) NOT NULL,
  `web` VARCHAR(100) NOT NULL,
  `country` VARCHAR(45) NOT NULL,
  `city` VARCHAR(45) NOT NULL,
  `state` VARCHAR(45) NOT NULL,
  `zip` VARCHAR(11) NOT NULL,
  PRIMARY KEY (`uni_id`),
  UNIQUE INDEX `uni_id_UNIQUE` (`uni_id` ASC) VISIBLE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`department`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`department` (
  `dept_id` (11) NOT NULL,
  `dept_name` VARCHAR(45) NOT NULL,
  `uni_id` (11) NOT NULL,
  PRIMARY KEY (`dept_id`),
  INDEX `department_ibfk_1` (`uni_id` ASC) VISIBLE,
  CONSTRAINT `department_ibfk_1`
    FOREIGN KEY (`uni_id`)
    REFERENCES `linkkariyermood`.`university` (`uni_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`instructor`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`instructor` (
  `username` VARCHAR(16) NOT NULL,
  `instructor_type` VARCHAR(20) NOT NULL,
  `dept_id` (11) NOT NULL,
  PRIMARY KEY (`username`),
  INDEX `instructor_ibfk_2` (`dept_id` ASC) VISIBLE,
  CONSTRAINT `instructor_ibfk_1`
    FOREIGN KEY (`username`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `instructor_ibfk_2`
    FOREIGN KEY (`dept_id`)
    REFERENCES `linkkariyermood`.`department` (`dept_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----------------------------------------------------
-- Table `linkkariyermood`.`course`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`course` (
  `course_id` (11) NOT NULL,
  `course_name` VARCHAR(45) NOT NULL,
  `course_type` VARCHAR(45) NOT NULL,
  `dept_id` (11) NOT NULL,
  `facmem` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`course_id`),
  INDEX `course_ibfk_1` (`dept_id` ASC) VISIBLE,
  INDEX `course_ibfk_2` (`facmem` ASC) VISIBLE,
  CONSTRAINT `course_ibfk_1`
    FOREIGN KEY (`dept_id`)
    REFERENCES `linkkariyermood`.`department` (`dept_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `course_ibfk_2`
    FOREIGN KEY (`facmem`)
    REFERENCES `linkkariyermood`.`instructor` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`assistantship`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`assistantship` (
  `ins_user` VARCHAR(16) NOT NULL,
  `course_id` (11) NOT NULL,
  PRIMARY KEY (`ins_user`, `course_id`),
  INDEX `assistantship_ibfk_2` (`course_id` ASC) VISIBLE,
  CONSTRAINT `assistantship_ibfk_1`
    FOREIGN KEY (`ins_user`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `assistantship_ibfk_2`
    FOREIGN KEY (`course_id`)
    REFERENCES `linkkariyermood`.`course` (`course_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----------------------------------------------------
-- Table `linkkariyermood`.`community`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`community` (
  `community_id` (11) NOT NULL,
  `comm_name` VARCHAR(45) NOT NULL,
  `comm_type` VARCHAR(45) NOT NULL,
  `creator_user` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`community_id`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`comm_member`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`comm_member` (
  `username` VARCHAR(16) NOT NULL,
  `comm_id` (11) NOT NULL,
  `isAdmin` (4) NULL DEFAULT '0',
  PRIMARY KEY (`username`, `comm_id`),
  INDEX `comm_id` (`comm_id` ASC) VISIBLE,
  CONSTRAINT `comm_member_ibfk_1`
    FOREIGN KEY (`username`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `comm_member_ibfk_2`
    FOREIGN KEY (`comm_id`)
    REFERENCES `linkkariyermood`.`community` (`community_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`post`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`post` (
  `post_id` (11) NOT NULL,
  `text` VARCHAR(280) NOT NULL,
  `img_url` VARCHAR(200) NULL DEFAULT NULL,
  `poster_comm` (11) NULL DEFAULT NULL,
  `poster_user` VARCHAR(16) NOT NULL,
  `post_time`  NOT NULL,
  `isEdited` (4) NOT NULL DEFAULT '0',
  PRIMARY KEY (`post_id`),
  INDEX `post_ibfk_2` (`poster_user` ASC) VISIBLE,
  INDEX `poster_comm` (`poster_comm` ASC) VISIBLE,
  CONSTRAINT `post_ibfk_2`
    FOREIGN KEY (`poster_user`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `post_ibfk_3`
    FOREIGN KEY (`poster_comm`)
```

```
    REFERENCES `linkkariyermood`.`community` (`community_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
AUTO_INCREMENT = 6
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`comment`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`comment` (
  `comment_id` (11) NOT NULL,
  `username` VARCHAR(16) NOT NULL,
  `post_id` (11) NOT NULL,
  `content` VARCHAR(280) NOT NULL,
  `comment_time`  NOT NULL,
  PRIMARY KEY (`comment_id`),
  INDEX `comment_ibfk_1` (`username` ASC) VISIBLE,
  INDEX `post_id` (`post_id` ASC) VISIBLE,
  CONSTRAINT `comment_ibfk_1`
    FOREIGN KEY (`username`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `comment_ibfk_2`
    FOREIGN KEY (`post_id`)
     REFERENCES `linkkariyermood`.`post` (`post_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
AUTO_INCREMENT = 4
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`company`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`company` (
  `comp_id` (11) NOT NULL,
  `comp_name` VARCHAR(200) NOT NULL,
  `sector` VARCHAR(45) NOT NULL,
  `web` VARCHAR(100) NOT NULL,
  `country` VARCHAR(45) NOT NULL,
  `city` VARCHAR(45) NOT NULL,
  `state` VARCHAR(45) NOT NULL,
  `zip` VARCHAR(11) NOT NULL,
  PRIMARY KEY (`comp_id`),
  UNIQUE INDEX `comp_id_UNIQUE` (`comp_id` ASC) VISIBLE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----------------------------------------------------
-- Table `linkkariyermood`.`comp_phones`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`comp_phones` (
  `comp_id` (11) NOT NULL,
  `phone` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`comp_id`, `phone`),
  CONSTRAINT `comp_phones_ibfk_1`
    FOREIGN KEY (`comp_id`)
    REFERENCES `linkkariyermood`.`company` (`comp_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`follow`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`follow` (
  `follower` VARCHAR(16) NOT NULL,
  `following` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`follower`, `following`),
  INDEX `follow_ibfk_2` (`following` ASC) VISIBLE,
  CONSTRAINT `follow_ibfk_1`
    FOREIGN KEY (`follower`)
      REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `follow_ibfk_2`
    FOREIGN KEY (`following`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`grad_course`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`grad_course` (
  `course_id` (11) NOT NULL,
  `grad_course_type` VARCHAR(6) NOT NULL,
  PRIMARY KEY (`course_id`),
  CONSTRAINT `grad_course_ibfk_1`
    FOREIGN KEY (`course_id`)
    REFERENCES `linkkariyermood`.`course` (`course_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----------------------------------------------------
-- Table `linkkariyermood`.`grad_student`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`grad_student` (
  `username` VARCHAR(16) NOT NULL,
  `grad_student_type` VARCHAR(6) NOT NULL,
  PRIMARY KEY (`username`),
  CONSTRAINT `grad_student_ibfk_1`
    FOREIGN KEY (`username`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`grad_enroll`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`grad_enroll` (
  `username` VARCHAR(16) NOT NULL,
  `course_id` (11) NOT NULL,
  `midterm` (3) NULL DEFAULT NULL,
  `final` (3) NULL DEFAULT NULL,
  `makeup` (3) NULL DEFAULT NULL,
  `lab` (3) NULL DEFAULT NULL,
  PRIMARY KEY (`username`, `course_id`),
  INDEX `grad_enroll_ibfk_2` (`course_id` ASC) VISIBLE,
  CONSTRAINT `grad_enroll_ibfk_1`
    FOREIGN KEY (`username`)
    REFERENCES `linkkariyermood`.`grad_student` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `grad_enroll_ibfk_2`
    FOREIGN KEY (`course_id`)
    REFERENCES `linkkariyermood`.`grad_course` (`course_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`job_advert`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`job_advert` (
  `advert_name` VARCHAR(100) NOT NULL,
  `comp_id` (11) NOT NULL,
  `working_type` VARCHAR(45) NOT NULL,
  `advert_time`  NOT NULL,
  `content` VARCHAR(280) NOT NULL,
  PRIMARY KEY (`advert_name`, `comp_id`),
  INDEX `job_advert_ibfk_1` (`comp_id` ASC) VISIBLE,
  CONSTRAINT `job_advert_ibfk_1`
    FOREIGN KEY (`comp_id`)
    REFERENCES `linkkariyermood`.`company` (`comp_id`)
    ON DELETE CASCADE
```

```
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`job_app`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`job_app` (
  `username` VARCHAR(16) NOT NULL,
  `advert_name` VARCHAR(100) NOT NULL,
  `comp_id` (11) NOT NULL,
  `app_date` DATE NOT NULL,
  PRIMARY KEY (`username`, `advert_name`, `comp_id`),
  INDEX `job_app_ibfk_2` (`advert_name` ASC, `comp_id` ASC) VISIBLE,
  CONSTRAINT `job_app_ibfk_1`
    FOREIGN KEY (`username`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `job_app_ibfk_2`
    FOREIGN KEY (`advert_name` , `comp_id`)
    REFERENCES `linkkariyermood`.`job_advert` (`advert_name` , `comp_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`license_student`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`license_student` (
  `username` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`username`),
  CONSTRAINT `license_student_ibfk_1`
    FOREIGN KEY (`username`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`license_enroll`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`license_enroll` (
  `username` VARCHAR(16) NOT NULL,
  `course_id` (11) NOT NULL,
  `midterm` (3) NULL DEFAULT NULL,
  `final` (3) NULL DEFAULT NULL,
  `makeup` (3) NULL DEFAULT NULL,
  `lab` (3) NULL DEFAULT NULL,
  PRIMARY KEY (`username`, `course_id`),
  INDEX `license_enroll_ibfk_2` (`course_id` ASC) VISIBLE,
  CONSTRAINT `license_enroll_ibfk_1`
    FOREIGN KEY (`username`)
```

```
    REFERENCES `linkkariyermood`.`license_student` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `license_enroll_ibfk_2`
    FOREIGN KEY (`course_id`)
    REFERENCES `linkkariyermood`.`course` (`course_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`like`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`like` (
  `username` VARCHAR(16) NOT NULL,
  `post_id` (11) NOT NULL,
  PRIMARY KEY (`username`, `post_id`),
  INDEX `post_id` (`post_id` ASC) VISIBLE,
  CONSTRAINT `like_ibfk_1`
    FOREIGN KEY (`username`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `like_ibfk_2`
    FOREIGN KEY (`post_id`)
    REFERENCES `linkkariyermood`.`post` (`post_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`message`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`message` (
  `message_id` (11) NOT NULL,
  `sender` VARCHAR(16) NOT NULL,
  `receiver` VARCHAR(16) NOT NULL,
  `content` VARCHAR(280) NOT NULL,
  `message_time`  NOT NULL,
  PRIMARY KEY (`message_id`),
  INDEX `message_ibfk_1` (`sender` ASC) VISIBLE,
  INDEX `message_ibfk_2` (`receiver` ASC) VISIBLE,
  CONSTRAINT `message_ibfk_1`
    FOREIGN KEY (`sender`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `message_ibfk_2`
    FOREIGN KEY (`receiver`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
```

```
AUTO_INCREMENT = 22
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`project`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`project` (
  `project_name` VARCHAR(45) NOT NULL,
  `course_id` (11) NOT NULL,
  `project_content` VARCHAR(1000) NOT NULL,
  PRIMARY KEY (`project_name`, `course_id`),
  INDEX `project_ibfk_1` (`course_id` ASC) VISIBLE,
  CONSTRAINT `project_ibfk_1`
    FOREIGN KEY (`course_id`)
    REFERENCES `linkkariyermood`.`course` (`course_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`skill`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`skill` (
  `skill_id` (11) NOT NULL,
  `skill_name` VARCHAR(200) NOT NULL,
  PRIMARY KEY (`skill_id`))
ENGINE = InnoDB
AUTO_INCREMENT = 52
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`student_dept`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`student_dept` (
  `username` VARCHAR(16) NOT NULL,
  `dept_id` (11) NOT NULL,
  `student_no` (11) NOT NULL,
  `start_year` YEAR(4) NOT NULL,
  `grade` (1) NOT NULL,
  `gpa` DOUBLE NOT NULL DEFAULT '0',
  `grad_year` YEAR(4) NULL DEFAULT NULL,
  PRIMARY KEY (`username`, `dept_id`),
  INDEX `student_dept_ibfk_2` (`dept_id` ASC) VISIBLE,
  CONSTRAINT `student_dept_ibfk_1`
    FOREIGN KEY (`username`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `student_dept_ibfk_2`
    FOREIGN KEY (`dept_id`)
    REFERENCES `linkkariyermood`.`department` (`dept_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
```

```
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`student_project`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`student_project` (
  `username` VARCHAR(16) NOT NULL,
  `project_name` VARCHAR(45) NOT NULL,
  `course_id` (11) NOT NULL,
  `grade` (3) NULL DEFAULT NULL,
  PRIMARY KEY (`username`, `course_id`, `project_name`),
  INDEX `student_project_ibfk_2` (`project_name` ASC, `course_id` ASC) VISIBLE,
  CONSTRAINT `student_project_ibfk_1`
    FOREIGN KEY (`username`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `student_project_ibfk_2`
    FOREIGN KEY (`project_name` , `course_id`)
    REFERENCES `linkkariyermood`.`project` (`project_name` , `course_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`user_skill`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`user_skill` (
  `username` VARCHAR(16) NOT NULL,
  `skill_id` (11) NOT NULL,
  PRIMARY KEY (`username`, `skill_id`),
  INDEX `skill_id` (`skill_id` ASC) VISIBLE,
  CONSTRAINT `user_skill_ibfk_1`
    FOREIGN KEY (`username`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `user_skill_ibfk_2`
    FOREIGN KEY (`skill_id`)
    REFERENCES `linkkariyermood`.`skill` (`skill_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----------------------------------------------------
-- Table `linkkariyermood`.`user_endorsed_skill`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`user_endorsed_skill` (
  `username` VARCHAR(16) NOT NULL,
  `skill_id` (11) NOT NULL,
  `endorsed_by` VARCHAR(16) NOT NULL,
  PRIMARY KEY (`username`, `skill_id`, `endorsed_by`),
  INDEX `endorsed_by` (`endorsed_by` ASC) VISIBLE,
  CONSTRAINT `user_endorsed_skill_ibfk_1`
    FOREIGN KEY (`username` , `skill_id`)
    REFERENCES `linkkariyermood`.`user_skill` (`username` , `skill_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `user_endorsed_skill_ibfk_2`
    FOREIGN KEY (`endorsed_by`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`user_lang`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`user_lang` (
  `username` VARCHAR(16) NOT NULL,
  `lang` VARCHAR(45) NOT NULL,
  PRIMARY KEY (`username`, `lang`),
  CONSTRAINT `user_lang_ibfk_1`
    FOREIGN KEY (`username`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
-- -----------------------------------------------------
-- Table `linkkariyermood`.`user_phones`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`user_phones` (
  `username` VARCHAR(16) NOT NULL,
  `phone` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`username`, `phone`),
  CONSTRAINT `user_phones_ibfk_1`
    FOREIGN KEY (`username`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;
```

```
-- -----------------------------------------------------
-- Table `linkkariyermood`.`works_for`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `linkkariyermood`.`works_for` (
  `username` VARCHAR(16) NOT NULL,
  `comp_id` (11) NOT NULL,
  `role` VARCHAR(100) NOT NULL,
  `working_type` VARCHAR(45) NOT NULL,
  `start_date` DATE NOT NULL,
  `end_date` DATE NULL DEFAULT NULL,
  PRIMARY KEY (`username`, `comp_id`, `start_date`),
  INDEX `works_for_ibfk_2` (`comp_id` ASC) VISIBLE,
  CONSTRAINT `works_for_ibfk_1`
    FOREIGN KEY (`username`)
    REFERENCES `linkkariyermood`.`user` (`username`)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
  CONSTRAINT `works_for_ibfk_2`
    FOREIGN KEY (`comp_id`)
    REFERENCES `linkkariyermood`.`company` (`comp_id`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8mb4
COLLATE = utf8mb4_0900_ai_ci;

USE `linkkariyermood` ;
```

Assertion of LinkKariyerMood

It is written to prevent users under the age of 18 from entering the system.

```
CREATE
TRIGGER `linkkariyermood`.`check_bDateOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`user`
FOR EACH ROW
BEGIN
        IF(NOT(YEAR(NEW.bdate) < (YEAR(CURRENT_TIMESTAMP) - 17))) THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: 18 yaşından küçükler sisteme
        kaydolamaz.';
        END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_bDateOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`user`
FOR EACH ROW
BEGIN
        IF(NOT(YEAR(NEW.bdate) < (YEAR(CURRENT_TIMESTAMP) - 17))) THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: 18 yaşından küçükler sisteme
        kaydolamaz.';
        END IF;
END
```

One student is prevented from having more than one advisor while studying in a department.

```
CREATE
TRIGGER `linkkariyermood`.`check_advisorDeptDuplicate`
BEFORE INSERT ON `linkkariyermood`.`advisor`
FOR EACH ROW
BEGIN
        IF (NEW.student_user IN(
   SELECT advisor.student_user
   FROM advisor, instructor
   WHERE advisor.ins_user IN(
                        SELECT instructor.username
                        FROM instructor
                        WHERE instructor.dept_id in (
                                        SELECT instructor.dept_id
            FROM instructor
            WHERE instructor.username = new.ins_user)))) THEN
     SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Bir öğrencinin aynı departmanda 2 farklı danışmanı olamaz.';
   END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_advisorDeptDuplicateOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`advisor`
FOR EACH ROW
BEGIN
        IF (NEW.student_user IN(
  SELECT advisor.student_user
  FROM advisor, instructor
  WHERE advisor.ins_user IN(
                        SELECT instructor.username
                        FROM instructor
                        WHERE instructor.dept_id in (
                                        SELECT instructor.dept_id
          FROM instructor
          WHERE instructor.username = new.ins_user)))) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Bir öğrencinin aynı departmanda 2 farklı danışmanı olamaz.';
  END IF;
END
```

A student's advisor in the department to which he / she is affiliated must also be affiliated to the same department.

```
CREATE
TRIGGER `linkkariyermood`.`check_studentInsEqualDept`
BEFORE INSERT ON `linkkariyermood`.`advisor`
FOR EACH ROW
BEGIN
        IF (NEW.student_user NOT IN(
  SELECT sd.username
  FROM student_dept AS sd, instructor AS ins
  WHERE NEW.ins_user=ins.username AND ins.dept_id = sd.dept_id )) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Öğrenci ile danışmanı aynı departmanda bulunmalıdır';
  END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_studentInsEqualDeptOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`advisor`
FOR EACH ROW
BEGIN
        IF (NEW.student_user NOT IN(
  SELECT sd.username
  FROM student_dept AS sd, instructor AS ins
  WHERE NEW.ins_user=ins.username AND ins.dept_id = sd.dept_id )) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Öğrenci ile danışmanı aynı departmanda bulunmalıdır';
  END IF;
END
```

Checks whether the user is an instructor.

```
CREATE
TRIGGER `linkkariyermood`.`check_isInstructor`
BEFORE INSERT ON `linkkariyermood`.`instructor`
FOR EACH ROW
BEGIN
IF (NEW.username IN(SELECT username FROM user WHERE isInstructor=0)) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Bu kullanıcı öğretim görevlisi değil';
  END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_isInstructorOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`instructor`
FOR EACH ROW
BEGIN
   IF (NEW.username IN(SELECT username FROM user WHERE isInstructor=0)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Bu kullanıcı öğretim görevlisi değil';
   END IF;
END
```

Checks whether the lecturer is working in the department.

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `linkkariyermood`.`check_coursedept`
BEFORE INSERT ON `linkkariyermood`.`course`
FOR EACH ROW
BEGIN
            IF (NEW.dept_id NOT IN(
   SELECT i.dept_id
   FROM instructor AS i
   WHERE NEW.facmem = i.username)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Dersi verecek kişi dersin olduğu departmanda çalışmalıdır.';
   END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_coursedeptOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`course`
FOR EACH ROW
BEGIN
            IF (NEW.dept_id NOT IN(
   SELECT i.dept_id
   FROM instructor AS i
   WHERE NEW.facmem = i.username)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Dersi verecek kişi dersin olduğu departmanda çalışmalıdır.';
   END IF;
END
```

The type of instructor should be faculty member.

```
CREATE
TRIGGER `linkkariyermood`.`check_facmem`
BEFORE INSERT ON `linkkariyermood`.`course`
FOR EACH ROW
BEGIN
            IF (NEW.facmem NOT IN(
   SELECT i.username
   FROM instructor AS i
   WHERE NEW.facmem = i.username AND i.instructor_type="faculty_member")) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Dersi verecek öğretim üyesinin tipi faculty_member olmalıdır.';
   END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_facmemOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`course`
FOR EACH ROW
BEGIN
        IF (NEW.facmem NOT IN(
   SELECT i.username
   FROM instructor AS i
   WHERE NEW.facmem = i.username AND i.instructor_type="faculty_member")) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Dersi verecek öğretim üyesinin tipi faculty_member olmalıdır.';
   END IF;
END
```

Only research assistants can assist.

```
CREATE
TRIGGER `linkkariyermood`.`check_assistant`
BEFORE INSERT ON `linkkariyermood`.`assistantship`
FOR EACH ROW
BEGIN
   IF (NEW.ins_user NOT IN(
   SELECT username
   FROM instructor AS i
   WHERE NEW.ins_user = i.username AND i.instructor_type = "research_assistant")) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Asistanlık yapacak öğretim üyesinin tipi araştırma görevlisi
olmalıdır.';
   END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_assistantOnUpdt`
BEFORE UPDATE ON `linkkariyermood`.`assistantship`
FOR EACH ROW
BEGIN
  IF (NEW.ins_user NOT IN(
   SELECT username
   FROM instructor AS i
   WHERE NEW.ins_user = i.username AND i.instructor_type = "research_assistant")) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Asistanlık yapacak öğretim üyesinin tipi araştırma
görevlisi olmalıdır.';
   END IF;
END
```

Research assistants should assist in the department.

```
CREATE
TRIGGER `linkkariyermood`.`check_assistantCourse`
BEFORE INSERT ON `linkkariyermood`.`assistantship`
FOR EACH ROW
BEGIN
        IF (NEW.course_id NOT IN(
   SELECT course.course_id
   FROM course, instructor AS i
   WHERE course.dept_id = i.dept_id)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Araştırma görevlisi çalışmadığı departmanda asistanlık
yapamaz.';
   END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_assistantCourseOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`assistantship`
FOR EACH ROW
BEGIN
        IF (NEW.course_id NOT IN(
   SELECT course.course_id
   FROM course, instructor AS i
   WHERE course.dept_id = i.dept_id)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Araştırma görevlisi çalışmadığı departmanda
asistanlık yapamaz.';
   END IF;
END
```

The data of the creator of the community cannot be updated.

```
CREATE
TRIGGER `linkkariyermood`.`editCreator`
BEFORE UPDATE ON `linkkariyermood`.`community`
FOR EACH ROW
BEGIN
        IF(NOT(new.creator_user=old.creator_user)) then
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = "HATA: Topluluğu oluşturan kullanıcı değiştirilemez.";
        END IF;
END
```

Only community admins can share posts.

```
CREATE
TRIGGER `linkkariyermood`.`check_isPosterAdmin`
BEFORE INSERT ON `linkkariyermood`.`post`
FOR EACH ROW
BEGIN
        IF(NOT(NEW.poster_comm is null)) THEN
                IF (NEW.poster_user NOT IN(
    SELECT c.username
    FROM comm_member as c, user
    WHERE user.username = c.username AND c.comm_id = NEW.poster_comm
    )) THEN
                        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Gönderiyi paylaşan kişi topluluğun
yöneticisi değil.';
                END IF;
        END IF;
END
```

Comments cannot be updated.

```
CREATE
TRIGGER `linkkariyermood`.`editcomment`
BEFORE UPDATE ON `linkkariyermood`.`comment`
FOR EACH ROW
BEGIN
        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Yorum Güncellenemez';
END
```

The user cannot follow itself. This was prevented.

```
CREATE
TRIGGER `linkkariyermood`.`check_followingEqualsFollower`
BEFORE INSERT ON `linkkariyermood`.`follow`
FOR EACH ROW
BEGIN
	IF(NEW.following=NEW.follower) THEN
		SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Kullanıcı kendini takip edemez.';
	END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
DEFINER=`root`@`localhost`
TRIGGER `linkkariyermood`.`check_followingEqualsFollowerOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`follow`
FOR EACH ROW
BEGIN
	IF(NEW.following=NEW.follower) THEN
		SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Kullanıcı kendini takip edemez.';
	END IF;
END
```

License courses cannot be entered into the Grad course.

```
CREATE
TRIGGER `linkkariyermood`.`check_courseType`
BEFORE INSERT ON `linkkariyermood`.`grad_course`
FOR EACH ROW
BEGIN
	IF (NEW.course_id  IN(
   SELECT course_id
   FROM course
   WHERE course.course_type = "license" )) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Lisans Dersi Girişi Yapılamaz.';
   END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_courseTypeOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`grad_course`
FOR EACH ROW
BEGIN
	IF (NEW.course_id  IN(
   SELECT course_id
   FROM course
   WHERE course.course_type = "license" )) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Lisans Dersi Girişi Yapılamaz.';
   END IF;
END
```

For Master or Phd courses, a graduate degree is required.

```
CREATE
TRIGGER `linkkariyermood`.`check_isGrad`
BEFORE INSERT ON `linkkariyermood`.`grad_student`
FOR EACH ROW
BEGIN
        IF (NEW.username  IN(
   SELECT student_dept.username
   FROM student_dept
   WHERE grad_year is null )) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Master veya Phd dersleri için önce lisans mezunu olması
gerekir.';
   END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_isGradOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`grad_student`
FOR EACH ROW
BEGIN
        IF (NEW.username  IN(
   SELECT student_dept.username
   FROM student_dept
   WHERE grad_year is null )) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Master veya Phd dersleri için önce lisans mezunu olması
gerekir.';
   END IF;
END
```

Checks whether the user is an student.

```
CREATE
TRIGGER `linkkariyermood`.`check_isStudent2`
BEFORE INSERT ON `linkkariyermood`.`grad_student`
FOR EACH ROW
BEGIN
   IF (NEW.username IN(SELECT username FROM user WHERE isStudent=0)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Bu kullanıcı öğrenci değil';
   END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_isStudent2OnUpd`
BEFORE UPDATE ON `linkkariyermood`.`grad_student`
FOR EACH ROW
BEGIN
   IF (NEW.username IN(SELECT username FROM user WHERE isStudent=0)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Bu kullanıcı öğrenci değil';
   END IF;
END
```

The department in which the course is given and the department to which the student is affiliated must be the same.

```
CREATE
TRIGGER `linkkariyermood`.`check_userCourseEqualDept`
BEFORE INSERT ON `linkkariyermood`.`grad_enroll`
FOR EACH ROW
BEGIN
        IF (NEW.course_id NOT IN(
  SELECT course.course_id
  FROM student_dept, course
  WHERE  NEW.username = student_dept.username AND student_dept.dept_id = course.dept_id
                )) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Course ile öğrenci aynı departmana bağlı olmalıdır.';
  END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_userCourseEqualDeptOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`grad_enroll`
FOR EACH ROW
BEGIN
        IF (NEW.course_id NOT IN(
  SELECT course.course_id
  FROM student_dept, course
  WHERE  NEW.username = student_dept.username AND student_dept.dept_id = course.dept_id
                )) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Course ile öğrenci aynı departmana bağlı olmalıdır.';
  END IF;
END
```

Only grad student students can take Grad course classes.

```
CREATE
TRIGGER `linkkariyermood`.`check_userCourseType`
BEFORE INSERT ON `linkkariyermood`.`grad_enroll`
FOR EACH ROW
BEGIN
  IF (NEW.username NOT IN(
   SELECT grad_student.username
   FROM grad_student, grad_course
   WHERE NEW.course_id=grad_course.course_id
                AND grad_course.grad_course_type = grad_student.grad_student_type)) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Öğrenci kendi seviyesindeki kursu alabilir.';
  END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_userCourseTypeOnUpdt`
BEFORE UPDATE ON `linkkariyermood`.`grad_enroll`
FOR EACH ROW
BEGIN
  IF (NEW.username NOT IN(
   SELECT grad_student.username
   FROM grad_student, grad_course
   WHERE NEW.course_id=grad_course.course_id
                AND grad_course.grad_course_type = grad_student.grad_student_type)) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Öğrenci kendi seviyesindeki kursu alabilir.';
  END IF;
END
```

Only students can apply for a job.

```
CREATE
TRIGGER `linkkariyermood`.`check_isStudent3`
BEFORE INSERT ON `linkkariyermood`.`job_app`
FOR EACH ROW
BEGIN
   IF (NEW.username IN(SELECT username FROM user WHERE isStudent=0)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: İş Başvurusuna Sadece Öğrenciler Başvurabilir';
   END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_isStudent3OnUpd`
BEFORE UPDATE ON `linkkariyermood`.`job_app`
FOR EACH ROW
BEGIN
   IF (NEW.username IN(SELECT username FROM user WHERE isStudent=0)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: İş Başvurusuna Sadece Öğrenciler Başvurabilir';
   END IF;
END
```

Checks whether the user is an student.

```
CREATE
TRIGGER `linkkariyermood`.`check_isStudent`
BEFORE INSERT ON `linkkariyermood`.`license_student`
FOR EACH ROW
BEGIN
   IF (NEW.username IN(SELECT username FROM user WHERE isStudent=0)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Bu kullanıcı öğrenci değil';
   END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_isStudentOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`license_student`
FOR EACH ROW
BEGIN
   IF (NEW.username IN(SELECT username FROM user WHERE isStudent=0)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Bu kullanıcı öğrenci değil';
   END IF;
END
```

The department in which the course is given and the department to which the student is affiliated must be the same.

```
CREATE
TRIGGER `linkkariyermood`.`check_licenseCourseEqualDept`
BEFORE INSERT ON `linkkariyermood`.`license_enroll`
FOR EACH ROW
BEGIN
            IF (NEW.username NOT IN(
   SELECT student_dept.username
   FROM student_dept, course
   WHERE new.course_id = course.course_id and course.dept_id = student_dept.dept_id)) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Course ile öğrenci aynı departmana bağlı olmalıdır.';
   END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_licenseCourseEqualDeptOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`license_enroll`
FOR EACH ROW
BEGIN
        IF (NEW.username NOT IN(
                SELECT student_dept.username
                FROM student_dept, course
                WHERE new.course_id = course.course_id and course.dept_id = student_dept.dept_id)) THEN
                        SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Course ile öğrenci aynı departmana
bağlı olmalıdır.';
   END IF;
END
```

Only license student students can take license course classes.

```
CREATE
TRIGGER `linkkariyermood`.`check_licenseCourseType`
BEFORE INSERT ON `linkkariyermood`.`license_enroll`
FOR EACH ROW
BEGIN
        IF (NEW.course_id NOT IN(
   SELECT course.course_id
    FROM student_dept, course
   WHERE NEW.username=student_dept.username
                AND student_dept.dept_id= course.dept_id AND course.course_type = "license")) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Öğrenci kendi seviyesindeki kursu alabilir.';
   END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_licenseCourseTypeOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`license_enroll`
FOR EACH ROW
BEGIN
        IF (NEW.course_id NOT IN(
    SELECT course.course_id
   FROM student_dept, course
   WHERE NEW.username=student_dept.username
                AND student_dept.dept_id= course.dept_id AND course.course_type = "license")) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Öğrenci kendi seviyesindeki kursu alabilir.';
   END IF;
END
```

The user cannot send messages itself. This was prevented.

```
CREATE
TRIGGER `linkkariyermood`.`check_senderEqualsReceiver`
BEFORE INSERT ON `linkkariyermood`.`message`
FOR EACH ROW
BEGIN
        IF(NEW.sender=NEW.receiver) THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Kullanıcı kendine mesaj gönderemez.';
        END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_senderEqualsReceiverOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`message`
FOR EACH ROW
BEGIN
	IF(NEW.sender=NEW.receiver) THEN
		SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Kullanıcı kendine mesaj gönderemez.';
	END IF;
END
```

Message information and message content cannot be updated.

```
CREATE
TRIGGER `linkkariyermood`.`editMessage`
BEFORE UPDATE ON `linkkariyermood`.`message`
FOR EACH ROW
BEGIN
	IF(NOT((NEW.content=OLD.content)and(new.sender=old.sender)and(new.receiver=old.receiver))) THEN
		SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Mesaj bilgileri ve içeriği
değiştirilemez.';
	END IF;
END
```

Checks that students at the same university have different numbers.

```
CREATE
TRIGGER `linkkariyermood`.`check_studentnoCantEqualSameUni`
BEFORE INSERT ON `linkkariyermood`.`student_dept`
FOR EACH ROW
BEGIN
	IF (NEW.student_no IN(
   SELECT student_dept.student_no
   FROM department, student_dept, university
   WHERE student_dept.dept_id = department.dept_id AND department.uni_id IN(SELECT department.uni_id
                                                                             FROM department
                                                                             WHERE NEW.dept_id=department.dept_id AND department.uni_id =
university.uni_id))) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Aynı üniversitedeki öğrencilerin numarası farklı olmalıdır.';
   END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_studentnoCantEqualSameUniOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`student_dept`
FOR EACH ROW
BEGIN
	IF (NEW.student_no IN(
   SELECT student_dept.student_no
   FROM department, student_dept, university
   WHERE student_dept.dept_id = department.dept_id AND department.uni_id IN(SELECT department.uni_id
                                                                             FROM department
                                                                             WHERE NEW.dept_id=department.dept_id AND department.uni_id =
university.uni_id))) THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Aynı üniversitedeki öğrencilerin numarası farklı olmalıdır.';
   END IF;
END
```

Only license students can do license project.
Only grad students can do grad project.

```
CREATE
TRIGGER `linkkariyermood`.`check_isEnrollCourse`
BEFORE INSERT ON `linkkariyermood`.`student_project`
FOR EACH ROW
BEGIN
IF (NEW.course_id IN(SELECT course.course_id FROM course WHERE course.course_type="license")) THEN
        IF (NEW.username NOT IN(
  SELECT license_enroll.username
  FROM license_enroll
  WHERE NEW.course_id = license_enroll.course_id )) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Öğrenci Almadığı Dersin Projesini Yapamaz.';
  END IF;
END IF;
IF (NEW.course_id IN(SELECT course.course_id FROM course WHERE course.course_type="grad")) THEN
  IF (NEW.username NOT IN(
  SELECT grad_enroll.username
  FROM grad_enroll
  WHERE NEW.course_id = grad_enroll.course_id )) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Öğrenci Almadığı Dersin Projesini Yapamaz.';
  END IF;
END IF;
END
```

Checks whether the user is an employee.

```
CREATE
TRIGGER `linkkariyermood`.`check_isEmployee`
BEFORE INSERT ON `linkkariyermood`.`works_for`
FOR EACH ROW
BEGIN
  IF (NEW.username IN(SELECT username FROM user WHERE isEmployee=0)) THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Bu kullanıcı çalışan değil';
  END IF;
END
```

Triggers of LinkKariyerMood

The user's creation time must be the same as the current time.

```
CREATE
TRIGGER `linkkariyermood`.`check_createdtime`
BEFORE INSERT ON `linkkariyermood`.`user`
FOR EACH ROW
BEGIN
    SET NEW.created_date = CURRENT_TIMESTAMP;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_updatecreatedtime`
BEFORE UPDATE ON `linkkariyermood`.`user`
FOR EACH ROW
BEGIN
    SET NEW.created_date = OLD.created_date;
END
```

If the user is a community name, that value needs to be updated.

```
CREATE
TRIGGER `linkkariyermood`.`addMember`
AFTER INSERT ON `linkkariyermood`.`community`
FOR EACH ROW
BEGIN
        INSERT INTO `linkkariyermood`.`comm_member` (`username`, `comm_id`, `isAdmin`) VALUES (new.creator_user,
new.community_id, 1);
END
```

Checks if a post has been updated.

```
CREATE
TRIGGER `linkkariyermood`.`editpost`
BEFORE UPDATE ON `linkkariyermood`.`post`
FOR EACH ROW
BEGIN
        SET NEW.post_id = OLD.post_id;
  SET NEW.poster_comm = OLD.poster_comm;
  SET NEW.poster_user = OLD.poster_user;
  SET NEW.post_time = OLD.post_time;
  IF(NOT((NEW.text=OLD.text) AND (NEW.img_url=OLD.img_url))) THEN
            SET NEW.isEdited = 1;
  END IF;
END
```

The post's creation time must be the same as the current time.

```
CREATE
TRIGGER `linkkariyermood`.`posttime`
BEFORE INSERT ON `linkkariyermood`.`post`
FOR EACH ROW
BEGIN
        SET NEW.post_time = CURRENT_TIMESTAMP;
END
```

The comment's creation time must be the same as the current time.

```
CREATE
TRIGGER `linkkariyermood`.`commenttime`
BEFORE INSERT ON `linkkariyermood`.`comment`
FOR EACH ROW
BEGIN
        SET NEW.comment_time = CURRENT_TIMESTAMP;
END
```

The job advert's creation time must be the same as the current time.

```
CREATE
TRIGGER `linkkariyermood`.`check_adverttime`
BEFORE INSERT ON `linkkariyermood`.`job_advert`
FOR EACH ROW
BEGIN
    SET NEW.advert_time = CURRENT_TIMESTAMP;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_updateadverttime`
BEFORE UPDATE ON `linkkariyermood`.`job_advert`
FOR EACH ROW
BEGIN
    SET NEW.advert_time = CURRENT_TIMESTAMP;
END
```

The job application's creation time must be the same as the current date.

```
CREATE
TRIGGER `linkkariyermood`.`check_apptime`
BEFORE INSERT ON `linkkariyermood`.`job_app`
FOR EACH ROW
BEGIN
    SET NEW.app_date = CURRENT_DATE;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_updateapptime`
BEFORE UPDATE ON `linkkariyermood`.`job_app`
FOR EACH ROW
BEGIN
    SET NEW.app_date = OLD.app_date;
END
```

The message's time must be the same as the current time.

```
CREATE
TRIGGER `linkkariyermood`.`messagetime`
BEFORE INSERT ON `linkkariyermood`.`message`
FOR EACH ROW
BEGIN
        SET NEW.message_time = CURRENT_TIMESTAMP;
END
```

Check whether the student is the type of license student and adds to the license student table.

```
CREATE
TRIGGER `linkkariyermood`.`check_isLicStu`
BEFORE INSERT ON `linkkariyermood`.`student_dept`
FOR EACH ROW
BEGIN
        IF (NEW.username NOT IN (
                SELECT license_student.username
                FROM license_student, course
                WHERE new.dept_id=course.dept_id and course.course_type = "license")) THEN
                        INSERT INTO license_student VALUES(NEW.username);
  END IF;
END
```

Repeats the same control when updating a row.

```
CREATE
TRIGGER `linkkariyermood`.`check_isLicStuOnUpd`
BEFORE UPDATE ON `linkkariyermood`.`student_dept`
FOR EACH ROW
BEGIN
        IF (NEW.username NOT IN (
                SELECT license_student.username
                FROM license_student, course
                WHERE new.dept_id=course.dept_id and course.course_type = "license")) THEN
                        INSERT INTO license_student VALUES(NEW.username);
   END IF;
END
```

Checks that the user will not self-assess

```
CREATE DEFINER=`root`@`localhost` TRIGGER `check_userEqualEndorsed` BEFORE INSERT ON `user_endorsed_skill` FOR
EACH ROW BEGIN
        IF(NEW.username=NEW.endorsed_by) THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Kullanıcı kendini değerlendiremez.';
        END IF;
END
```

Repeats the same control when updating a row.

```
CREATE DEFINER=`root`@`localhost` TRIGGER `check_userEqualEndorsedOnUpd` BEFORE UPDATE ON
`user_endorsed_skill` FOR EACH ROW BEGIN
        IF(NEW.username=NEW.endorsed_by) THEN
                SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'HATA: Kullanıcı kendini değerlendiremez.';
        END IF;
END
```

Check constraints of LinkKariyerMood

Restricts whether the course type is license or grad.

```
CONSTRAINT `course_chk_1` CHECK ((`course_type` in ('license','grad')))
```

Restricts whether the grad course type is a master or PhD.

```
CONSTRAINT `grad_course_chk_1` CHECK ((`grad_course_type` in ('master','phd')))
```

The grad's note restricts the data to be between 0 and 100.

```
CONSTRAINT `grad_enroll_chk_1` CHECK (((`midterm` >= 0) and (`midterm` <= 100))),
CONSTRAINT `grad_enroll_chk_2` CHECK (((`final` >= 0) and (`final` <= 100))),
CONSTRAINT `grad_enroll_chk_3` CHECK (((`makeup` >= 0) and (`makeup` <= 100))),
CONSTRAINT `grad_enroll_chk_4` CHECK (((`lab` >= 0) and (`lab` <= 100)))
```

Restricts whether the grad student type is a master or PhD.

```
CONSTRAINT `grad_student_chk_1` CHECK ((`grad_student_type` in ('master','phd')))
```

Restricts whether the instructor type is a faculty member or research assistant.

```
CONSTRAINT `instructor_chk_1` CHECK ((`instructor_type` in ('faculty_member','research_assistant')))
```

Restricts whether the job advert's working type is fulltime or parttime or intern.

```
CONSTRAINT `job_advert_chk_1` CHECK ((`working_type` in ('fulltime','parttime','intern')))
```

The license's note restricts the data to be between 0 and 100.

```
CONSTRAINT `license_enroll_chk_1` CHECK (((`midterm` >= 0) and (`midterm` <= 100))),
CONSTRAINT `license_enroll_chk_2` CHECK (((`final` >= 0) and (`final` <= 100))),
CONSTRAINT `license_enroll_chk_3` CHECK (((`makeup` >= 0) and (`makeup` <= 100))),
CONSTRAINT `license_enroll_chk_4` CHECK (((`lab` >= 0) and (`lab` <= 100)))
```

Restricts that the graduation year is not before the start year.

```
CONSTRAINT `student_dept_chk_1` CHECK ((`grad_year` > `start_year`)),
```

Restricts the class of the year to be minimum 1 maximum 6.

```
CONSTRAINT `student_dept_chk_2` CHECK (((`grade` > 0) and (`grade` < 7))),
```

Restricts the grade point average between 0 and 4.

```
CONSTRAINT `student_dept_chk_3` CHECK (((`gpa` >= 0) and (`gpa` <= 4))),
```

Restricts the student number to be greater than 0.

```
CONSTRAINT `student_dept_chk_4` CHECK ((`student_no` > 0))
```

Restricts the grades of the students' projects to be between 0 and 100.

```
CONSTRAINT `student_project_chk_1` CHECK (((`grade` <= 100) and (`grade` >= 0)))
```

Restricts a user to be an employee or student or instructor.

```
CONSTRAINT `user_chk_1` CHECK (((`isEmployee` <> 0) or (`isStudent` <> 0) or (`isInstructor` <> 0))),
```

Restricts the user's gender to be entered as E or K.

```
CONSTRAINT `user_chk_2` CHECK ((`sex` in ('E','K')))
```

Restricts whether the company's working type is fulltime or parttime or intern.

```
CONSTRAINT `works_for_chk_1` CHECK ((`working_type` in ('fulltime','parttime','intern'))),
```

Restricts that the working year is not before the start year.

```
CONSTRAINT `works_for_chk_2` CHECK ((`end_date` >= `start_date`))
```

Insert, Delete and Update statements of LinkKariyerMood

Data can be insert to the user table as follows:

```
INSERT INTO `linkkariyermood`.`user`
(`username`,`fname`,`lname`,`email`,`bdate`,`city`,`country`,`sex`,`isEmployee`,`isStudent`,`isInstructor`)
VALUES('abdidjama', 'Abdi Djama', 'Waberi', 'abdi@hotmail.com', '1998-06-14', 'Istanbul', 'Turkiye', 'E', '0', '1', '0');
```

Updating data in user table. The data update for the sample student is as follows:

```
UPDATE `linkkariyermood`.`user`
SET`username` ='abdidjama',`fname` = 'Abdi Djama',`lname` = 'Waberi',`email` ='abdi@hotmail.com',`bdate` = '1998-06-14',`
city` ='Istanbul',`country` = 'Turkiye',
`sex` ='E',`isEmployee` = '0',`isStudent` ='1',`isInstructor` = '0'
WHERE `username` = 'alicankayabasi';
```

Deleting the user in the example:

```
DELETE FROM `linkkariyermood`.`user`
WHERE `username` = 'alicankayabasi';
```

Data can be insert to the course table as follows:

```
INSERT INTO `linkkariyermood`.`course`
(`course_id`,`course_name`,`course_type`,`dept_id`,`facmem`)
VALUES('10101', 'Algoritma 2', 'license', '101', 'cenkerdur');
```

The instructor data in the course table can be updated as follows:

```
UPDATE `linkkariyermood`.`course`
SET
`facmem` = 'leventtoker',
WHERE `course_id` = '10101';
```

Deleting the course in the example:

```
DELETE FROM `linkkariyermood`.`course`
WHERE dept_id=102;
```

Data can be insert to the job_advert table as follows:

```
INSERT INTO `linkkariyermood`.`job_advert`
(`advert_name`,`comp_id`,`working_type`,`content`)
VALUES('Ambar Sorumlusu', '6', 'parttime', 'Malzeme devir hızını takip etmek ve stoklu, stoksuz tüm parçaların aylık raporlamasını
yapmak');
```

The content and the working_type data in the job_advert table can be updated as follows:

```
UPDATE `linkkariyermood`.`job_advert`
SET
`working_type` =’fulltime’
`content` ='İhtiyaç tanımlama ve kurgu oluşturma/olgunlaştırma adımlarını Talep Sahibi ile birlikte yapılması, teknik fizibilitenin
gerçekleştirilmesi'
WHERE `advert_name` = 'Ambar Sorumlusu' AND `comp_id` = '6';
```

Deleting the job_advert in the example:

```
DELETE FROM `linkkariyermood`.`job_advert`
WHERE `working_type` ='intern';
```

Queries of LinkKariyerMood

1.Select statements using one table

Lists first names that used by more than one person alphabetically and its usage count

```
SELECT user.fname as "First Name", COUNT(*) as "# of Person"
FROM user
GROUP BY user.fname
HAVING COUNT(*) > 1
ORDER BY fname ASC;
```

Lists distinct departments that have "Muhendis" keyword in their name

```
SELECT DISTINCT department.dept_name as "Department"
FROM department
WHERE department.dept_name LIKE "%Muhendis%";
```

Lists usernames of students that on 4th grade, not graduated and has a gpa over 3.0

```
SELECT sd.username
FROM student_dept as sd
WHERE sd.grade = 4 AND sd. grad_year is NULL AND sd.gpa > 3.0;
```

2.Select statements using two tables

Lists companies that have a job advertisement for interns

```
SELECT DISTINCT company.comp_name as "Company"
FROM company, job_advert
WHERE job_advert.comp_id = company.comp_id AND job_advert.working_type = "intern";
```

Lists users that joined more than one community and community count that they joined

```
SELECT user.fname as "First Name", COUNT(*) as "# of Comm"
FROM comm_member, user
WHERE user.username = comm_member.username
GROUP BY comm_member.username
HAVING COUNT(*) > 1
ORDER BY fname ASC;
```

Lists username of instructors that have a student took 100 from any of their project

```
SELECT DISTINCT c.facmem
FROM course as c, student_project as sp
WHERE sp.grade=100 AND sp.course_id = c.course_id;
```

Lists username of faculty members who instruct any course that has no projects

```
SELECT DISTINCT c.facmem
FROM course as c
WHERE NOT EXISTS(
                SELECT c.course_id
                FROM project as p
                WHERE p.course_id = c.course_id);
```

3.Select statements using minimum three tables

Lists users that both working and instructing with their company name

```
SELECT u.fname, u.lname, c.comp_name
FROM user as u, company as c, works_for as w
WHERE u.isInstructor = 1 AND u.username = w.username AND w.comp_id = c.comp_id;
```

Lists students who applicated an internship in last 3 months

```
SELECT DISTINCT u.fname as "First Name", u.lname as "Last Name"
FROM user as u, job_app as app, job_advert as ad
WHERE u.username = app.username AND app.advert_name = ad.advert_name AND app.comp_id = ad.comp_id
        AND     ad.working_type = "intern" AND current_timestamp >= now()-interval 3 month
ORDER BY fname ASC, lname ASC;
```

Lists advisors and license students who took over 70 from midterm of the courses that has assistant

```
SELECT stu.fname as "Student First Name", stu.lname as "Student Last Name",
        ins.fname as "Advisor First Name", ins.lname as "Advisor Last Name"
FROM user as ins, user as stu, license_enroll as l, assistantship as a, advisor as adv
WHERE a.course_id = l.course_id AND l.midterm > 70 AND l.username = stu.username AND
        l.username = adv.student_user AND adv.ins_user = ins.username;
```

4.Original select statements

Lists female students who are living out of Istanbul but applicated an advert of a company located in Istanbul

```
SELECT DISTINCT u.fname as "First Name", u.lname as "Last Name"
FROM user as u, company as c, job_app as app, job_advert as ad
WHERE c.city = "Istanbul" AND c.comp_id = ad.comp_id AND ad.advert_name = app.advert_name AND ad.comp_id =
app.comp_id
        AND app.username = u.username AND app.username not in (
                        SELECT u.username
      WHERE u.sex="E" or u.city="Istanbul")
ORDER BY fname ASC, lname ASC;
```

Lists students who have consulting skill and applicated any job

```
SELECT DISTINCT u.fname as "First Name", u.lname as "Last Name", j.advert_name as "Applicated to"
FROM user as u, user_skill as us, job_app as j, student_dept as sd, department as d, university as uni, skill as s
WHERE uni.uni_id = d.uni_id and d.dept_id = sd.dept_id and sd.username = j.username and j.username = us.username
        and us.skill_id = 1 and us.username = u.username and s.skill_id = us.skill_id;
```

Lists students who graduated from "Universidade Nova de Lisboa" and currently studies as graduated student at any university

```
SELECT u.fname as "First Name", u.lname as "Last Name"
FROM user as u, student_dept as sd, department as d, grad_student as gs
WHERE sd.grad_year is not null and sd.dept_id = d.dept_id and
        sd.username = gs.username and gs.username = u.username and d.uni_id IN
  ( SELECT uni.uni_id
          FROM university as uni
   WHERE uni.uni_name ="Universidade Nova de Lisboa");
```

Lists students who graduated in 2019 with a gpa over 2.5 and knows German

```
SELECT u.fname as "First Name", u.lname as "Last Name"
FROM user as u, user_lang as ul, student_dept as sd
WHERE ul.lang = "German" and ul.username = sd.username and sd.gpa > 2.5 and sd.grad_year = 2019 and sd.username =
u.username;
```

It lists how many people have endorsed their SKILLs for users who have a GPA higher than the average GPA and who live in Istanbul

```
SELECT ues.username, count(ues.username)
FROM user,user_endorsed_skill as ues
WHERE ues.username          IN(
   SELECT user.username
            WHERE user.city = "Istanbul"  AND user.username IN (
                        SELECT username
                        FROM student_dept as sd
                        WHERE sd.gpa > (SELECT avg(sd.gpa)
                        FROM student_dept as sd)));
```

4.Select statements with wiew

Shows the project grades of the students applying for a job.

```
CREATE VIEW project_notes AS
SELECT DISTINCT u.fname as "First Name",u.lname as "Last Name",  course.course_name as "Project Course",
student_project.grade as "Project Grade"
FROM user as u, job_app, student_project, course
WHERE course.course_id = student_project.course_id AND  job_app.username = student_project.username
    AND u.username = student_project.username AND student_project.grade is not null ;
```

Shows the exam grades (midterm, final, make-up, laboratory) of the courses taken by the grad student.

```
CREATE VIEW grad_exam_notes AS
SELECT DISTINCT u.fname as "First Name",u.lname as "Last Name",  course.course_name as "Course", grad_enroll.midterm as
"Midterm", grad_enroll.final as "Final", grad_enroll.makeup as "Make-Up", grad_enroll.lab as "Laboratory"
FROM user as u, job_app, grad_enroll, course
WHERE course.course_id = grad_enroll.course_id AND  job_app.username = grad_enroll.username
    AND u.username = grad_enroll.username  ;
```

Shows the exam grades (midterm, final, make-up, laboratory) of the courses taken by the license student.

```
CREATE VIEW license_exam_notes AS
SELECT DISTINCT u.fname as "First Name",u.lname as "Last Name",  course.course_name as "Course", license_enroll.midterm
as "Midterm", license_enroll.final as "Final", license_enroll.makeup as "Make-Up", license_enroll.lab as "Laboratory"
FROM user as u, job_app, license_enroll, course
WHERE course.course_id = license_enroll.course_id AND  job_app.username = license_enroll.username
    AND u.username = license_enroll.username  ;
```

Shows how many people have applied for job advertisements.

```
CREATE VIEW job_app_count AS
SELECT company.comp_name as "Company Name",job_app.advert_name as "Advertisement",COUNT(*) as "Number Of
Application"
FROM job_app, company
WHERE job_app.comp_id = company.comp_id
GROUP BY advert_name, job_app.comp_id;
```