EGE UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

DATABASE MANAGEMENT

2021–2022 FALL SEMESTER

TERM PROJECT REPORT

(LinkedinMoodle Database)

DELIVERY DATE

02/02/2022

PREPARED BY

05190000875, Mehmet Nuraydın

05180000075, Emircan Bahadır

05180000066, Bülent Ruhat Karataş

05180000123, Rahime Taşdemir

Table of Contents

Analysis

1.Brief Explanation of Applications

Linkedin

LinkedIn is a social platform that allows its users to create profiles about their careers and interact with other professionals. This platform consists of accounts for personal users and organizations. Users have personal information, training, skills, past experiences, languages, and so on. Furthermore, they can share the projects they have contributed to over the years, the people they have worked with, which organization they were working for at the time, and post about their interests in that sector. Users can also connect with other users, and they can like, comment, and answer content shared by other users or organizations. And if the user wants, they can create an organization account in the system, manage its employees, share a post about new job openings, and so forth.

Moodle

Moodle is a learning management system that serves teachers and students. Teachers create courses, and students can take these courses by enrolling. Each course includes various assignments like exams and projects. These assignments can only get assigned by the teachers that created the course. Subsequently, students can upload these assignments before deadlines. In summary, Moodle is a system that consists of courses, teachers giving those courses, students who take those courses, assignments given by teachers, et cetera.

2.Analysis Report

2.a.Aim of each application

Linkedin

The main purpose of the LinkedIn application is to keep career information of personal and corporate accounts and let them create a professional network by connecting, being interested in companies that are well-known in their sectors.

Moodle

The main purpose of the Moodle system is to bring together students and teachers in an online environment.

2.b.Main entities

Linkedin Database

- USER
- ORGANIZATION
- PROJECT
- ACCOUNT
- POST
- COMMENT
- LIKABLE_CONTENT

Moodle Database

- STUDENT
- TEACHER
- COURSE
- ASSIGNMENT

Linkedin-Moodle Database

- USER
- ORGANIZATION
- ACCOUNT
- MOODLE_MEMBER
- STUDENT
- TEACHER
- COURSE
- ASSIGNMENT
- PROJECT
- LIKABLE_CONTENT
- POST
- COMMENT
- ANSWER

2.c.Characteristics of Each Entity

Linkedin Database

- USER: Strong Entity
- ORGANIZATION : Strong Entity
- PROJECT : Strong Entity
- ACCOUNT : Strong Entity
- POST : Weak Entity
- COMMENT : Weak Entity
- ANSWER : Weak Entity
- LIKABLE_CONTENT : Strong Entity

Moodle Database

- STUDENT : Strong Entity
- TEACHER : Strong Entity
- COURSE : Strong Entity
- ASSIGNMENT : Weak Entity

Linkedin-Moodle Database

- USER : Strong Entity
- ORGANIZATION : Strong Entity
- ACCOUNT : Strong Entity
- MOODLE_MEMBER : Strong Entity
- STUDENT : Strong Entity
- TEACHER : Strong Entity
- COURSE : Strong Entity
- ASSIGNMENT : Weak Entity
- PROJECT : Strong Entity
- LIKABLE_CONTENT : Strong Entity
- POST : Weak Entity
- COMMENT : Weak Entity
- ANSWER : Weak Entity

2.d.Relationships Among Entities

Linkedin Database

- USER.Ssn → ACCOUNT.Account_id
- USER_SKILLS. User_ssn → USER.Ssn
- USER_LANGS.User_ssn → USER.Ssn
- CONNECT.User_ssn → USER.Ssn
- CONNECT.Connected_Ssn → USER.Ssn
- ORGANIZATION.Id →ACCOUNT.Account_id
- ORGANIZATION.Admin_ssn → USER.Ssn
- PROJECT.Org_id →ORGANIZATION.Id
- WORKS_FOR_ORG.User_ssn → USER.Ssn
- WORKS_FOR_ORG.Org_id → ORGANIZATION.Id
- POST.Account_id → ACCOUNT.Account_id
- POST.Likable_content_id → LIKABLE_CONTENT.Id
- WORKS_ON_ROJECT.User_ssn → USER.Ssn
- WORKS_ON_ROJECT.Project_id → PROJECT.Id
- COMMENT.User_ssn → USER.Ssn
- COMMENT.Post_id,Account_id → POST.(Id,Account_id)
- COMMENT.Likable_content_id → LIKABLE_CONTENT.Id
- ANSWER.User_ssn → USER.Ssn
- ANSWER.Comment_id,Account_id,Post_id) →COMMENT.(Comment_id,Account_id,Post_id),
- ANSWER.(Post_id,Account_id) → POST.(Id,Account_id)
- ANSWER.(Likable_content_id) → LIKABLE_CONTENT.Id
- LIKES.User_ssn → USER.Ssn
- LIKES.Likable_content_id) → LIKABLE_CONTENT.Id

Moodle Database

Each teacher can create a number of course and a course should created by only one teacher.

Each student can enrolls number of course and there may be number of students that enrolls that course.

- COURSE.Teacher_ssn → TEACHER.Ssn
- ASSIGNMENT.Course_id → COURSE.Id
- ASSIGNMENT_UPLOAD.Assignment_id,Course_id) → ASSIGNMENT. (Assignment_id,Course_id)
- ASSIGNMENT_UPLOAD.Student_ssn) → STUDENT.Ssn
- ASSIGNMENT_UPLOAD.Student_ssn) → USER.Ssn
- ASSIGNMENT_GRADE.Assignment_id,Course_id) → ASSIGNMENT.(Assignment_id,Course_id)
- ASSIGNMENT_GRADE.Student_ssn) → STUDENT.Ssn
- ASSIGNMENT_GRADE.Student_ssn) → USER.Ssn
- ENROLLS.Course_id) → COURSE.Id
- ENROLLS.Student_ssn) → STUDENT.Ssn
- ENROLLS.Student_ssn) → USER.Ssn

Linkedin-Moodle Database

- USER.Ssn → ACCOUNT.Account_id
- USER_SKILLS. User_ssn → USER.Ssn
- USER_LANGS.User_ssn → USER.Ssn
- CONNECT.User_ssn → USER.Ssn
- CONNECT.Connected_Ssn → USER.Ssn
- ORGANIZATION.Id →ACCOUNT.Account_id
- ORGANIZATION.Admin_ssn → USER.Ssn
- PROJECT.Org_id →ORGANIZATION.Id
- WORKS_FOR_ORG.User_ssn → USER.Ssn
- WORKS_FOR_ORG.Org_id → ORGANIZATION.Id

- POST.Account_id → ACCOUNT.Account_id
- POST.Likable_content_id → LIKABLE_CONTENT.Id
- WORKS_ON_ROJECT.User_ssn → USER.Ssn
- WORKS_ON_ROJECT.Project_id → PROJECT.Id
- COMMENT.User_ssn → USER.Ssn
- COMMENT.Post_id,Account_id → POST.(Id,Account_id)
- COMMENT.Likable_content_id → LIKABLE_CONTENT.Id
- ANSWER.User_ssn → USER.Ssn
- ANSWER.Comment_id,Account_id,Post_id) →COMMENT.(Comment_id,Account_id,Post_id),
- ANSWER.(Post_id,Account_id) → POST.(Id,Account_id)
- ANSWER.(Likable_content_id) → LIKABLE_CONTENT.Id
- COURSE.Teacher_ssn → TEACHER.Ssn
- ASSIGNMENT.Course_id → COURSE.Id
- ASSIGNMENT_UPLOAD.Assignment_id,Course_id) → ASSIGNMENT. (Assignment_id,Course_id)
- ASSIGNMENT_UPLOAD.Student_ssn) → STUDENT.Ssn
- ASSIGNMENT_UPLOAD.Student_ssn) → USER.Ssn
- ASSIGNMENT_GRADE.Assignment_id,Course_id) → ASSIGNMENT.(Assignment_id,Course_id)
- ASSIGNMENT_GRADE.Student_ssn) → STUDENT.Ssn
- ASSIGNMENT_GRADE.Student_ssn) → USER.Ssn
- ENROLLS.Course_id) → COURSE.Id
- ENROLLS.Student_ssn) → STUDENT.Ssn
- ENROLLS.Student_ssn) → USER.Ssn
- LIKES.User_ssn → USER.Ssn
- LIKES.Likable_content_id) → LIKABLE_CONTENT.Id

2.e.Constraints, Characteristics and Relationships

LINKEDIN DATABASE

The LinkedIn database includes USER, ORGANIZATION, PROJECT, ACCOUNT, POST, COMMENT, ANSWER, and LIKABLE_CONTENT entities. User, organization, project, account, and likable content. Post, a comment, and an answer are weak entities. Account entity is a strong entity, and the post entity is a strong entity of the comment entity, and the comment entity is a strong entity of the answer entity. Therefore;

-A post record with an account id that doesn't exist in our database can't be inserted.

-A comment record with a post id that doesn't exist in our database can't be inserted.

-An answer record with a comment id that doesn't exist in our database can't be inserted.

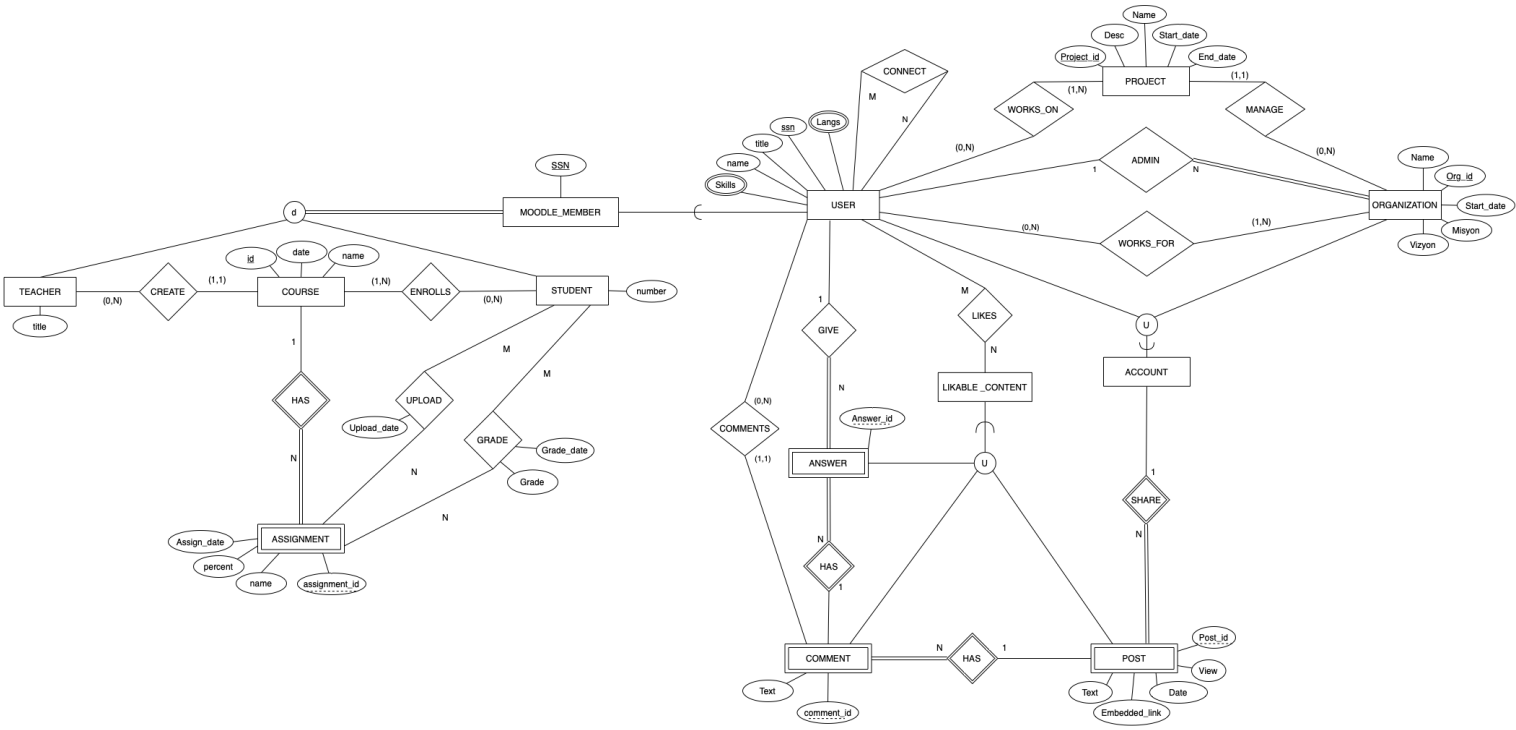
When we insert a post record, comment record, or answer record we create a likable content record for them.

Organizations can create posts but they can't interact with them in any way or form.

MOODLE DATABSE

The Moodle database includes STUDENT, TEACHER COURSE, and ASSIGNMENT entities. Student, teachers, and courses are strong entities. The assignment entity is a weak entity, and its strong entity is the course. If there is no course entity in the system, the assignment entity cannot be created.A course can be created by one teacher, so course_id can not be repeated.

Conceptual Design



Mapping

Iteration-1

STEP-1
ORGANIZATION

Org_id	Name	Start_date	Mission	Vision

PROJECT

Project_id	Name	Desc	Start_date	End-date

COURSE

id	name	date

STEP-2
ASSIGNMENT

Course_id	Assignment_id	Name	percent	Assign_date

STEP-3
——

STEP-4
PROJECT

...	org_id

STEP-5-6-7
——

STEP-8
USER

Ssn	name	title

STEP-9
ACCOUNT

Account_id

USER

Ssn	name	title	account_id

ORGANIZATION

...	account_id

Iteration-2

STEP-1
——

STEP-2
POST

account_id	post_id	veiw	date	embedded_link	text

STEP-3
——

STEP-4

ORGANIZATION

…	user_ssn

STEP-5
WORKS_ON

user_ssn	project_id

WORKS_FOR

user_ssn	organization_id

CONNECT

user_ssn	connected_ssn

STEP-6
USER_LANGS

user_ssn	language

USER_SKILLS

user_ssn	skill

STEP-7
——

STEP-8
TEACHER

Ssn	title

STUDENT

Ssn	number

STEP-9
——

Iteration-3

STEP-1
——

STEP-2
COMMENT

account_id	post_id	comment_id	text

STEP-3
——

STEP-4
COURSE

…	Teacher_ssn

COMMENT

…	user-ssn

STEP-5
ENROLLS

student_ssn	course_id

UPLOAD

course_id	assignment_id	student_ssn	uploaded

GRADE

course_id	assignment_id	student_ssn	grade	grade_date

STEP-6-7-8-9
——

Iteration-4

STEP-1
——

STEP-2
ANSWER

account_id	post_id	comment_id	answer_id	text

STEP-3
——

STEP-4
ANSWER

…	user_ssn

STEP-5-6-7-8
——

STEP-9
LIKABLE_CONTENT

likable_content_id

ANSWER

…	likable_content_id

COMMENT

…	likable_content_id

POST

…	likable_content_id

Relational model

Organization

Org_id	Name	Start_Date	Mission	Vision	Acct_id	user_ssn

Project

project_id	Org_id	Name	Desc	Start_Date	End_Date

Course

id	Name	Date	Teacher_ssn

Assignment

Course_id	Assignment_id	Name	percent	Assign_date

User

ssn	name	title	Acct_id

Account

Account_id

Post

Acct_id	post_id	view	Date	Embedded_link	text	likable_content_id

Works_on

user_ssn	project_id

Works_for

user_ssn	organization_id

Connect

user_ssn	Connected_user_ssn

User_langs

user_ssn	language

User_skills

user_ssn	skill

Teacher

ssn	title

Student

ssn	number

Comment

Acct_id	Post_id	Comment_id	Text	user_ssn	likable_content_id

Enrolls

Student_ssn	course_id

Upload

Course_id	Assign_id	Student_ssn	Upload_Date

Grade

Course_id	Assign_id	Studentt_ssn	Grade	Grade_Date

Likable_content

likable_content_id

Answer

Acct_id	Post_id	content_id	answer_id	user_ssn	likable_content_id

Like

user_ssn	likable_content_id

Implementation

6.SQL Scripts

```
CREATE TABLE ACCOUNT(
    Account_id CHAR(9) NOT NULL PRIMARY KEY
            );
 CREATE TABLE USER(
    Ssn CHAR(9) NOT NULL PRIMARY KEY,
    title VARCHAR(30) NOT NULL,
    name VARCHAR(100) NOT NULL,
    last_logged TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT BADCHARS CHECK(name LIKE "%*%" OR name LIKE "%-%" OR name LIKE "%+%" OR name LIKE "%~%")
    FOREIGN KEY(Ssn) REFERENCES ACCOUNT(Account_id)
                ON UPDATE CASCADE ON DELETE CASCADE
 );
 CREATE TABLE USER_SKILLS(
    User_ssn CHAR(9) NOT NULL,
    Sname VARCHAR(30) NOT NULL,
    FOREIGN KEY(User_ssn) REFERENCES USER(Ssn),
    PRIMARY KEY(User_ssn,Sname)
 );
```

```
CREATE TABLE USER_LANGS(
    User_ssn CHAR(9) NOT NULL,
    Lname VARCHAR(30) NOT NULL,
    FOREIGN KEY(User_ssn) REFERENCES USER(Ssn),
    PRIMARY KEY(User_ssn,Lname)
);
CREATE TABLE CONNECT(
    User_ssn CHAR(9) NOT NULL,
    Connected_ssn CHAR(9) NOT NULL,
    CONSTRAINT SLFCNCTN CHECK (User_ssn != Connected_ssn),
    FOREIGN KEY(User_ssn) REFERENCES USER(Ssn)
    ON DELETE NO ACTION ON UPDATE NO ACTION,
    FOREIGN KEY(Connected_Ssn) REFERENCES USER(Ssn)
    ON DELETE NO ACTION ON UPDATE NO ACTION,
    PRIMARY KEY(User_ssn,Connected_Ssn)
);
CREATE TABLE ORGANIZATION(
    Id CHAR(9),
    name VARCHAR(255) NOT NULL,
    Start_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Admin_ssn CHAR(9) NOT NULL,
    Mission VARCHAR(255) DEFAULT "ORGANIZATION_MISSION",
    Vision VARCHAR(255) DEFAULT "ORGRANIZATION_VISION",
    PRIMARY KEY(Id),
    FOREIGN KEY(Id) REFERENCES ACCOUNT(Account_id)
```

```
                    ON UPDATE CASCADE ON DELETE CASCADE,
    FOREIGN KEY(Admin_ssn) REFERENCES USER(Ssn)
);
CREATE TABLE WORKS_FOR_ORG(
    User_ssn CHAR(9) NOT NULL,
    Org_id CHAR(9) NOT NULL,
    FOREIGN KEY(User_ssn) REFERENCES USER(Ssn),
    FOREIGN KEY(Org_id) REFERENCES ORGANIZATION(Id),
    PRIMARY KEY(User_ssn,Org_id)
);

CREATE TABLE LIKABLE_CONTENT(
    Id INTEGER AUTO_INCREMENT NOT NULL PRIMARY KEY
);
CREATE TABLE POST(
    Id INTEGER NOT NULL AUTO_INCREMENT,
    Account_id CHAR(9) NOT NULL,
    Created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Content VARCHAR(255) NOT NULL DEFAULT "POST_CONTENT",
    Likable_content_id INTEGER NOT NULL,
    CONSTRAINT NOBADPOST CHECK(Content NOT LIKE "%fuck%"),
    PRIMARY KEY(Id,Account_id) ,
    FOREIGN KEY(Account_id) REFERENCES ACCOUNT(Account_id),
    FOREIGN KEY(Likable_content_id) REFERENCES LIKABLE_CONTENT(Id)
);
```

```
CREATE TABLE PROJECT(
    Id INTEGER NOT NULL,
    Pname VARCHAR(100) NOT NULL,
    Pdesc VARCHAR(255) DEFAULT "NO DESC PROVIDED",
    Start_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Org_id CHAR(9) NOT NULL,
    End_date TIMESTAMP NULL DEFAULT NULL,
    CONSTRAINT CheckEndLaterThanStart CHECK (End_date=NULL OR End_date >=
Start_date),
    PRIMARY KEY(Id),
    FOREIGN KEY(Org_id) REFERENCES ORGANIZATION(Id)
);
CREATE TABLE WORKS_ON_PROJECT(
    User_ssn char(9) NOT NULL,
    Project_id INTEGER NOT NULL,
    FOREIGN KEY(User_ssn) REFERENCES USER(Ssn),
    FOREIGN KEY(Project_id) REFERENCES PROJECT(Id),
    PRIMARY KEY(User_ssn,Project_id)
);
CREATE TABLE COMMENT(
    Comment_id INTEGER NOT NULL AUTO_INCREMENT,
    Account_id CHAR(9) NOT NULL,
    Post_id INTEGER NOT NULL,
    User_ssn CHAR(9) NOT NULL,
    Body VARCHAR(255) DEFAULT "Comment_body",
    Likable_content_id INTEGER NOT NULL,
```

```
    CONSTRAINT NOBADCOMMENT CHECK(Body NOT LIKE "%fuck%"),
    PRIMARY KEY(Comment_id,Account_id,Post_id),
    FOREIGN KEY(User_ssn) REFERENCES USER(Ssn),
    FOREIGN KEY(Post_id,Account_id) REFERENCES POST(Id,Account_id),
    FOREIGN KEY(Likable_content_id) REFERENCES LIKABLE_CONTENT(Id)

);
CREATE TABLE ANSWER(
    Answer_id INTEGER NOT NULL AUTO_INCREMENT,
    Account_id CHAR(9) NOT NULL,
    Comment_id INTEGER NOT NULL,
    Post_id INTEGER NOT NULL,
    User_ssn CHAR(9) NOT NULL,
    Body VARCHAR(255) DEFAULT "ANSWER_BODY",
    Likable_content_id INTEGER NOT NULL,
    CONSTRAINT NOBADANSWER CHECK(Body NOT LIKE "%fuck%"),
    PRIMARY KEY(Answer_id,Account_id,Comment_id,Post_id),
    FOREIGN KEY(User_ssn) REFERENCES USER(Ssn),
    FOREIGN KEY(Comment_id,Account_id,Post_id) REFERENCES
COMMENT(Comment_id,Account_id,Post_id),
    FOREIGN KEY(Post_id,Account_id) REFERENCES POST(Id,Account_id),
    FOREIGN KEY(Likable_content_id) REFERENCES LIKABLE_CONTENT(Id)
);
```

```
CREATE TABLE STUDENT(
    Ssn CHAR(9) NOT NULL,
    Snumber CHAR(11) NOT NULL UNIQUE,
    PRIMARY KEY(Ssn)
);
CREATE TABLE TEACHER(
    Ssn CHAR(9) NOT NULL,
    title VARCHAR(50) NOT NULL DEFAULT "Asistant",
    PRIMARY KEY(Ssn)
);
CREATE TABLE COURSE(
    Id INTEGER AUTO_INCREMENT,
    Cname VARCHAR(75) NOT NULL,
    Cdate TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    Teacher_ssn CHAR(9) NOT NULL,
    PRIMARY KEY(Id),
    FOREIGN KEY(Teacher_ssn) REFERENCES TEACHER(Ssn)
);
CREATE TABLE ASSIGNMENT(
    Assignment_id INTEGER NOT NULL AUTO_INCREMENT,
    Course_id INTEGER NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    Aname VARCHAR(100) NOT NULL,
    PRIMARY KEY(Assignment_id,Course_id),
    FOREIGN KEY(Course_id) REFERENCES COURSE(Id)
);
```

```
CREATE TABLE ASSIGNMENT_UPLOAD(
    Assignment_id INTEGER NOT NULL,
    Course_id INTEGER NOT NULL,
    Student_ssn CHAR(9) NOT NULL,
    Upload_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY(Assignment_id,Course_id) REFERENCES
ASSIGNMENT(Assignment_id,Course_id),
    FOREIGN KEY(Student_ssn) REFERENCES STUDENT(Ssn),
    FOREIGN KEY(Student_ssn) REFERENCES USER(Ssn),
    PRIMARY KEY(Assignment_id,Course_id,Student_ssn)
);

CREATE TABLE ASSIGNMENT_GRADE(
    Assignment_id INTEGER NOT NULL,
    Course_id INTEGER NOT NULL,
    Student_ssn CHAR(9) NOT NULL,
    Grade INTEGER NOT NULL DEFAULT 0,
    Grade_date TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT GRADECONSTRAINT CHECK(Grade>=0 and Grade<=100),
    FOREIGN KEY(Assignment_id,Course_id) REFERENCES
ASSIGNMENT(Assignment_id,Course_id),
    FOREIGN KEY(Student_ssn) REFERENCES STUDENT(Ssn),
    FOREIGN KEY(Student_ssn) REFERENCES USER(Ssn),
    PRIMARY KEY(Assignment_id,Student_ssn,Course_id)
);
```

```
CREATE TABLE ENROLLS(
   Course_id INTEGER NOT NULL,
   Student_ssn CHAR(9) NOT NULL,
   Enroll_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   FOREIGN KEY(Course_id) REFERENCES COURSE(Id),
   FOREIGN KEY(Student_ssn) REFERENCES STUDENT(Ssn),
   FOREIGN KEY(Student_ssn) REFERENCES USER(Ssn),
   PRIMARY KEY(Course_id,Student_ssn)
);
CREATE TABLE LIKES(
   User_ssn CHAR(9) NOT NULL,
   Likable_content_id INTEGER NOT NULL,
   PRIMARY KEY(User_ssn , Likable_content_id),
   FOREIGN KEY(User_ssn) REFERENCES USER(Ssn),
   FOREIGN KEY(Likable_content_id) REFERENCES LIKABLE_CONTENT(Id)
);
```

8.TRIGGERS

The CONNECT_CONSTRAINT trigger is for disabling two users to connect for two times.

```sql
/*#bir kullanıcının bağlandığı kişi ile tekrar bağlanamamalı*/
/*(a,b) mevcutsa (b,a) mevcut olmamalı*/
DELIMITER $$
CREATE TRIGGER CONNECT_CONSTRAINT
BEFORE INSE
    ON CONNECT
      FOR EACH ROW
        BEGIN
          IF EXISTS(
                SELECT *
                FROM CONNECT as A
                WHERE CONCAT(A.User_ssn,A.Connected_ssn) =
CONCAT(NEW.Connected_ssn,NEW.User_ssn))
              THEN SIGNAL SQLSTATE '45000'
                  SET MESSAGE_TEXT = 'They are already connected';
          END IF;
        END;
$$
```

The ORGANIZATION_ADMIN_INSERT trigger is for automatically updating WORKS_FOR_ORG table with organization admin.

```sql
DELIMITER $$
/*Eğer bir user bir organizasyonun admini ise orda zaten çalışıyor olmalıdır*/
CREATE TRIGGER ORGANIZATION_ADMIN_INSERT
AFTER INSERT
```

```
        ON ORGANIZATION
            FOR EACH ROW
              BEGIN
                IF NOT EXISTS(
                        SELECT *
                        FROM WORKS_FOR_ORG
                        WHERE NEW.Admin_ssn = WORKS_FOR_ORG.User_ssn and
WORKS_FOR_ORG.Org_id = NEW.Id )
                  THEN INSERT INTO WORKS_FOR_ORG(User_ssn,Org_id)
                      VALUES (NEW.Admin_ssn,NEW.Id);
                END IF;
              END;
$$
```

The triggers will be activated when user and organization tables have some insertion. Finally, the ACCOUNT table will be updated with the triggers.

```sql
DEMLIMITTER $$
CREATE TRIGGER USER_ACCOUNT_INSERTION
BEFORE INSERT
    ON USER
FOR EACH ROW
      BEGIN
        INSERT INTO ACCOUNT(Account_id)
        VALUES (NEW.Ssn);
      END;
$$
```

```
CREATE TRIGGER ORG_ACCOUNT_INSERTION
BEFORE INSERT
    ON ORGANIZATION
FOR EACH ROW
      BEGIN
        INSERT INTO ACCOUNT(Account_id)
        VALUES (NEW.Id);
      END;
$$


```

The NOPERMISSION_ASG_UPLOAD is for disabling to users who are not enrolled to a specific course which is assignment upload for.

```sql
DELIMITER $$

CREATE TRIGGER NOPERMISSION_ASG_UPLOAD
BEFORE INSERT
    ON ASSIGNMENT_UPLOAD
      FOR EACH ROW
        BEGIN
          IF NOT EXISTS(
            SELECT *
            FROM ASSIGNMENT AS ASG  , ENROLLS AS ENR
            WHERE NEW.Assignment_id = ASG.Assignment_id
                    AND  ASG.Course_id = ENR.Course_id
                        AND NEW.Student_ssn = ENR.Student_ssn)
            THEN SIGNAL SQLSTATE '45000'
                SET MESSAGE_TEXT = 'No permission to the user. Not enrolled to the course!';
```

```
            END IF;
        END;
$$
```

The triggers will be activated when POST,COMMENT and ANSWER tables have some insertion. Finally, the LIKABLE_CONTENT table will be updated with the triggers.

```sql
CREATE TRIGGER LIKABLE_POST_INSERTION
BEFORE INSERT
    ON POST
FOR EACH ROW
    BEGIN
        INSERT INTO LIKABLE_CONTENT(Id)
        VALUES (NEW.Likable_content_id);
    END;
$$
CREATE TRIGGER LIKABLE_COMMENT_INSERTION
BEFORE INSERT
    ON COMMENT
FOR EACH ROW
    BEGIN
        INSERT INTO LIKABLE_CONTENT(Id)
        VALUES (NEW.Likable_content_id);
    END;
$$
```

```
CREATE TRIGGER LIKABLE_ANSWER_INSERTION
BEFORE INSERT
      ON ANSWER
FOR EACH ROW
         BEGIN
            INSERT INTO LIKABLE_CONTENT(Id)
            VALUES (NEW.Likable_content_id);
         END;
$$
```

9.Constraints and Assertions

The SLFCNCTN constraint is for disabling self connection on system.

```sql
CREATE TABLE CONNECT(
   User_ssn CHAR(9) NOT NULL,
   Connected_ssn CHAR(9) NOT NULL,
   CONSTRAINT SLFCNCTN CHECK (User_ssn != Connected_ssn),
   **
);
```

The CheckEndLaterThanStart constraint is for checking if start and end dates is valid.

```sql
CREATE TABLE PROJECT(
   **
   Start_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   End_date TIMESTAMP NULL DEFAULT NULL,
   CONSTRAINT CheckEndLaterThanStart CHECK (End_date=NULL OR End_date >= Start_date),
            **
```

);

```

The GRADECONSTRAINT constraint is for checking if grade value is valid.

```sql
CREATE TABLE ASSIGNMENT_GRADE(
   **
   Grade INTEGER NOT NULL DEFAULT 0,
   CONSTRAINT GRADECONSTRAINT CHECK(Grade>=0 and Grade<=100),
   **
);
```

The NOBADCOMMENT constraint is for disabling swear words to use.

```sql
CREATE TABLE COMMENT(
   **
   Body VARCHAR(255) DEFAULT "Comment_body",
   CONSTRAINT NOBADCOMMENT CHECK(Body NOT LIKE "%fuck%"),
   **
);
```

The BADCHARS constraint is for disabling unwanted chars to be used!

```sql
CREATE TABLE USER(
      ***
      name VARCHAR(100) NOT NULL,
      CONSTRAINT BADCHARS CHECK(name LIKE "%*%" OR name LIKE "%-%" OR name LIKE "%+%"
                  OR name LIKE "%~%")
      ***
   );
```
```

```

The CONNECT_CONSTRAINT is for disabling repeated connections to occur.

```sql
DELIMITER $$
CREATE TRIGGER CONNECT_CONSTRAINT
BEFORE INSERT
    ON CONNECT
      FOR EACH ROW
        BEGIN
          IF EXISTS(
                SELECT *
                FROM CONNECT as A
                WHERE CONCAT(A.User_ssn,A.Connected_ssn) =
CONCAT(NEW.Connected_ssn,NEW.User_ssn))
              THEN SIGNAL SQLSTATE '45000'
                  SET MESSAGE_TEXT = 'They are already connected';
          END IF;
        END;
$$
```
```

10.SQL Statements

1. INSERTIONS-DELETIONS-UPDATES

   1. INSERTIONS

      1. TABLE USER

         ```sql
         INSERT INTO USER (Ssn,title,name,last_logged)
         VALUES
          (500000010,"Student","Nissim Deleon","2021-11-22"),
          (500000011,"Data Scientist","Avram Golden","2021-11-22"),
          (500000012,"Teacher","Nayda Rios","2021-11-22"),
          (500000013,"Entrepreneur","Harding Wooten","2021-11-22"),
          (500000014,"Member","Whoopi Lindsay","2021-11-22"),
          (500000015,"Manager","Grant Poole","2021-11-22"),
          (500000016,"Entrepreneur","Stuart Everett","2021-11-22"),
          (500000017,"Doctor","Garrett Rowland","2021-11-22"),
          (500000018,"Prof","Alfreda Reeves","2021-11-22"),
          (500000019,"Entrepreneur","Meredith Cotton","2021-11-22");
         ```

        b. TABLE CONNECT

      ```sql
      INSERT INTO CONNECT (User_ssn,Connected_ssn)
      VALUES
       (500000010,500000011),
       (500000010,500000012),
       (500000010,500000013),
       (500000010,500000014),
       (500000010,500000015),
       (500000015,500000016),
      ```

(500000018,500000017),

(500000005,500000018),

(500000013,500000012),

(500000018,500000019);

```

c. TABLE POST

```sql

INSERT INTO POST(Id,Account_id,Content,Likable_content_id )

VALUES

(31,500000069,"The system is all about memorization!",900000031),

(32,500000071,"Engineering harder than anybody think",900000032),

(33,500000073,"Harder you work luckier you get!",900000033),

(34,500000075,"It's all about consistency!",900000034),

(35,500000077,"Life is too short.İf you don't look around sometimes you could miss it.",900000035),

(37,500000081,"With great power comes great responsibility",900000037);

```

b. DELETIONS

1. TABLE USER

```sql

DELETE FROM USER

WHERE Ssn=500000010;

```

2. TABLE CONNECT

```sql

DELETE FROM CONNECT

WHERE  user_ssn = 500000010

```
    AND connected_ssn = 500000011;
```
```

3. TABLE POST

```sql
DELETE FROM POST
WHERE Content LIKE '%fuck%';
```

c. UPDATES

1. TABLE ASSIGNMENT_GRADE

```sql
UPDATE ASSIGNMENT_GRADE
SET Grade = 90
WHERE Student_ssn =500000086;
```

2. TABLE USER

```sql
UPDATE USER
SET title="Prof"
WHERE Ssn=500000106;
```

3. TABLE ORGANIZATION

```sql
UPDATE ORGANIZATION
SET Admin_ssn=500000009
WHERE Id=600000007;
```
```

2 THE 10 STATEMENTS

1. Maximum One Table

1.Information of active organizations registered in the system and established after 2000...

```sql
SELECT
  name,
  Start_date,
  Mission,
  Vision
FROM
  ORGANIZATION
WHERE
  YEAR(Start_date)> 2000;
```

2.Posts in the system that contain phone numbers...

```sql
SELECT
  Id,
  Account_id,
  Created_at,
  Content
FROM
  POST
WHERE
  Content LIKE "%5_____%"
  OR Content LIKE "%5__ ___ __ __%"
  OR Content LIKE "%(5__) ___ __ __%";
```

3.Data for comments with content longer than 25 characters...

```sql
SELECT
  Comment_id,
  Account_id,
  Body
FROM
  COMMENT
WHERE
  CHAR_LENGTH(Body) > 25;
```

2. Minimum Two Tables

1.The name of the course that has more than one assignment in the 2020 and the number of how many assignments it has in descending order...

```sql
SELECT
  C.Cname as Course_Name,
  COUNT(*) as Ass_Count
FROM
  COURSE AS C,
  ASSIGNMENT AS A
WHERE
  C.Id = A.Course_id
  AND YEAR(C.Cdate)= 2020
GROUP BY
  C.Id
ORDER BY
  Ass_Count DESC;
```

2.Data from teachers teaching Calculus 1.

```sql
SELECT
  U.Ssn,
  U.name,
  C.Cdate as Course_date
FROM
  COURSE AS C,
  USER AS U
WHERE
  C.Teacher_ssn = U.Ssn
  AND C.Cname = "Calculus 1";
```

3.Data of most skilled 10 users in descending order

```sql
SELECT title,
      name,
      last_logged,
      COUNT(*) AS Skill_count
FROM USER,
    USER_SKILLS
WHERE Ssn = User_ssn
GROUP BY Ssn
ORDER BY Skill_count DESC
LIMIT 10;
```

4.Let a post list the replies to the comment that received more likes than the comment made.

```sql
SELECT C.Comment_id,
      AN.Answer_id,
      C.Body AS C_body,
      AN.Body AS A_body
FROM COMMENT AS C,
            ANSWER AS AN
WHERE
   (SELECT COUNT(*)
    FROM LIKES AS L
    WHERE AN.Likable_content_id = L.Likable_content_id
     AND AN.Comment_id = C.Comment_id) >
   (SELECT COUNT(*)
    FROM LIKES AS L
    WHERE L.Likable_content_id = C.Likable_content_id);
```

c. Minimum Three Tables

1.The data of the projects worked by the users whose grade point average is higher than 50 from the assignments they have uploaded...

```sql
SELECT U.Ssn,
      U.name,
      P.Pname,
      P.Start_date,
      P.End_date
FROM USER AS U,
          WORKS_ON_PROJECT AS WOP,
```

```
        PROJECT AS P
WHERE P.Id = WOP.Project_id
  AND WOP.User_ssn = U.Ssn
  AND (U.Ssn) IN
    (SELECT AG.Student_ssn
     FROM ASSIGNMENT_GRADE AS AG
     GROUP BY AG.Student_ssn
     HAVING AVG(AG.Grade)>50) ;
```

2.The names,titles and last login dates of the five users with the most connections who have data on moodle ...

```sql
SELECT U.Name,
       U.title,
       U.last_logged
FROM USER AS U,
           TEACHER AS T,
           STUDENT AS S
WHERE (U.Ssn = T.Ssn
       OR U.SSN = S.Ssn)
  AND U.Ssn IN
    (SELECT U.Ssn
     FROM USER AS U,
               CONNECT AS C
     WHERE U.Ssn = C.User_ssn
       OR U.Ssn = C.Connected_ssn
     GROUP BY U.Ssn
     ORDER BY Count(*) DESC)
LIMIT 5 ;
```

```

3.Number of assignments submitted and graded by students who enrolled to the most courses...

```sql
WITH RECURSIVE MOST_ENROLLED_USERS(Ssn, total)AS
  (SELECT U.Ssn,
        COUNT(*) AS total
   FROM STUDENT AS U,
      ENROLLS AS E
   WHERE U.Ssn = E.Student_ssn
   GROUP BY U.Ssn
   UNION SELECT M.Ssn,
             M.total
   FROM MOST_ENROLLED_USERS AS M)
SELECT M.Ssn,
      M.total,
      USER.name,
  (SELECT COUNT(*)
   FROM ASSIGNMENT_UPLOAD AS A
   WHERE A.Student_ssn = M.Ssn) AS uploaded_as ,
  (SELECT COUNT(*)
   FROM ASSIGNMENT_GRADE AS A
   WHERE A.Student_ssn = M.Ssn) AS graded_as
FROM MOST_ENROLLED_USERS AS M,
    USER
WHERE M.Ssn = USER.Ssn
ORDER BY total DESC
LIMIT 5;
```
```

3 THE 5 ORIGINAL STATEMENT

1.The number of talents of students who have not uploaded any assignment they are responsible for...

```sql
SELECT S.Ssn,
    COUNT(*) AS skill_c
FROM STUDENT AS S,
    ASSIGNMENT AS A,
        USER_SKILLS AS SKILL
WHERE SKILL.User_ssn = S.Ssn
  AND NOT EXISTS
    (SELECT *
     FROM ASSIGNMENT_UPLOAD AS UPLOAD
     WHERE S.Ssn = UPLOAD.Student_ssn )
  AND A.Assignment_id IN
    (SELECT A.Assignment_id
     FROM ENROLLS AS E
     WHERE A.Course_id = E.Course_id
      AND S.Ssn = E.Student_ssn )
GROUP BY SKILL.User_ssn;
```

2.Project data including students who enroll in lessons given by teachers who speak more than one language...

```sql
SELECT E.Student_ssn,
    P.Pname,
    P.Pdesc,
    P.Start_date
FROM ENROLLS AS E,
```

```
    COURSE AS CRS,
    PROJECT AS P,
    WORKS_ON_PROJECT AS WOP
WHERE WOP.User_ssn = E.Student_ssn
 AND WOP.Project_id = P.Id
 AND CRS.Id IN
  (SELECT CRS.Id
   FROM TEACHER AS T
   WHERE CRS.Teacher_ssn = T.Ssn
    AND T.Ssn IN
     (SELECT T.Ssn
      FROM USER_LANGS AS UL
      WHERE T.Ssn = UL.User_ssn
      GROUP BY T.Ssn
      HAVING COUNT(*)>1) );
```

3.As a result of the evaluation of the teachers registered in the system based on Assignments, the data of the 5 teachers with the best evaluation...

In evaluation:

- The time which spend to teacher to evaluate submitted Assignment (Low Good)

- Grade point averages of the students in the relevant course (High Good)

```sql
SELECT C.Teacher_ssn,
    U.name AS Teacher_name,
    AVG(TIMESTAMPDIFF(HOUR, AU.Upload_date, AG.Grade_date)) AS avg_time,
    AVG(AG.Grade) AS t_avg_grade,
    count(*) AS std_count
FROM ASSIGNMENT_UPLOAD AS AU,
    ASSIGNMENT_GRADE AS AG,
```

```
    COURSE AS C,
    USER AS U
WHERE C.Teacher_ssn = U.Ssn
  AND C.Id = AG.Course_id
  AND AU.Assignment_id = AG.Assignment_id
GROUP BY C.Teacher_ssn
ORDER BY t_avg_grade DESC,
        avg_time ASC ;
```

4.The data of the teacher of the course that gives more than three Assignment and how many students take this course..

```sql
SELECT U.Ssn AS T_ssn,
       U.name AS Tname,
       C.Cname,
       Count(*) AS St_Count
FROM USER AS U,
           COURSE AS C,
           ENROLLS AS E
WHERE C.Teacher_ssn = U.Ssn
  AND C.Id = E.Course_id
  AND C.Id IN
    (SELECT A.Course_id
     FROM ASSIGNMENT AS A,
                  COURSE AS C
     WHERE A.Course_id = C.Id
     GROUP BY A.Course_id
     HAVING COUNT(*)>3)
GROUP BY E.Course_id;
```

```

5.The names, titles, average grades, the number of languages they know and the number of talents they have of users with the highest average according to graded assignments.

```sql
WITH RECURSIVE AVG_GRADES(Ssn, Avarage_grade) AS
 (SELECT Student_ssn,
        AVG(Grade)
  FROM ASSIGNMENT_GRADE
  GROUP BY Student_ssn
  UNION SELECT AG.Ssn,
            Ag.Avarage_grade
  FROM AVG_GRADES AS AG)
SELECT name,
      title,
      AG.Avarage_grade,
 (SELECT COUNT(*)
  FROM USER_LANGS AS UL
  WHERE UL.User_ssn = AG.Ssn) AS lang_count,

 (SELECT COUNT(*)
  FROM USER_SKILLS AS US
  WHERE US.User_ssn = AG.Ssn) AS skill_count
FROM AVG_GRADES AS AG,
    USER
WHERE USER.Ssn = AG.Ssn
ORDER BY AG.Avarage_grade DESC
LIMIT 5;
```
```