



Bilkent University

Department of Computer Engineering

---

# CS 353 Term Project

*CSCareer: A Quiz Based Hiring System*

## Design Report

**Group 22:** Hamza PEHLİVAN, Meryem EFE, Fırat YÖNAK, Selen UYSAL

Instructor: Özgür ULUSOY

Teaching Assistant: Arif USTA

# Contents

<b>1. Revised E/R Model</b>	<b>5</b>
<b>2. Relation Schemas</b>	<b>7</b>
2.1. User	7
2.2. Developer	7
2.3. Representative	8
2.4. Company	9
2.5. Admin	9
2.6. Profile	10
2.7. PreferredWorkingLocations	11
2.8. PreferredCompanies	11
2.9. RolePreferences	12
2.10. CompanyBlackList	13
2.11. ProjectInfo	13
2.12. WorkInfo	14
2.13. EducationInfo	15
2.14. Question	16
2.15. Choice	16
2.16. Category	17
2.17. CategorizedAs	18
2.18. Quiz	18
2.19. Quiz_questions	19
2.20. QuizTrial	20
2.21. Answer	21
2.22. Tries	21

2.23. Request	22
<b>3. Functional Dependencies and Normalization of Tables</b>	<b>23</b>
<b>4. Functional Components</b>	<b>23</b>
4.1. Use Cases / Scenarios	23
4.1.1. Developer	23
4.1.2. Company Representative	25
4.1.3. Admin	26
4.2. Algorithms	27
4.2.1. Algorithm of Calculating the Quiz Results	27
4.2.2. Algorithm of Sorting and Listing the Developers	28
4.2.3. Algorithm of Company Representative Request	28
<b>5. User Interface Design and Corresponding SQL Statements</b>	<b>29</b>
5.1. Homepage	29
5.2. Sign Up For Developer Page	30
5.3. Sign In For Developer Page	31
5.4. Developers' Edit Profile Page	32
5.4.1. Edit General Information	32
5.4.2. Edit Education Information	35
5.4.3. Edit Work Information	36
5.4.4. Edit Project Information	37
5.5. Sign Up For Representatives Page	38
5.6. Sign In For Representatives Page	40
5.7. Developer Takes a Quiz	41
5.8. Developer Solves a Question	42
5.9. Developer Views Quiz Results	44
5.10. Admin Creates a New Quiz	45

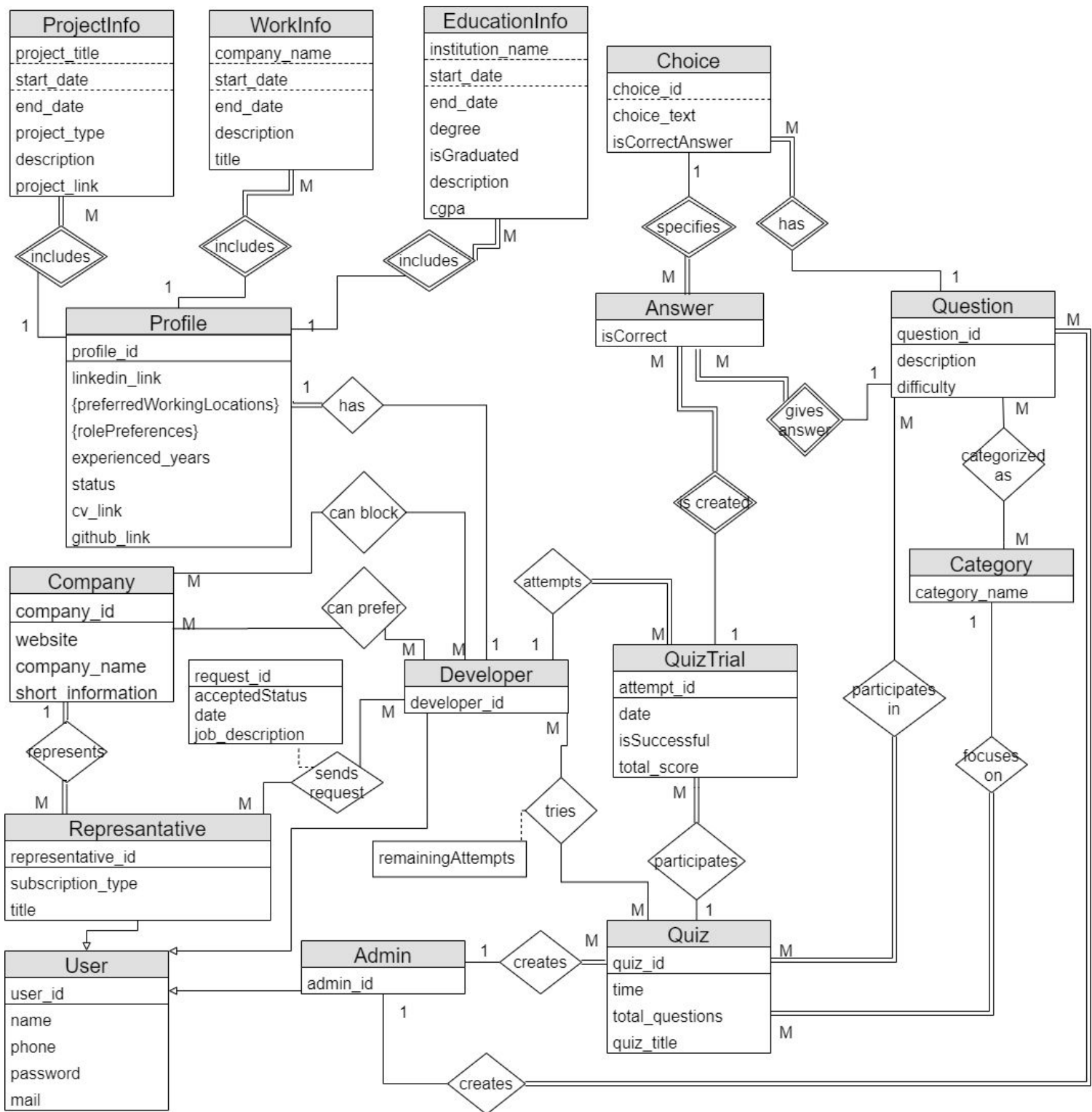
5.11. Enter and Choose the Quiz's Properties	46
5.12. Add Question and Complete Preparing the Quiz	47
5.13. Representative Send an Interview Request	48
5.13.1. Developer Responds to Interview Request	49
<b>6. Implementation Plan</b>	<b>50</b>
<b>7. Website</b>	<b>50</b>

# Design Report

*CSCareer: A Quiz Based Hiring System*

## 1. Revised E/R Model

- Various mistakes related to full participation are corrected.
- Account entity is replaced with User entity. Developer, Representative and Admin inherit from User.
- Date attribute is added to Send Request relation.
- Choice entity is added as a weak entity. That is because, choices cannot exist without a question.
- Category entity is added.
- Categorized As relation is added between Question and Category entities. There is many to many relation because a question can be included in many categories. Also, it is required that each question should have at least one category.
- QuizResult entity is removed and its attributes added to QuizTrial entity.
- New attributes are added to Quiz entity such as time and quiz title.
- Weak entity Answer is added between QuizTrial, Question and Choice entities. That is because, which answers given to which questions in a particular attempt should be addressed.
- Tries relation is added between Developer and Quiz entities to keep track of remaining attempts for a certain quiz.
- Company blacklist and preferred companies relations are added.



## 2. Relation Schemas

### 2.1. User

#### Relational Model:

User (user\_id, email, password, fullname, phone)

#### Functional Dependencies:

user\_id  $\rightarrow$  email, password, fullname, phone

email  $\rightarrow$  user\_id, password, fullname, phone

#### Candidate Keys:

{ (user\_id), (email) }

#### Normal Form:

BCNF

#### Table Definition:

```
CREATE TABLE User(  
    user_id      INT PRIMARY KEY AUTO_INCREMENT,  
    email        VARCHAR(45) NOT NULL UNIQUE,  
    password     VARCHAR(45) NOT NULL,  
    fullname     VARCHAR(45) NOT NULL,  
    phone        VARCHAR(10) );
```

### 2.2. Developer

#### Relational Model:

Developer (developer\_id, profile\_id)

#### Functional Dependencies:

developer\_id  $\rightarrow$  profile\_id

profile\_id  $\rightarrow$  developer\_id

**Candidate Keys:**

{ (developer\_id), (profile\_id) }

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE Developer(  
    developer_id      INT PRIMARY KEY,  
    profile_id        INT NOT NULL,  
    FOREIGN KEY (developer_id) REFERENCES User (user_id)  
    FOREIGN KEY (profile_id) REFERENCES Profile (profile_id) );
```

## 2.3. Representative

**Relational Model:**

Representative (representative\_id, subscription\_type, title, company\_id)

**Functional Dependencies:**

representative\_id → subscription\_type, title, company\_id

**Candidate Keys:**

{ (representative\_id) }

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE Representative(  
    representative_id      INT PRIMARY KEY,  
    subscription_type      VARCHAR(5) NOT NULL,  
    title                  VARCHAR(45) NOT NULL,  
    company_id             INT NOT NULL,
```



FOREIGN KEY (representative\_id) REFERENCES User (user\_id),  
FOREIGN KEY (company\_id) REFERENCES Company (company\_id),  
CHECK ( subscription\_type IN ('month', 'year') ) );

## 2.4. Company

### Relational Model:

Company (company\_id, website, company\_name, short\_information)

### Functional Dependencies:

company\_id → website, company\_name, short\_information

website → company\_id, company\_name, short\_information

### Candidate Keys:

{ (company\_id), (website) }

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE Company(  
    company_id      INT PRIMARY KEY AUTO_INCREMENT,  
    website         VARCHAR(45) NOT NULL UNIQUE,  
    company_name    VARCHAR(45) NOT NULL,  
    short_information TEXT(200) NOT NULL );
```

## 2.5. Admin

### Relational Model:

Admin (admin\_id)

**Functional Dependencies:**

None

**Candidate Keys:**

{ (admin\_id) }

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE Admin(  
    admin_id          INT PRIMARY KEY,  
    FOREIGN KEY (admin_id) REFERENCES User (user_id) );
```

## 2.6. Profile

**Relational Model:**

Profile (profile\_id, linkedin\_link, experienced\_years, status, cv\_link, github\_link)

**Functional Dependencies:**

profile\_id → linkedin\_link, experienced\_years, status, cv\_link, github\_link

**Candidate Keys:**

( {profile\_id} )

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE Profile (  
    profile_id          INT PRIMARY KEY AUTO_INCREMENT,  
    linkedin_link        VARCHAR(50) UNIQUE,  
    experienced_years    INT,
```

status	TINYINT(1) NOT NULL,
cv_link	VARCHAR(50) UNIQUE,
github_link	VARCHAR(50) UNIQUE );

## 2.7. PreferredWorkingLocations

### Relational Model:

PreferredWorkingLocations (developer\_id, city)

### Functional Dependencies:

None

### Candidate Keys:

{ (developer\_id, city) }

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE PreferredWorkingLocations(
    developer_id      INT PRIMARY KEY,
    city              VARCHAR(15) PRIMARY KEY,
    FOREIGN KEY (developer_id) REFERENCES Developer (developer_id) );
```

## 2.8. PreferredCompanies

### Relational Model:

PreferredCompanies (developer\_id, company\_id)

### Functional Dependencies:

None

**Candidate Keys:**

{ (developer\_id, company\_id) }

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE PreferredCompanies(  
    developer_id      INT PRIMARY KEY,  
    company_id        INT PRIMARY KEY,  
    FOREIGN KEY (developer_id) REFERENCES Developer (developer_id),  
    FOREIGN KEY (company_id) REFERENCES Company (company_id) );
```

## 2.9. RolePreferences

**Relational Model:**

RolePreferences (profile\_id, position)

**Functional Dependencies:**

None

**Candidate Keys:**

{ (profile\_id, position) }

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE PreferredWorkingLocations(  
    profile_id      INT PRIMARY KEY,  
    position        VARCHAR(30) PRIMARY KEY,  
    FOREIGN KEY (profile_id) REFERENCES Profile (profile_id) );
```

## 2.10. CompanyBlackList

### Relational Model:

CompanyBlackList (profile\_id, company\_id)

### Functional Dependencies:

None

### Candidate Keys:

{ (profile\_id, company\_id) }

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE CompanyBlackList (  
    profile_id    INT PRIMARY KEY,  
    company_id   INT PRIMARY KEY,  
    FOREIGN KEY (profile_id) REFERENCES Profile (profile_id),  
    FOREIGN KEY (company_id) REFERENCES Company (company_id) );
```

## 2.11. ProjectInfo

### Relational Model:

ProjectInfo (profile\_id, project\_title, start\_date, end\_date, project\_type, description, project\_link)

### Functional Dependencies:

profile\_id, project\_title, start\_date → end\_date, project\_type, description, project\_link

### Candidate Keys:

( {profile\_id, project\_title, start\_date} )

### Normal Form:

BCNF

**Table Definition:**

```
CREATE TABLE ProjectInfo (  
    profile_id          INT PRIMARY KEY,  
    project_title       VARCHAR(30) PRIMARY KEY,  
    start_date          DATE PRIMARY KEY,  
    end_date            DATE,  
    project_type        VARCHAR(20),  
    description         VARCHAR(100) NOT NULL,  
    project_link        VARCHAR(100),  
    FOREIGN KEY (profile_id) REFERENCES Profile (profile_id) );
```

**2.12. WorkInfo****Relational Model:**

WorkInfo (profile\_id, company\_name, start\_date, end\_date, title, description)

**Functional Dependencies:**

profile\_id, company\_name, start\_date → title, end\_date, description

**Candidate Keys:**

( {profile\_id , company\_name, date} )

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE WorkInfo (  
    profile_id          INT PRIMARY KEY,  
    company_name       VARCHAR(45) PRIMARY KEY,  
    start_date         DATE PRIMARY KEY,
```

end\_date            DATE,  
 title                VARCHAR(20) NOT NULL,  
 description        VARCHAR(100) NOT NULL,  
 FOREIGN KEY (profile\_id) REFERENCES profile (profile\_id );

## 2.13. EducationInfo

### Relational Model:

EducationInfo (profile\_id, instution\_name, start\_date, end\_date, degree, cgpa, isGraduated, description)

### Functional Dependencies:

profile\_id, institution\_name, start\_date → end\_date, degree, cgpa, isGraduated, description

### Candidate Keys:

( {profile\_id, institution\_name, date} )

### Normal Form:

BCNF

### Table Definition:

```

CREATE TABLE WorkInfo (
    profile_id            INT PRIMARY KEY,
    institution_name      VARCHAR(45) PRIMARY KEY,
    start_date            DATE PRIMARY KEY,
    end_date              DATE,
    isGraduted            TINYINT(1) NOT NULL,
    cgpa                  DOUBLE NOT NULL,
    description           VARCHAR(100),
    FOREIGN KEY (profile_id) REFERENCES Profile (profile_id );
  
```

## 2.14. Question

### Relational Model:

Question (question\_id, admin\_id, description, difficulty)

### Functional Dependencies:

question\_id  $\rightarrow$  admin\_id, description, difficulty

description  $\rightarrow$  question\_id, admin\_id, difficulty

### Candidate Keys:

( {question\_id}, {description} )

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE question (  
    question_id      INT PRIMARY KEY AUTO_INCREMENT,  
    admin_id         INT NOT NULL,  
    description       varchar(100) NOT NULL,  
    difficulty        ENUM('easy','medium','hard') NOT NULL,  
    UNIQUE KEY description_UNIQUE (description),  
    UNIQUE KEY question_id_UNIQUE (question_id),  
    FOREIGN KEY (admin_id) REFERENCES admin (admin_id) );
```

## 2.15. Choice

### Relational Model:

Choice (question\_id, choice\_id, choice\_text, isCorrectAnswer)

### Functional Dependencies:

question\_id, choice\_id  $\rightarrow$  choice\_text, isCorrectAnswer



**Candidate Keys:**

( {question\_id, choice\_id} )

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE choice (  
    choice_id          INT PRIMARY KEY,  
    question_id       INT PRIMARY KEY,  
    choice_text        VARCHAR(45) NOT NULL,  
    isCorrectAnswer    TINYINT(1) NOT NULL,  
    FOREIGN KEY (question_id) REFERENCES question (question_id) );
```

## 2.16. Category

**Relational Model:**

Category(category\_name)

**Functional Dependencies:**

None

**Candidate Keys:**

( {category\_name} )

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE category (  
    category_name VARCHAR(15) PRIMARY KEY );
```

## 2.17. CategorizedAs

### Relational Model:

CategorizedAs (category\_name,question\_id)

### Functional Dependencies:

None

### Candidate Keys:

( {category\_name,question\_id} )

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE categorized_as (  
    category_name    VARCHAR(15) PRIMARY KEY,  
    question_id      INT PRIMARY KEY,  
    FOREIGN KEY (category_name) REFERENCES category (category_name),  
    FOREIGN KEY (question_id) REFERENCES question (question_id) );
```

## 2.18. Quiz

### Relational Model:

Quiz (quiz\_id, admin\_id, category\_name, total\_questions, quiz\_title, time)

### Functional Dependencies:

quiz\_id → admin\_id, category\_name, total\_questions, quiz\_title, time

### Candidate Keys:

( {quiz\_id} )

### Normal Form:

BCNF

**Table Definition:**

```
CREATE TABLE quiz (  
    quiz_id          INT PRIMARY KEY AUTO_INCREMENT,  
    admin_id         INT NOT NULL,  
    category_name    VARCHAR(15) NOT NULL,  
    total_questions  INT NOT NULL,  
    quiz_title       VARCHAR(10) NOT NULL,  
    time             TIME NOT NULL,  
    UNIQUE KEY quiz_id_UNIQUE (quiz_id),  
    FOREIGN KEY (admin_id) REFERENCES admin (admin_id),  
    FOREIGN KEY (category_name) REFERENCES category (category_name));
```

**2.19. Quiz\_questions****Relational Model:**

Quiz\_questions (quiz\_id, question\_id)

**Functional Dependencies:**

None

**Candidate Keys:**

( {quiz\_id, question\_id} )

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE quiz_questions (  
    quiz_id          INT PRIMARY KEY,  
    question_id      INT PRIMARY KEY,  
    FOREIGN KEY (question_id) REFERENCES question (question_id),
```

FOREIGN KEY (quiz\_id) REFERENCES quiz (quiz\_id) );

## 2.20. QuizTrial

### Relational Model:

QuizTrial (attempt\_id, developer\_id, quiz\_id, date, isSuccessful, total\_score)

### Functional Dependencies:

attempt\_id → developer\_id, quiz\_id, date, isSuccessful, total\_score

### Candidate Keys:

( {attempt\_id} )

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE quiztrial (  
    attempt_id          INT PRIMARY KEY AUTO_INCREMENT,  
    developer_id        INT NOT NULL,  
    quiz_id             INT NOT NULL,  
    date                DATE NOT NULL,  
    isSuccessful         TINYINT(1),  
    total_score          INT,  
    UNIQUE KEY attempt_id_UNIQUE (attempt_id),  
    FOREIGN KEY (developer_id) REFERENCES developer (developer_id),  
    FOREIGN KEY (quiz_id) REFERENCES quiz (quiz_id) );
```

## 2.21. Answer

### Relational Model:

Answer(attempt\_id,question\_id,choice\_id, isCorrect)

### Functional Dependencies:

attempt\_id, question\_id, choice\_id → isCorrect

### Candidate Keys:

( {attempt\_id, question\_id, choice\_id} )

### Normal Form:

BCNF

### Table Definition:

```
CREATE TABLE answer (  
    attempt_id          INT PRIMARY KEY,  
    question_id         INT PRIMARY KEY,  
    choice_id           INT PRIMARY KEY,  
    iscorrect           TINYINT(1) NOT NULL,  
    FOREIGN KEY (attempt_id) REFERENCES quiztrial (attempt_id),  
    FOREIGN KEY (question_id) REFERENCES question (question_id),  
    FOREIGN KEY (choice_id) REFERENCES choice(choice_id) );
```

## 2.22. Tries

### Relational Model:

Tries (developer\_id,quiz\_id, remainingAttempts)

### Functional Dependencies:

developer\_id, quiz\_id → remainingAttempts

**Candidate Keys:**

( {developer\_id, quiz\_id} )

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE tries(  
    developer_id          INT PRIMARY KEY,  
    quiz_id               INT PRIMARY KEY,  
    remaining_attempts    INT NOT NULL DEFAULT 3,  
    FOREIGN KEY (developer_id) REFERENCES developer (developer_id)  
    FOREIGN KEY (quiz_id) REFERENCES quiz (quiz_id) );
```

## 2.23. Request

**Relational Model:**

Request (request\_id, acceptedStatus, date, job\_description, developer\_id, representative\_id)

**Functional Dependencies:**

request\_id → developer\_id, representative\_id, acceptedStatus, date, job\_description

**Candidate Keys:**

{ (request\_id) }

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE Request(  
    request_id            INT PRIMARY KEY,  
    acceptedStatus        ENUM ('accepted', 'declined', 'waiting') NOT NULL,
```

```

date            DATE NOT NULL,
job_description  VARCHAR (100) NOT NULL,
developer_id     INT NOT NULL,
representative_id INT NOT NULL,

FOREIGN KEY (developer_id) REFERENCES Developer (developer_id),

FOREIGN KEY (representative_id) REFERENCES Representative
(representative_id) );

```

### 3. Functional Dependencies and Normalization of Tables

All of the relations in our design report are in BCNF (Boyce-Codd Normal Form). Since it is a strict form of normalization that removes the redundancies caused by functional dependencies, there is no need for additional normalization.

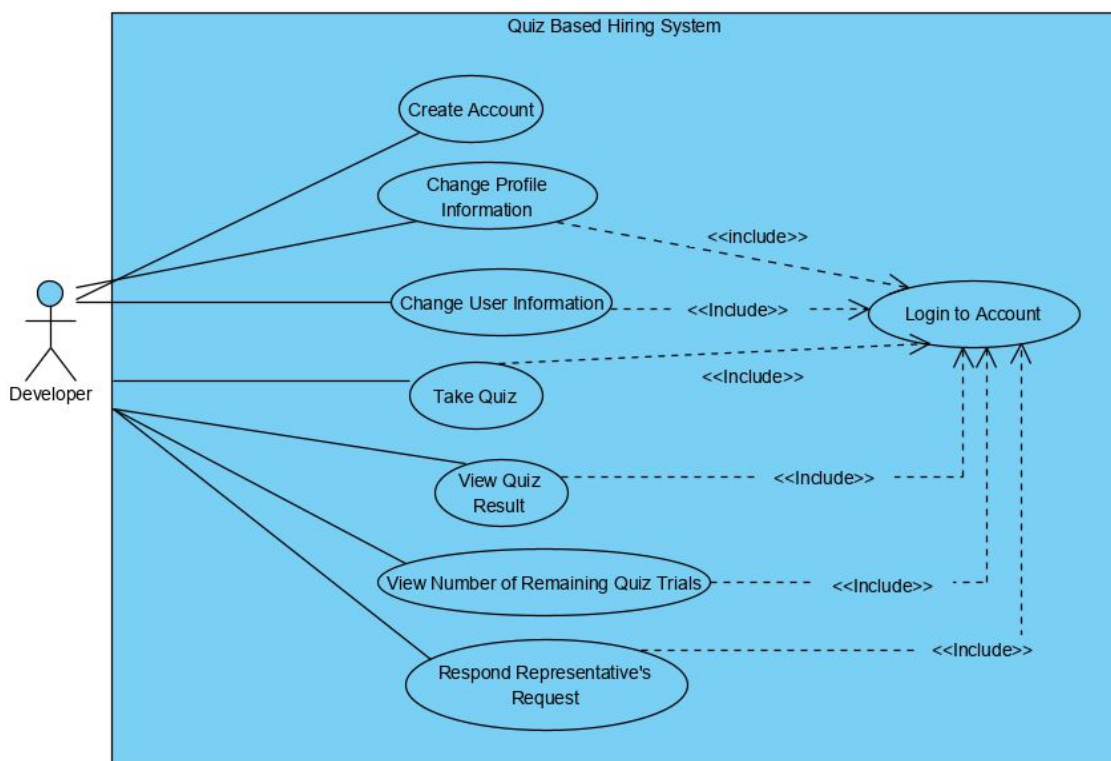
## 4. Functional Components

### 4.1. Use Cases / Scenarios

#### 4.1.1. Developer

- **Create Account:** Developers can create an account if they do not have one already. They enter their email, name, phone and password to create it.
- **Login to Account:** Developers can login to their existing accounts. They need to login to change their profile and user information, take quizzes, view their quiz results and number of remaining quiz trials as well as responding to the representatives' requests.
- **Change Profile Information:** Developers can change their profile information such as their GitHub link, LinkedIn link, preferred working locations, preferred companies, role preferences, experienced years, company blacklist (which companies should not see their profile), status and the link to their CV.

- Change User Information: Developers can change their user information such as their name, phone and password. However, they cannot change their mail, they need to create a new account with the desired mail if that is the case.
- Take Quiz: Developers can take quizzes in different categories to show their knowledge in that category.
- View Quiz Result: Developers can view their quiz result and get feedback after completing each quiz.
- View Number of Remaining Quiz Trials: Developers have three trials for each quiz. They can view the remaining number of trials for each quiz.
- Respond Representative's Request: When the representatives want to communicate with a developer, they send a request. Developers will be able to accept or decline that request.

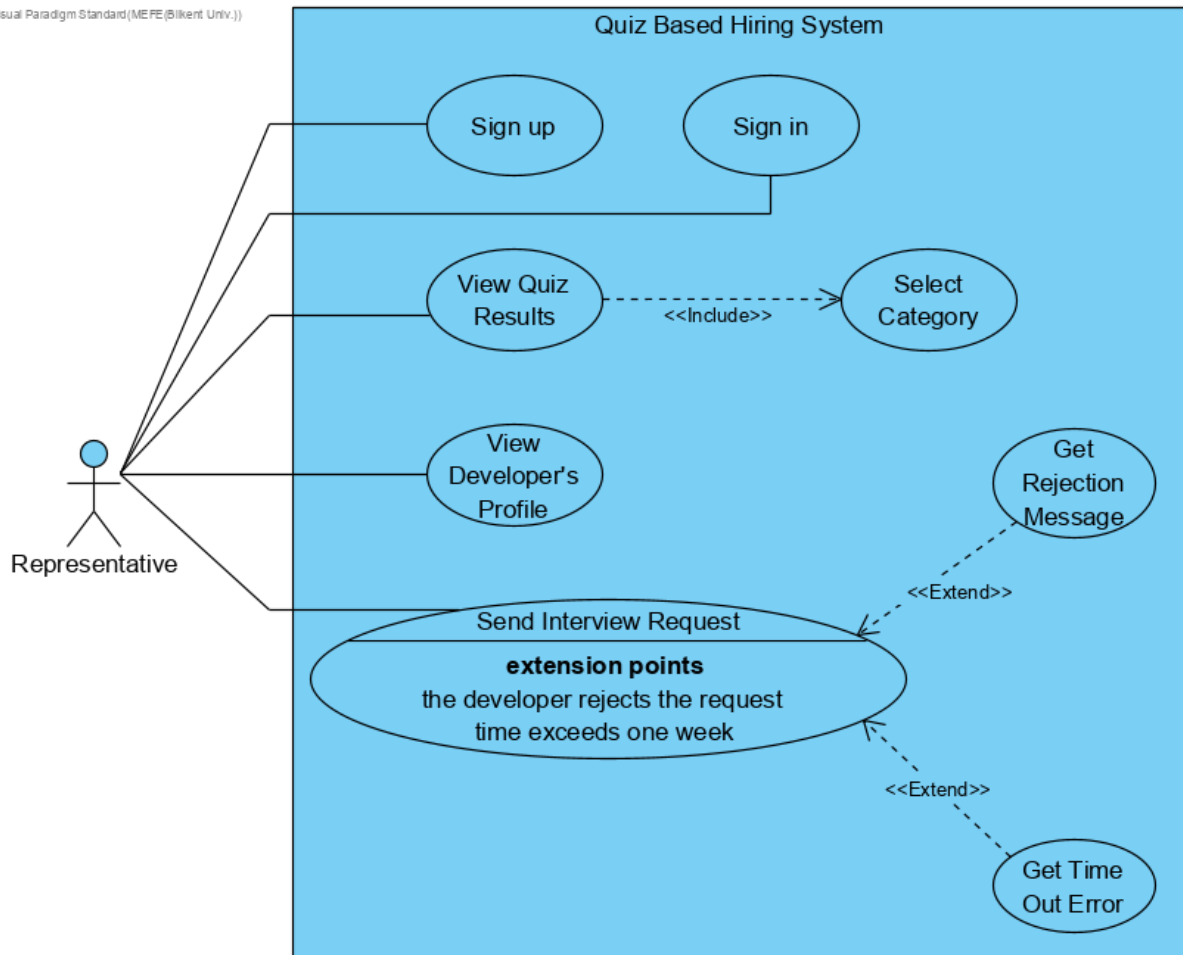


**Figure 1.** Use Case of Developer



#### **4.1.2. Company Representative**

- Sign up: If the company of representative is registered, the representative can sign up to the system by entering her/his information and selecting the company.
- Sign in: The representatives can sign in to the system by email and password if they already have account. In order to perform the following functions, the representative should sign in to the system.
- View Quiz Results: The representatives can view all the quiz results in the descending score order.
- Select Category: The representatives can select a category or categories if the company needs a developer in a particular category. Then, they can show the results in that category.
- View Developer's Profile: The representative can review developers' profiles and see the other quiz results.
- Send Interview Request: The representatives can send interview request to a developer. Then, the system sends the contact information with the representative unless there is any exception as the followings.
- Get Rejection Message: After sending the interview request, if the developer rejects this request, the developer is informed and gets a rejection message.
- Get Time Out Error: After sending the interview request, if one week passes and there is no response, the developer is informed and gets time error message.

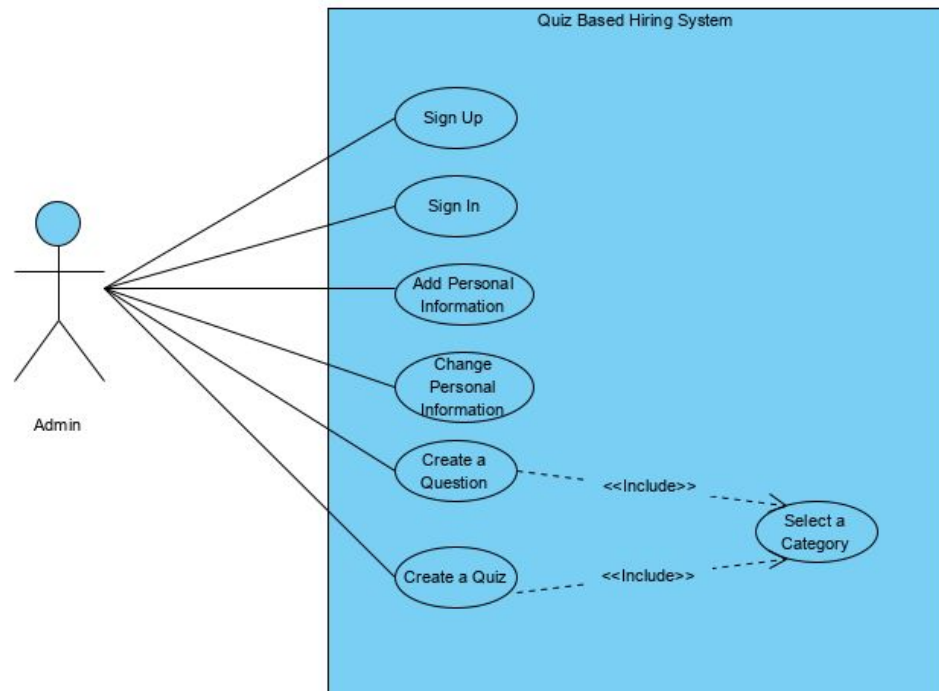


**Figure 2.** Use Case of Representative

#### 4.1.3. Admin

- Sign Up: Admins must sign up when they first enter to system like each user and specify themselves as an admin.
- Sign In: Admins have to sign in to the system with their password and email like every user if they have an account. Sign in process is necessary to do following actions.
- Add Personal Information: Admin can add their name and phone number into their user account.
- Change Personal Information: Admins are able to change their user information like phone, name and password; however, they are not allowed to change their email like other users.

- Create a Question: Questions are created by admins in this system. They can create questions if they provide necessary information about question.
- Create a Quiz: Admins can create a quiz on this system. They benefit from existing questions when they prepare a quiz.
- Select a Category: Admins need to select category when they prepare either a question or quiz because system categorize them by this information.



**Figure 3.** Use Case of Admin

## 4.2. Algorithms

### 4.2.1. Algorithm of Calculating the Quiz Results

Every developer can take any kind of quiz from this system to show his or her skills and knowledge. To display their capabilities, their results should be calculated at the end of a quiz. Each quiz has its own scoring distribution; that's why, every one of them needs to be calculated question by question. When a developer complete the quiz, system compare developer's choices with answers one by one. If the developer's answer is wrong, he or she cannot get any point from that question. If developer's answer is true, s/he gets points by question category and difficulty. Each

difficulty level has different score. Receiving points increase from easy to hard. Scoring is also different for different category. Each quiz can include questions from different categories and every one of them gives points by the category of quiz. The system checks difficulty level of question and the importance of category of it for quiz category. According to its checking process, it calculates final result of developer.

#### **4.2.2. Algorithm of Sorting and Listing the Developers**

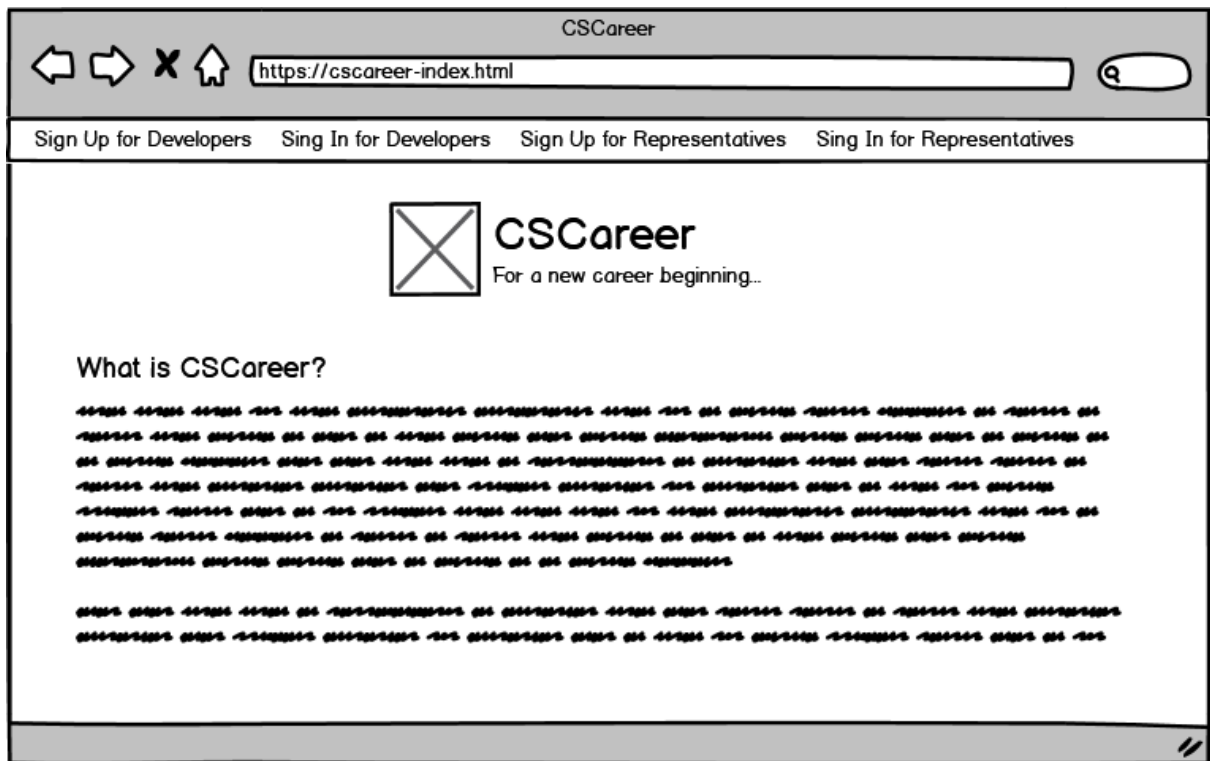
According to the total scores that the developers got from the quizzes in a specific category, they will be listed so that the developers with better scores will be listed at the top. Therefore, the company representatives can see the developers' names and their scores in a descending order. Representatives will be able to view the list in each category.

#### **4.2.3. Algorithm of Company Representative Request**

A company may need a developer who is good at a particular area. Therefore, the representative can select the desired category and see the developers' results in that category. If a developer doesn't want to reveal her/his results to this company, the representative cannot see them in the list. If the company likes a developer's result and wants to interview with the developer, the representative can send an interview request to the developer on behalf of the company. After the interview request, the developer can accept or reject it. If the request is accepted, the system shares the contact information of the developer with the representative. Otherwise, the system asks the developer whether s/he wants to block this company or not. The developer can block the company and never see this company's requests again or s/he rejects only this request.

## 5. User Interface Design and Corresponding SQL Statements

### 5.1. Homepage



In the homepage, there will be a brief description about what CSCareer is. Users will be able to get detailed informations by clicking *How to Use* link which will be in the header. Also, in the header, there will be links to sign up and sign in for both developers and company representatives.

## 5.2. Sign Up For Developer Page

CSCareer

Sign Up for Developer

Full Name \*:

E-mail \*:

Password \*:

Confirm Password \*:

Phone Number:

Sign Up

If the developer has no account, s/he can create an account by entering the necessary informations in this page. In the form, it is mandatory to fill in the sections with an asterisk. Also, the ones written in the password and confirm password sections must be the same.

**Inputs:** @fullname, @email, @password, @confirmpassword, @phonenumber

### SQL Statements

- Firstly, values are inserted to *user* table:

```
INSERT INTO user (email, password, fullname, phone)
VALUES (@email, @password, @fullname, @phonenumber)
```

- Secondly, a new insertion is made in *developer* table since this user is a developer:

```
INSERT INTO developer
VALUES (SELECT user_id FROM user WHERE email = @email)
```

### 5.3. Sign In For Developer Page

The screenshot shows a web browser window with the title 'CSCareer'. The address bar contains the URL 'https://cscareer-developer-signin.html'. The browser's navigation bar includes links for 'Homepage', 'Sign Up for Developers', 'Sign Up for Representatives', 'Sing In for Representatives', and 'How to Use'. The main content area features the 'CSCareer' logo on the left and the title 'Sign In for Developer' on the right. A central form box contains the following elements: an 'E-mail:' label followed by a text input field with an '@' symbol inside; a 'Password:' label followed by a text input field; and a 'Sign In' button below the password field. The browser's status bar at the bottom right shows a double-slash icon.

In this page, developers will be able to sign in to the system by entering their email and password.

**Inputs:** @email, @password

#### SQL Statements

```
SELECT *  
FROM user  
WHERE ( email = @email AND  
        password = @password AND  
        user_id IN (SELECT developer_id FROM developer) );
```

## 5.4. Developers' Edit Profile Page

In the Edit Profile page, there will be four options. Developer will be able to edit general informations, edit / add education informations, edit / add project informations, and edit / add work informations.

### 5.4.1. Edit General Information

The screenshot shows a web browser window with the title 'CSCareer' and the URL 'https://cscareer-developer-profile-edit-general.html'. The page has a header with the 'CSCareer' logo and a 'Edit Profile' link. The main content area is titled 'General Information' and contains the following fields:

- LinkedIn Link:
- Github Link:
- CV Link:
- Preferred Working Locations:
- Preferred Companies:
- Role Preferences:
- Experienced Years:
- Company Black List:
- ☐ I am open to job opportunities.

A 'Save Changes' button is located at the bottom right of the form.

**Inputs:** @profile\_id, @developer\_id, @linkedin\_link, @github\_link, @cv\_link, @preferred\_cities, @preferred\_companies, @preferred\_roles, @experienced\_years, @blocked\_companies, @status

### SQL Statements

In this part, there will be changes in several tables.



### **Update profile informations:**

UPDATE profile

SET

linkedin\_link = @linkedin\_link,

github\_link = @github\_link,

cv\_link = @cv\_link,

experienced\_years = @experienced\_years,

status = @status,

WHERE ( profile\_id = @profile\_id);

### **Add Preferred Working Location:**

INSERT INTO preferredworkinglocations

VALUES (@profile\_id, @preferred\_cities)

### **Delete Preferred Working Location:**

DELETE FROM preferredworkinglocations

WHERE profile\_id = @profile\_id AND preferred\_cities = @preferred\_cities

### **Add Preferred Company:**

INSERT INTO preferredcompanies

VALUES (@profile\_id, @preferred\_companies)

**Delete Preferred Company:**

```
DELETE FROM preferredcompanies  
  
WHERE    developer_id = @developer_id AND  
         preferred_companies= @preferred_companies
```

**Add Role Preferences:**

```
INSERT INTO rolepreferences  
  
VALUES (profile_id, preferred_roles)
```

**Delete Role Preferences:**

```
DELETE FROM rolepreferences  
  
WHERE profile_id = @profile_id AND preferred_roles = @preferred_roles
```

**Add Company to Black List:**

```
INSERT INTO companyblacklist  
  
VALUES (@profile_id, @bocked_companies)
```

**Delete Company from Black List:**

```
DELETE FROM companyblacklist  
  
WHERE    developer_id = @developer_id AND  
         bocked_companies = @bocked_companies
```

### 5.4.2. Edit Education Information

The screenshot shows a web browser window titled 'CSCareer' with the URL 'https://cscareer-developer-profile-edit-education.html'. The page has a header with 'CSCareer' and a search bar. The main content area is titled 'Edit Profile' and 'Education Information'. It contains a form for editing education information. The form has a title 'Education Information' and a list of fields: 'Bilkent University', 'Start Date: September 19, 2016', 'End Date: -', 'Department : Computer Science', 'CGPA: 3.25', 'Not Graduated', and 'Description: I studied BUSEL one year. I am planning to graduate'. There is an 'Add New Education Information' button at the bottom of the form and a 'Save Changes' button on the right. A small edit icon (pencil) is next to the form fields.

**Inputs:** @profile\_id, @instution\_name, @start\_date, @end\_date, @degree, @cgpa, @isGraduated, @description

### SQL Statements

#### Update education information:

```
UPDATE EducationInfo
SET  end_date = @end_date,
      degree = @degree,
      cgpa = @cgpa,
      isGraduated = @isGraduated,
      description = @description
WHERE profile_id = @profile_id AND instution_name = @instution_name
AND start_date = @start_date;
```

### Add new education information:

```
INSERT INTO EducationInfo
```

```
VALUES ( @profile_id, @instution_name, @start_date, @end_date,  
@degree, @cgpa, @isGraduated, @description)
```

### 5.4.3. Edit Work Information

The screenshot shows a web browser window titled 'CSCareer' with the URL 'https://cscareer-developer-profile-edit-education.html'. The page has a header with 'CSCareer' and a search bar. The main content area is titled 'Edit Profile' and 'Work Information'. It contains a form for editing work information. The form has a title 'XXX Game', start date 'July 15, 2018', end date 'September 15, 2018', title 'Summer Intern', and description 'I worked in game development department on a new mobile application by using Flutter.' There is an 'Add New Work Information' button and a 'Save Changes' button.

**Inputs:** @profile\_id, @company\_name, @start\_date, @end\_date, @title, @description

### SQL Statements

#### Update work information:

```
UPDATE WorkInfo
```

```
SET end_date = @end_date,
```

```
title = @title,
```

```
description = @description
```

```
WHERE profile_id = @profile_id AND company_name = @company_name  
AND start_date = @start_date;
```

#### **Add new work information:**

```
INSERT INTO WorkInfo
```

```
VALUES ( @profile_id, @company_name, @start_date, @end_date, @title,  
@description)
```

#### **5.4.4. Edit Project Information**

The screenshot shows a web browser window titled 'CSCareer' with the URL 'https://cscareer-developer-profile-edit-education.html'. The page has a header with a 'CSCareer' logo and a navigation bar with 'Edit Profile' and 'Project Information' links. The main content area is titled 'Project Information' and contains a form for editing project details. The form includes a title 'Quiz Based Hiring System', start and end dates (February 1, 2020 and April 1, 2020), project type 'Database Management', description 'In this project, ...', and project link 'github/mrrobot/CS353\_Term\_Project'. There is an 'Add New Project Information' button at the bottom of the form and a 'Save Changes' button to the right. A vertical scrollbar is visible on the right side of the form area.

**Inputs:** @profile\_id, @project\_title, @start\_date, @end\_date, @project\_type,  
@description, @project\_link

#### **SQL Statements**

##### **Update project information:**

```
UPDATE ProjectInfo
```

```
SET end_date = @end_date,
```

project\_type = @project\_type,

description = @description,

project\_link = @project\_link

WHERE profile\_id = @profile\_id AND project\_title = @project\_title AND  
start\_date = @start\_date;

### Add new project information:

INSERT INTO ProjectInfo

VALUES ( @profile\_id, @project\_title, @date, @project\_type, @description,  
@project\_link)

## 5.5. Sign Up For Representatives Page

The screenshot shows a web browser window titled "CSCareer" with the address bar displaying "https://cscareer-representative-signup.html". The browser's navigation bar includes links for "Homepage", "Sign Up for Developers", "Sing In for Developers", "Sing In for Representatives", and "How to Use". The main content area features the "CSCareer" logo and the heading "Sign Up for Representative". Below this, a form is presented with the following fields: "Full Name \*" (text input), "E-mail \*" (text input with an "@" symbol), "Password \*" (text input), "Confirm Password \*" (text input), "Phone Number:" (text input with a "( )" symbol), "Company \*" (dropdown menu showing "Company List"), "Title \*" (text input), and "Subscription Type \*" (dropdown menu showing "Subscription Types"). A "Sign Up" button is located at the bottom right of the form. The browser's status bar at the bottom shows a double-slash icon.

Representatives' sign up page will be similar to developers' sign up page. However, in this form, differently from the developer' sign up page, there will be company, title,

and subscription type sections. Representative should select her/his company from the list. Also, s/he should type her/his position in the company to title section.

**Inputs:** @fullname, @email, @password, @confirmpassword, @phonenumber, @company\_id, @title, @subscriptiontype

### **SQL Statements**

- Values are inserted to *user* table.

```
INSERT INTO user (email, password, fullname, phone)
VALUES (@email, @password, @fullname, @phonenumber)
```

- Since this user is representative, rest of the informations are inserted to *representative* table.

```
INSERT INTO representative
VALUES ( (SELECT user_id FROM user WHERE email = @email),
@subscription_type, @title, @company_id )
```

## 5.6. Sign In For Representatives Page

The screenshot shows a web browser window with the title 'CSCareer'. The address bar contains the URL 'https://cscareer-representative-signin.html'. The browser's navigation bar includes links: 'Homepage', 'Sign Up for Developers', 'Sing In for Developers', 'Sign Up for Representatives', and 'How to Use'. The main content area features the 'CSCareer' logo on the left and the title 'Sign In for Representative' on the right. A central form box contains the following elements: an 'E-mail:' label followed by a text input field with an '@' symbol inside; a 'Password:' label followed by a text input field; and a 'Sign In' button below the password field. The browser's status bar at the bottom right shows a double-slash icon.

In this page, representatives will be able to sign in to the system by entering their email and password.

**Inputs:** @email, @password

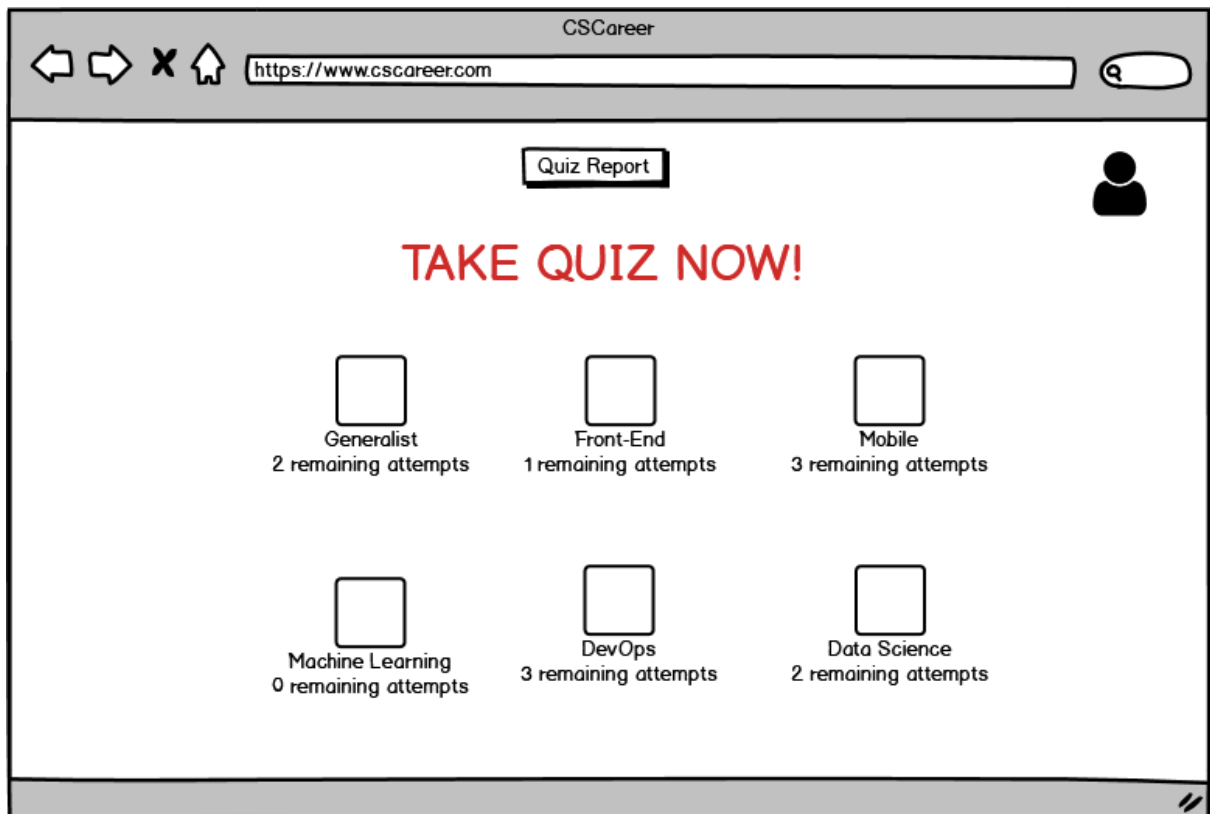
### SQL Statements

```
SELECT *  
FROM user  
WHERE ( email = @username AND  
        password = @password AND  
        user_id IN (SELECT representative_id FROM representative) );
```

\*\*\* Since sign up/sign in functions of admin will be similar to developer and representative, the use cases related to admin is not added to this report.



## 5.7. Developer Takes a Quiz



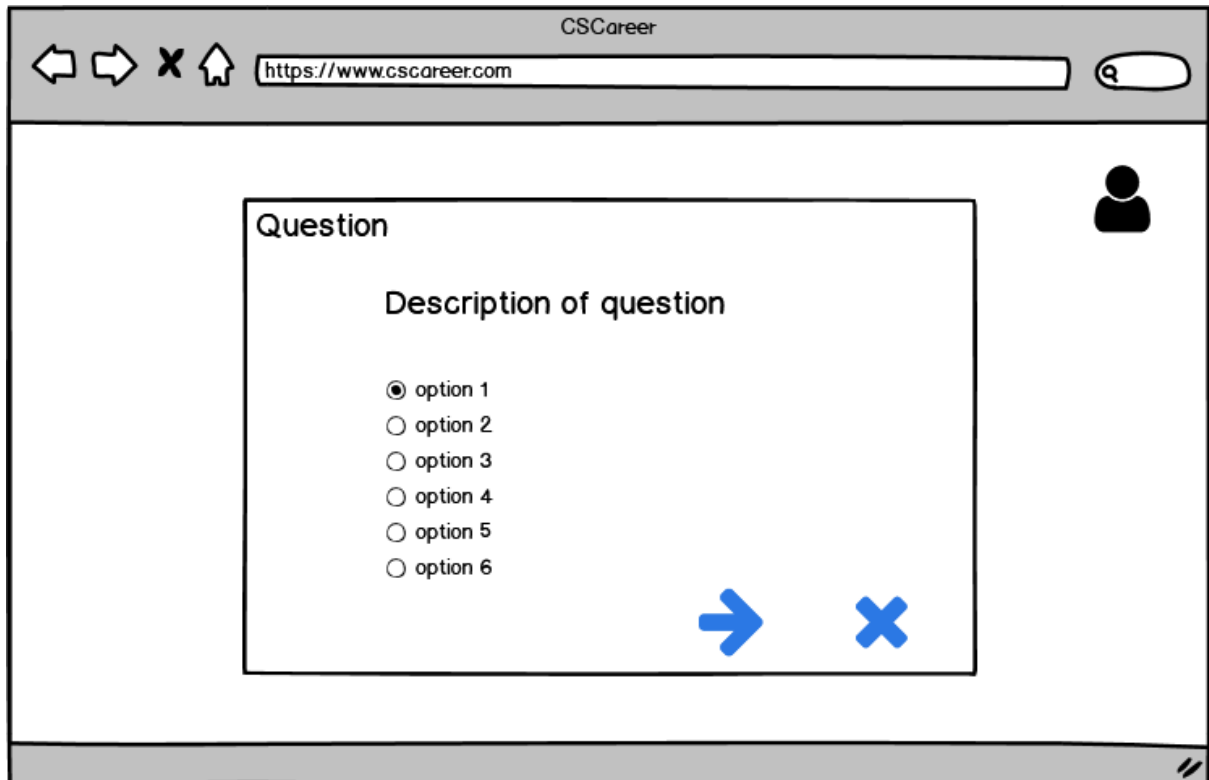
The developer can see different quizzes and how many attempts they have for a quiz. This information will be displayed using view.

```
create view view_quizzes as  
select quiz_id, remainingAttempts, category_name  
from tries natural join quiz  
where developer_id = @developer_id
```

When developer selects a quiz, a new quiz trial will be added.

```
insert into quiztrial (attempt_id, developer_id, quiz_id, date, isSuccessful,  
total_score)  
values (@developer_id, @quiz_id, @date, null, null)
```

## 5.8. Developer Solves a Question



The questions should be found after selecting a quiz.

```
select question_id, question_description
from quiz_question natural join question
where quiz_id = @quiz_id
order by question_id asc
```

For each question\_id found by above query, choices will be found.

```
select choice_id, choice_text
from choice
where question_id = @question_id
```

When developer chooses an answer and presses next button, answer for the question will be recorded. The developer can also pass the question by not marking

an option, then pressing next button. Passing a question corresponds to the choice\_id = 0.

First, it will be determined whether the choice is correct or not.

```
select isCorrectAnswer
from choice
where question_id = @question_id and choice_id = @choice_id
```

Then answer will be recorded in answer table together with isCorrectAnswer value.

```
insert into answer
values ( @attempt_id, @question_id, @choice_id, isCorrectAnswer)
```

If the developer presses end the quiz button (denoted as X in above mock-up), isSuccessful and total\_score values will be updated.

Firstly, total\_score will be found

```
select sum(*) as total_score
from answer
where isCorrect = 1
```

Secondly, it will be determined whether the result is successful or not. If the found total score > 15, the quiz is successful.

Finally, quiz trial will be updated with found values

```
update quiztrial
set isSuccessful = @succesful, total_score = @total_score
where attempt_id = @attempt_id
```

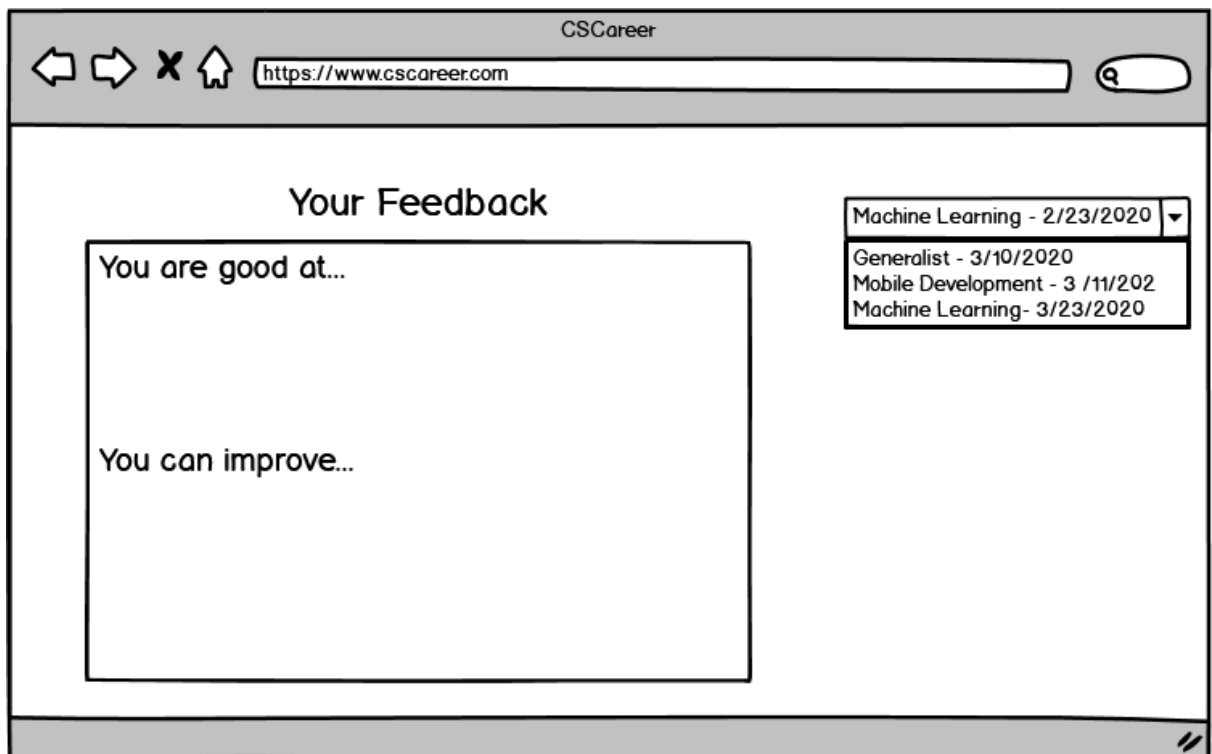
After finishing quiz, remaining attempts for the quiz will be decreased by one.

**update** tries

**set** remainingAttempts = remainingAttempts – 1

**where** quiz\_id = @quiz\_id and developer\_id = @developer\_id

## 5.9. Developer Views Quiz Results



The drop-down menu will be displayed using following command.

**create view** view\_result **as**

**select** attempt\_id, category\_name, date

**from** quiztrial natural join quiz

**where** developer\_id = @developer\_id

Feedback will be displayed according to chosen attempt. If the score calculated by below query is greater than 10, the category will be displayed in “You are good at” part. Otherwise, it will be displayed in “You can improve” part.

**create view** feedback **as**

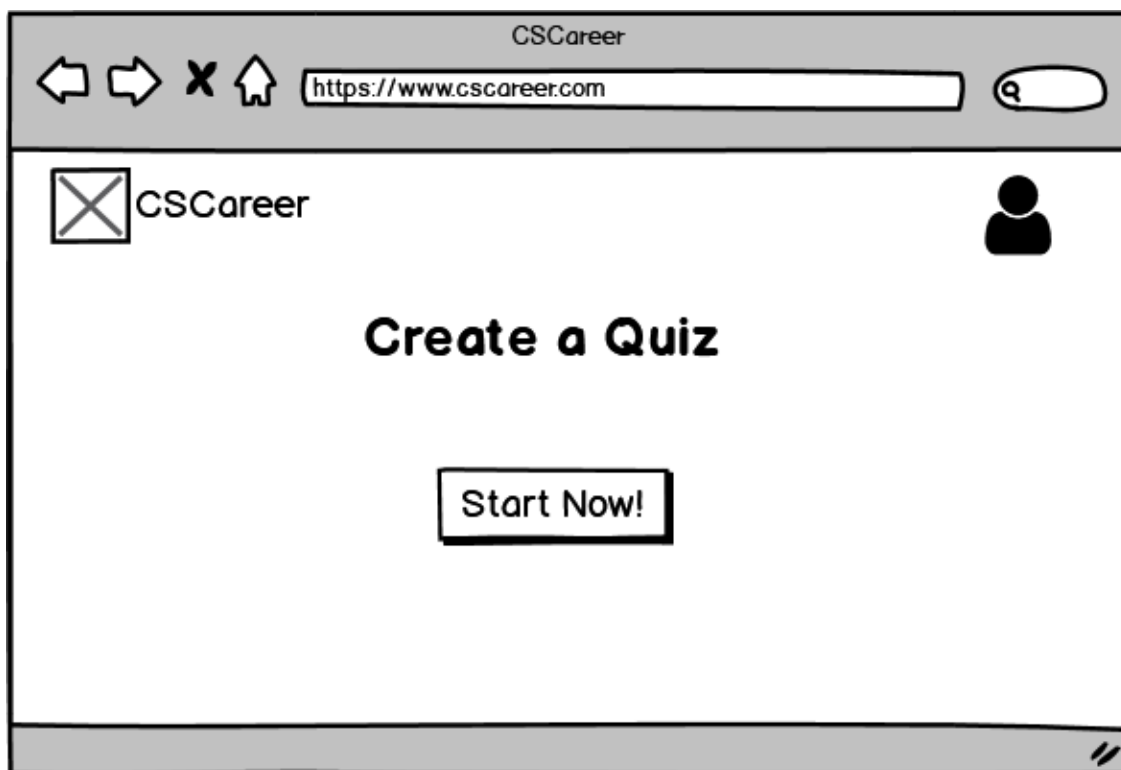
**select** category\_name, **sum** (**case when** isCorrect = 1 **then** 1 **else** -1 **end**)  
**as** score

**from** answer **natural join** categorized\_as

**where** attempt\_id = @attempt\_id

**group by** category\_name

### 5.10. Admin Creates a New Quiz



Only admin can create a quiz. Admin can start creating by the “Start Now!” button.

## 5.11. Enter and Choose the Quiz's Properties

The screenshot shows a web browser window titled "CSCareer" with the URL "https://www.cscareer.com". The page features a header with a "CSCareer" logo and a user profile icon. The main content area contains a form for creating a quiz, which is divided into two sections. The top section includes a text input field labeled "Name the Quiz", a dropdown menu labeled "Select its Category", and a button labeled "Create New Category". The bottom section includes a dropdown menu labeled "Select the maximum time given to the quiz" and a "Next" button.

**Inputs:** @quiz\_title, @category\_name, @time

**SQL Statements:**

**Creating a quiz:**

```
INSERT INTO quiz (quiz_id, admin_id, category_name, total_questions,
quiz_title, time)
```

```
VALUES (@quiz_id, @admin_id, @category_name, 0, @quiz_title, @time)
```

**Creating a new category:**

```
INSERT INTO category
```

```
VALUES (@category_name)
```

## 5.12. Add Question and Complete Preparing the Quiz

The screenshot shows a web browser window titled 'CSCareer' with the URL 'https://www.cscareer.com'. The page features a 'Machine Learning Quiz' form. The form includes a text input for 'Enter your question', two radio buttons for 'Enter an answer choice', a plus icon and text for 'Add a new answer choice', a dropdown for 'Select the correct answer', and another dropdown for 'Select its difficulty'. At the bottom of the form are 'Add Question' and 'Done' buttons. A user profile icon is visible in the top right corner of the page.

**Inputs:** @description, @choice\_text, @isCorrectAnswer, @difficulty

**SQL Statements:**

**Creating a new choice:**

```
INSERT INTO choice (question_id, choice_id, choice_text, isCorrectAnswer)
VALUES (@question_id, @choice_id, @choice_text, @isCorrectAnswer)
```

**Creating a new question:**

```
INSERT INTO question (question_id, admin_id, description, difficulty)
VALUES (@question_id, @admin_id, @description, @difficulty)
INSERT INTO quiz_questions (quiz_id, question_id)
VALUES (@quiz_id, @question_id)
```

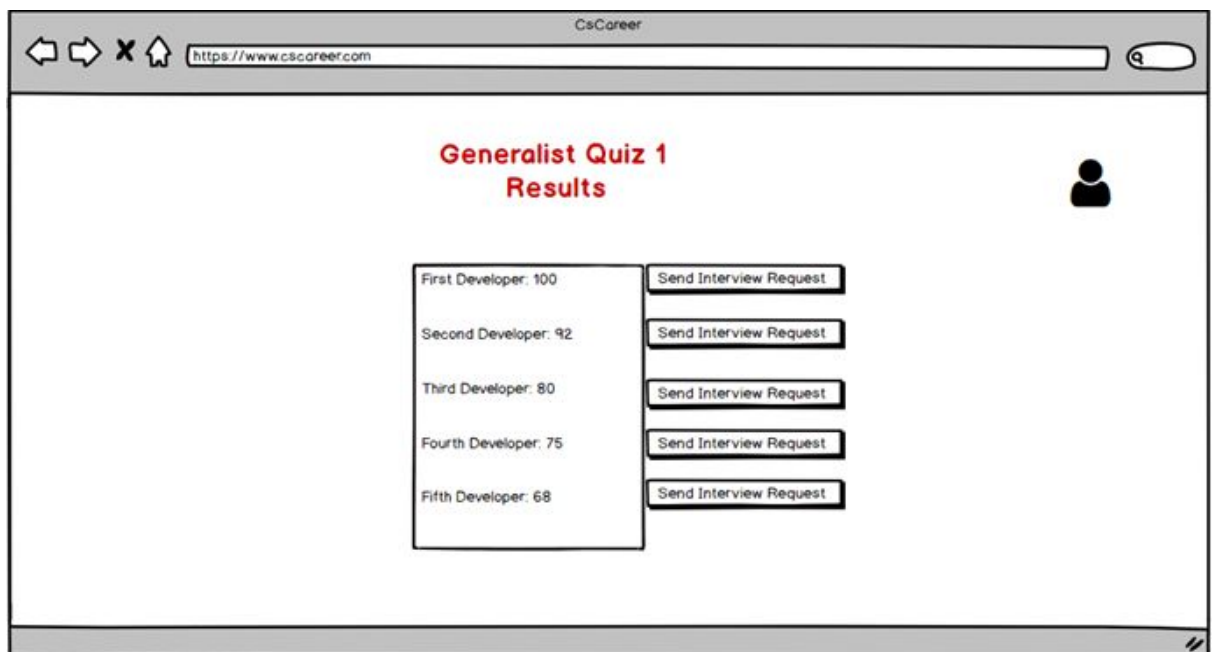
Incrementing the number of questions in the quiz by 1 after each question is added:

```
UPDATE quiz (total_questions)
```

```
SET total_questions = total_questions + 1
```

```
WHERE quiz_id = @quiz_id AND admin_id = @admin_id
```

### 5.13. Representative Send an Interview Request



It is the result page example for each quiz. Company representatives can see the score of developers in descending order. According to this table, representatives can send an interview request to any developer.

```
insert into sendRequest ( request_id, acceptedStatus, date, job_description)
```

```
values ( @request_id, @acceptedStatus, @date, @job_description)
```



### 5.13.1. Developer Responds to Interview Request



When developer clicks on any interview request, he or she will access to this page. They can see interview request with its description. They can accept or decline the request.

If developer accepts the request:

**update** status

**set** acceptedStatus = accepted

**where** request\_id = @request\_id and developer\_id = @developer\_id

If developer declines the request

**update** status

**set** acceptedStatus = declined

**where** request\_id = @request\_id and developer\_id = @developer\_id

## 6. Implementation Plan

In this project, MySQL is going to be used for database while PHP is going to be used for server side. To design user interface, HTML, CSS, and JavaScript are going to be used.

We shared the work as offered in the Project Functionality Document. According to the plan, each team member is responsible from one part as follows:

Meryem Efe → Common functionalities & Additional requirements

Hamza Pehlivan → Developer takes quiz.

Selen Uysal → Admin prepares questions.

Fırat Yönak → Representative sends interview request.

## 7. Website

Our website for further project information:

<https://hamzapehlivan.github.io/Database-Project/>