CS 353 TERM PROJECT

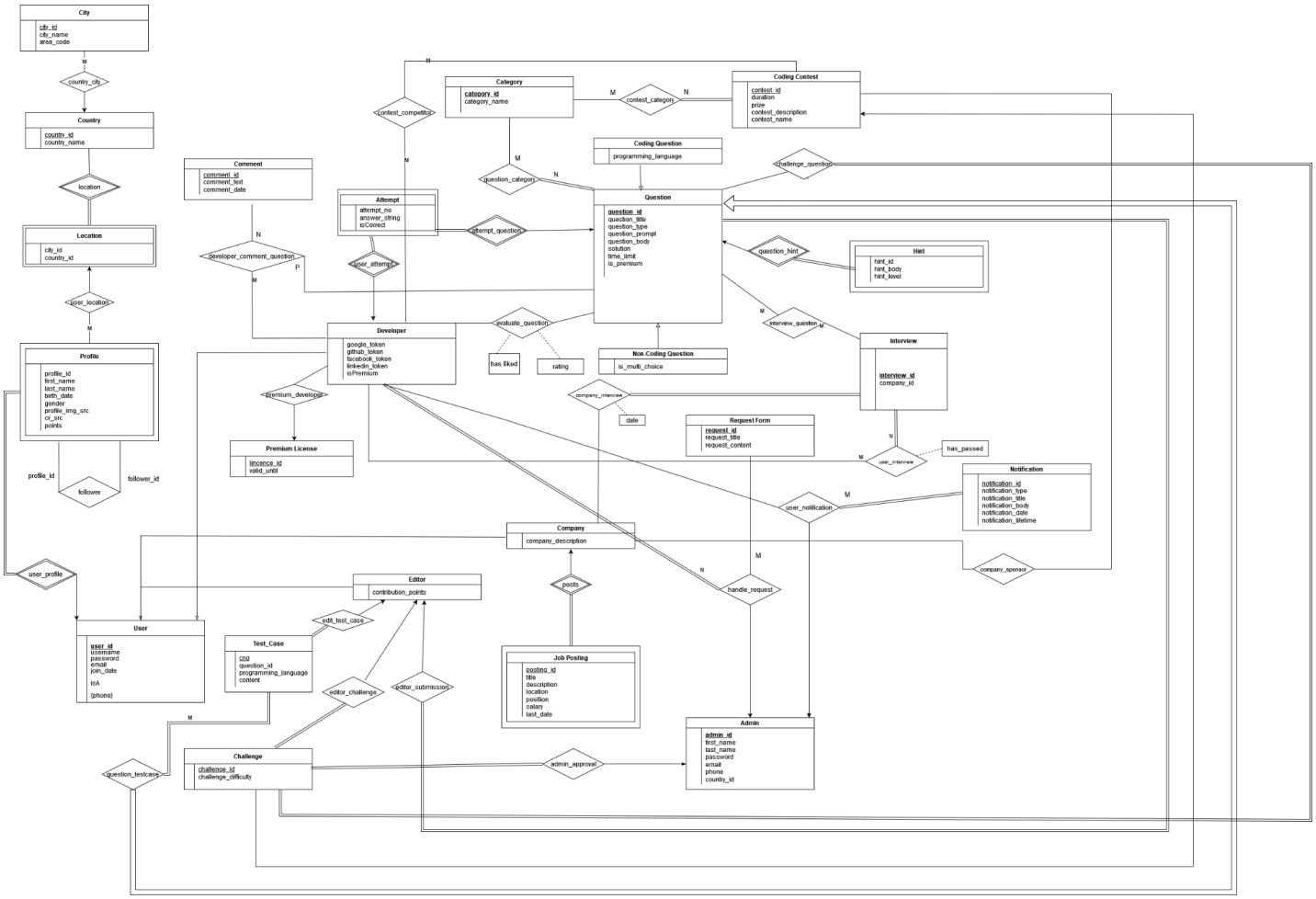
Online Coding Platform

PROJECT DESIGN REPORT

Group 11:
Ali Eren Günaltılı 21801897
Ceyda Şahin 219023447
Bora Altınok 21902206
Mustafa Anıl Taşdan 21601653

Table of Contents

1. Revised ER



The high resolution image can be found at
https://raw.githubusercontent.com/ceydas/codeatrapton/gh-pages/er.png

2. Table Schemas

Relational Model:
user (user_id, isA, username, password, email, join_date, phone)
Functional Dependencies:
user_id -> isA username password email join_date phone

Candidate Keys:
{(user_id)}

Normal Form:
3NF

Table Definition:

```
Create Table user(
user_id      int not null,
isA          varchar(20),
username     varchar(40) not null,
password     varchar(20) not null,
email        varchar(30) not null,
join_date    date not null,
phone        varchar(20) not null,
primary key(user_id)
);
```

—----------

Relational Model:
developer (user_id, isA, username, password, email, join_date, phone, google_token, github_token, facebook_token, linkedin_token)

FK: user_id references user
FK: isA references user
FK: username references user
FK: password references user
FK: email references user
FK: join_date references user
FK: phone references user

Functional Dependencies:

user_id -> isA username password email join_date phone google_token github_token facebook_token linkedin_token

Candidate Keys:
{(user_id), (username)}

Normal Form:
3NF

Table Definition:
Create Table developer**(**
user_id **int** not null,
isA **varchar**(20),
username **varchar**(40) not null,
password **varchar**(20) not null,
email **varchar**(30) not null,
join_date **date** not null,
phone **varchar**(20) not null,
google_token **varchar**(100),
github_token **varchar**(100),
facebook_token **varchar**(100),
linkedin_token **varchar**(100),
primary key(user_id)**);**
—----------

Relational Model:
category (<u>category_id</u>, category_name)
Functional Dependencies:
category_id -> category_name

Candidate Keys:
{(category_id)}

Normal Form:
3NF

Table Definition:
Create Table category**(**
category_id **int** not null,
category_name **varchar**(20),
primary key(category_id)

);
—----------

Relational Model:

question (question_id, question_title, difficulty, question_type, question_prompt, question_body, solution, time_limit, is_premium)

Functional Dependencies:

Question_id -> question_title question_type question_prompt question_body solution time_limit is_premium

Candidate Keys:

{(question_id)}

Normal Form:

3NF

Table Definition:

```
Create Table question(
question_id     int not null,
question_title varchar(50),
difficulty varchar(20),
question_type not null varchar(20),
question_prompt         varchar(200) not null,
question_body           varchar(500) not null,
solution          varchar(500) not null,
time_limit        int not null,
is_premium        boolean not null,
primary key(question_id)
);
```

—----------

Relational Model:

non_coding_question (question_id, question_title, question_prompt, question_body, solution, time_limit, is_premium, is_multi_choice)

FK: question_id references question
FK: question_title references question
FK: question_prompt references question
FK: question_body references question
FK: solution references question
FK: time_limit references question
FK: is_premium references question

Functional Dependencies:

question_id -> question_title question_propmt question_body solution time_limit is_multi_choice

Candidate Keys:

{(question_id)}

Normal Form:

3NF

Table Definition:

Create Table non_coding_question**(**
question_id **int not null,**
question_title **varchar(50),**
question_type **not null varchar(20),**
question_prompt **varchar(200)** not null,
question_body **varchar(500)** not null,
solution **varchar(500)** not null,
time_limit **int** not null,
is_premium **boolean** not null,
Is_multi_choice **boolean** not null,
primary key(question_id),
foreign key(question_title)references question,
foreign key(question_type)references question,
foreign key(question_prompt)references question,
foreign key(question_body)references question,
foreign key(solution)references question,
foreign key(time_limit)references question,
foreign key(is_premium)references question,
);

—----------

Relational Model:

coding_question (question_id, programming_language, question_title, question_prompt, question_body, solution, time_limit, is_premium)

FK: question_id references question
FK: question_title references question
FK: question_prompt references question
FK: question_body references question
FK: solution references question
FK: time_limit references question
FK: is_premium references question

Functional Dependencies:

question_id -> programming_language question_title question_prompt question_Body solution_time time_limit is_premium

Candidate Keys:

{(question_id)}

Normal Form:

3NF

Table Definition:

Create Table coding_question**(**

question_id int not null,
programming_language **varchar(20)** not null**,**
question_title **varchar(50),**
question_type **not null varchar(20),**
question_prompt **varchar(200)** not null,
question_body **varchar(500)** not null,
solution **varchar(500)** not null,
time_limit **int** not null,
is_premium **boolean** not null,
primary key(question_id),
foreign key(question_title) references question,
foreign key(question_type)references question,
foreign key(question_prompt)references question,
foreign key(question_body)references question,
foreign key(solution)references question,
foreign key(time_limit)references question,
foreign key(is_premium)references question,
);
—----------

Relational Model:
question_category (category_id, question_id)
FK: category_id references category
FK: question_id references question
Functional Dependencies:
None
Candidate Keys:
{(category_id, question_id)}

Normal Form:
3NF

Table Definition:
Create Table question_category**(**
category_id int not null**,**
question_id int not null**,**
primary key(category_id, question_id**),**
foreign key(category_id) references category,
foreign key(question_id)references question,
);
—----------
Relational Model:
coding_contest (contest_id, duration, prize, contest_description, contest_name)
Functional Dependencies:

contest_id -> duration prize contest_description contest_name

Candidate Keys:
{(contest_id)}

Normal Form:
3NF

Table Definition:
Create Table coding_contest**(**
contest_id **int not null,**
duration **int not null,**
prize **varchar(100),**
contest_description **varchar(500),**
contest_name **varchar(50),**
primary key(contest_id**)**
);
—----------
Relational Model:
contest_category (category_id, contest_id)
FK: category_id references category
FK: contest_id references contest
Functional Dependencies:
None

Candidate Keys:
{(category_id, contest_id)}

Normal Form:
3NF

Table Definition:
Create Table contest_category**(**
category_id **int** not null**,**
contest_id **int** not null**,**
primary key(category_id**,** contest_id**),**
foreign key(category_id**)** references category**,**
foreign key(contest_id**)** references contest

);
—----------
Relational Model:
challenge_question (challenge_id, question_id)

FK: challenge_id references challenge
FK: question_id references question

Functional Dependencies:
None

Candidate Keys:
{(challenge_id, question_id)}

Normal Form:

Table Definition:
Create Table contest_question**(**
challange_id **int** not null**,**
question_id **int** not null**,**
primary key(challange_id**,** question_id**),**
foreign key(challange_id**)** references challenge**,**
foreign key(question_id**)** references question
);
—----------

Relational Model:
question_hint (question_id, hint_id, hint_body, hint_level)
FK: question_id references question

Functional Dependencies:
question_id hint_id -> hint_body hint_level

Candidate Keys:
{(question_id, hint_id)}

Normal Form:
3NF

Table Definition:
Create Table question_hint**(**
question_id **int** not null,
hint_id **int** not null,
hint_body **varchar(100)** not null,
hint_level **int** not null**,**
primary key(question_id, hint_id**),**
foreign key(question_id**)** references question,
foreign key(hint_id**)** references hint
);
—----------

Relational Model:

interview (interview_id, company_id)
Functional Dependencies:
interview_id -> company_id

Candidate Keys:
{(interview_id)}

Normal Form:
3NF

Table Definition:
Create Table interview**(**
interview_id int not null,
company_id int not null,
primary key(interview_id**),**
foreign key(company_id**)** references company
);
—----------
Relational Model:
interview_question (interview_id, question_id)
FK: question_id references question
FK: interview_id references interview
Functional Dependencies:
None

Candidate Keys:
{(interview_id, question_id)}

Normal Form:
3NF

Table Definition:
Create Table interview_question**(**
interview_id **int** not null**,**
question_id **int** not null**,**
primary key(interview_id**,** question_id**),**
foreign key(interview_id**)** references interview**,**
foreign key(question_id**)** references question
);
—----------
Relational Model:
evaluate_question (user_id, question_id, has_liked, rating)
FK: user_id references user

FK: question_id references question

Functional Dependencies:

user_id question_id -> has_liked rating

Candidate Keys:

{(user_id, question_id)}

Normal Form:

3NF

Table Definition:

```
Create Table evaluate_question(
user_id        int not null,
question_id    int not null,
has_liked      boolean,
rating         int,
primary key(user_id, question_id),
foreign key(user_id) references user,
foreign key(question_id) references question

);
```

—----------

Relational Model:

comment(<u>comment_id</u>, comment_text, comment_date)

Functional Dependencies:

comment_id -> comment_text comment_date

Candidate Keys:

{(comment_id)}

Normal Form:

3NF

Table Definition:

```
Create Table comment(
comment_id    int not null,
comment_text varchar(200),
```

comment_date **Date,**
primary key(comment_id**),**
);
—----------
Relational Model:
premium_license(license_id, start, valid_until)
Functional Dependencies:
license_id > start valid_until

Candidate Keys:
{(license_id)}

Normal Form:
3NF

Table Definition:
Create Table premium_license**(**
license_id **int** not null,
start **Date** not null,
valid_until **Date** not null,
primary key(license_id)
);
—----------
Relational Model:
premium_developer(license_id, user_id)
FK: license_id references premium_license
FK: user_id references user
Functional Dependencies:
None
Candidate Keys:
{(license_id), (user_id)}

Normal Form:
3NF

Table Definition:
Create Table premium_developer**(**
license_id **int** not null**,**
user_id **int** not null**,**
primary key(license_id**),**
foreign key(license_id**)** references premium_license**,**
foreign key(user_id**)** references user**);**

—----------

Relational Model:

developer_comment_question(comment_id, user_id, question_id)

FK: comment_id references comment

FK: user_id references user

FK: question_id references question

Functional Dependencies:

None

Candidate Keys:

{(comment_id, user_id, question_id)}

Normal Form:

Table Definition:

Create Table developer_comment_question**(**

comment_id **int** not null**,**

user_id **int** not null**,**

question_id **int** not null**,**

primary key(comment_id**,** user_id, question_id**),**

foreign key(comment_id**)** references comment**,**

foreign key(user_id**)** references user,

foreign key(question_id **)** references question**);**

—----------

Relational Model:

attempt(attempt_id, answer_string)

Functional Dependencies:

attempt_id -> answer_string

Candidate Keys:

{(attempt_id)}

Normal Form:

3NF

Table Definition:

CREATE TABLE attempt(

attempt _id int NOT NULL,

primary key(attempt_id));

—----------

Relational Model:
user_attempt(attempt_id, user_id)
FK: attempt_id references attempt
FK: user_id references user

Functional Dependencies:
attempt_id -> user_id

Candidate Keys:
{(attempt_id)}

Normal Form:
3NF

Table Definition:
CREATE TABLE user_attempt(
attempt _id INT NOT NULL,
user_id INT NOT NULL
primary key(attempt_id),
foreign key(attempt_id) references attempt,
foreign key(user_id) references user,
);

—----------

Relational Model:
attempt_question(attempt_id, question_id, isCorrect)
FK: question_id references question

Functional Dependencies:
attempt_id ->question_id isCorrect

Candidate Keys:
{(attempt_id)}

Normal Form:
3NF

Table Definition:
Table Definition:

CREATE TABLE attempt_question (

attempt _id INT NOT NULL,
question_id INT NOT NULL
primary key(attempt_id),
foreign key(attempt_id) references attempt,
foreign key(question_id) references question,
);
—----------

Relational Model:
company(user_id, isA, username, password, email, join_date, phone, company_description)
FK: user_id references user
FK: isA references user
FK: username references user
FK: password references user
FK: email references user
FK: join_date references user
FK: phone references user
Functional Dependencies:
user_id -> isA username password email join_date phone company_description

Candidate Keys:
{(user_id)}

Normal Form:
3NF

Table Definition:
Create Table company**(**
user_id **int** not null,
isA **varchar**(20),
username **varchar**(40) not null,
password **varchar**(20) not null,
email **varchar**(30) not null,
join_date **date** not null,
phone **varchar**(20) not null,
company_description **varchar**(500),
primary key(user_id),
foreign key(user_id**)** references user**,**
foreign key(isA**)** references user,
foreign key(username **)** references user**),**
foreign key(password**)** references user**,**
foreign key(email**)** references user,
foreign key(join_date **)** references user,

foreign key(phone **)** references user,
);
—----------
Relational Model:
admin (admin_id, first_name, last_name, password, email, phone, country_id)
Functional Dependencies:
admin_id -> first_name last_name password email phone country_id

Candidate Keys:
{(admin_id)}

Normal Form:
3NF

Table Definition:
Create Table admin**(**
admin_id **int not null,**
first_name **varchar**(20) not null,
last_name **varchar**(20) not null,
password **varchar**(20) not null,
email **varchar**(50) not null,
phone **varchar**(20) not null,
country_id **int** not null,
primary key(admin_id)
);
—----------
Relational Model:
profile (profile_id, user_id, first_name, last_name, birth_date, gender, profile_img_src, cv_src, points)
Functional Dependencies:
profile_id user_id -> first_name last_name birth_date gender profile_img_src cv_src points

Candidate Keys:
{(profile_id, user_id)}

Normal Form:
3NF

Table Definition:
Create Table profile**(**
profile_id **int not null,**
user_id **int not null,**

first_name **varchar**(20) not null,
last_name **varchar**(20) not null,
birth_date **Date not null,**
gender **varchar(**15**),**
profile_img_src **varchar(**50**),**
cv_src **varchar(**50**),**
points **int,**
primary key(profile_id**,** user_id**),**
foreign key(user_id**)** references user

);
—----------

Relational Model:
challenge (challenge_id, challenge_difficulty)
Functional Dependencies:
chellange_id -> challenge_difficulty

Candidate Keys:
{(chellange_id)}

Normal Form:
3NF

Table Definition:
Create Table challenge**(**
challenge_id **int** not null,
challenge_difficulty **varchar**(10),
primary key(challenge_id)
);
—----------
Relational Model:
test_case (cno, question_id, programming_language, content)
FK: question_id references question
Functional Dependencies:
cno question_id -> programming_language content

Candidate Keys:
{(cno, question_id)}

Normal Form:
3NF

Table Definition:

```
Create Table test_case(
cno     int not null,
question_id   int not null,
programming_language        varchar(20),
content          varchar(100),
primary key(cno, question_id),
foreign key(question_id) references question
);
```

—----------

Relational Model:

editor_challange (challenge_id,user_id)

FK: challenge id references challenge

FK: user_id references user

Functional Dependencies:

None

Candidate Keys:

{(challange_id, user_id)}

Normal Form:

3NF

Table Definition:

```
Create Table editor_challange(
challange_id   int not null,
user_id          int not null,
primary key(challange_id, user_id),
foreign key(challange_id) references challenge,
foreign key(user_id) references user);
```

—----------

Relational Model:

job_posting (posting_id, user_id, title, description, location, position, salary, last_date)

FK: user_id references user

Functional Dependencies:

posting_id user_id -> title description location position salary last_date

Candidate Keys:

{(posting_id, user_id)}

Normal Form:
3NF

Table Definition:
Create Table job_posting**(**
posting_id int not null,
user_id int not null,
title **varchar(50),**
description **varchar(200)** not null,
location **varchar(100)** not null,
position **varchar(50)** not null,
salary **int not null,**
last_date **Date not null,**
primary key(posting_id, user_id**),**
foreign key(user_id**)** references user

);
—----------
Relational Model:
edit_test_case (user_id, cno)
FK: user_id references user
FK: cno references test_case
Functional Dependencies:
None

Candidate Keys:
{(user_id, cno)}

Normal Form:
3NF

Table Definition:
Create Table edit_test_case**(**
challange_id **int** not null**,**
user_id **int** not null**,**
primary key(user_id**,** cno**),**
foreign key(user_id**)** references user**,**
foreign key(cno**)** references test_case**);**
—----------
Relational Model:
request_form (request_id, request_title, request_content)
Functional Dependencies:
request_id -> request_title request_content

Candidate Keys:
{(request_id)}

Normal Form:
3NF

Table Definition:
Create Table request_form**(**
request_id **int** not null,
request_title **varchar(**50**),**
request_content **varchar(**200**),**
primary_key(request_id**)**
);
—----------
Relational Model:
handle_request (admin_id, request_id, user_id)
FK: admin_id references admin
FK: request_id references request_form
FK: user_id references user
Functional Dependencies:
None

Candidate Keys:
{(admin_id, request_id, user_id)}

Normal Form:
3NF

Table Definition:
Create Table handle_request**(**
admin_id **int** not null**,**
request_id **int** not null**,**
user_id **int** not null**,**
primary key(admin_id**,** request_id, user_id**),**
foreign key(admin_id**)** references admin**,**
foreign key(request_id**)** references request_form,
foreign key(user_id **)** references user**);**
—----------
Relational Model:
notification (notification_id, notification_type, notification_title, notification_body, notification_date, notification_lifetime)

Functional Dependencies:
notification_id -> notification_type notification_title notification_body notification_date notification_lifetime

Candidate Keys:
{(notification_id)}

Normal Form:
3NF

Table Definition:
Create Table notification**(**
notification_id **int** not null,
notification_type **varchar(**20**)** not null**,**
notification_title **varchar(**50**)** not null**,**
notification_body **varchar(**200**)** not null**,**
notification_date **Date** not null**,**
notification_lifetime **varchar(**200**)** not null**,**
primary_key(request_id**));**
—----------
Relational Model:
user_notification (notification_id, admin_id, user_id)
FK: notification_id references notification
FK: admin_id references admin
FK: user_id references user
Functional Dependencies:
None

Candidate Keys:
{(notification_id, admin_id , user_id)}

Normal Form:
3NF

Table Definition:
Create Table user_notification**(**
notification_id **int** not null**,**
admin_id **int** not null**,**
user_id **int** not null**,**
primary key(notification_id**,** admin_id, user_id**),**
foreign key(notification_id**)** references notification**,**
foreign key(admin_id**)** references admin,

foreign key(user_id **)** references user**);**

—----------

Relational Model:

company_sponsor (user_id, contest_id)

FK: user_id references user

FK: contest _id references coding_contest

Functional Dependencies:

None

Candidate Keys:

{(user_id, contest_id)}

Normal Form:

3NF

Table Definition:

Create Table company_sponsor**(**

user_id **int** not null**,**

user_id **int** not null**,**

primary key(user_id**,** contest_id**),**

foreign key(user_id**)** references user**,**

foreign key(contest_id**)** references coding_contest,

—----------

Relational Model:

hint (question_id, hint_id, hint_body, hint_level)

FK: question_id references question

Functional Dependencies:

question_id hint_id -> hint_body hint_level

Candidate Keys:

{(question_id, hint_id)}

Normal Form:

3NF

Table Definition:

Create Table hint**(**

question_id **int** not null,

hint_id **int** not null,

hint_body **varchar**(200) not null,

hint_level **int** not null,

primary key(question_id),

foreign key(question_id) references question,
foreign key(hint_id)references hint
);
—----------
Relational Model:
user_interview (user_id, interview_id, net_score, result)
FK: user_id references user
FK: interview_id references interview
Functional Dependencies:
user_id interview_id -> net_score result

Candidate Keys:
{(user_id, interview_id)}

Normal Form:
3NF

Table Definition:
Create Table user_interview**(**
user_id **int** not null,
interview_id **int** not null,
net_score **int** not null,
result **int** not null,
primary key(user_id, interview_id),
foreign key(user_id) references user,
foreign key(interview_id)references interview**);**
—----------
Relational Model:
country (country_id, country_name)
Functional Dependencies:
country_id -> country_name

Candidate Keys:
{(country_id)}

Normal Form:
3NF

Table Definition:
Create Table country**(**
country_id **int not null,**
country_name **varchar(20),**

primary key(country_id**)**
);
—----------

city (city_id, city_name, area_code)
Functional Dependencies:
City_id -> city_name area_code

Candidate Keys:

Normal Form:
3NF

Table Definition:
Create Table city**();**
—----------
Relational Model:
country_city (city_id, country_id)
FK: city_id references city
FK: country_id references country
Functional Dependencies:
city_id -> country_id

Candidate Keys:
{(city_id)}

Normal Form:
3NF

Table Definition:
Create Table country_city**(**
city_id **int** not null,
country_id **int** not null,
primary key(city_id),
foreign key(city_id)references city,
foreign key(coutnry_id) references country
);
—----------
Relational Model:
location (user_id, country_id, city_id)
FK: user_id references user
FK: country_id references country
FK: city_id references city

Functional Dependencies:
user_id -> country_id city_id

Candidate Keys:
{(user_id)}

Normal Form:
3NF

Table Definition:
Create Table location**(**
user_id **int** not null**,**
country_id **int** not null**,**
city_id **int** not null**,**
primary key(user_id**),**
foreign key(user_id**)** references user,
foreign key(country_id**)** references country,
foreign key(city_id**)** references city**,**

);
—----------

Relational Model:
contest_competitor (user_id, contest_id, prize_collected)
FK: user_id references user
FK: contest_id references contest

Functional Dependencies:
user_id contest id-> prize_collected

Candidate Keys:
{(user_id, contest_id)}

Normal Form:
3NF

Table Definition:
Create Table contest_competitor **(**
user_id **int** not null**,**
contest**int** not null**,**

prize_collected **int,**
primary key(user_id,contest_id**),**
foreign key(user_id**)** references user,
foreign key(contest_id**)** references contest);
—----------

Relational Model:
company_sponsor (user_id, contest_id)
FK: user_id references company
FK: contest_id references contest

Functional Dependencies:
None

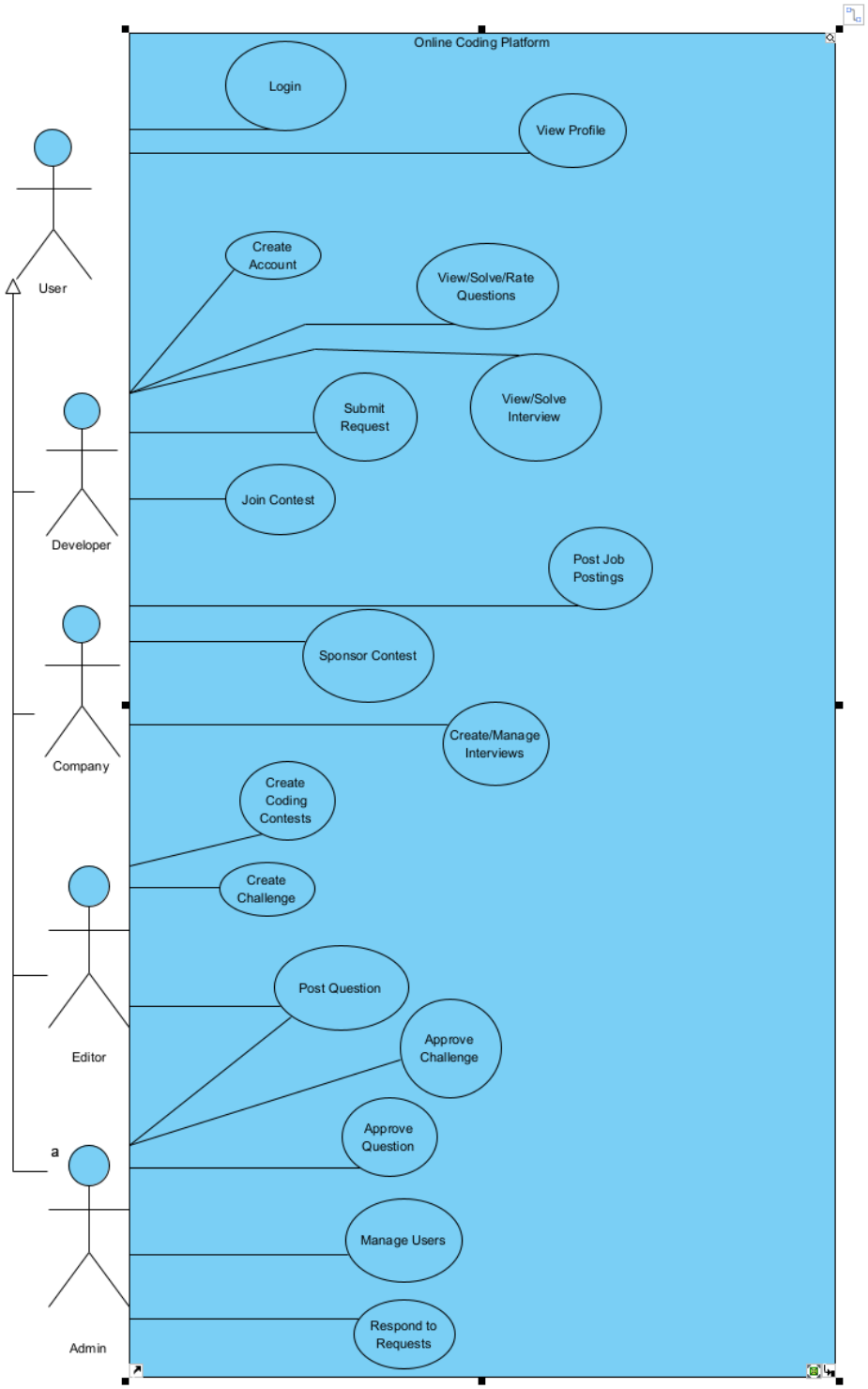
Candidate Keys:
{(user_id, contest_id)}

Normal Form:
3NF

Table Definition:
Create Table company_sponsor **(**
user_id **int** not null**,**
contest**int** not null**,**
primary key(user_id,contest_id**),**
foreign key(user_id**)** references user,
foreign key(contest_id**)** references contest);
—----------

3. Use Case Scenarios

3.1 Use-Case Model



3.2 Use-Case Scenarios

Use case table Login

Participating Actors:	User
Stakeholders:	Users want to use the system.
Flow of events:	1.User enters their credentials to the system.
Pre-conditions	Users must have an account.
Post-conditions	Users are redirected to their dashboard.
Exit conditions:	User does not have an account on the system or they enter their credentials incorrectly.

Use case table View Profile

Participating Actors:	User
Stakeholders:	User wants to see their profile data.
Flow of events:	1. At any time, the user clicks on their profile page.
Pre-conditions	Users must be logged in.
Post-conditions	Profile data and profile alteration methods are displayed to the user.
Exit conditions:	None

Use case table Create Account

Participating Actors:	Developer
Stakeholders:	Developer wants to be enrolled into the system.
Flow of events:	1. Developer clicks on “register” in the login screen. 2. Developers enter their personal data.
Pre-conditions	None

Post-conditions	Developer is redirected to their dashboard.
Exit conditions:	Developer enters invalid data to register forms.

Use case table View/Solve/Rate Questions

Participating Actors:	Developer
Stakeholders:	Developer wants to interact with the questions system.
Flow of events:	1. Developer goes to the questions screen to view available coding or non-coding questions.
Pre-conditions	Developer must be logged in.
Post-conditions	Developer is redirected to their selected question's screen.
Exit conditions:	Developer quits the question screen without taking action.
Alternative Scenarios:	Developers can post their answers to the question. They will be shown prompts depending on the validity of their answer. Developers can rate the question for other users.

Use case table View/Solve Interview

Participating Actors:	Developer
Stakeholders:	Developer wants to join an interview.
Flow of events:	1. Developer clicks on the "interviews" button to see their available interview tests. 2. Developers can solve the questions on the test and submit them to the interviewing company.
Pre-conditions	Developers must be logged in and have an active interview request.
Post-conditions	Developer is shown a prompt and redirected to their dashboard.

Exit conditions:	Developer quits the interview without completing it.

Use case table Submit Request

Participating Actors:	Developer
Stakeholders:	Developer wants to submit a request to site admins.
Flow of events:	1. Developer fills out a request form to be sent to the admin.
Pre-conditions	Developer must be logged in.
Post-conditions	Developer is shown a prompt and redirected to their dashboard.
Exit conditions:	Developer quits the request screen without taking any action.

Use case table Join Contest

Participating Actors:	Developer
Stakeholders:	Developer wants to join a coding contest..
Flow of events:	1. Developers join the contest. 2. Developer participates in the contest and does the coding and non-coding questions.
Pre-conditions	Developer must be logged in. Developers must join the contest by clicking a related button.
Post-conditions	The Developer's score is saved into the database.
Exit conditions:	Developer quit the contest.

Use case table Post Job Postings

Participating Actors:	Company
Stakeholders:	Company decides to post a Job Posting
Flow of events:	1.Company fills the “job posting form” and submits. 2.Submitted form is saved into the database.
Pre-conditions	Companies must login to their accounts.
Post-conditions	Companies publish their job postings and students can see it.
Exit conditions:	Company does not have an account on the system or they enter their credentials incorrectly.

Use case table Sponsor Contest

Participating Actors:	Company
Stakeholders:	Company decides to sponsor a contest.
Flow of events:	1.Editor creates a contest. 2.Company sponsors the contest.
Pre-conditions	Editor must login and create a contest. Companies must login and sponsor the contest.
Post-conditions	Contest created by the editor is sponsored by the company from now on.
Exit conditions:	There is no contest currently.

Use case table Create Interviews

Participating Actors:	Company
Stakeholders:	Company decides to create an interview for the students.
Flow of events:	1.Companies login and create a form for publishing their interviews. 2.Companies prepare their coding and

	non-coding questions. 3.Companies submit their interview. 4.Students can see the interview.
Pre-conditions	Company must login. Companies must prepare questions.
Post-conditions	Interview is posted on the website.
Exit conditions:	No question is submitted.

Use case table Manage Interviews

Participating Actors:	Company
Stakeholders:	Companies manage the interviews done by the students.
Flow of events:	Students do the interview questions posted by companies. Companies manage the interviews and evaluate the participating interviews.
Pre-conditions	Companies published the interviews. Students participate in the interviews.
Post-conditions	Participated students' interviews are evaluated.
Exit conditions:	Student quits the interview in the middle.

Use case table Create Coding Contests

Participating Actors:	Editor

Stakeholders:	Editor decides to create a coding contest consisting of coding and non-coding questions for students to participate.
Flow of events:	Editor prepares the non-coding questions and coding questions for the contest. Editor submits the contest. With admin's approval, the contest can be seen by the developers and developers can participate in the contest.
Pre-conditions	Admin must accept and approve the editor's submitted contest's challenges.
Post-conditions	The contest can be seen by the developers and developers can participate in the contest.
Exit conditions:	Admin rejects the contest's challenges.

Use case table Create Challenge

Participating Actors:	Editor
Stakeholders:	Editor decides to create a coding contest consisting of coding and non-coding questions for students to participate.
Flow of events:	Editor prepares the non-coding questions and coding questions as challenges. Editor submits the challenges.. With admin's approval, the challenge can be seen by the developers and developers can try to solve the challenge.
Pre-conditions	Admin must accept and approve the editor's submitted challenge.
Post-conditions	The challenge can be seen by the developers and developers can participate in the contest.
Exit conditions:	Admin rejects the challenge.

,

Use case table Post Question

Participating Actors:	Editor, Admin
Stakeholders:	Editor or Admin decides to post a particular question for developers to solve.
Flow of events:	Editor or Admin creates the coding or non-coding question by filling the appropriate fields of the question. Admin's submission can be directly seen by the developers however Editor's submission must be approved by the admin before it can be seen by the developers.
Pre-conditions	Admin must accept and approve the editor's submitted question.
Post-conditions	The challenge can be seen by the developers and developers can participate in the question.
Exit conditions:	Admin rejects the editor's question.

Use case table Approve Question

Participating Actors:	Admin
Stakeholders:	Admin decides to accept or reject the question submitted by the editor.
Flow of events:	Editor submits a question. Admin checks the question. Rejects the question or accepts the question.
Pre-conditions	Editor should have submitted a question.
Post-conditions	If rejected, developers can not see the question submitted by the editor. If accepted, developers can see the question and solve it.
Exit conditions:	Admin rejects the editor's question.

Use case table Manage Users

Participating Actors:	Admin
Stakeholders:	Admin decides to give accounts to users.
Flow of events:	Admin prepares username and password for the user.
Pre-conditions	Admin must prepare username and password.
Post-conditions	Users or editors now can login with the credentials provided by the admin.
Exit conditions:	Admin did not prepare username and password.

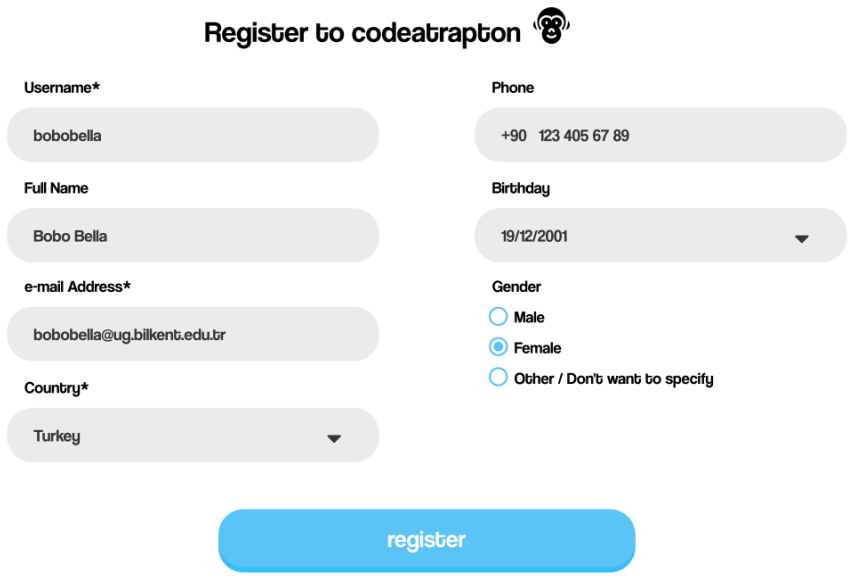
Use case table Respond to Requests

Participating Actors:	Admin
Stakeholders:	Admin decides to respond to the messages or request done by the developers or editors.
Flow of events:	Editor or developers request a functionality or
Pre-conditions	Admin must prepare username and password.
Post-conditions	Users or editors now can login with the credentials provided by the admin.
Exit conditions:	Admin did not prepare username and password.

4. User Interface Design and SQL Statements

4.1 Signup Page

4.1.1. Signup Page for Developers



Signup:

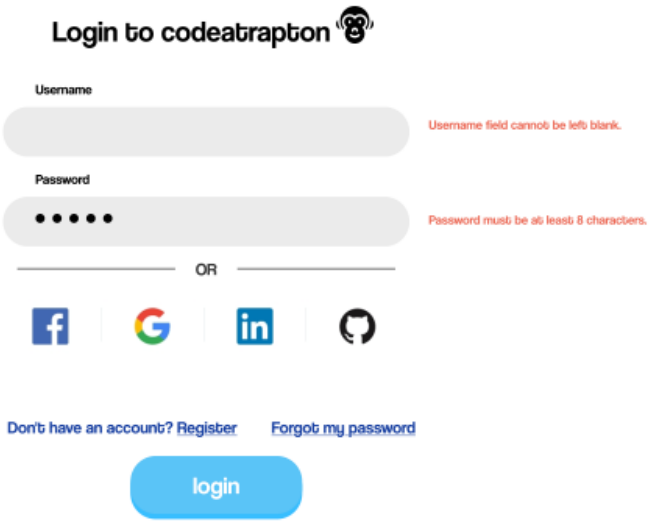
INSERT INTO user(next_available_p_id, developer, username,password,email,current_date,phone)

INSERT INTO developer(next_available_p_id, developer, username, password, email, current_date, phone, NULL,NULL,NULL,NULL)

INSERT INTO profile (next_available_p_id, id, first_name, last_name, birth_date, gender, NULL, NULL, 0)

4.2 Login Page

4.1.1 Login Page for Developer

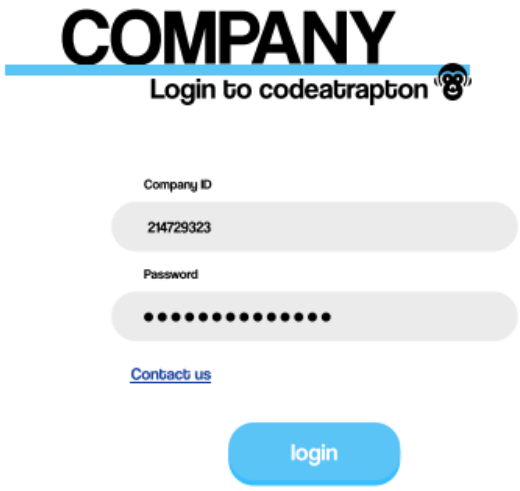


Login as user:
SELECT * from user
WHERE username = @username AND password =@password AND isA = ‘developer’

4.1.2 Login Page for Editor

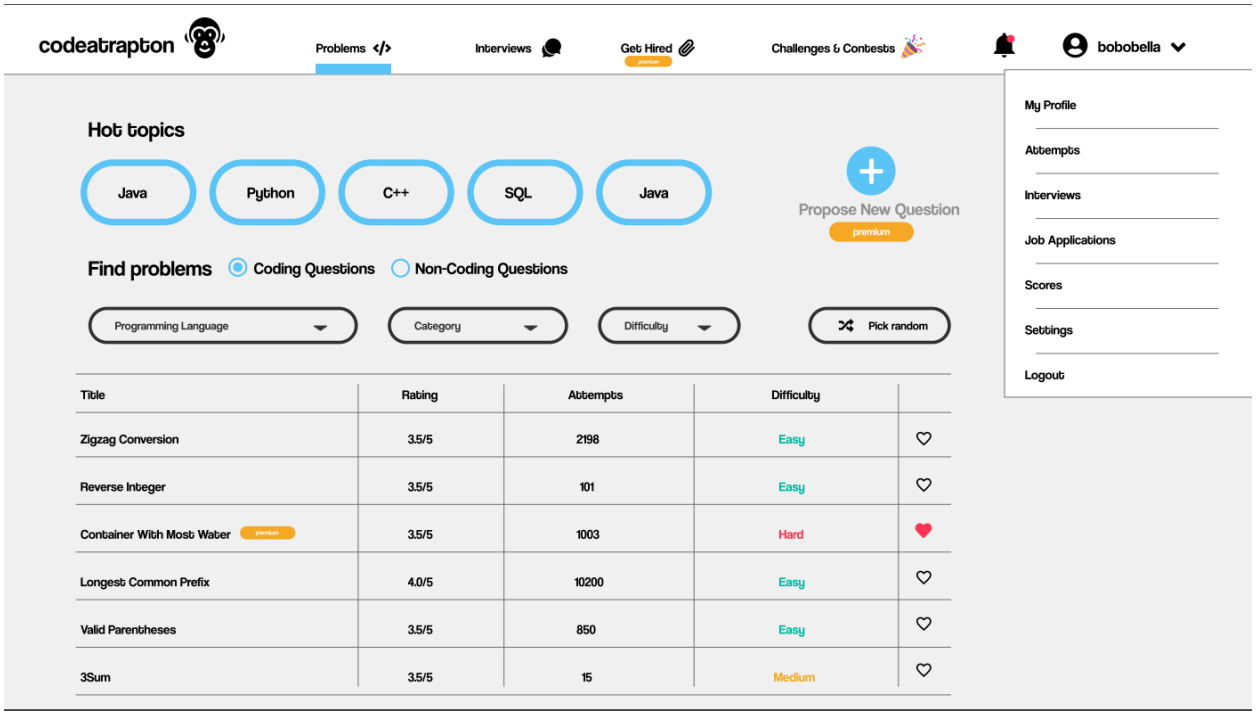
SELECT * from user
WHERE username = @username AND password =@user_password AND isA = ‘editor’

4.1.3 Login Page for Company



SELECT * from user
WHERE username = @username AND password =@user_password AND isA = ‘company’

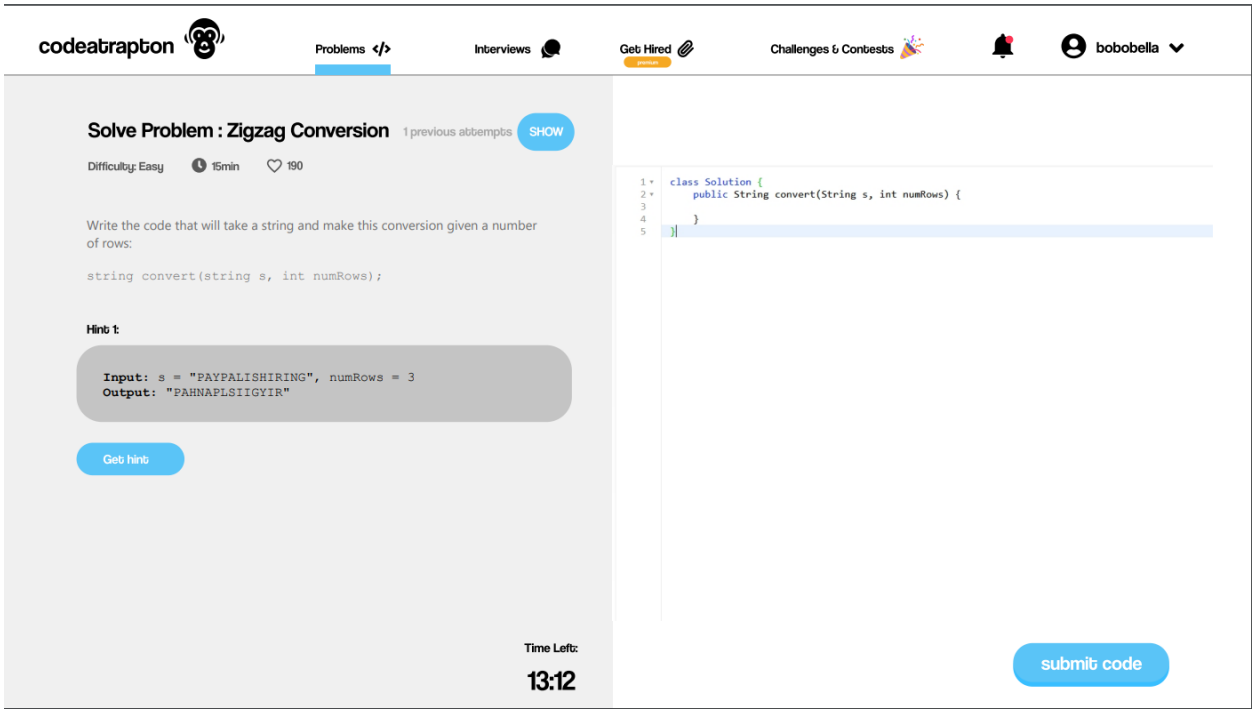
4.3 Problems Page (seen by Developer)



List Problem Table:

```
SELECT difficulty, question_title, is_premium
FROM question
WHERE question_type = 'coding'

SELECT difficulty, question_title, is_premium
FROM question
WHERE question_type = 'non_coding'

SELECT hasLiked
FROM evaluate_question
WHERE user_id = @uid

SELECT * FROM question_rating;

SELECT * FROM AttemptCount;
```

4.4 Attempt Page (seen by Developer)



Get Problem Data:

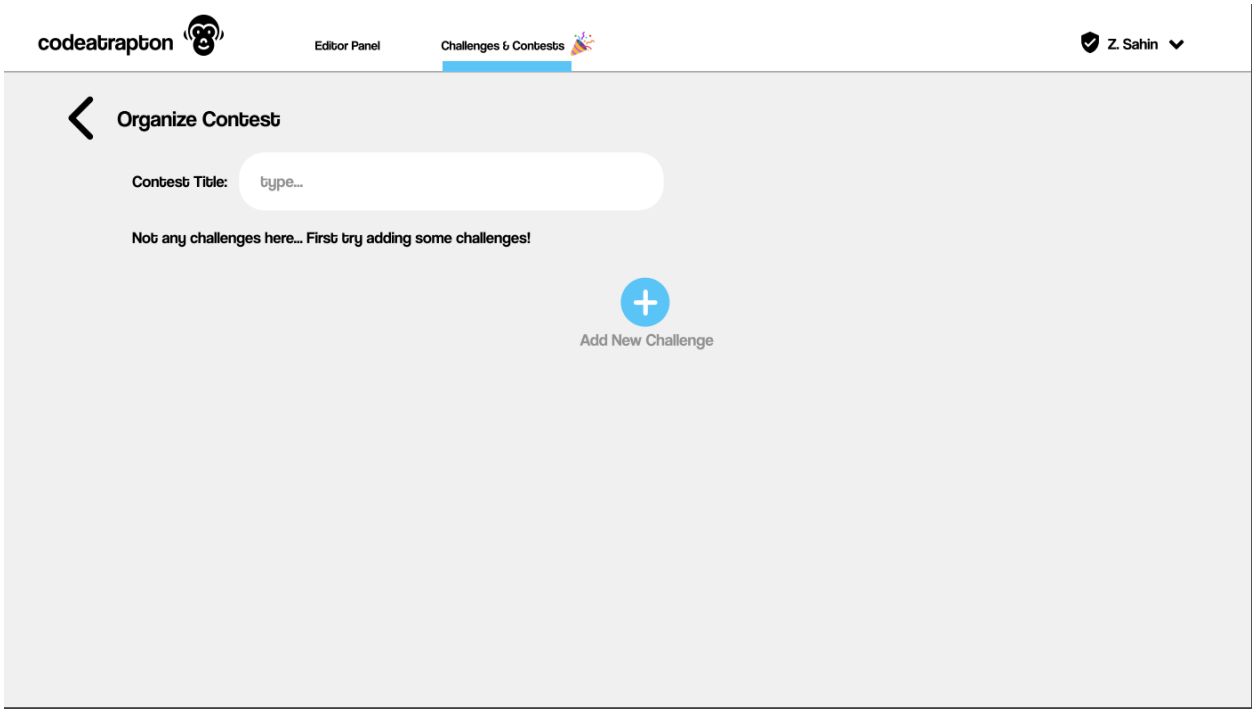
SELECT question_title,programming_language,question_prompt,question_body,solution,time_limit)
FROM coding_question cq
WHERE cq.question_id = selected_cq_id

Get Hint Data:
SELECT hint_body,hint_level
FROM hint h
WHERE h.question_id = selected_cq_id

Submit Attempt:
INSERT INTO attempt VALUES(@next_attempt_no,@current_user_id,@answer_string,@isCorrect)

Get Like Count:
SELECT * FROM question_rating
WHERE question_id = @qid

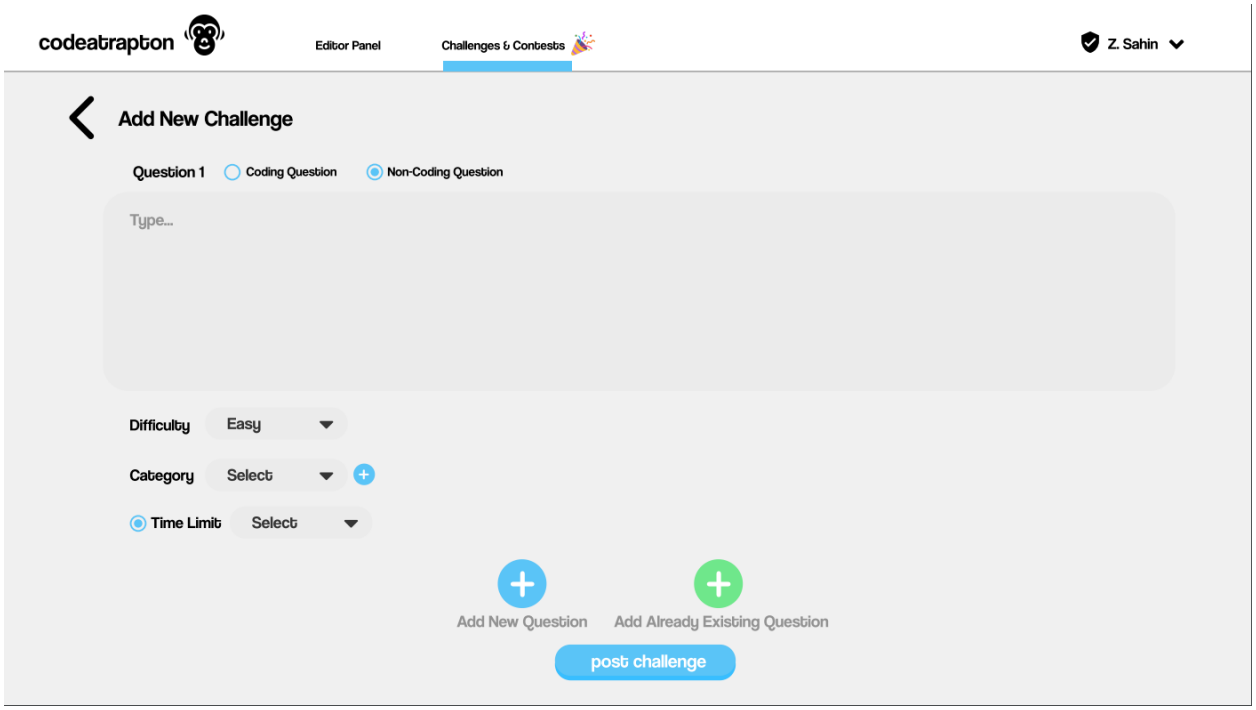
4.5 Create Contest Page (seen by Editor)



Submit Contest:

```
INSERT INTO coding_contest(next_contest_id,prize,contest_description_contest_name)
INSERT INTO contest_category(next_category_id,next_contest_id)
```

4.6 Create Challenge Page (seen by Editor)



Create a question for a contest:

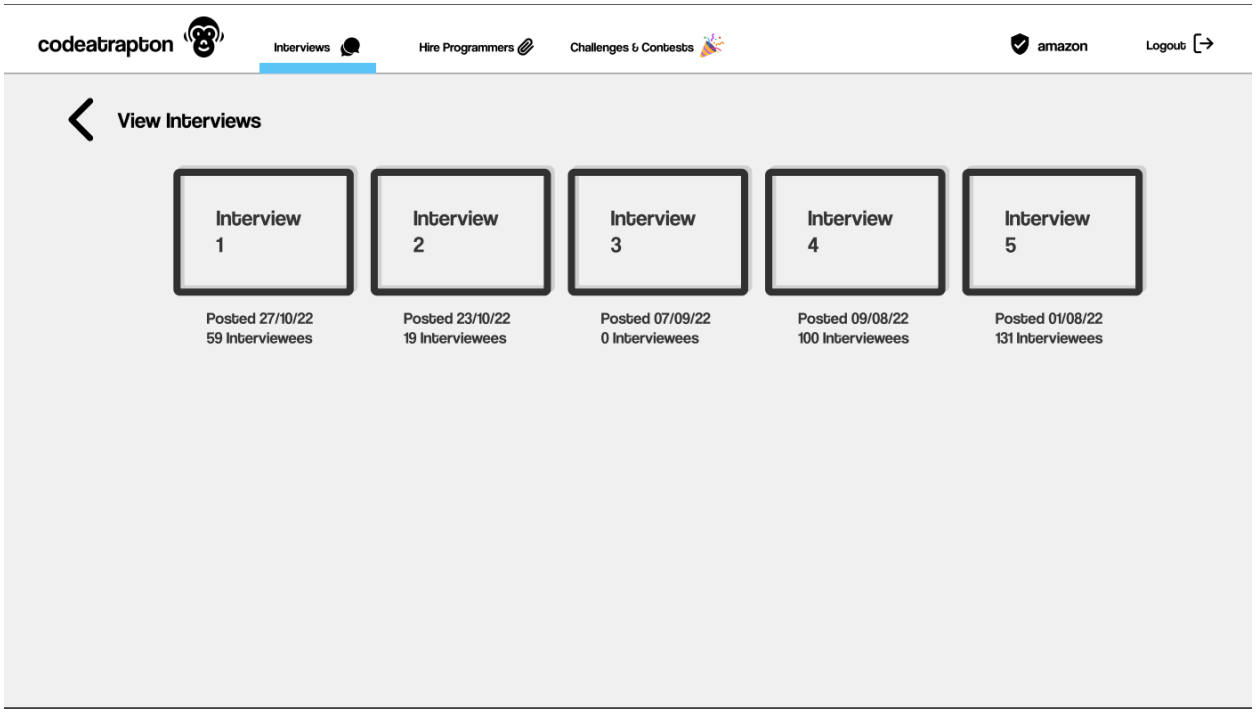
INSERT INTO coding_question VALUES(next_question_id,programming_language,question_title,question_prompt,question_body,time_limit,is_premium)

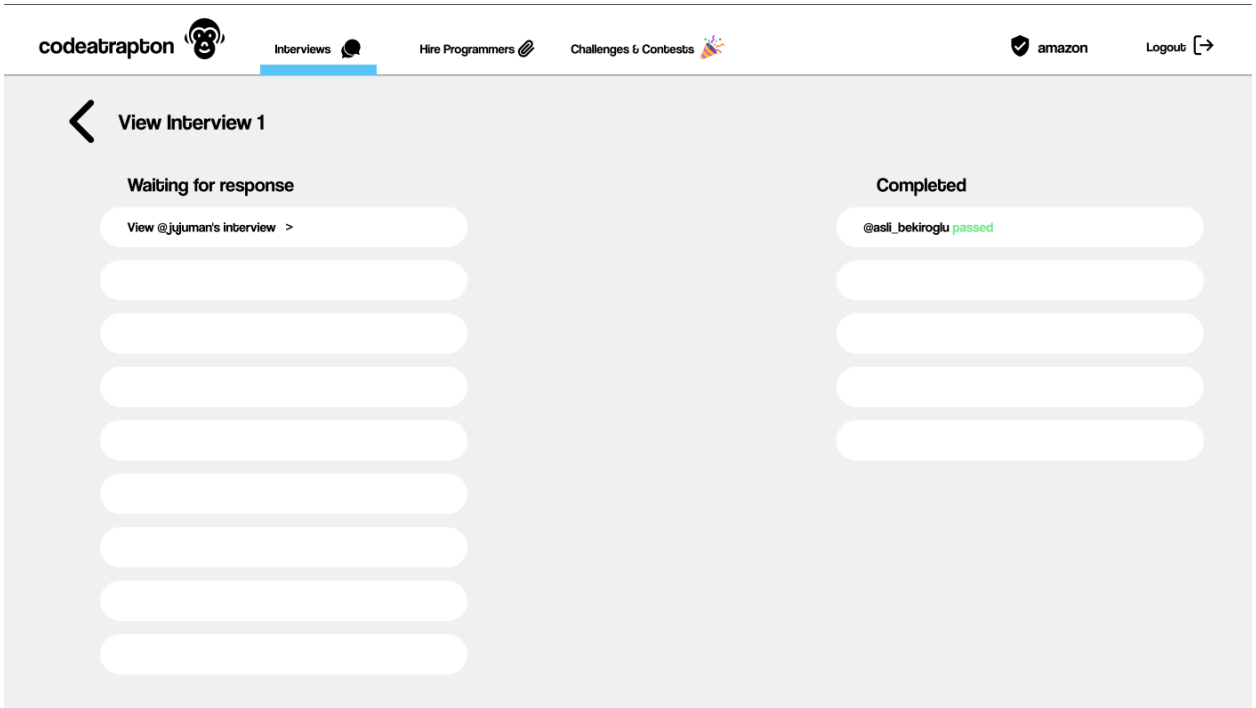
-OR-

INSERT INTO non_coding_question VALUES(next_question_id,question_title,question_prompt,question_body,solution,time_limit,is_premium,is_multi_choice)

INSERT INTO challenge_question (selected_challenge_id,next_question_id)
INSERT INTO challenge (selected_challenge_id,challenge_difficulty)

4.7 Interview Pages (seen by Company)





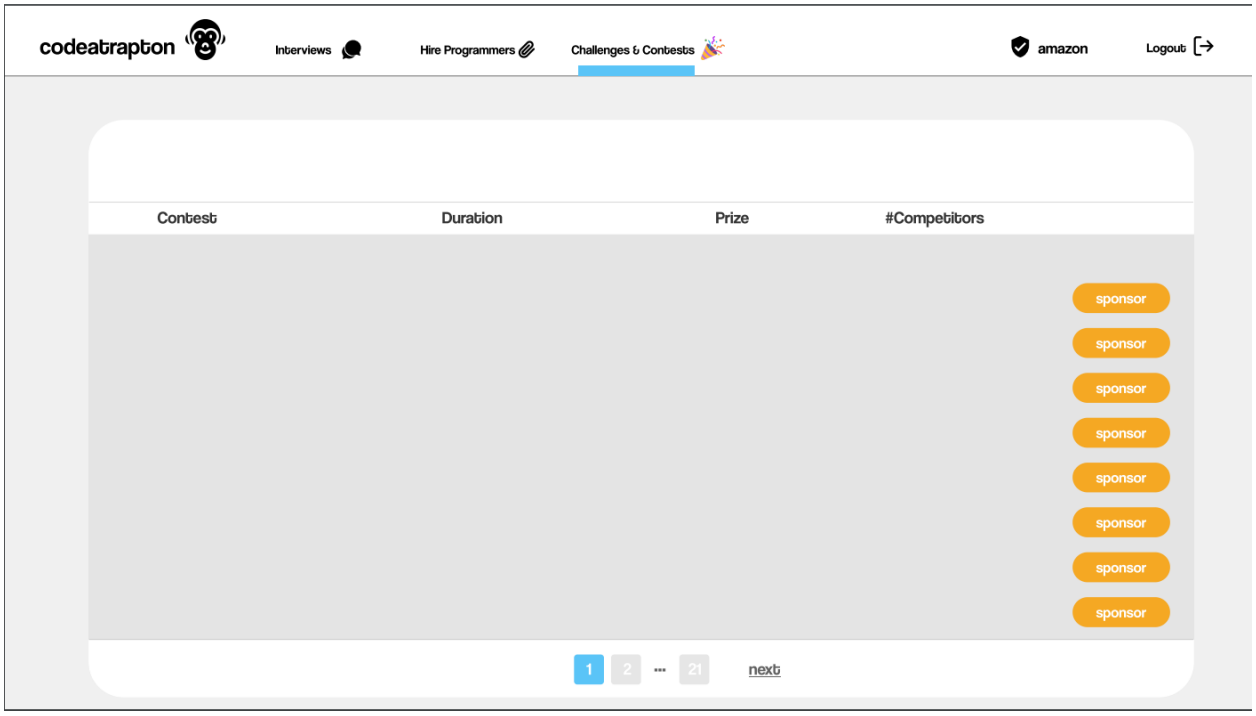
Get General Interview Data
interview (interview_id, company_id)

```
SELECT interview_id
FROM interview i
WHERE i.company_id = current_company_id
```

Get Specific Interview Data

```
SELECT username,net_score,result

FROM user u INNER JOIN user_interview ui ON u.user_id = ui.user_id
WHERE interview_id = selected_interview_id
```

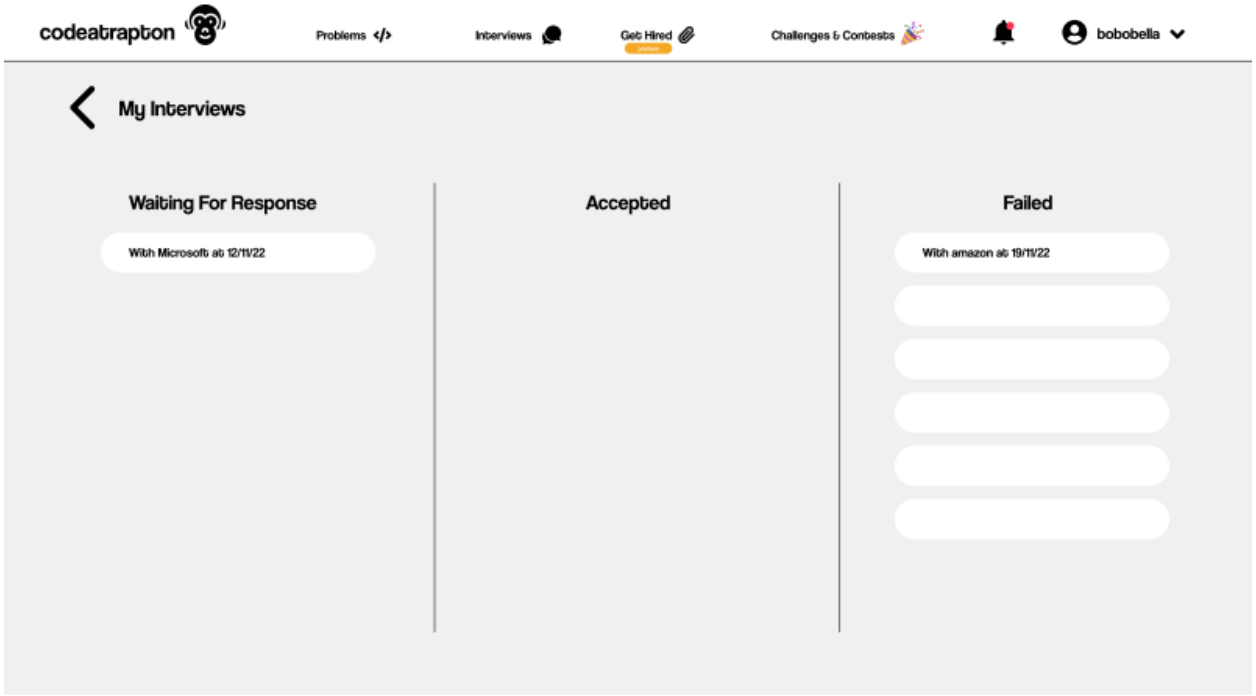
4.8 Sponsor Contest Page (seen by Company)



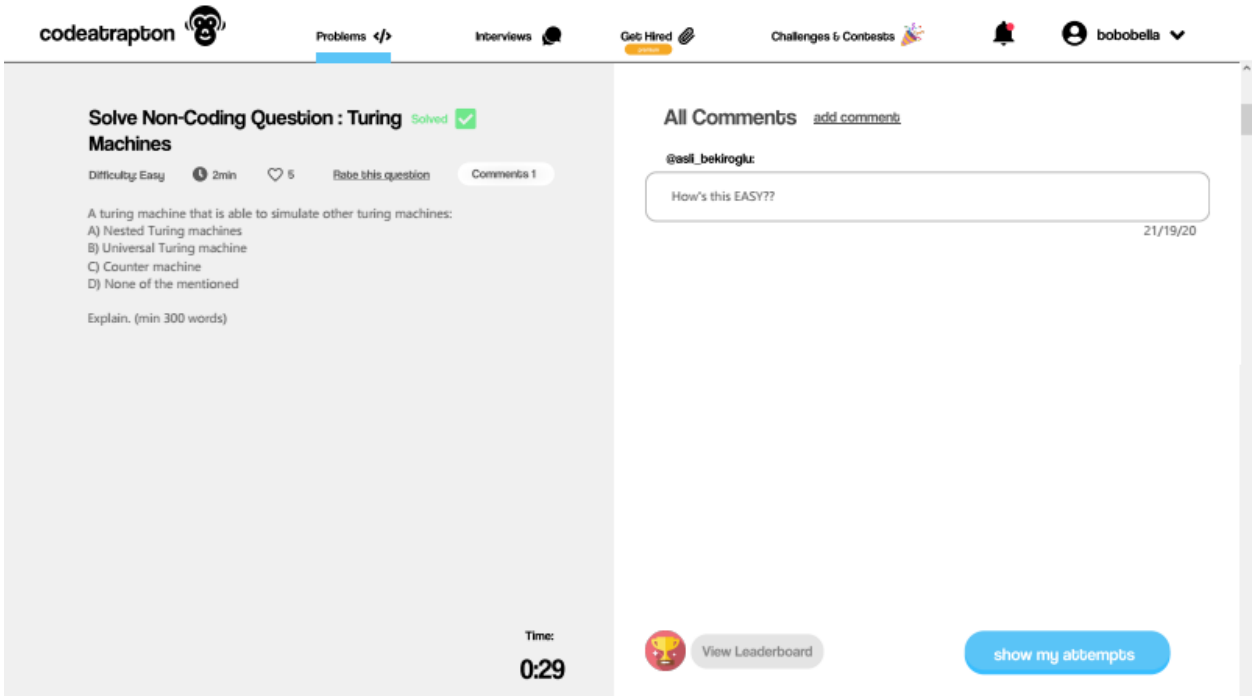
Sponsor Contest:

```
INSERT INTO company_contest VALUES(current_company_id,selected_contest_id)
```

4.9 Interview Page (seen by Developer)



4.10. Comments Page /seen by Developer)



SELECT question_comment

5. Advanced Database Components

5.1. Reports

5.1.1 Hot (Most attempted) Problems of All Time

```
SELECT question_id, count(*) as cnt
FROM user_attempt NATURAL JOIN attempt_question
GROUP BY question_id
ORDER BY cnt DESC
LIMIT 10;
```

5.1.2 Number of Interviews for Each Company

```
SELECT username,count(*) as InterviewCount
FROM company c JOIN company_interview ci ON c.user_id = ci.company_id
GROUP BY username
ORDER BY InterviewCount DESC
```

5.1.3 Number of Attempts on a Question

```
SELECT username, count(*) as AttemptCount
FROM developer d JOIN attempt a ON d.user_id = a.user_id
GROUP BY username
ORDER BY AttemptCount DESC
```

5.1.4 Average Rating for Questions

```
SELECT avg(rating) as AvgRate
FROM evaluate_question
```

5.1.5. Number of Premium Members

```
SELECT count (user_id)
FROM user
WHERE is_premium = ‘1’
```

5.1.6. Longest Period of Premium Membership

```
SELECT pd.user_id, max (date_diff)
FROM (
SELECT DATEDIFF(day, pl.start, pl.valid_until) AS date_diff
FROM premium_developer pd JOIN premium_license pl ON pd.license_id =
pl.license_id) as dd;
```

5.2. Views

5.2.1. Question Likes View

```
CREATE VIEW question_likes AS
    (SELECT count(hasLiked) FROM evaluate_question WHERE hasLiked = '1')
```

5.2.2. Leaderboard View

```
CREATE VIEW leaderboard AS
    (SELECT u.user_id, u.username, max(prize_collected) as maxp
    FROM contest_competitor c JOIN user u ON u.user_id = c.user_id
    ORDER BY maxp DESC
     );
```

5.2.3 Rating for A Question

```
CREATE VIEW question_rating AS
    (SELECT question_id, avg(rating)
    FROM evaluate_question
    WHERE qid = @qid
    GROUP BY qid
    ORDER BY avg_rate DESC);
```

5.2.4 Comments For Question

```
CREATE VIEW question_comments AS
    (SELECT user, avg(rating)
    FROM developer_comment_question
    WHERE qid = @qid
    GROUP BY qid
```

```
ORDER BY avg_rate DESC);
```

5.2.5. Successful Interviewees View

```
create view Successful_Interviewees
select *
from user_interview natural join Developer
where has_passed = true
```

5.2.6. Solved Premium Questions

```
Create view solved_prem_questions
select *
from Attempt natural join Question
where isCorrect = true
```

5.3. Triggers

- Forgot My Password Trigger

- Update Age Trigger
  When a developer registers specifying a birth year, their age on their profile is automatically updated.

- Complete Coding Contest Trigger
  When a developer completes a contest or a challenge within a contest, they should receive the prize points immediately.

- Leaderboard Update
  When a contest is over, the leader will be updated in the leaderboard by considering points taken in the contest.

5.4. Constraints

The system cannot be used without an account.

The system requires at least a username, a password and a valid email address to be enrolled as a developer.

The password has to contain at least one letter and one special character.

Only validated company accounts can post job opportunities.

Only validated company accounts can conduct interviews.

Developer cannot re-enter an interview after leaving it.

Developer cannot submit another solution to the already solved question.

5.5. Stored Procedures

5.1.1 Login and Signup

```
CREATE PROCEDURE login
AS
SELECT *
FROM user
WHERE username = @Username AND password = @Password

CREATE PROCEDURE signup
AS
INSERT INTO user(next_available_p_id, developer,
@Username,@password,@email,current_date,@phone)

INSERT INTO developer(next_available_p_id, developer, @username,
@password, @email, current_date,@phone, NULL,NULL,NULL,NULL)

INSERT INTO profile ( next_available_p_id, id, @first_name,@ last_name,
@birth_date, @gender, NULL, NULL, 0)
```

- Constantly reused processes such as login, signup, attempt submission, question creation, rating a question, contest creation, interview creation and data retrieval for such processes will be stored processes, so we will not have to rewrite queries during development.

6. Technologies Used in Implementation

We plan to use HTML/CSS/Javascript for the frontend, PHP for the backend and MySQL for the database of our project implementation.