Bilkent University

Department of Computer Engineering

CS353 Term Project Design Report
CodeInt

Assigned TA

Mustafa Can Çavdar

Group 12

Team Members

Barış Can

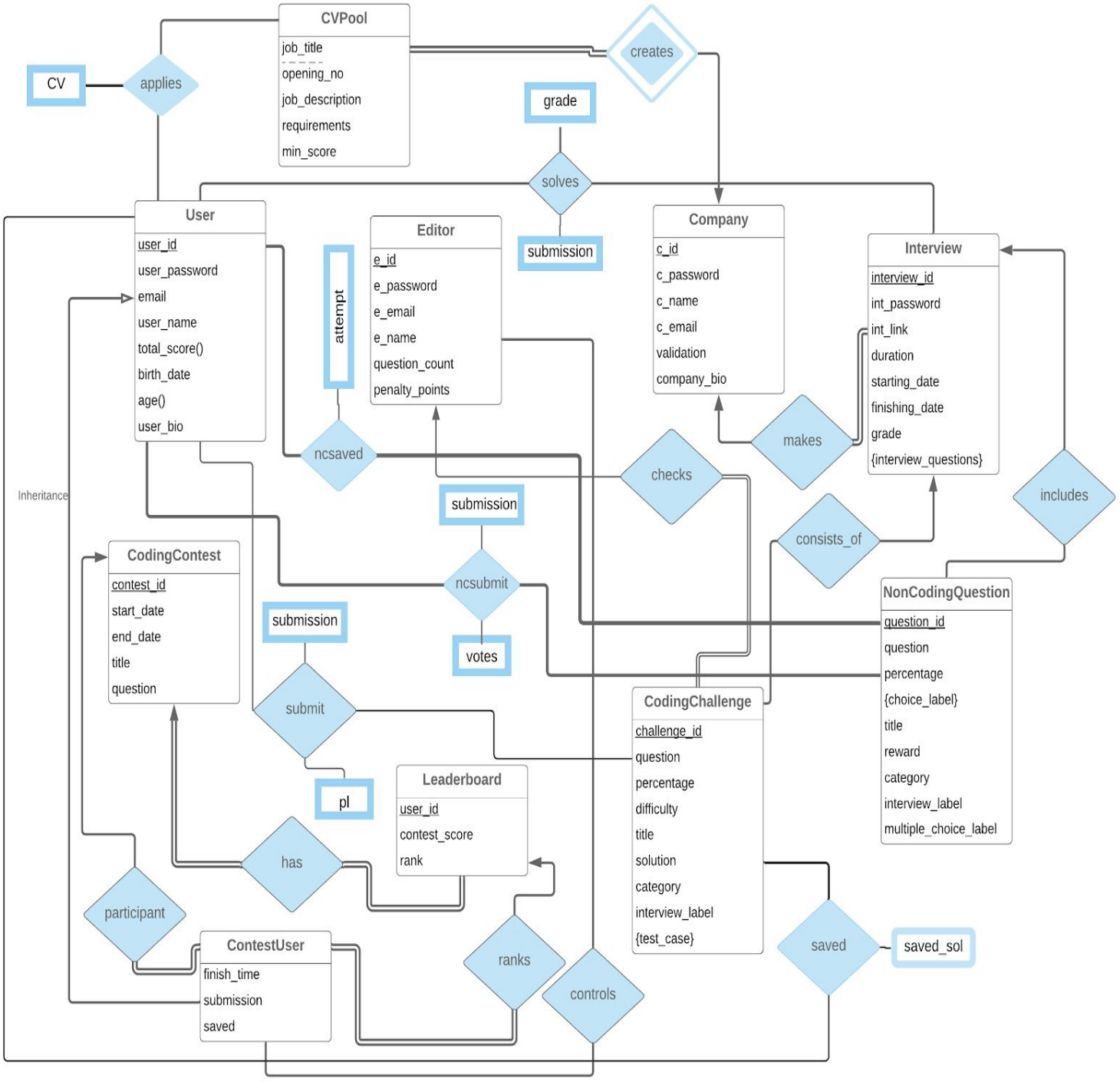
Fazilet Simge Er

Merve Kılıçarslan

Pınar Ayaz

Contents

1. Revised ER


CVPool
job_title
opening_no
job_description
requirements
min_score
CV
applies
creates
grade
solves
submission
User
user_id
user_password
email
user_name
total_score()
birth_date
age()
user_bio
Editor
e_id
e_password
e_email
e_name
question_count
penalty_points
attempt
ncsaved
Company
c_id
c_password
c_name
c_email
validation
company_bio
Interview
interview_id
int_password
int_link
duration
starting_date
finishing_date
grade
{interview_questions}
makes
checks
includes
consists_of
Inheritance
submission
ncsubmit
votes
CodingContest
contest_id
start_date
end_date
title
question
submission
submit
pl
NonCodingQuestion
question_id
question
percentage
{choice_label}
title
reward
category
interview_label
multiple_choice_label
CodingChallenge
challenge_id
question
percentage
difficulty
title
solution
category
interview_label
{test_case}
Leaderboard
user_id
contest_score
rank
has
participant
ranks
controls
saved
saved_sol
ContestUser
finish_time
submission
saved

2. Table Schemas

2.1 User

Relational Model

User(<u>user_id</u>, user_password, email, user_name, total_score, birth_date, age, user_bio)

Functional Dependencies

user_id → user_password email user_name total_score birth_date age user_bio

Candidate Keys

{(user_id)}

Normal Form

3NF

Table Definition

```
create table User(
user_id          int not null auto_increment,
user_password    varchar(10) not null,
email            varchar(30) not null,
user_name        varchar(15) not null,
total_score      int not null,
birth_date       date,
age              int,
user_bio         varchar(100),
primary key (user_id)
);
```

2.2 Editor

Relational Model

Editor(e_id, e_password, e_email, e_name, question_count, penalty_points)

Functional Dependencies

e_id → e_password e_email e_name question_count penalty_points

Candidate Keys

{(e_id)}

Normal Form

3NF

Table Definition

```
create table Editor(
e_id            int not null auto_increment,
e_password      varchar(10) not null,
e_email         varchar(30) not null,
e_name          varchar(15) not null,
question_count  int not null,
penalty_points  int not null,
primary key(e_id)
);
```

2.3 Company

Relational Model

Company(c_id, c_password, c_name, c_email, validation, company_bio)

Functional Dependencies

c_id → c_password c_name c_email validation company_bio

Candidate Keys

{(c_id)}

Normal Form

3NF

Table Definition

```
create table Company(
c_id            int not null auto_increment,
c_password      varchar(10) not null,
c_email         varchar(30) not null,
c_name          varchar(30) not null,
validation      boolean not null,
company_bio     varchar(100),
primary key(c_id)
);
```

2.4 Interview

Relational Model

Interview(<u>interview_id, c_id</u>, int_password, int_link, duration, starting_date, finishing_date)

FK: c_id references Company

Functional Dependencies

interview_id c_id → int_password int_link duration starting_date finishing_date

Candidate Keys

{(interview_id, c_id)}

Normal Form

3NF

Table Definition

```
create table Interview(
interview_id        int not null auto_increment,
c_id                int not null,
int_password        varchar(10) not null,
int_link            varchar(100) not null,
starting_date       date not null,
duration            smallint not null,
finishing_date      date not null,
primary key(c_id, interview),
foreign key(c_id) references Company
);
```

2.5 CodingContest

Relational Model

CodingContest(<u>contest_id</u>, start_date, end_date, title, question)

Functional Dependencies

contest_id → start_date end_date title question

Candidate Keys

{(contest_id)}

Normal Form

3NF

Table Definition

```
create table CodingContest(
contest_id          int not null auto_increment,
start_date          date not null,
end_date            date not null,
title               varchar(20) not null,
question            varchar(500) not null,
primary key(contest_id)
);
```

2.6 Leaderboard

Relational Model

Leaderboard(<u>user_id</u>, contest_score, rank)
FK: user_id references ContestUser

Functional Dependencies

user_id → contest_score rank

Candidate Keys

{(user_id)}

Normal Form

3NF

Table Definition

```
create table Leaderboard(
user_id          int not null,
contest_score    int not null,
rank             int not null,
primary key(user_id),
foreign key(user_id) references ContestUser
);
```

2.7 CodingChallenge

Relational Model

CodingChallenge(<u>challenge_id, e_id,</u> question, percentage, difficulty, title, solution, category, interview_label)
FK: e_id references Editor

Functional Dependencies

challenge_id e_id → question percentage difficulty title solution category interview_label

Candidate Keys

{(challenge_id, e_id)}

Normal Form

3NF

Table Definition

```
create table CodingChallenge(
challenge_id        int not null auto_increment,
e_id                int not null,
question            varchar(500) not null,
percentage          numeric(2,2) not null,
difficulty          varchar(10) not null,
title               varchar(20) not null,
solution            varchar(5000),
category            varchar(20) not null,
interview_label     boolean not null,
primary key(challenge_id, e_id),
foreign key(e_id) references Editor
);
```

2.8 NonCodingQuestion

Relational Model

NonCodingQuestion(question_id, question, percentage, title, reward, category, interview_label, multiple_choice_label)

Functional Dependencies

question_id → question percentage title reward category interview_label multiple_choice_label

Candidate Keys

{(question_id)}

Normal Form

3NF

Table Definition

```
create table NonCodingQuestion(
question_id          int not null auto_increment,
question             varchar(500) not null,
percentage           numeric(2,2) not null,
title                varchar(20) not null,
reward               int not null,
category             varchar(20) not null,
interview_label      boolean not null,
multiple_choice_label boolean not null,
primary key(question_id)
);
```

2.9 NCQuestionChoices

Relational Model

NCQuestionChoices(question_id, choice_label)
FK: question_id references NonCodingQuestion

Functional Dependencies

None

Candidate Keys

{(question_id, choice_label)}

Normal Form

3NF

Table Definition

```
create table NCQuestionChoices(
question_id          int,
choice_label         varchar(50),
primary key(question_id, choice_label),
foreign key(question_id) references NonCodingQuestion
);
```

2.10 ContestUser

Relational Model

ContestUser(<u>user_id, contest_id</u>, finish_time, submission, saved)
FK: user_id references User
FK: contest_id references CodingContest

Functional Dependencies

user_id contest_id → finish_time submission saved

Candidate Keys

{(user_id, contest_id)}

Normal Form

3NF

Table Definition

```
create table ContestUser(
user_id          int not null,
contest_id       int not null,
finish_time      smallint,
submission       varchar(5000),
saved            varchar(5000),
primary key(user_id, contest_id),
foreign key(user_id) references Leaderboard,
foreign key(contest_id) references CodingContest
);
```

2.11 CVPool

Relational Model

CVPool(c_id, job_title, job_description, requirements, opening_no, min_score)
FK: c_id references Company

Functional Dependencies

c_id job_title → job_description requirements opening_no min_score

Candidate Keys

{(c_id, job_title)}

Normal Form

3NF

Table Definition

```
create table CVPool(
c_id                int not null,
job_title           varchar(50) not null,
job_description     varchar(1000) not null,
requirements        varchar(500),
opening_no          smallint not null,
min_sore            int,
primary key(c_id, job_title),
foreign key (c_id) references Company
);
```

2.12 applies

Relational Model

applies(<u>user_id, c_id, job_title</u>, CV)
FK: user_id references User
FK: c_id, job_title references CVPool

Functional Dependencies

user_id c_id job_title → CV

Candidate Keys

{(user_id, c_id, job_title)}

Normal Form

3NF

Table Definition

```
create table applies(
user_id         int not null,
c_id            int not null,
job_title       varchar(50) not null,
CV              varchar(50) not null,
primary key(user_id, c_id, job_title),
foreign key (user_id) references User,
foreign key (c_id, job_title) references CVPool
);
```

2.13 solves

Relational Model

solves(<u>user_id, interview_id</u>, grade, submission)
FK: user_id references User
FK: interview_id references Interview

Functional Dependencies

user_id interview_id → grade submission

Candidate Keys

{(user_id, interview_id)}

Normal Form

3NF

Table Definition

```
create table solves(
user_id          int not null,
interview_id     int not null,
grade            int,
submission       varchar(5000),
primary key(user_id, interview_id),
foreign key (user_id) references User,
foreign key (interview_id) references Interview
);
```

2.14 submit

Relational Model

submit(<u>user_id, challenge_id</u>, submission, pl)
FK: user_id references User
FK: challenge_id references CodingChallenge

Functional Dependencies

user_id challenge_id → submission pl

Candidate Keys

({user_id, challenge_id)}

Normal Form

3NF

Table Definition

```
create table submit(
user_id          int not null,
challenge_id     int not null,
submission       varchar(5000),
pl               varchar(10) not null,
primary key(user_id, challenge_id),
foreign key (user_id) references User,
foreign key (challenge_id) references CodingChallenge
);
```

2.15 consists_of

Relational Model

consists_of(interview_id, challenge_id)
FK: interview_id references Interview
FK: challenge_id references CodingChallenge

Functional Dependencies

None

Candidate Keys

{(interview_id, challenge_id)}

Normal Form

3NF

Table Definition

```
create table consists_of(
interview_id        int not null,
challenge_id        int not null,
primary key(interview_id, challenge_id),
foreign key (interview_id) references Interview,
foreign key (challenge_id) references CodingChallenge
);
```

2.16 includes

Relational Model

includes(<u>interview_id, question_id</u>)
FK: interview_id references Interview
FK: question_id references NonCodingQuestion

Functional Dependencies

None

Candidate Keys

{(interview_id, question_id)}

Normal Form

3NF

Table Definition

```
create table includes(
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 NonCodingChallenge
);
```

2.17 has

Relational Model

has(<u>user_id, contest_id</u>)
FK: user_id references Leaderboard
FK: contest_id references CodingContest

Functional Dependencies

None

Candidate Keys

{(user_id, contest_id)}

Normal Form

3NF

Table Definition

```
create table has(
user_id          int not null,
contest_id       int not null,
primary key(user_id, contest_id),
foreign key (user_id) references Leaderboard,
foreign key (contest_id) references CodingChallenge
);
```

2.18 controls

Relational Model

controls(e_id, user_id, contest_id)
FK: e_id references Editor
FK: user_id references ContestUser
FK: contest_id references ContestUser

Functional Dependencies

None

Candidate Keys

{(e_id, user_id, contest_id)}

Normal Form

3NF

Table Definition

```
create table controls(
e_id            int not null,
user_id         int not null,
contest_id      int not null,
primary key(e_id, user_id, contest_id),
foreign key (e_id) references Editor,
foreign key (user_id) references ContestUser,
foreign key (contest_id) references ContestUser
);
```

2.19 saved

Relational Model

saved(<u>challenge_id, user_id,</u> saved_sol)
FK: challenge_id references CodingChallenge
FK: user_id references User

Functional Dependencies

challenge_id user_id → saved_sol

Candidate Keys

{(challenge_id, user_id)}

Normal Form

3NF

Table Definition

```
create table saved(
challenge_id        int not null,
user_id             int not null,
saved_sol           varchar(5000),
primary key(user_id, challenge_id),
foreign key (challenge_id) references CodingChallenge,
foreign key (user_id) references User,
);
```

2.20 ncsaved

Relational Model

ncsaved(<u>question_id, user_id,</u> attempt)
FK: question_id references NonCodingQuestion
FK: user_id references User

Functional Dependencies

question_id user_id -> attempt

Candidate Keys

{(question_id, user_id)}

Normal Form

3NF

Table Definition

```
create table ncsaved(
question_id        int not null,
user_id            int not null,
attempt            varchar(5000),
primary key(user_id, question_id),
foreign key (question_id) references NonCodingQuestion,
foreign key (user_id) references User,
);
```

2.21 ncsubmit

Relational Model

ncsubmit(<u>user_id, question_id</u>, submission, votes)
FK: user_id references User
FK: question_id references CodingChallenge

Functional Dependencies

user_id question_id→ submission votes

Candidate Keys

({user_id, question_id)}

Normal Form

3NF

Table Definition

```
create table ncsubmit(
user_id          int not null,
question_id      int not null,
submission       varchar(5000),
votes            int,
primary key(user_id, question_id),
foreign key (user_id) references User,
foreign key (question_id) references NonCodingQuestion
);
```

2.22 TestCases

Relational Model

TestCases(<u>challenge_id, test_case</u>)
FK: challenge_id references CodingChallenge

Functional Dependencies

None

Candidate Keys

({challenge_id, test_case)}

Normal Form

3NF

Table Definition

```
create table TestCases(
challenge_id        int not null,
test_case           varchar(100) not null,
primary key(challenge_id, test_case),
foreign key (challenge_id) references CodingChallenge,
);
```

2.23 InterviewQuestions

Relational Model

interview_questions(interview_id, c_id, interview_question)
FK: interview_id references Interview
FK: c_id references Company

Functional Dependencies

None

Candidate Keys

({interview_id, c_id, interview_question)}

Normal Form

3NF

Table Definition

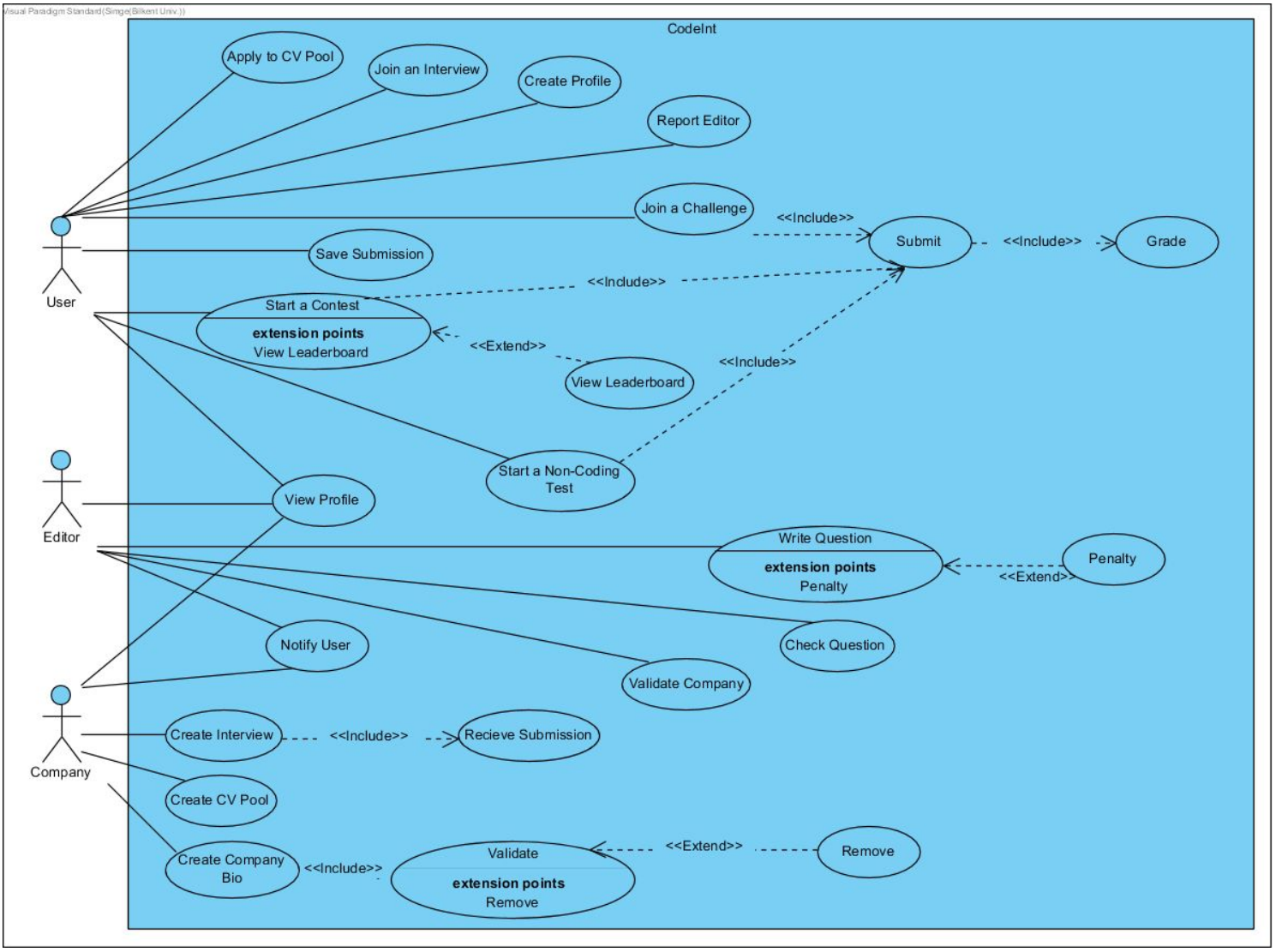
```
create table InterviewQuestions(
interview_id            int not null,
c_id                    int not null,
interview_question              varchar(500) not null,
primary key(interview_id, c_id, interview_question),
foreign key (interview_id) references Interview,
foreign key (c_id) references Company,
);
```

3. Functional Components

In this section, we discuss the functional components of our project.

3.1 Use-Case Model

The diagram below captures the possible end-user interactions with CodeInt. Each of the use cases represented here and their respective flow of the events are explained in detail with the use case tables that follow.



3.2 Use-case Scenarios

Use case Name:	**JoinInterview**
Participating actors:	User
Stakeholders/Interests :	User decides to join an interview created by a company
The flow of events:	1. With the link and password that came from the company, the user will be able to join the interview 2. User submits his/her answer to database 3. Submission is sent to the company directly to be evaluated later.
Pre-conditions:	User has to have an account and must log in to the website User must have a URL specified by a company
Post-conditions:	User joins to the interview
Exit conditions:	The company may not send an interview to the user

Use case Name:	**CreateProfile**
Participating actors:	User
Stakeholders/Interests:	User decides to edit or create his/her profile
The flow of events:	1. User sign ups to the website 2. User clicks his/her profile button 3. User clicks create profile button 4. User can change his/her profile
Pre-conditions:	User has to have an account and must log in to the website
Post-conditions:	User changes his/her profile
Exit conditions:	User may close the website and changes may not be saved Changes may not be saved due to a network problem If the user doesn't provide necessary information, sign up is rejected.
Alternative Scenarios	3.1 If the user has already a profile, she/he can change it by clicking the edit profile button

Use case Name:	**ApplyToCVPool**
Participating actors:	User
Stakeholders/Interests:	User decides to apply a job in the CVPool
The flow of events:	1. User logs in to website 2. User clicks "CV Pool" button 3. User chooses job titles she/he wishes to apply 4. User clicks to "Continue With Application" button. 5. User uploads his/her CV to CVPool. 5. The CV and the profile of the user is sent to the company to be evaluated later
Pre-conditions:	User has to have an account and must log in to the website.
Post-conditions:	User's CV will be added to the company pool.
Exit conditions:	User may not add his/her CV to the pool because of a network problem
Alternative Scenarios:	5.1 User chooses to cancel the application to some job titles. 5.2 User chooses to apply to CVPool without a CV.

Use case Name:	**JoinAContest**
Participating actors:	User
Stakeholders/Interests:	User decides to join a contest
The flow of events:	1. User logs in to the website 2. User clicks to the contest bar from the home page 3. User clicks to join contest button
Pre-conditions:	User has to have an account and must log in to the website There must be an available contest
Post-conditions:	User joins to the contest
Exit conditions:	Contest's time limit ends
Alternative Scenarios:	None

Use case Name:	**StartAChallenge**
Participating actors:	User
Stakeholders/Interests:	User decides to start a challenge
The flow of events:	1. User logs in to the website 2. User clicks to the challenge bar from the home page 3. User decides the category and the topic of the challenge 4. User clicks to the challenge
Pre-conditions:	User has to have an account and must log in to the website
Post-conditions:	User starts the contest
Exit conditions:	None
Alternative Scenarios:	3.1 User chooses a challenge without using filters

Use case Name:	**StartNonlcodingTest**
Participating actors:	User
Stakeholders/Interests:	User decides to start a non-coding test
The flow of events:	1. User logs in to the website 2. User clicks to the non-coding questions' bar from the home page 3. User decides the category and topic of the questions 3. User clicks to the title of the questions
Pre-conditions:	User has to have an account and must log in to the website
Post-conditions:	User starts the test
Exit conditions:	There might be a network error between saving the questions
Alternative Scenarios:	3.1 User chooses a challenge without using filters

Use case Name:	**ReportEditor**
Participating actors:	User
Stakeholders/Interests:	User decides to report an editor for questions' evaluation
The flow of events:	1. User logs in to website 2. User clicks his/her submission page 3. User clicks to a particular submission 4. User sees his/her score and the editor that evaluated the question 5. If the user is not satisfied with his/her score, he/she might report the editor and send the question re-evaluation by clicking the report question button
Pre-conditions:	User has to submit an answer for a coding contest or challenge
Post-conditions:	User reports the editor and sends the question re-evaluation
Exit conditions:	None
Alternative Scenarios::	None

Use case Name:	**WriteQuestions**
Participating actors:	Editor
Stakeholders/Interests:	Editor writes questions for database.
The flow of events:	1. Editor logs into the system. 2. Editor clicks to write question button. 3. Editor writes a question specifying the constraints. 4. Editor clicks to send question button
Pre-conditions:	Editor is in the database of the program.
Post-conditions:	Editor submits the question to the database. Editor question count increases.
Exit conditions:	Editor submits the question and logs out of the system. Editor decides to write another question.
Alternative Scenarios:	2.1 System warns the editor about penalty points if the editor is below the expected question count.

Use case Name:	**ValidateCompany**
Participating actors:	Editor
Stakeholders/Interests:	Editor validates the integrity of a company.
The flow of events:	1. Editor logs into the system. 2. Editor goes to their main page. 3. Editor clicks to “Assigned Companies” button. 4. Editor chooses a company to validate. 5. Editor checks the implied content of the company. 6. Editor makes a decision about the company’s validation. 7. Editor submits the decision to the system.
Pre-conditions:	There must be companies waiting for validation. Company has to provide contact information.
Post-conditions:	Company is joined to the system or rejected.
Exit conditions:	After submitting the controlled question editor redirected to the main page.
Alternative Scenarios:	6.1 Editor approves that the company can be joined to the system. 6.1.1 Editor sends an email to the company for validation. 6.2 Editor disapproves that the company can be joined to the system. 6.2.1 Editor sends an email to company for further clarification. 6.2.2 After 2 months if the company does not reply to email it is out of the system. 6.2.3 If the company replies to email giving satisfying information, the company is rechecked for validation.

Use case Name:	**CreateCVPool**
Participating actors:	Company
Stakeholders/Interests:	Company creates a job application for recruitment.
The flow of events:	1. Company logs into the system. 2. Company pushes the create CVPool Button. 3. Company gives specific information about the job. 4. Company pushes the finalize button. 5. Company's CVPool is created in the system.
Pre-conditions:	The company must be validated before hand. Company has to offer all the necessary conditions.
Post-conditions:	Company can change specifications after creating the CVPool. Only the company can see the applications. Company redirected to the main page after creation.
Exit conditions:	None

Use case Name:	**CreateInterview**
Participating actors:	Company
Stakeholders/Interests:	Company creates a specific interview for a participant.
The flow of events:	1. Company logs into the system. 2. Company pushes create interview button. 3. Company chooses questions from the database or writes their own questions. 4. Company specifies the constraints of the interview. 5. Company pushes finalize button. 6. Interview is created in the system. 7. System generates a unique URL and a password for the interview. 8. Interview URL is emailed to the participant via email by company.
Pre-conditions:	Both company and the participant must be the users of the system. Company has to be validated.
Post-conditions:	Participant completes the interview in the giving time frame. Participant's submission is redirected to the company by the system. After the specified time frame if the interview is not attempted to be solved, it is deleted from database.
Exit conditions:	Company is redirected to the main page.
Alternative Scenarios:	Company closes the system without finishing the interview. Company decides to delete the interview.

Use case Name:	**CreateCompanyBio**
Participating actors:	Company
Stakeholders/Interests:	Company creates an public information page.
The flow of events:	1. Company logs into the system. 2. Company provides specific information about itself. 3. Company applies for validation. 4. Company is validated by Editors. 5. Company receives an email about a validation link. 6. Company completes the validation by clicking the link.
Pre-conditions:	Company must provide communication information.
Post-conditions:	Company is created in the database. Company's Bio page is visible by all users. Company can re-apply to the system if rejected.
Exit conditions:	After submitting the controlled question editor redirected to the main page.
Alternative Scenarios:	3.1 Validation is rejected. 3.1.1 Company is removed from the database.

Use case Name:	**NotifyUser**
Participating actors:	Company, Editor
Stakeholders/Interests:	Actors want to notify the users.
The flow of events:	1. Actors log into the system. 2. Actors choose the users to be notified. 3. Actors provide a reason for notification.
Pre-conditions:	None
Post-conditions:	Users are notified by actors. Users are able to see information about the notification.
Exit conditions:	None
Alternative Scenarios:	None

3.3 Algorithms

3.3.1 Vote Calculation Algorithm

Editors will not check non-coding questions' answers, so user votes will be very important for our system. Therefore, for each answer to a non-coding question, there will be a vote count that starts from 0 and increases with each user vote.

3.3.2 Penalty Points Algorithm

Since all questions except non-coding questions are checked by editors, penalty points are very important to keep the questions and evaluations accurate. Penalty points will be gained by three ways. By not adding three questions in a week, by not doing accurate evaluation for a question and by writing a low quality question. All of these will have a different weight and the penalty points will be calculated by the algorithm. In addition, when an editor submits a new question or checks a challenge, his/her penalty points will be decreased by a small amount.

3.3.3 Total Score Algorithm

Each user will have a total score that is gained by solving a challenge, contest or non-coding question. These questions will have different weights and the total score will be calculated by the total score algorithm. After calculating these scores, users will be able to apply interviews that has some score requirement.

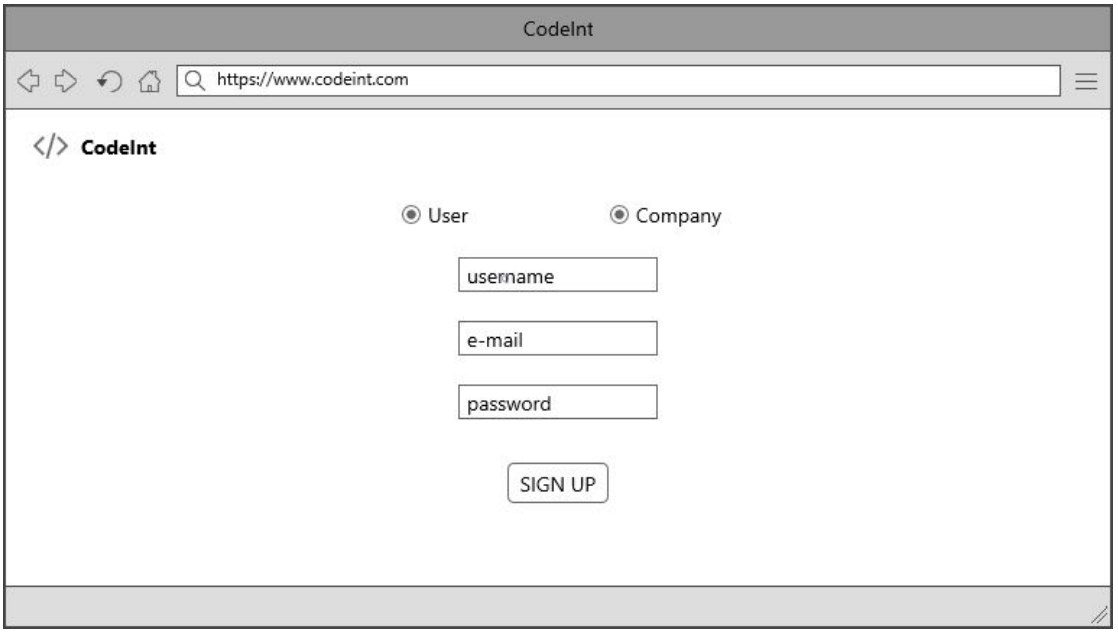
3.4 Data Structures

For the attribute domains, we use smallint for the small numbers like duration, boolean for flags like validation, numerics for percentage, int for IDs, Date type for dates and varchar type of MySQL for strings.

4 User Interface Design and SQL Statements

In this section, we show the user interface design and SQL statements of our project.

4.1 Sign Up Page



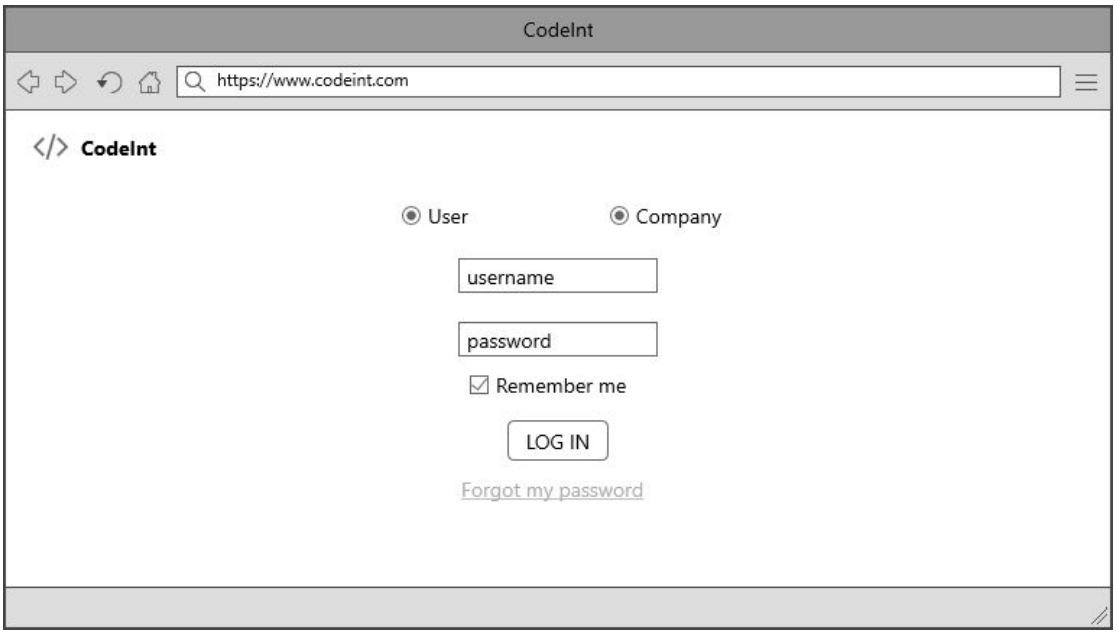
Sign up new user:

insert into User values(0, @password, @email, @username, 0, null, null, null)

Sign up new company:

insert into Company values(0, @password, @c_name, @email, false, null)

4.2 Login Page



Login as user:

select * from User where user_name = @user_name and user_password = @user_password

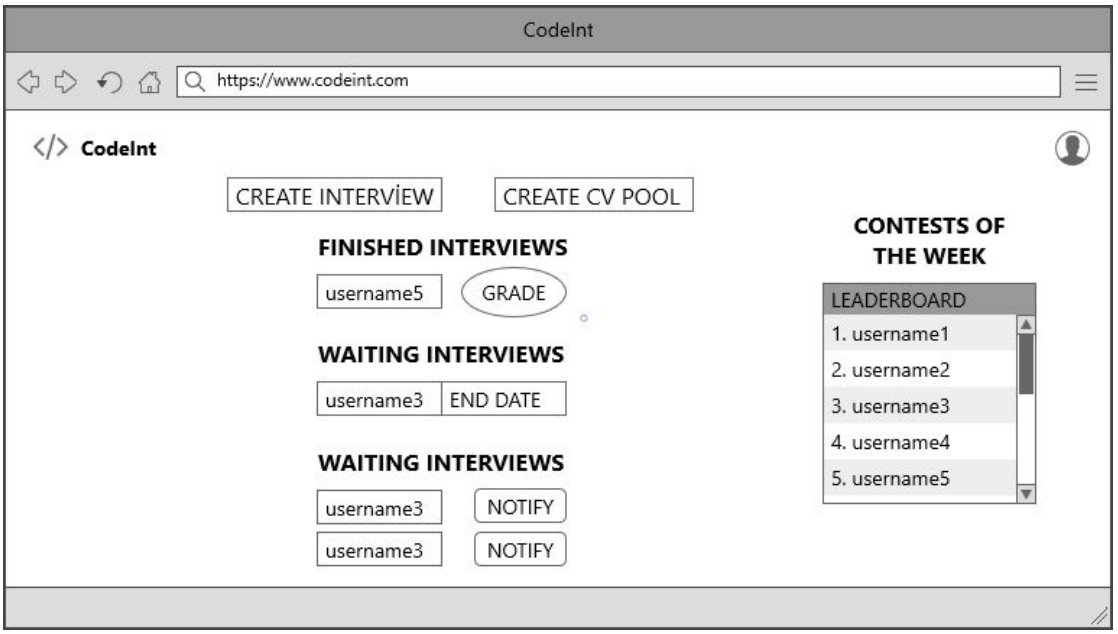
Login as editor:

select * from Editor where e_name = @e_name and e_password = @e_password

Login as company:

select * from Company where c_name = @c_name and c_password = @c_password

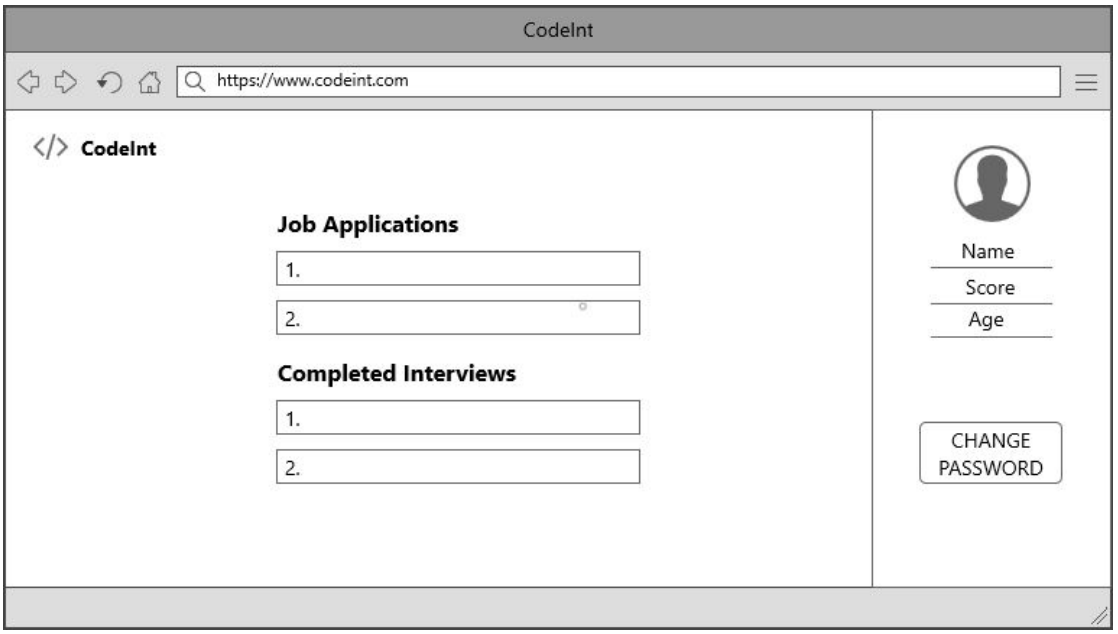
4.3 Company Main Page



Decide on the result of interview and notify the user as a company:

update solves set grade = @grade where interview_id = @interview_id

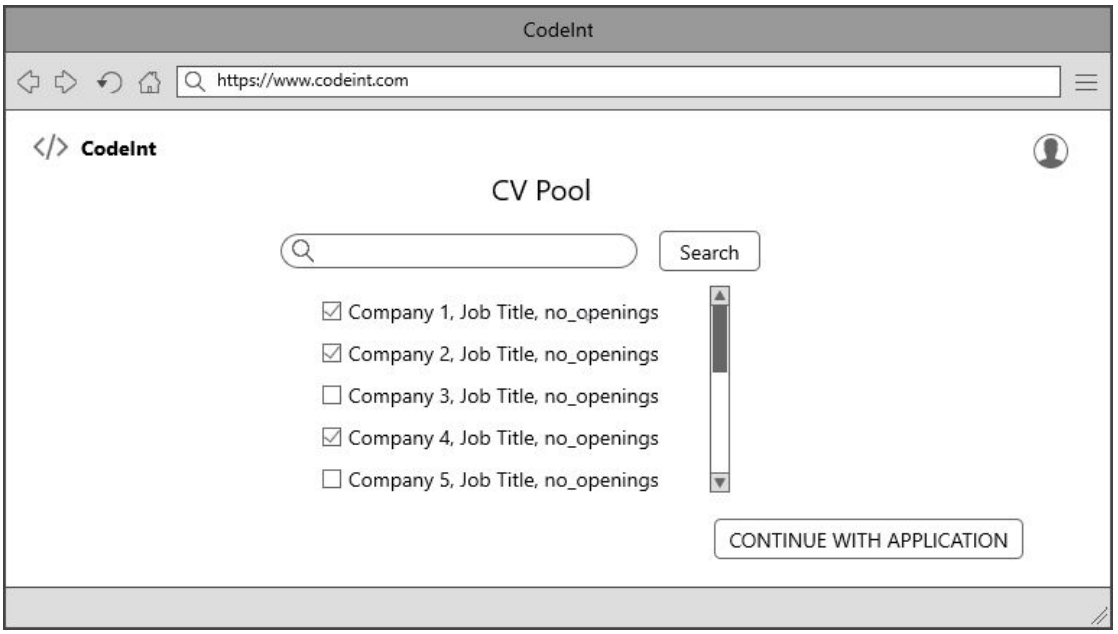
4.4 User Main Page



List previous interviews and results as a user:

select C.company_name, S.submission, S.grade from (Interview I natural join solves S) natural join Company C where I.interview_id = S.interview_id and C.c_id = I.c_id and user_id = @user_id

4.5 CV Pool Page



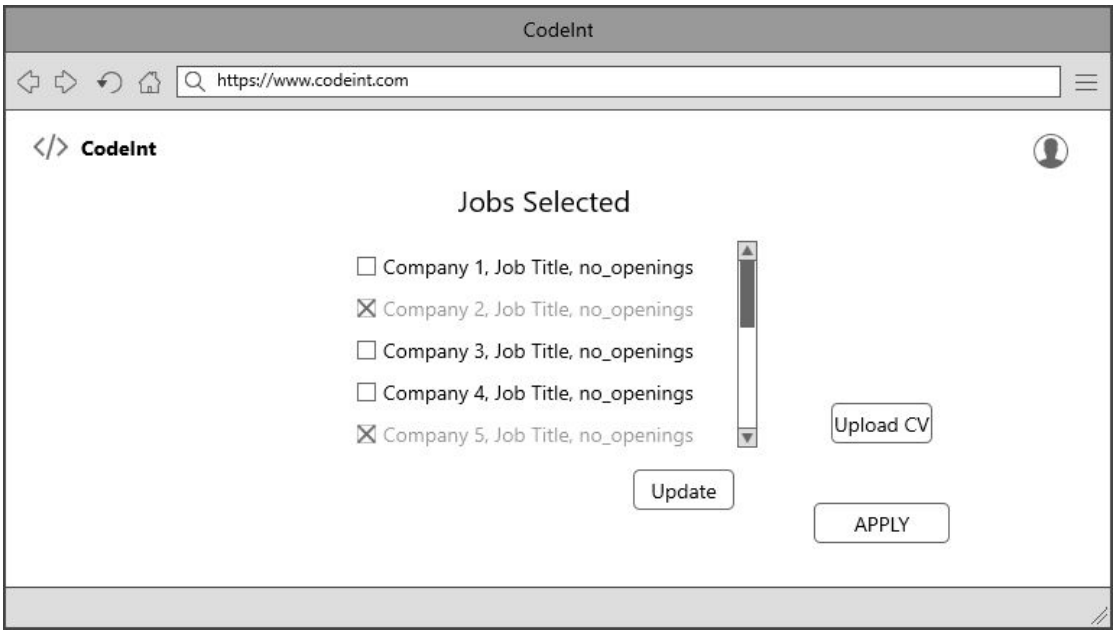
Search CV Pool:

select * from CVPool where job_title like '%@keyword%'

Select jobs to apply as user:

select * from CVPool where c_id = @c_id and job_title = @job_title and opening_no <> 0 as selected_jobs

4.6 Upload to CV Pool Page



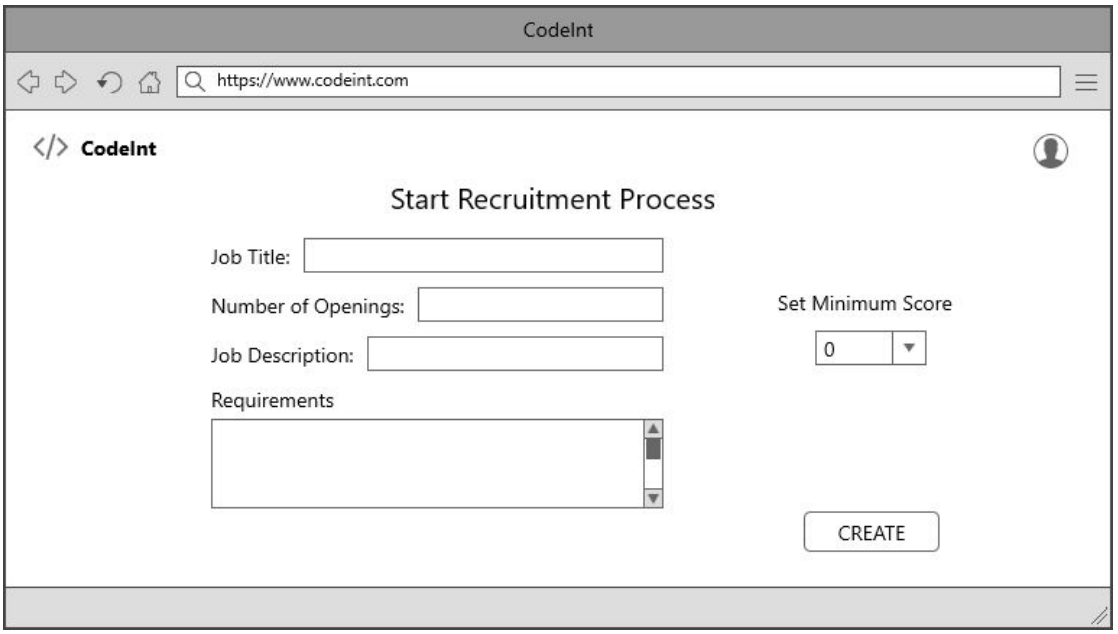
Remove deselected jobs:

delete from selected_jobs where @check = false

Upload CV to selected jobs:

insert into applies where user_id = @user_id, c_id = @c_id, job_title = @job_title, CV = @cv_path

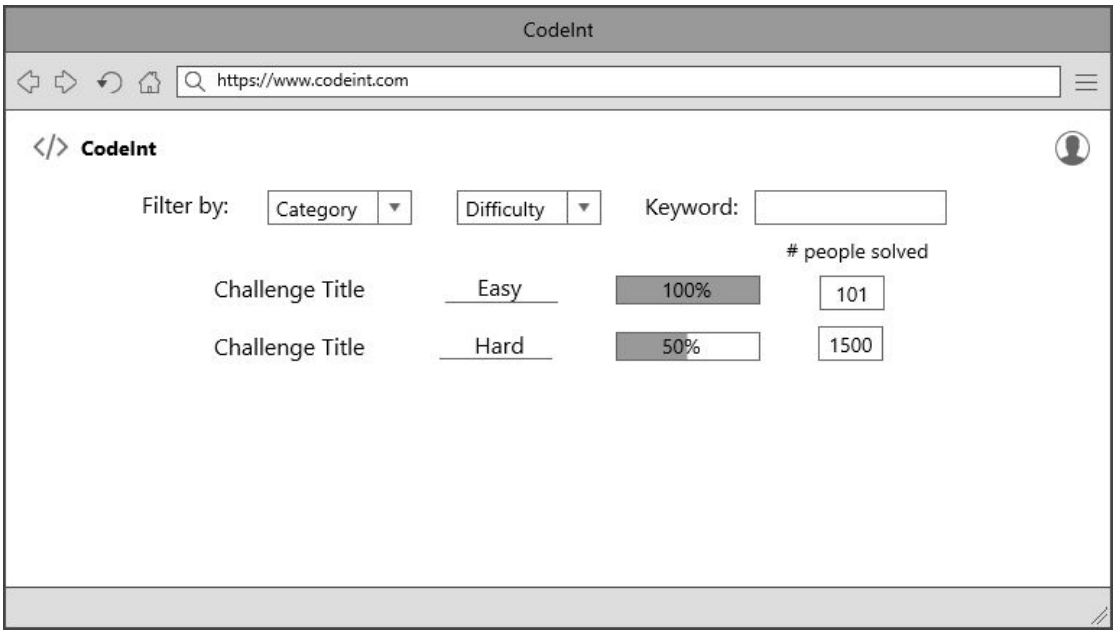
4.7 Create CV Pool



Create new CV Pool as company:

insert into CVPool values (@c_id, @job_title, @job_description, @requirements, @opening_no, @min_score)

4.8 Listing Available Coding Challenges and Questions



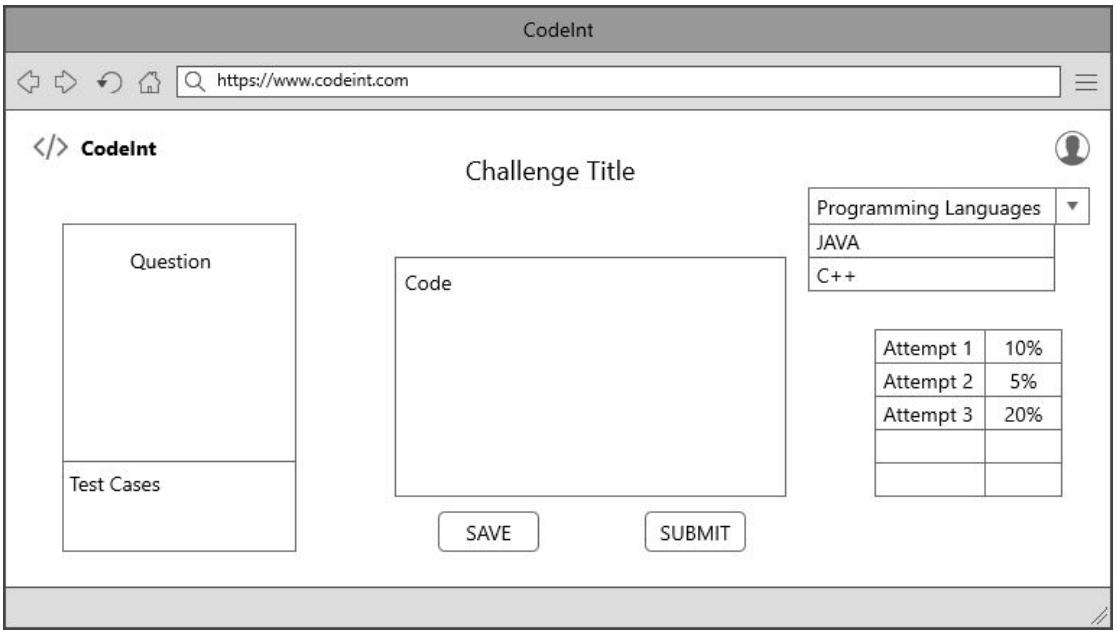
Listing available coding challenges and non coding questions:

select * from CodingChallenge
select * from NonCodingQuestion

Filtering the coding challenges and non coding questions:

select * from CodingChallenge where category = @category and difficulty = @difficulty and title like '%@keyword%'
select * from NonCodingQuestion where category = @category and title like '%@keyword%'

4.9 Solving Challenges



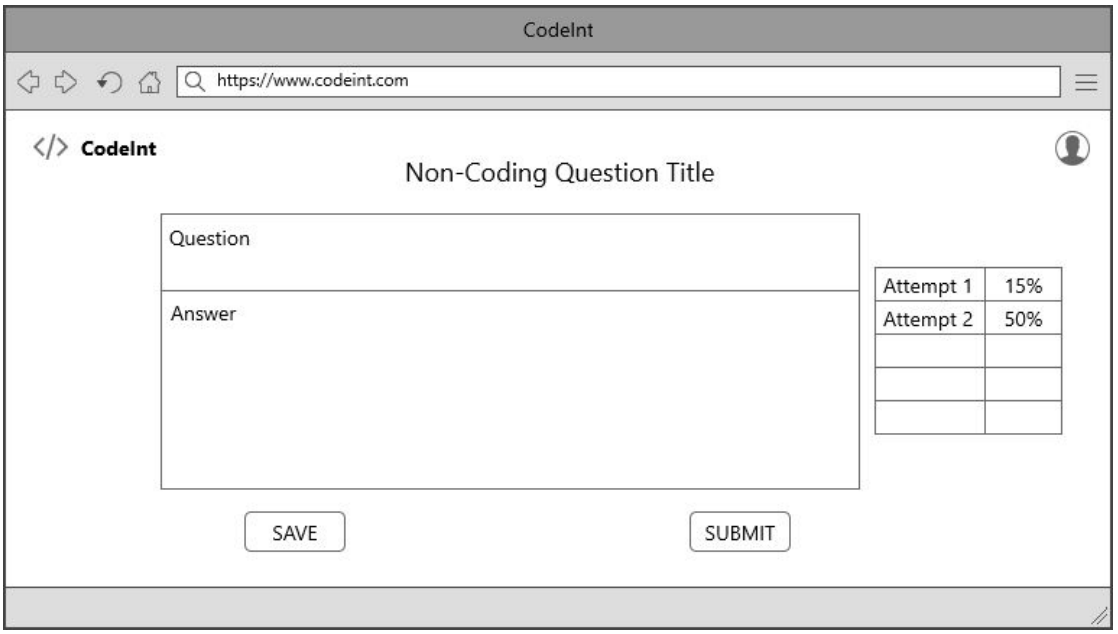
Listing previous attempts and results:

select saved_sol from saved where challenge_id = @challenge_id and user_id = @user_id

Make a new submission with the selected programming language:

insert into submit values (@user_id, @challenge_id, @submission, @pl)

4.10 Solving Non Coding Questions



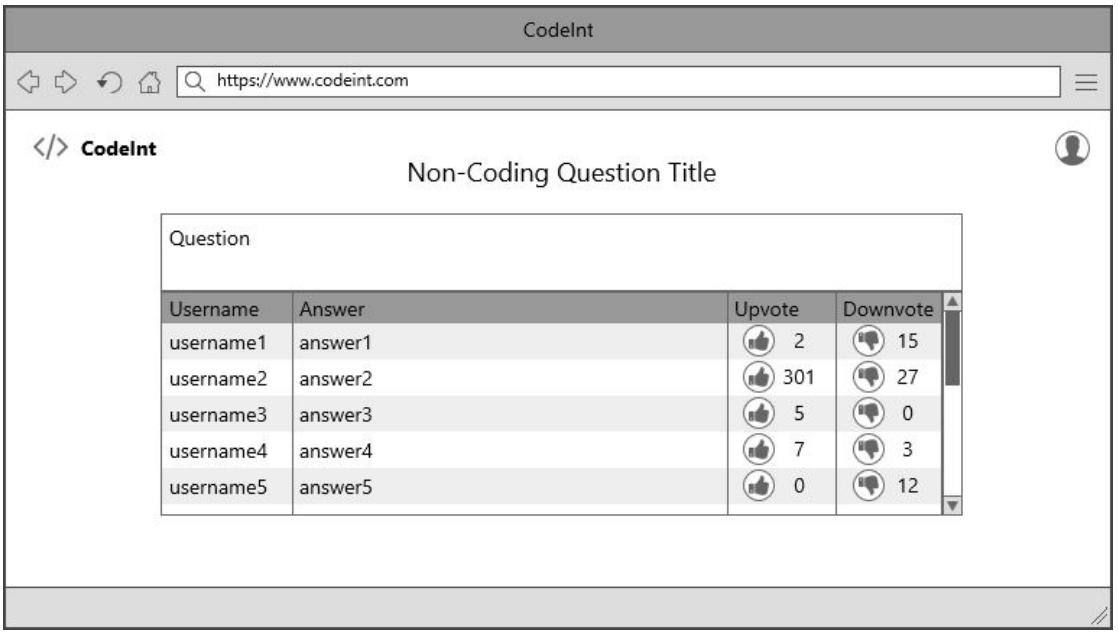
Save attempted answer to the question:

insert into ncsaved values(@question_id, @user_id, @attempt)

Answer the question:

insert into ncsubmit values(@question_id, @user_id, @submission, 0)

4.11 Evaluating Non Coding Questions



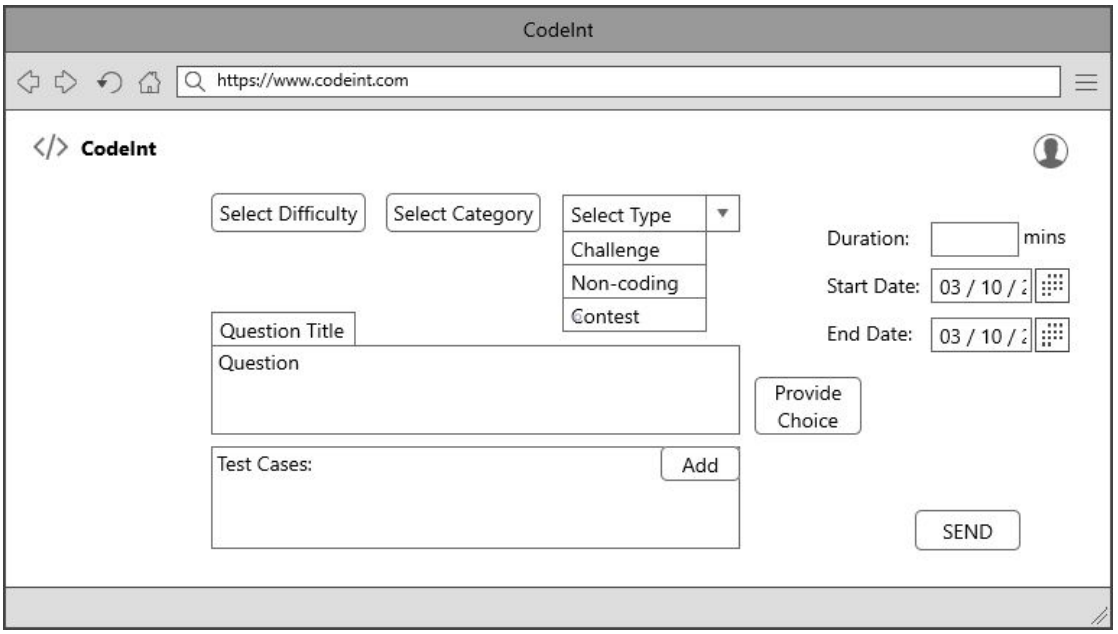
List all users' answers to corresponding question:

select submission from ncsubmit where question_id = @question_id

Upvote/downvote answers:

update ncsubmit set votes = votes + @new_vote where question_id = @question_id and user_id = @user_id

4.12 Preparing Questions for a Challenge/Contest/Non Coding Question



Prepare a coding challenge with difficulty and category as an editor:

insert into CodingChallenge values(0, @e_id, @question, 0, @difficulty, @title, @solution, @category, false)

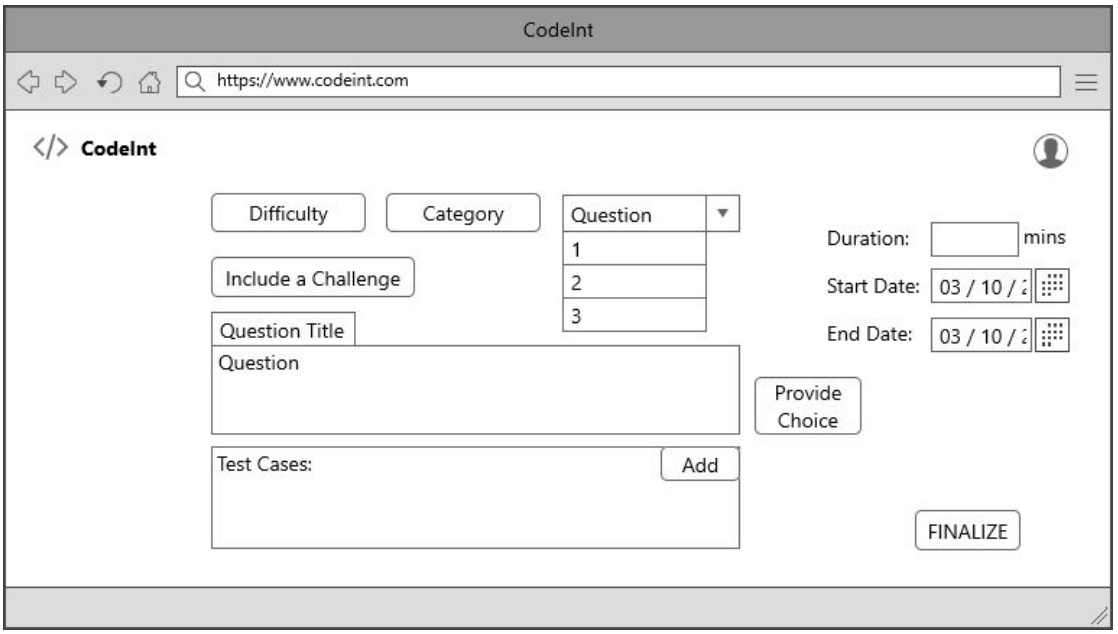
Prepare test cases for a challenge as an editor:

insert into TestCases values(@challenge_id, @test_case)

Create a coding contest with a series of challenges and a duration as an editor:

insert into CodingContest values (0, @start_date, @duration, @title, @questions)

4.13 Preparing Coding Interviews



Specify an interview with a specific date interval and duration as a company:

insert into Interview values(0, @c_id, @int_password, ‘’, ‘’, @duration, @starting_date, @finishing_date)

Add interview questions to an interview as a company:

insert into interview_questions values (@interview_id, @c_id, @question)

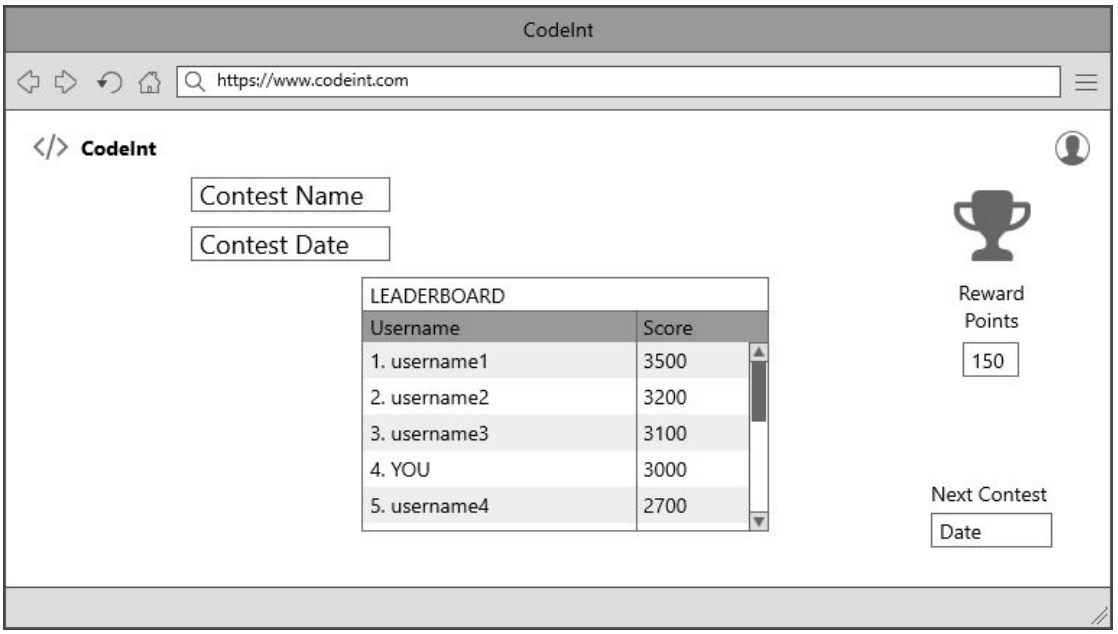
Select from existing challenges for an interview as a company:

insert into consists_of values (@interview_id, @challenge_id)

Select from existing questions for an interview as a company:

insert into includes values (@interview_id, @question_id)

4.14 Leaderboard



Set the ranks of users in the leaderboard:

update Leaderboard set rank as (row_number() over (order by contest_score desc))

See leaderboard of a contest by user:

select user_name, contest_score, rank from Leaderboard L natural join User U
where L.user_id = U.user_id

5 Advanced Database Components

In this section, we show the advanced database components of our project.

5.1 Reports

5.1.1 Total Number of Interviews for Each Company

```
with company_and_interview(c_id, c_name, interview_id) as (select C.c_id,
C.c_name, I.i_id from Company C natural join Interview I where C.c_id = I.c_id)

select c_id, c_name, count(interview_id) from company_and_interview group by
c_id, c_name
```

5.1.2 Average Interview Grade for Each User

```
with user_and_grade(user_id, user_name, interview_id, grade) as (select S.user_id,
U.user_name, S.interview_id, S.grade from solves S natural join user U where
S.user_id = U.user_id)

select user_id, user_name, avg(grade) from user_and_grade group by user_id,
user_name
```

5.1.3 Total Number of Coding Challenges for Each Editor

```
with editor_and_challenge(e_id, e_name, challenge_id) as (select E.e_id, E.e_name,
C.challenge_id from Editor E natural join Challenge C where E.e_id = C.e_id)

select e_id, e_name, count(challenge_id) from editor_and_challenge group by e_id,
e_name
```

5.1.4 Total Number of Submissions for Each Challenge

```
select C.challenge_id, C.title, count(*) from submit S natural join Challenge C where
S.challenge_id = C.challenge_id group by C.challenge_id, C.title
```

5.2 Views

5.2.1 Graded Interviews View

A company has a functionality as notifying a user after the user has completed an interview. Hence in the Company Main Page, notify option will be available only for the users who finished an interview and interview is graded. This view only contains the graded interviews for a company.

```
create view graded_interview as
        select user_name,
        from user
        where user_id in
        (select user_id
        from solves
        where grade is not null)
```

5.2.2 Graded Questions View

An editor has a functionality to notify the users. However an editor can only notify the users whom the editor has checked a question. The view will be used in the Editor main page. This view contains the user which can be notified by an editor.

```
create view graded_questions as
        select user_name,
        from user
        where user_id in
        (select user_id
        from submit, CodingChallenge
        where percentage is not null)
```

5.2.3 Number of Users Who Answered a Question View

The count of user who have successfully answered a question. This view will be used in the Listing Available Coding Challenges and Questions page. The will be show the number of users that solved a particular question.

```
        create view total_user as
        select count(*) from CodingChallenge
        where percentage = 100
```

5.2.4 Apply to CVPool Restriction View

When creating a CVPool, company can specify a minimum score for the users who wishes to apply. In the CV Pool Page, only the job applications that satisfied by the user's score will be shown. The view will show the job applications which the user can apply.

```
create view CVPool_restriction
select * from CVPool C1
where  job_title in(select job_title
                    from applies P1, User  U1
                    where C1.min_score < U1.total_score AND C1. job_title
            = P1.job_title)
```

5.2.5 Restrictive Profile View for Users View

Validated companies and editors are allowed to see user's email when viewing a user's profile. However, invalidated companies and users will only see user_name, age, score and user_bio. In order to achieve this restriction, RestrictiveProfile view will be used.

```
create view RestrictiveProfile
select user_name, total_score, user_bio, age
from User
```

5.3 Triggers

- **Leaderboard Update**

When a contest ends and another contest starts, old leaderboard will be deleted and a new leaderboard will be created automatically.

- **Link Generation for an Interview**

When a company creates an interview and finalizes the creation a link and a password will be generated automatically by the triggers.

- **User Main Page Update**

When the user applies to CVPools or completes an interview, the user main page where the job applications and completed interviews are shown will be updated.

- **Gain Points**

When the user completes a challenge, contest or a non-coding question successfully user score will increase automatically by triggers.

- **Decrease Penalty Points**

When the editor submits a new question to the system or checks a challenge penalty points of the Editor will be decreased automatically.

- **Calculate Age**

When the user enters their birth-date on their profile, the age field will be automatically calculated and entered into the system.

5.4 Constraints

1. The system cannot be used without an account.
2. The system requires at least a username, a password and a valid email address to be enrolled.
3. The password cannot be shorter than 6 characters, and cannot be longer than 10 characters.
4. A user can submit an answer to a coding contest 5 times.
5. A user cannot re-submit an answer when evaluation starts for a particular coding challenge question.
6. A user cannot re-submit an answer when time is up for a coding contest question.
7. Invalidated companies cannot create interviews and CV Pools.
8. Companies communicate with their candidates only through email, there are no extra features like messaging or calling on the website.
9. A question of any kind can only be created when every related parts are filled.

5.5 Stored Procedures

- Sign up and log in processes will be stored processes and they will check for validity during log in.

- Every kind of question like coding challenge, non-coding question and coding contest will be stored procedures. These questions will be used when checking for identical questions.

- The scores of users which are users' reward points for answering correctly will be a stored process. Whenever a user's score needs to be updated, it will be updated using the given parameters.

6 Implementation Details

For our project, we will use MySQL for the database management and InnoDB as our database engine. For back-end development we will use PHP and for front-end development we will use JavaScript.

7 Website

Our project repository where reports and source code can be found:

https://github.com/pinarayaz/CS353-Database-Systems-Project

8 References

[1]LeetCode - The World's Leading Online Programming Learning Platform. [Online]. Available: https://leetcode.com/subscription/. [Accessed: 04-Mar-2019].

[2]“Practice Live Job Interviews - For Free,” Pramp. [Online]. Available: https://www.pramp.com/#/. [Accessed: 04-Mar-2019].

[3]“Online Diagram Software & Visual Solution,” Lucidchart, 13-Nov-2018. [Online]. Available: https://www.lucidchart.com/. [Accessed: 04-Mar-2019].