CS 353 – Database Systems

Project Design Report

Group 8

Mustafa Bayraktar

Zeynep Gözel

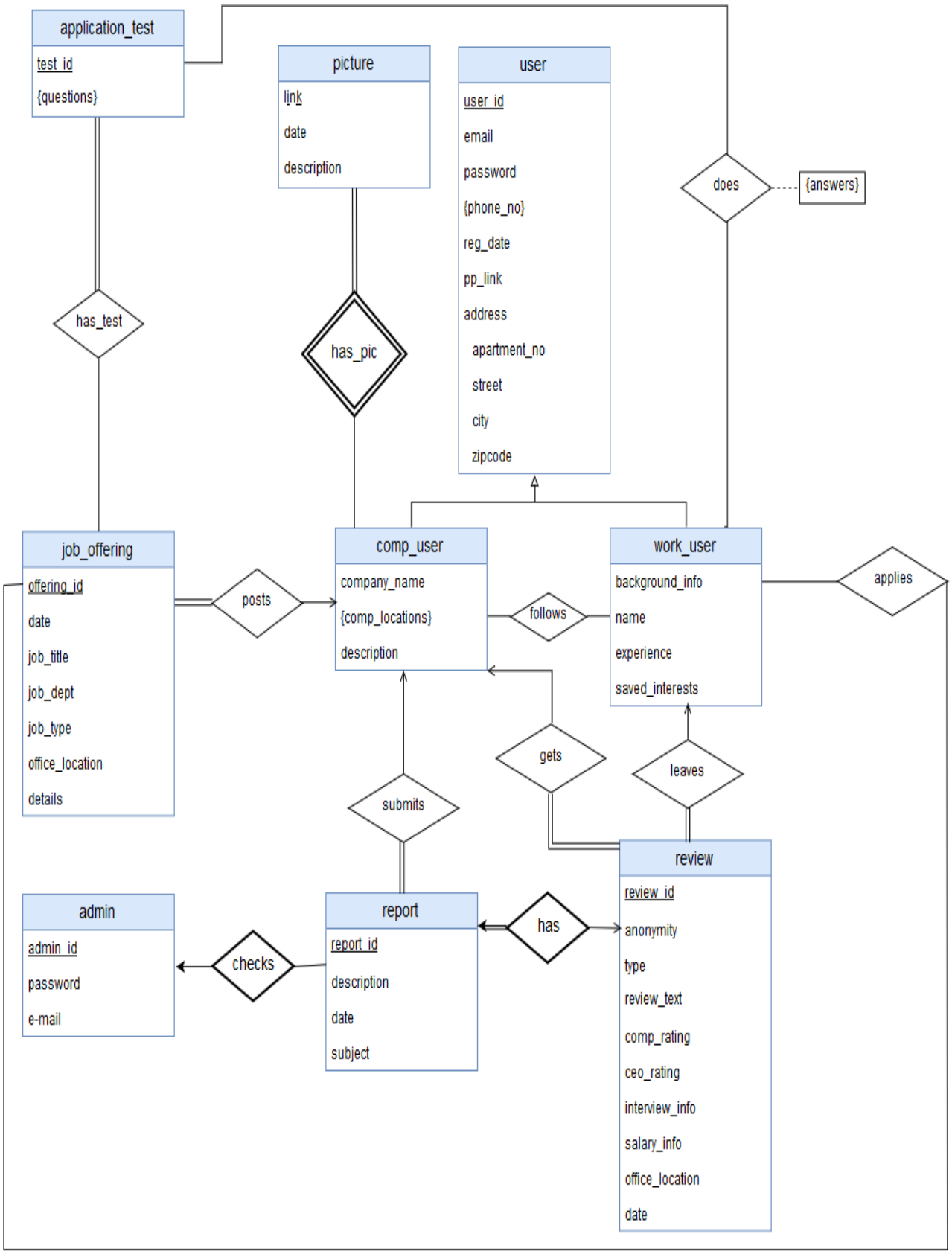
Yağız Efe Mertol

Ege Özcan

Table of Contents

1. Revised E/R Diagram

- Review was changed. Type attribute added to review which defines the reviews type.
- Picture was changed. Only “link” became partial key.
- Total participation added to has_pic
- avg_rating and ceo_rating removed from comp_user
- Total participation added to posts.
- Total participation added to report.
- Total participation added to leaves
- Total participation added to gets
- Applies becomes many to many
- Admin added to check reports.
- Application_test added to job offerings as additional feature.


application_test
test_id
{questions}
picture
link
date
description
user
user_id
email
password
{phone_no}
reg_date
pp_link
address
apartment_no
street
city
zipcode
does
{answers}
has_test
has_pic
job_offering
offering_id
date
job_title
job_dept
job_type
office_location
details
posts
comp_user
company_name
{comp_locations}
description
follows
work_user
background_info
name
experience
saved_interests
applies
gets
leaves
submits
review
review_id
anonymity
type
review_text
comp_rating
ceo_rating
interview_info
salary_info
office_location
date
admin
admin_id
password
e-mail
checks
report
report_id
description
date
subject
has

2. Table Schemas

2.1. User

Model:

User(user_ID,email, password, phone_no, reg_date, picture_link, apartment_no, street, city, zipcode)

Candidate Keys:

user_ID, email, phone_no

Primary Key:

user_ID

Functional Dependencies:

userID → email, password, phone_no, reg_date, picture_link, apartment_no, street, city, zipcode

email → userID, password, phone_no, reg_date, picture_link, apartment_no, street, city, zipcode

Normal Form:

3nf

Table Declaration:

```
CREATE TABLE USER(
        user_ID INT PRIMARY KEY AUTO_INCREMENT,
        email VARCHAR(32) NOT NULL UNIQUE,
        password VARCHAR(32) NOT NULL,
        phone_no NVARCHAR(32) NOT NULL UNIQUE,
        reg_date DATETIME DEFAULT CURRENT_TIMESTAMP,
        pp_link VARCHAR(32) DEFAULT NULL,
        apartment_no VARCHAR(32) DEFAULT NULL,
        street VARCHAR(32) DEFAULT NULL,
        city VARCHAR(32) DEFAULT NULL,
        zipcode VARCHAR(32) DEFAULT NULL
);
```

2.2. Comp_User

Model:

Comp_user(user_ID, company_name, comp_locations, description)

FK: user_ID to User

Candidate Keys:

user_ID, company_name

Primary Key:

user_ID

Functional Dependencies:

userID → company_name, comp_locations, description

company_name → user_ID, comp_locations, description

Normal Form:

3nf

Table Declaration:

```
CREATE TABLE comp_user(
        user_ID INT PRIMARY KEY,
        FOREIGN KEY(user_ID) REFERENCES User(user_ID)
                ON DELETE CASCADE,
        company_name VARCHAR(32) NOT NULL UNIQUE,
        comp_locations NVARCHAR(32) DEFAULT NULL,
        description VARCHAR(128) DEFAULT NULL
);
```

2.3. Work_user

Model:

work_user(user_ID, name, background_info, experience, saved_interest)

FK: user_ID to User

Candidate Keys:

user_ID

Primary Key:

user_ID

Functional Dependencies:

userID → name, background_info, experience, saved_interest

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE work_user(
        user_ID INT PRIMARY KEY,
        FOREIGN KEY(user_ID) REFERENCES User(user_ID)
                ON DELETE CASCADE,
        background_info VARCHAR(128) DEFAULT NULL,
        experience VARCHAR(128) DEFAULT NULL,
        saved_interests VARCHAR(128) DEFAULT NULL
);
```

2.4. Follows

Model:

follows(c_id, w_id)

FK: c_id to Comp_user

FK: w_id to work_user

Candidate Keys:

{c_id, w_id}

Primary Key:

{c_id, w_id}

Functional Dependencies:

There is no functional dependencies.

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE follows(
        c_id INT PRIMARY KEY,
        FOREIGN KEY(c_id) REFERENCES Comp_user(user_ID)
                ON DELETE CASCADE,
        w_id INT PRIMARY KEY,
        FOREIGN KEY(w_id) REFERENCES Work_user(user_ID)
                ON DELETE CASCADE,
);
```

2.5. Picture

Model:

picture(user_ID,link, date, description)

FK: user_ID to comp_user

Candidate Keys:

{user_ID, link}

Primary Key:

{user_ID, link}

Functional Dependencies:

user_ID,link → date, description

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE picture(
        user_ID INT PRIMARY KEY,
        FOREIGN KEY(user_ID) REFERENCES work_user(user_ID)
                ON DELETE CASCADE,
        link VARCHAR(32) NOT NULL,
        date DATETIME DEFAULT CURRENT_TIMESTAMP,
        description VARCHAR(128) DEFAULT NULL
);
```

2.6. Review

Model:

reviews(review_id, anonymity, type, review_text, comp_rating, ceo_rating, interview_info, salary_info,office_location, date)

Candidate Keys:

review_id

Primary Key:

review_id

Functional Dependencies:

review_id → anonymity, type, review_text, comp_rating, ceo_rating, interview_info, salary_info,office_location, date

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Review(
        review_id INT PRIMARY KEY AUTO_INCREMENT,
        anonymity TINYINT(1) NOT NULL,
        type VARCHAR(32) NOT NULL,
        review_text VARCHAR(128) NOT NULL,
        comp_rating INT DEFAULT NULL,
        ceo_rating INT DEFAULT NULL,
        interview_info VARCHAR(32) DEFAULT NULL,
        salary_info VARCHAR(32) DEFAULT NULL,
        office_location VARCHAR(32) DEFAULT NULL,
        date DATETIME DEFAULT CURRENT_TIMESTAMP,
        check( type in(“Full Time”, “Part Time”, “Internship”, “Interview”))
);
```

2.7. Leaves

Model:

leaves(user_id, review_id)

FK: user_id to work_user

FK: review_id to review

Candidate Keys:

{user_id, review_id}

Primary Key:

{user_id, review_id}

Functional Dependencies:

There is no functional dependencies.

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE follows(
        user_id INT PRIMARY KEY,
        FOREIGN KEY(user_id) REFERENCES work_user(user_ID)
                ON DELETE CASCADE,
        review_id INT PRIMARY KEY,
        FOREIGN KEY(review_id) REFERENCES Review(review_ID)
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

2.8. Gets

Model:

gets(user_id, review_id)

FK: user_id to user

FK: review_id to review

Candidate Keys:

{user_id, review_id}

Primary Key:

{user_id, review_id}

Functional Dependencies:

There is no functional dependencies.

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE gets(
        user_id INT PRIMARY KEY,
        FOREIGN KEY(user_id) REFERENCES comp_user(user_ID)
                ON DELETE CASCADE,
        review_id INT PRIMARY KEY,
        FOREIGN KEY(review_id) REFERENCES Review(review_ID)
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

2.9. Report

Model:

report(report_id, description, date, subject)

Candidate Keys:

report_id

Primary Key:

report_id

Functional Dependencies:

report_id → description, date, subject

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Report(
        report_id INT PRIMARY KEY AUTO_INCREMENT,
        description VARCHAR(128) NOT NULL,
        date DATETIME DEFAULT CURRENT_TIMESTAMP,
        subject VARCHAR(32) NOT NULL
);
```

2.10. Has

Model:

has(report_id, review_id)

FK: review_id to review

FK: report_id to report

Candidate Keys:

report_id, review_id

Primary Key:

report_id

Functional Dependencies:

There is no functional dependencies.

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE has(
        report_id INT PRIMARY KEY,
        FOREIGN KEY(report_id) REFERENCES report (report_id)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        review_id INT,
        FOREIGN KEY(review_id) REFERENCES Review(review_ID)
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

2.11. Submits

Model:

submits(user_id, report_id)

FK: user_id to comp_user

FK: report_id to report

Candidate Keys:

report_id

Primary Key:

report_id

Functional Dependencies:

there is no functional dependencies.

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE submits(
        user_id INT PRIMARY KEY,
        FOREIGN KEY(user_id) REFERENCES comp_user(user_ID)
                ON DELETE CASCADE,
        report_id INT PRIMARY KEY,
        FOREIGN KEY(report_id) REFERENCES report (report_id)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        review_id INT PRIMARY KEY,
);
```

2.12. Admin

Model:

admin(admin_id, password, e-mail)

Candidate Keys:

admin_id, e-mail

Primary Key:

admin_id

Functional Dependencies:

admin_id → password

e-mail → password

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE Admin(
        admin_id INT PRIMARY KEY AUTO_INCREMENT,
        password VARCHAR(32) NOT NULL,
        e-mail VARCHAR(32) NOT NULL UNIQUE
);
```

2.13. Checks

Model:

checks(report_id, admin_id)

FK: report_id to report

FK: admin_id to admin

Candidate Keys:

report_id, admin_id

Primary Key:

report_id

Functional Dependencies:

There is no fuctional dependencies

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE checks(
        report_id INT PRIMARY KEY,
        FOREIGN KEY(report_id) REFERENCES report (report_id)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        admin_id INT,
        FOREIGN KEY(admin_id) REFERENCES admin ( admin_id)
                ON DELETE CASCADE
);
```

2.14. Job_offering

Model:

job_offering(offering_id, date, job_title, job_dept, job_type, office_location, details)

Candidate Keys:

offering_id

Primary Key:

offering_id

Functional Dependencies:

offering_id → date, job_title, job_dept, job_type, office_location, details

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE job_offering(
        offering_id INT PRIMARY KEY AUTO_INCREMENT,
        date DATETIME DEFAULT CURRENT_TIMESTAMP,
        job_title VARCHAR(32) NOT NULL,
        job_dept VARCHAR(32) DEFAULT NULL,
        job_type VARCHAR(32) DEFAULT NULL,
        office_location VARCHAR(32) NOT NULL,
        details VARCHAR(128) DEFAULT NULL
);
```

2.15. Application_test

Model:

application_test(test_id, questions)

Candidate Keys:

test_id

Primary Key:

test_id

Functional Dependencies:

test_id → questions

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE application_test(
        test_id INT PRIMARY KEY AUTO_INCREMENT,
        questions NVARCHAR(128) NOT NULL
);
```

2.16. Has_test

Model:

has (offering_id, test_id)

FK: offering_id to job_offering

FK: test_id to application_test

Candidate Keys:

{offering_id, test_id}

Primary Key:

{offering_id, test_id}

Functional Dependencies:

There is no functional dependencies.

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE has_test(
        offering_id INT PRIMARY KEY,
        FOREIGN KEY(offering_id) REFERENCES job_offering (offering_id)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        test_id INT PRIMARY KEY,
        FOREIGN KEY(test_id) REFERENCES application_test ( test_id)
                ON DELETE CASCADE
                ON UPDATE CASCADE
);
```

2.17. Does

Model:

does(test_id, user_id, answers)

FK: test_id to application_test

FK: user_id to work_user

Candidate Keys:

{test_id, user_id}

Primary Key:

{test_id, user_id}

Functional Dependencies:

There is no functional dependencies.

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE does(
        test_id INT PRIMARY KEY,
        FOREIGN KEY(test_id) REFERENCES application_test ( test_id)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        user_id INT PRIMARY KEY,
        FOREIGN KEY(user_id) REFERENCES work_user (user_id)
                ON DELETE CASCADE,
        answers VARCHAR(32)
);
```

2.18. Posts

Model:

posts(offering_id, user_id)

FK: offering_id to job_offering

FK: user_id to user

Candidate Keys:

{ offering_id, user_id }

Primary Key:

{ offering_id, user_id }

Functional Dependencies:

There is no functional dependencies.

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE posts(
        offering_id INT PRIMARY KEY,
        FOREIGN KEY(offering_id) REFERENCES job_offering (offering_id)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
        user_id INT PRIMARY KEY,
        FOREIGN KEY(user_id) REFERENCES comp_user (user_id)
                ON DELETE CASCADE
);
```

2.19. Applies

Model:

applies(user_id, offering_id)

FK: user_id to work_user

FK: offering_id to job_offering

Candidate Keys:

{ user_id, offering_id }

Primary Key:

{ user_id, offering_id }

Functional Dependencies:

There is no functional dependencies.

Normal Form:

BCNF

Table Declaration:

```
CREATE TABLE applies(
        user_id INT PRIMARY KEY,
        FOREIGN KEY(user_id) REFERENCES work_user (user_id)
                ON DELETE CASCADE

        offering_id INT PRIMARY KEY,
        FOREIGN KEY(offering_id) REFERENCES job_offering (offering_id)
                ON DELETE CASCADE
                ON UPDATE CASCADE,
);
```

3. Functional Components

3.1. Algorithms

3.1.1. Job Offer Posting and Application Related Algorithms

Work users will be able to search for jobs based on job title, job type and job location.

Details of the application are taken into the database by following restrictions. Many users can apply for the same job and the applications will be stored in the database. Companies will be able to see applications on their job offerings.

How many job offerings a company user posts will be monitored and the job offerings will be monitored for duplicates.

3.1.2. Logical Requirements

Our system should work without encountering any logical errors. Any kind of attribute that is related to dates like registration dates, review dates, report dates and job offering dates should be scrutinised. It must be ensured that these dates do not start before the current date.

There are other dates that need to be checked.

- The report date must come after the review date

- The review date must come after registration date

- Job offering date must be after registration date

3.2. Data Structures

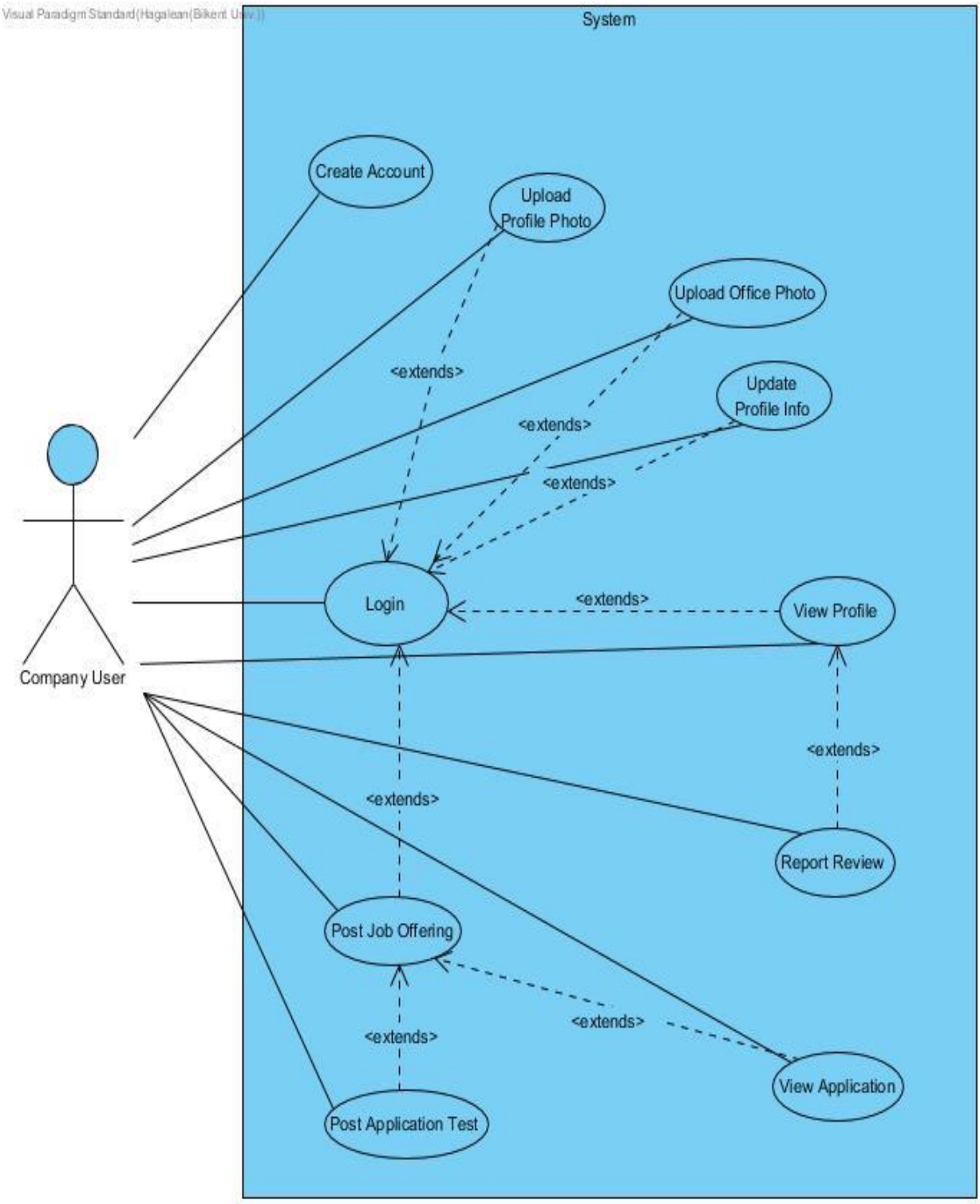
Our system relies on date, numeric and alphabetic types. Attributes that are numeric will be stored as INT. VARCHAR will be used to store string type attributes because the length of these strings are not predetermined. Date values will be stored conveniently as DATE type

3.3. Use Cases

3.3.1. Company User

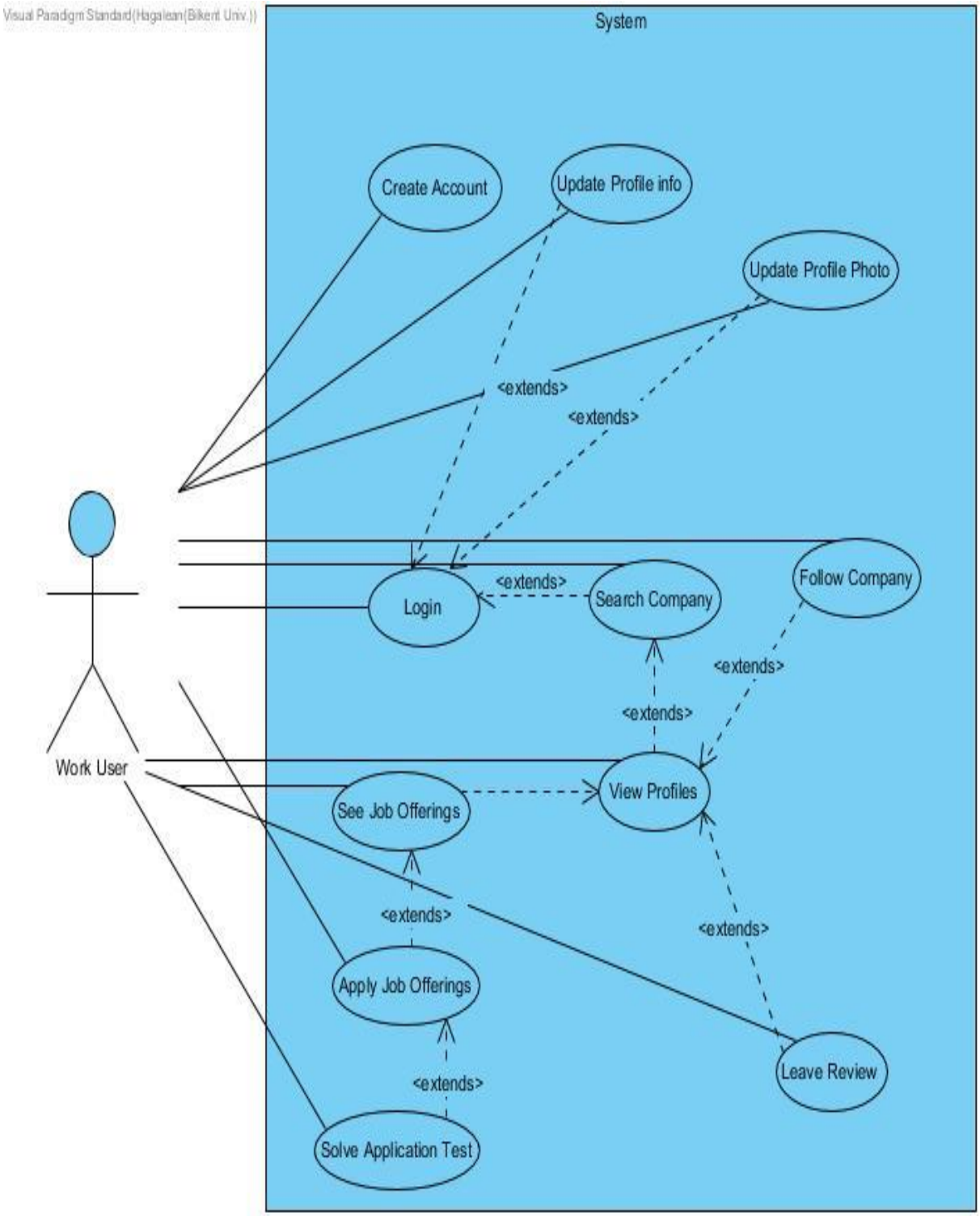
- **Create Account**: Company users can create acounts with company name, e-mail, password, phone number. e-mail and phone number should be unique for every company user.
- **Login:** Company users can login by entering their e-mail and password. After they logged the provided services by system can be used. These services are post job offerings, update profile information and picture, upload office pictures, view profiles and applications, report review.
- **Update Profile Info:** Company users can change their own profile information such as their password, phone number, address, other locations and description. However, they cannot change their e-mail, or company name.
- **Upload profile photo:** Company users can upload profile photo. However, there can be just one profile photo.
- **Upload office photo:** Company users can upload their office photos to their profile. These photos can be seen by other users.
- **Report review:** Company users can report reviews in their profile, if they think reviews are not fair.
- **Post Job Offering:** Company users can post new job offerings. job title and office location must be provided by company user. Also if they want they can also provide job department and details.
- **Post Application Test:** Company users can post new application tests to their job offerings.

- **View Applications:** Company users can see applications of their job offerings.
- **View Profile:** Company user can see its own profile.



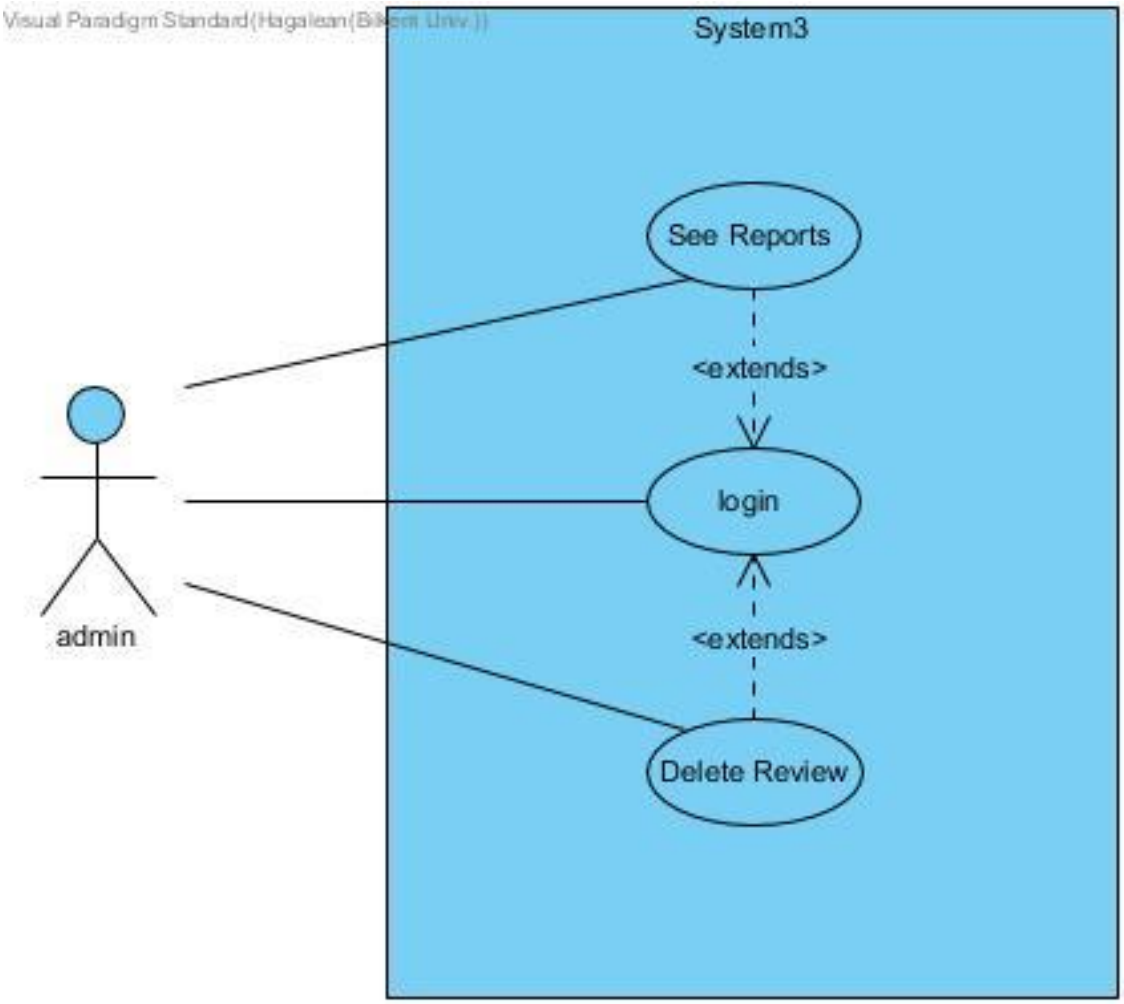
3.3.2. Work_User

- **Create Account:** Work users can create an acount with their name, e-mail, password, phone number. e-mail and phone number should be unique for every company user.
- **Login:** Work users can login by entering their e-mail and password. After they log in the provided services by system can be used. These services are apply and see job offerings, follow companies, leave reviews, update profile information and picture, view profiles and applications, search company by criteria.
- **Update Profile Info:** Work users can change their own profile information such as their password, phone number, address, background informations, experiences, interests. However, they cannot change their e-mail or name.
- **Upload Profile Photo:** Work users can a upload/update profile pictures. However, there can be just one profile photo.
- **Leave Review:** Work users can leave reviews to company profiles.
- **See Job Offerings:** Work users can see job offerings of companies.
- **Apply Job Offerings:** Work users can apply to job offerings.
- **Solve Application Test:** Work users can solve application tests of job offerings.
- **Follow Company:** Work users can follow companies by their profile page.
- **View Profiles:** Work user can see companies profiles
- **Search Company by Criteria:** Work users can search companies.


Visual Paradigm Standard(Hagalean(Bilkent Univ.))
System
Create Account
Update Profile info
Update Profile Photo
<extends>
<extends>
Login
<extends>
Search Company
Follow Company
<extends>
<extends>
Work User
See Job Offerings
View Profiles
<extends>
<extends>
Apply Job Offerings
Leave Review
<extends>
Solve Application Test

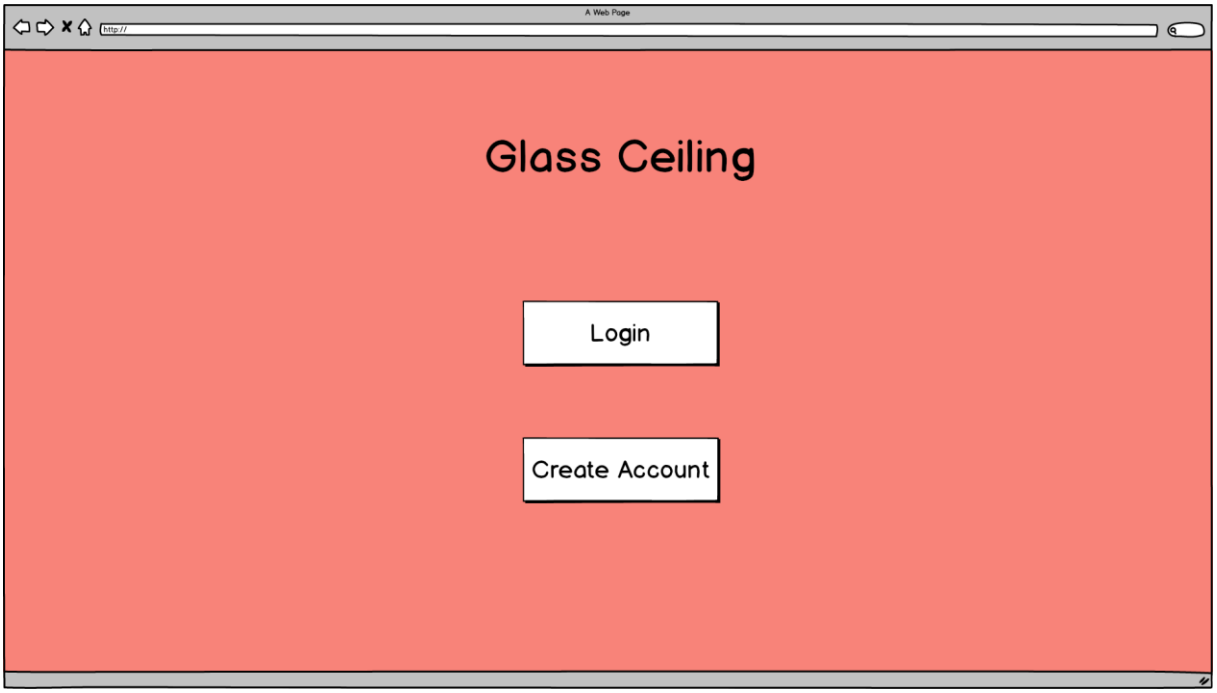
3.3.3. Admin

- **Login:** Admin can login by entering their e-mail and password. After they log in they can see reports and they can remove reviews.
- **See Reports:** Admin can see the reports left by companies.
- **Remove review:** Admin can remove reviews if the reviews are improper.



4. User interface design and corresponding SQL statements

4.1. Homepage

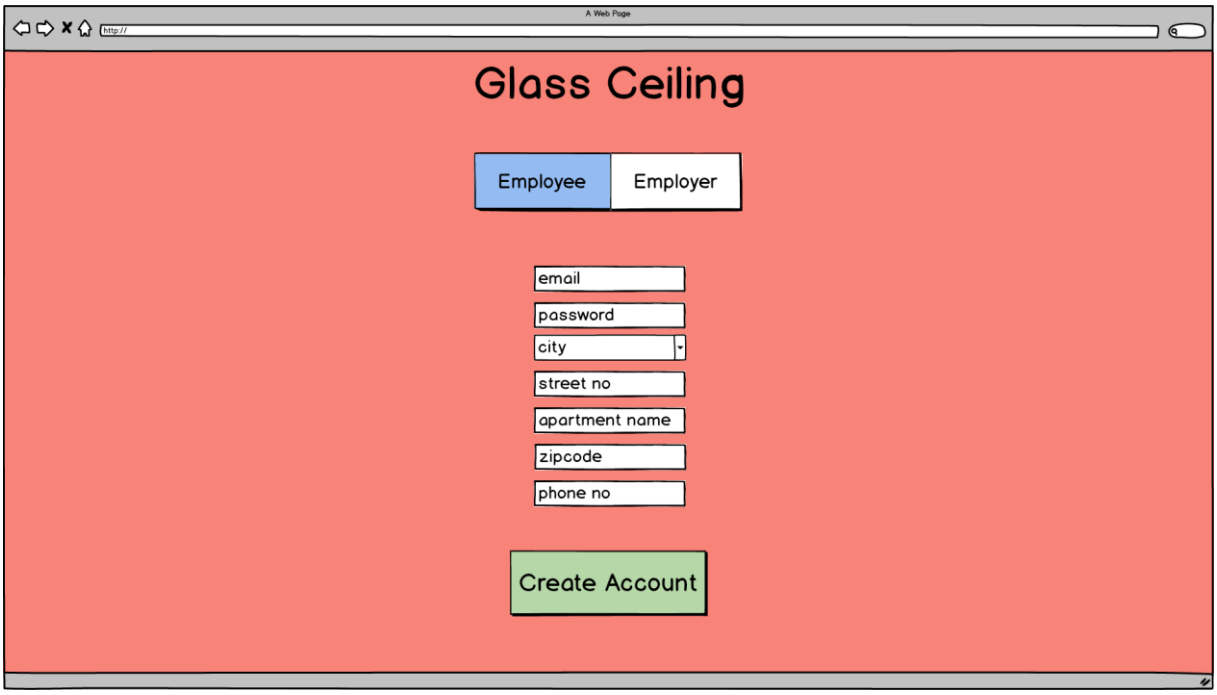


Inputs: -

Process: Users who have not logged in to the system will be greeted by the homepage. The homepage has two buttons. The login button redirects users to the login page and the create account button redirects users to the create account page.

SQL Statements: -

4.2. Create Account Page



Inputs: @email, @password, @city, @street_no,@apartment_no, @zipcode, @phone_no

Process: This page can be reached from the homepage. In this page users must enter a valid email address and a password to create an account. The other parts can be left blank. Both work and company users use this page to create an account and they need to use the toggle on top of the page to specify what type of a user they are.

SQL Statements:

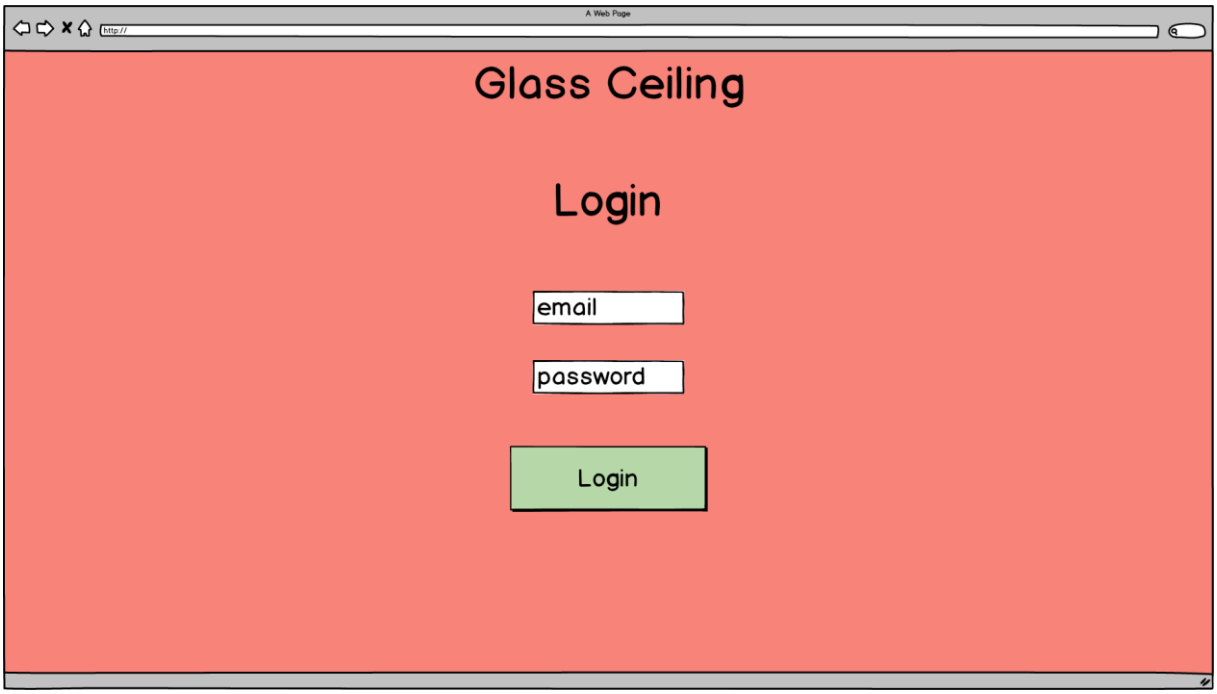
Create Account Pressed, Employee Toggle is Selected:

INSERT INTO work_user (email, password, city, street, apartment_no, zipcode, phone_no)
VALUES (@email, @password, @city, @street_no,@apartment_no, @zipcode, @phone_no)

Create Account Pressed, Employer Toggle is Selected:

INSERT INTO comp_user (email, password, city, street, apartment_no, zipcode, phone_no)
VALUES (@email, @password, @city, @street_no,@apartment_no, @zipcode, @phone_no)

4.3. Login Page



Inputs: @email, @password

Process: The login page can be reached from the homepage. This is the screen where users who have not already logged in to the system can enter their credentials(email and password) to login. Both work and company users can use this screen to login.

SQL Statements:

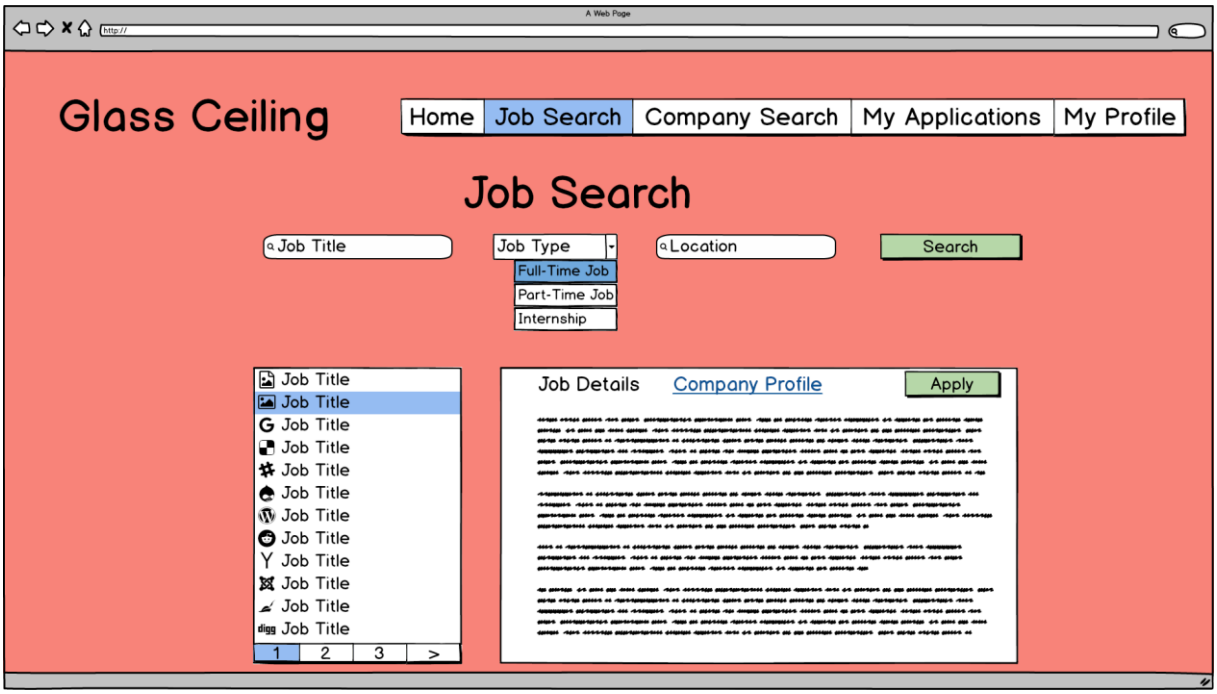
On Login Button:

SELECT user_id,email,password

FROM user

WHERE email = @email and password = @password ;

4.4. Job Search Page for Work User



Inputs: @job_title, @job_type, @location

Process: This is the page where a work user can search for different kinds of job opportunities. A logged in work user can access this page through the top navigation bar from any screen. The user is expected to enter a job title and then specify the type of their search (looking for a full time job, looking for an internship etc.) the user also needs to specify a location.

After the user presses the search button the results of the search are displayed on the screen. Available jobs that meet the search criteria are displayed on the left and the details of the job are displayed on the right of the screen. The user can then apply to a job from this screen or go to the profile page of the company that posted the job offering.

SQL Statements:

On search button pressed:

SELECT details, offering_id

FROM job_offering

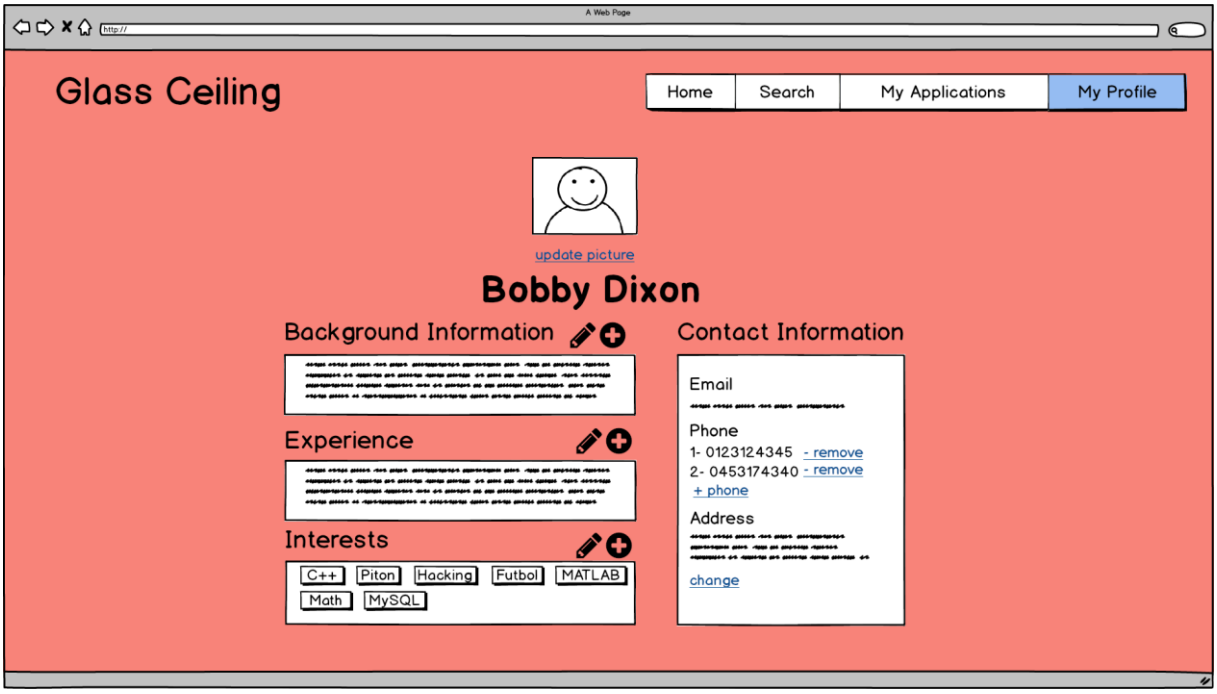
WHERE job_title = @job_title and job_type = @job_type and office_location = @location

On apply button pressed:

INSERT INTO applies (user_id,offering_id)

VALUES (@user_id, @offering_id)

4.5. My Profile Page for Work User



Inputs: @phone_no, @picture_link, @background, @experience,@address,@interest

Process: This is the page where a work user can look at their profile details and update profile information. A logged in work user can access this page through the top navigation bar by clicking on the my profile button from any screen.

SQL Statements:

On update picture pressed:

UPDATE user

SET picture_link = @picture_link

WHERE user_id = @user_id

On remove phone number pressed:

UPDATE user

SET phone_no = @phone_no

WHERE user_id = @user_id

On add phone number pressed:

INSERT INTO user(phone_no)

VALUES(@phone_no)

On change address pressed:

UPDATE user

SET address = @address

WHERE user_id = @user_id

On change background information pressed:

UPDATE work_user

SET background_info = @background

WHERE user_id = @user_id

On change experience pressed:

UPDATE work_user

SET experience = @experience

WHERE user_id = @user_id

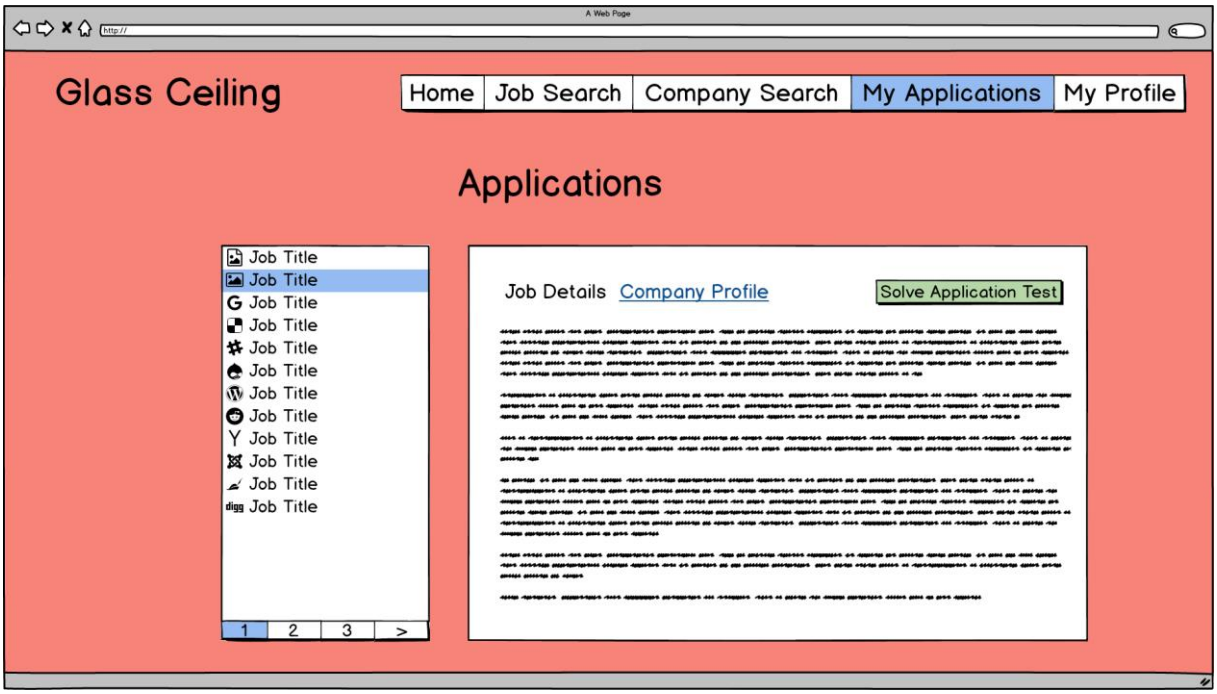
On change interests pressed:

UPDATE work_user

SET saved_interests = @interest

WHERE user_id = @user_id

4.6. My Applications Page for Work User



Inputs: -

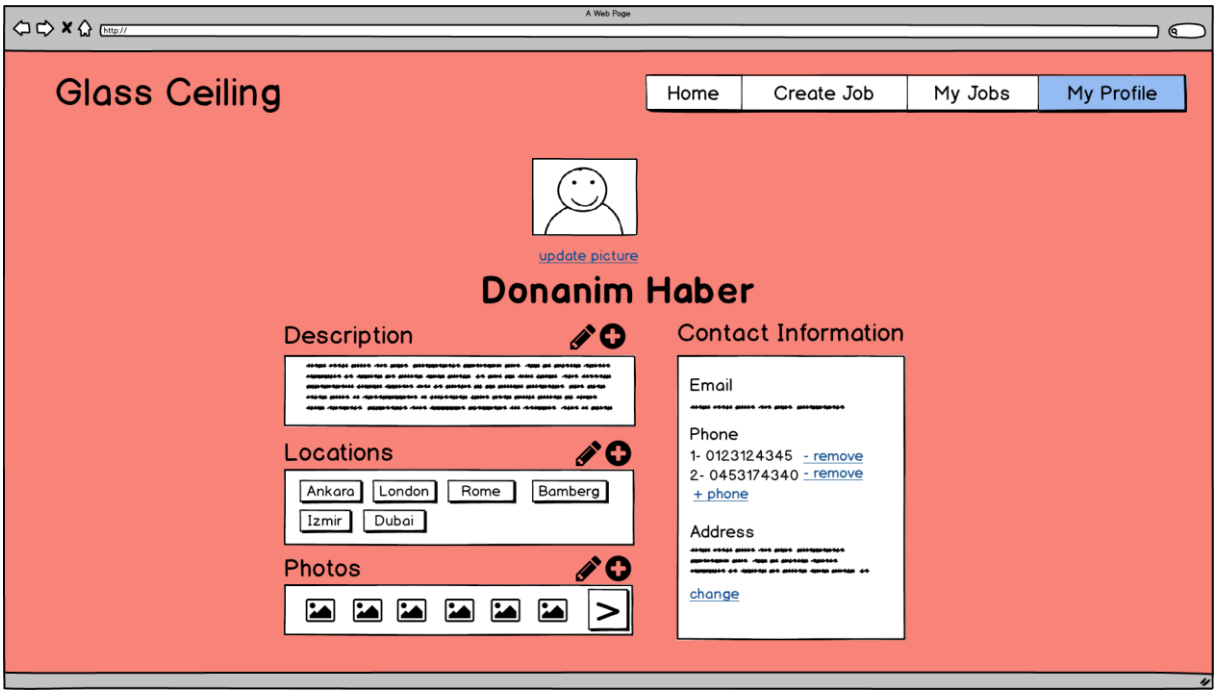
Process: On this screen, work user can see her/his applied job offerings. Also, can find links to navigate to the company's profile or solve an application test if available.

SQL Statements:

For view:

```
SELECT *
FROM applies natural join job_offerings
WHERE user_id = @user_id
```

4.7. My Profile Page for Company User



Inputs: @phone_no, @picture_link, @location, @description,@address,@interest,@photo_link

Process: This is the page where a company user can look at the company profile details and update profile information. A logged in company user can access this page through the top navigation bar by clicking on the my profile button from any screen.

SQL Statements:

On update picture pressed:

UPDATE user

SET picture_link = @picture_link

WHERE user_id = @user_id

On remove phone number pressed:

UPDATE user

SET phone_no = @phone_no

WHERE user_id = @user_id

On add phone number pressed:

```
INSERT INTO user(phone_no)
VALUES(@phone_no)
```

On change address pressed:

```
UPDATE user
SET address = @address
WHERE user_id = @user_id
```

On change description pressed:

```
UPDATE comp_user
SET description = @description
WHERE user_id = @user_id
```

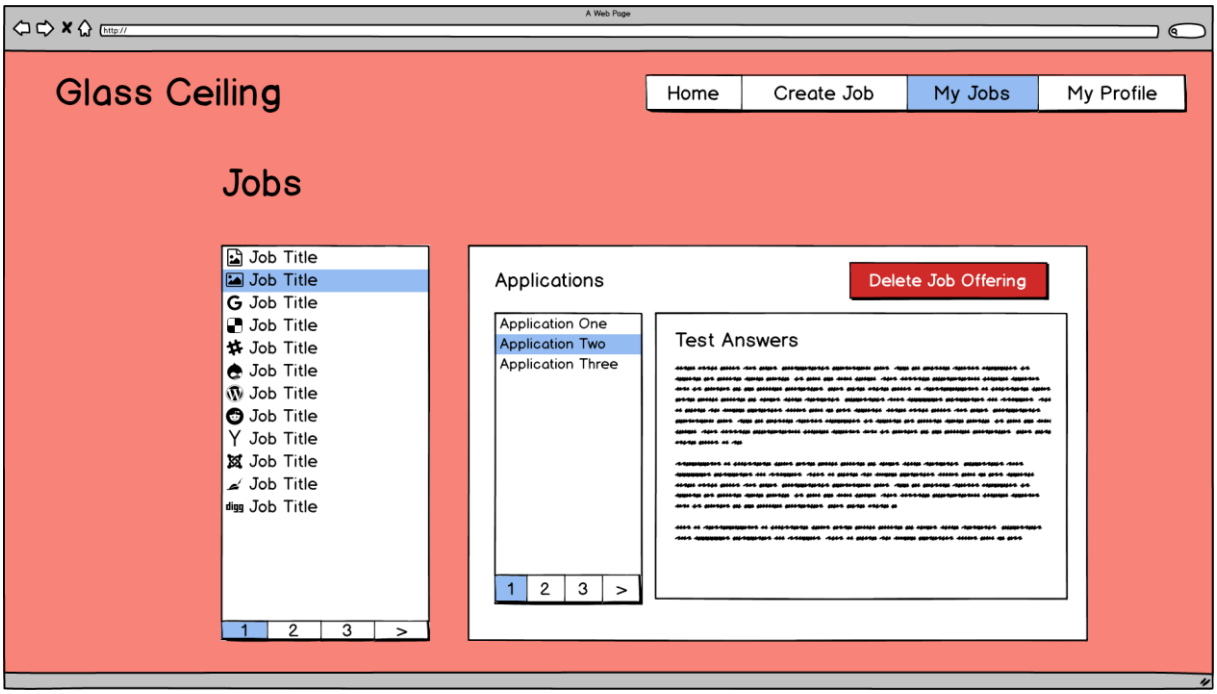
On change locations pressed:

```
UPDATE comp_user
SET location = @location
WHERE user_id = @user_id
```

On change photos pressed:

```
INSERT INTO picture(user_id, picture_link)
VALUES (@user_id,@photo_link)
```

4.8. My Jobs Page for Company User



Inputs: -

Process: On this screen, company user views applications to their job offerings and removes job offerings if needed.

SQL Statements:

For view:

SELECT *

FROM job_offering natural join posts

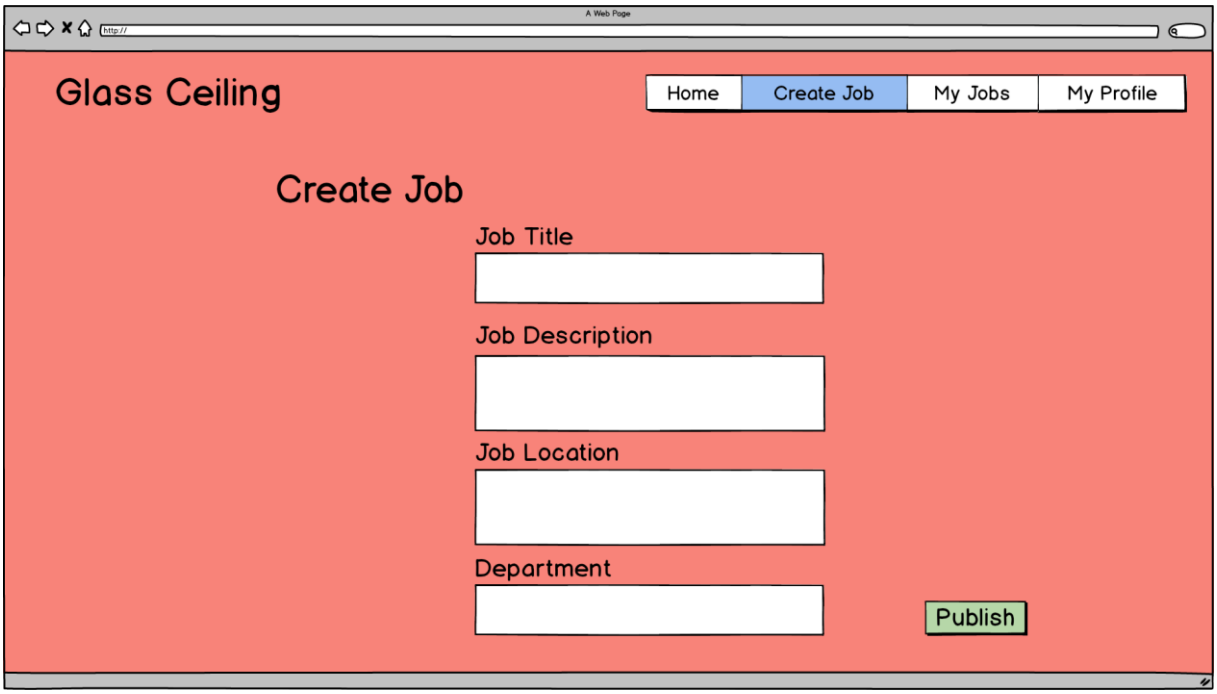
WHERE user_id = @user_id

On delete:

DELETE FROM job_offerings

WHERE offering_id = @offering_id

4.9. Create Job Page for Company User



Input: @job_title, @job_dept, @job_location, @details

Process: Company user can create a new job offering from this screen

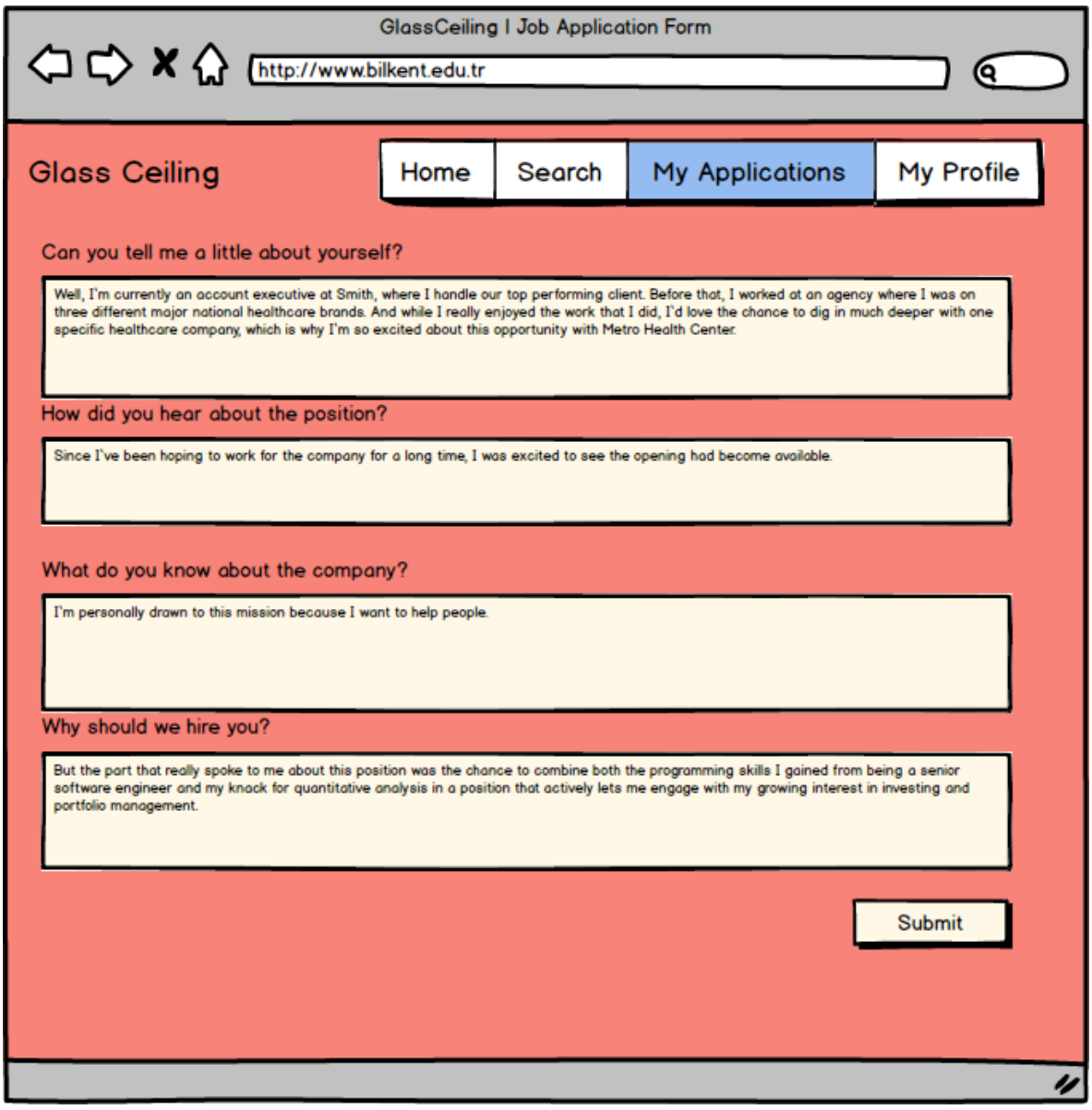
SQL Statements:

On publish:

Insert into job_offerings

Values(@offering_id, @date, @job_title, @job_dept, @job_location, @details)

4.10. Fill Job Application Form for Work User



Input: @answers

Process: While applying to a job offering, the job application form for the job will be queried and shown to the users. His answers will be in string named as answers.

View Application form questions:

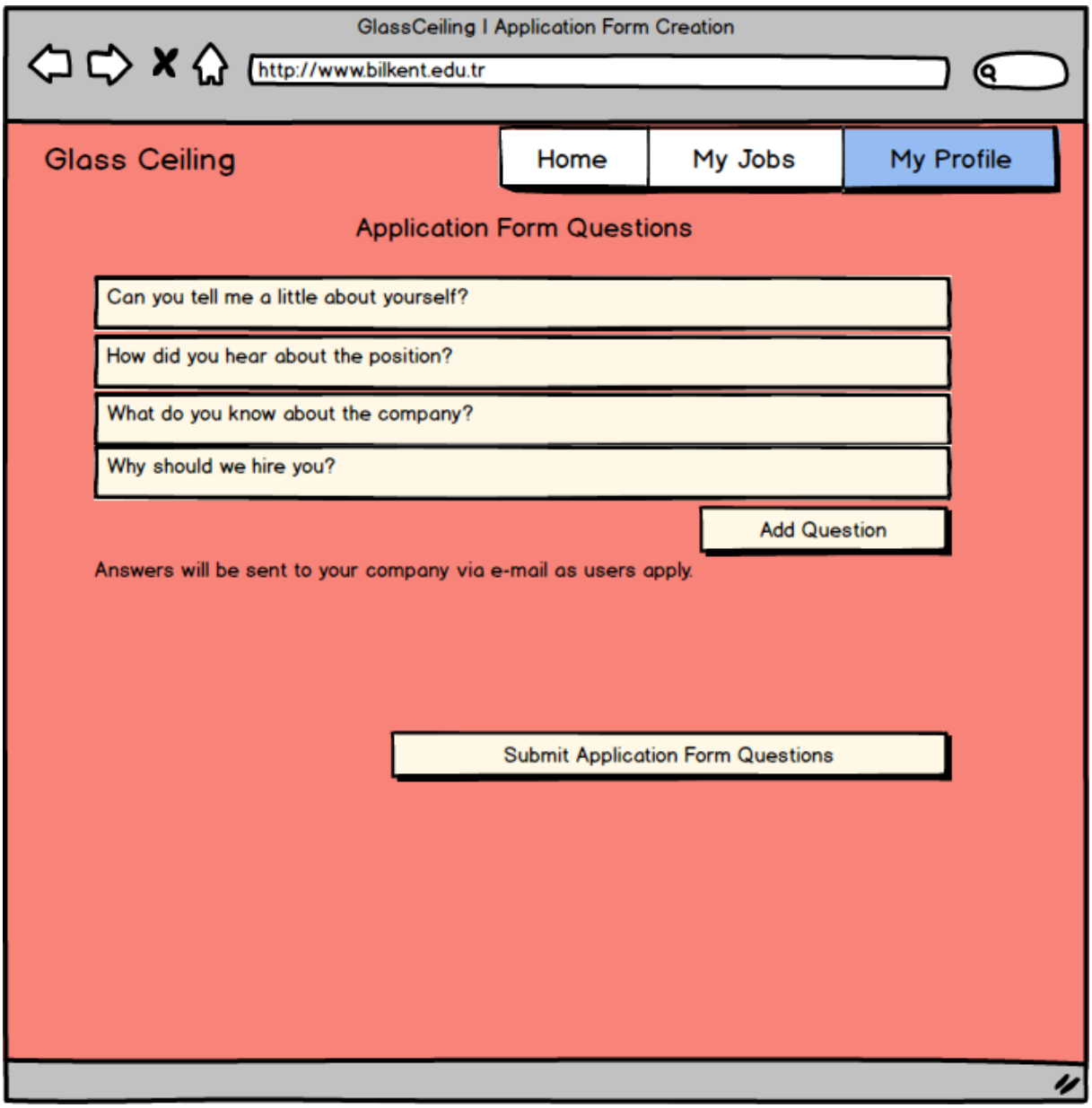
```
SELECT test_id, questions
FROM has_test NATURAL JOIN application_test
WHERE offering_id = @offering_id
```

On submit button:

INSERT INTO does

VALUES (@test_id, @user_id, @answers);

4.11. Application Form Creation for Company User



Input: @questions

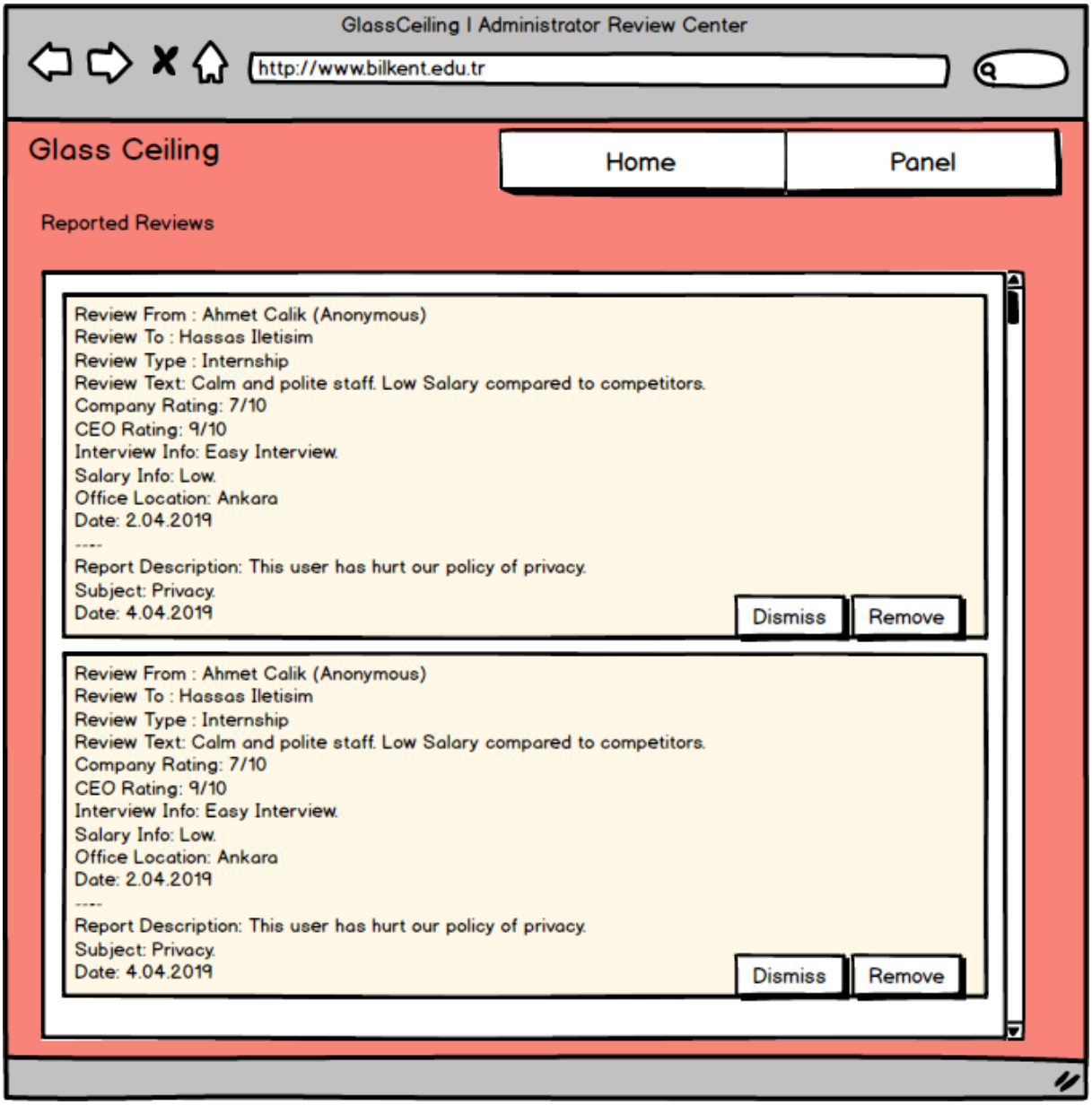
Process: The company user will create a job application form by entering questions in this page. Test ID will be generated by our system.

On Submit Application Form Questions:

INSERT INTO application_test

VALUES (@test_id, @questions);

4.12. Admin Review Center



Process: Only administrator user can access to this panel and the administrator will be able to see the reported reviews and evaluate them.

To view reported reviews:

SELECT *

FROM (checks natural join report natural join has) inner join review on has.review_id = review.review_id

WHERE admin_id= @user_id

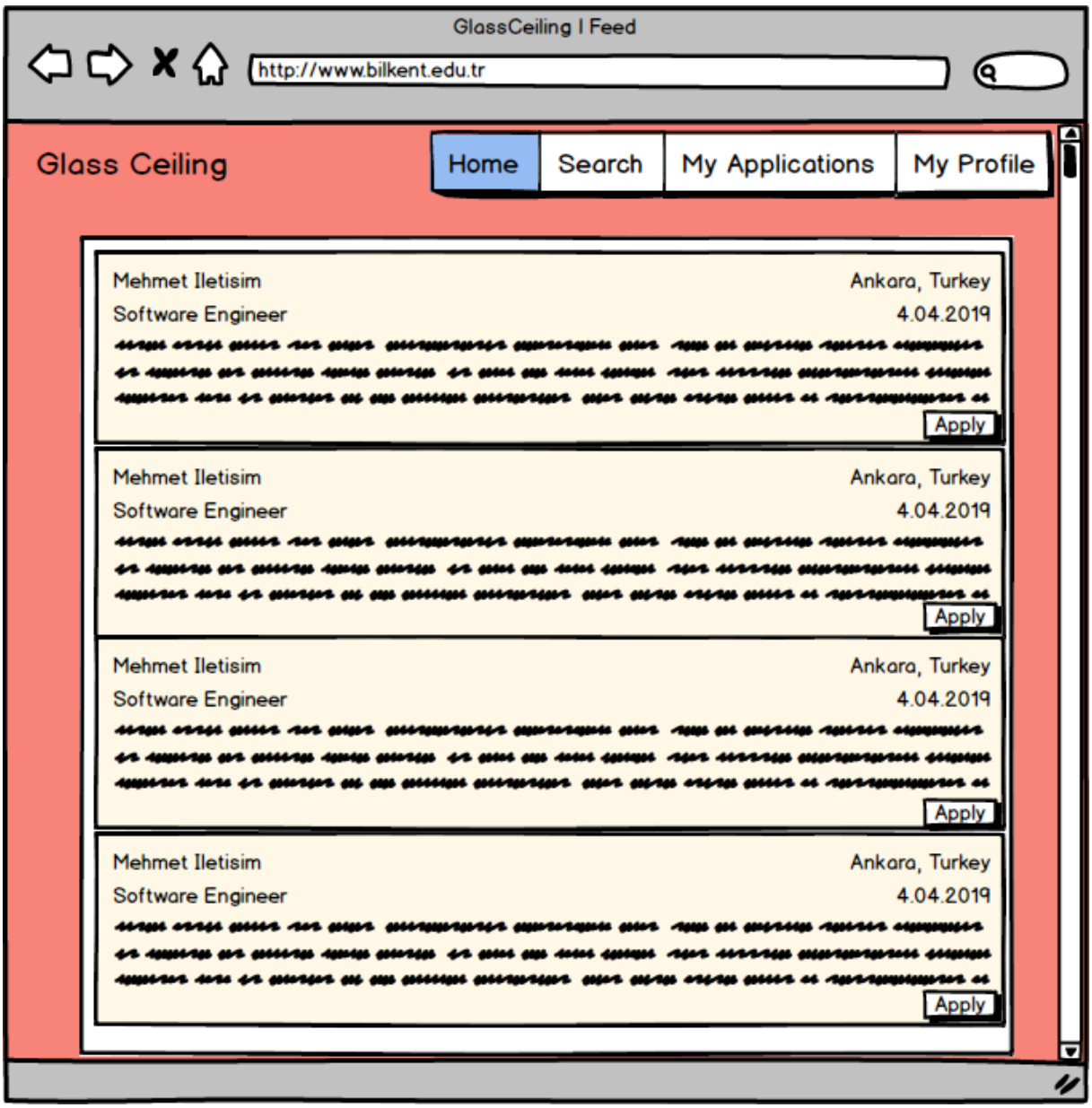
On Dismiss Button:

DELETE FROM report WHERE report_id = @report_id

On Remove Button:

DELETE FROM review WHERE review_id = @review_id

4.13. User Feed



Process: On user feed, job offering from the companies that the user follows will be shown.

View job offerings from followed companies:

SELECT * from (job offering NATURAL JOIN comp_user) NATURAL JOIN follows

WHERE w_id = @user_id

On Apply Button:

INSERT INTO applies

VALUES(@user_id, @offering_id

5. Advanced database components

5.1. Views

Feed_view: This view is used for feed to list the worker user's followed companies'

posted jobs.

CREATE VIEW feed_view(company_name, offering_id, date, job_title, job_dept, office_location, details)

AS SELECT company_name, offering_id, date, job_title, job_dept, office_location, details

from (job offering NATURAL JOIN comp_user) NATURAL JOIN follows

WHERE w_id = @user_id

ORDER BY date DESC;

Applications_view: This view is used to show the applications of a worker user.

CREATE VIEW applications_view(offering_id, date, job_title, job_dept, office_location, details)

AS SELECT offering_id, date, job_title, job_dept, office_location, details

FROM applies natural join job_offerings

WHERE user_id = @user_id

ORDER BY date DESC;

Report_view: This view is used for admins to see reports and subjected reviews.

CREATE VIEW report_view (report_id, description, date, subject, review_id, anonymity, type, review_text, comp_rating, ceo_rating, interview_info, salary_info, date)

AS SELECT report_id, description, date, subject, review_id, anonymity, type, review_text, comp_rating, ceo_rating, interview_info, salary_info, date

FROM (checks natural join report natural join has) inner join review on has.review_id = review.review_id

WHERE admin_id= @user_id

ORDER BY date DESC;

5.2. Triggers

- When the job offerings are added by company user the corresponding relation which is post will be updated.
- When a company is removed from the system all corresponding relations and entities will be updated or removed which are posts, job offerings, has_test, reviews, gets, leaves, reports, pictures.
- When a work user is removed from the system all corresponding relations will be updated or removed which are does, follows, applies,
- When a job offering is removed by a company user the corresponding relations which are posts and applies will be updated.
- When a review is deleted by admin, corresponding relations has, gets and leaves will be updated.

5.3. Constraints

- User entries will be limited by the system after he/she choose the type of the review.
- A company cannot post the same job offering twice.
- User cannot have more than one profile photo.
- Company user cannot report a review more than one time.

5.4. Stored Procedures

We will use stored procedures to make it easier to call common queries.

- A stored procedure with parameters will be created for job searching. The parameters will be job title, job type and job location. When this procedure is executed the jobs that satisfy the criteria will be returned.
- A stored procedure with parameters company name and location will be used when a work user searches for a company.

5.5. Reports

5.5.1. Job Application Report

Company users will be able to see how many new applications they had so far for different job offerings.

```
CREATE VIEW applications_report AS
(
SELECT count(*), offering_id
FROM comp_user NATURAL JOIN posts NATURAL JOIN job_offering NATURAL JOIN (applies NATURAL JOIN work_user)
WHERE user_id = @user_id
GROUP BY (offering_id)
)
```

6. Implementation

- We will use MySQL to implement our database.
- For the user interface and functionalies, we will use HTML, CSS, PHP and JavaScript.

7. Website

https://github.com/ege0zcan/CS353