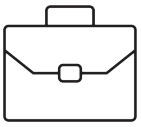
Concordia University

COMP 5531 Files and Databases

Main Project Report
Web Career Portal Database

Team ID:
cxc55311

Team Members:
Md Tanveer Alamgir
ID: 40014877
Craig Boucher
ID: 21295721
Osman Momoh
ID: 26220150
Fan Zou
ID: 40118112



Contents

1 Introduction

This project consists of two main components. The design of the database and the website that is used as a tool to implement the database and showcase various features. The database design consisted of creating an entity relation diagram to abstract the database to the core components necessary. This diagram was then used to project the relations that will be required and their associated attributes. From this schema, the functional dependencies were determined and rigorous normalization was performed to ensure the relations were all in at least third normal form.

The website, known as the web portal, has a front end to display the user interface and a back-end which connects the functionality between the MySQL database and the front end components. A model-view-controller architecture pattern is used to organize the code in the various files used to build the web portal.

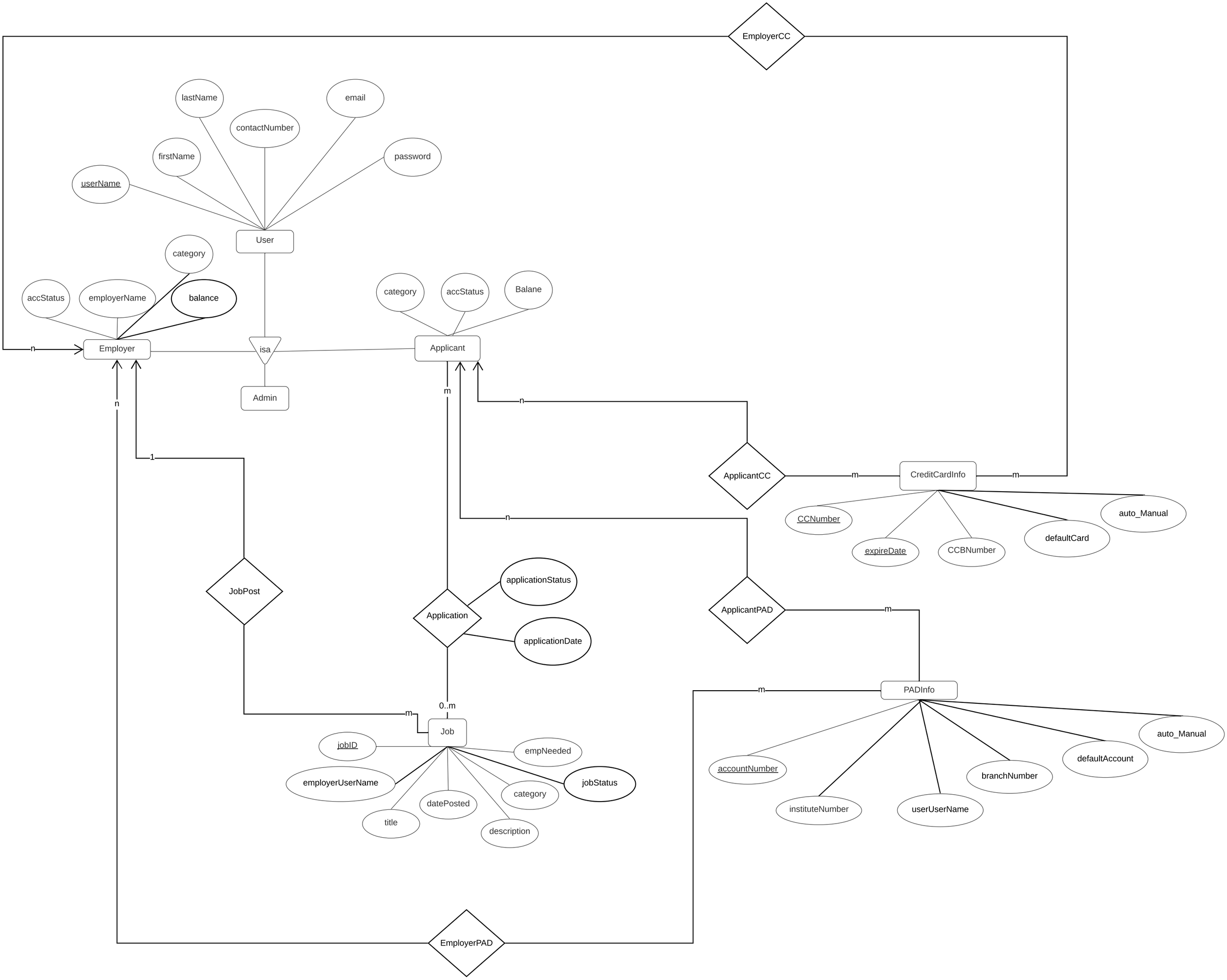
The website itself is a career portal interface. There are three types of accounts that have access to the website. Administrators, who have powers to modify other user accounts. Employers, who can post jobs and offer employment. Lastly, job seekers (also known as applicants), can search for job postings and apply to them. The website also handles payment information and payment processing.

2 Database Design

The reasonable assumptions made based on the requirements provided in the project description include the following. There will be a table for employers that will contain attributes like the name of the employer, their account category, contact information, username, and password. A table for job seekers ('ordinary' users) to hold data regarding their contact information, account balance, category type, username, and password. A table restricted to administrator accounts will also be necessary. These assumptions were used as a starting point to craft an entity relationship model.

2.1 Entity-Relation Diagram

The ER Diagram is on the following page.


EmployerCC
lastName
email
contactNumber
firstName
password
userName
User
category
accStatus
employerName
balance
category
accStatus
Balane
n
Employer
isa
Applicant
Admin
m
n
n
1
ApplicantCC
m
CreditCardInfo
m
CCNumber
expireDate
CCBNumber
defaultCard
auto_Manual
JobPost
applicationStatus
Application
applicationDate
ApplicantPAD
m
m
PADInfo
0..m
m
Job
jobID
empNeeded
employerUserName
jobStatus
datePosted
category
title
description
accountNumber
instituteNumber
userUserName
branchNumber
defaultAccount
auto_Manual
EmployerPAD

2.2 E/R Diagram to Relation Conversions

The database schema is on the following pages.

E/R Diagram to Relational Database Schema

Entity:

User (userName, firstName, lastName, email, contactNumber, password)

Employer (userName, employerName, accStatus, category, balance)

Foreign Key (userName) From entity User (userName)

Applicant (userName, category, accStatus, balance)

Foreign Key (userName) From entity User (userName)

Admin (userName)

Foreign Key (userName) From entity User (userName)

Job (jobID, employerUserName, title, datePosted, description, category, jobStatus, empNeeded)

Foreign Key (employerUserName) From entity Employer (userName)

CreditCardInfo (CCNumber, expireDate, CCBNumber, defaultCard, auto_manual)

PADInfo (accountNumber, instituteNumber, branchNumber, defaultAccount, auto_manual)

Relationship:

Application (applicantUserName, jobID, applicationStatus, applicationDate)

Foreign Key (applicantUserName) From entity Applicant (userName)

Foreign Key (jobID) From entity Job (jobID)

Many-to-Many: An applicant can apply for many jobs, A job can have many application from different applicant.

EmployerCC (employerUserName**,** CCNumber**)**

Foreign Key (employerUserName) From entity Employer (userName)

Foreign Key (CCNumber) From entity CreditCardInfo (CCNumber)

Many-to-One: An employer can have many Credit Cards, A credit card can be associated with only one employer.

EmployerPAD (employerUserName**,** accountNumer**)**

Foreign Key (employerUserName) From entity Employer (userName)

Foreign Key (accountNumer) From entity PADInfo (accountNumber)

Many-to-One: An employer can have many bank account, A bank account will be associated with only one employer.

ApplicantCC (applicantUserName**,** CCNumber**)**

Foreign Key (applicantUserName) From entity Applicant (userName)

Foreign Key (CCNumber) From entity CreditCardInfo (CCNumber)

Many-to-One: An applicant can have many Credit Card, A credit card will be associated with only one applicant.

ApplicantPAD (applicantUserName**,** accountNumer**)**

Foreign Key (applicantUserName) From entity Applicant (userName)

Foreign Key (accountNumer) From entity PADInfo (accountNumber)

Many-to-One: An applicant can have many bank account, A bank account will be associated with only one applicant.

2.3 Functional Dependencies

The following pages contain the normalization process for the relations.

2.3.1 Normalization

Functional Dependencies, Canonical cover & Normalization

- **User** (<u>userName</u>, firstName, lastName, email, contactNumber, password)

Candidate keys:

userName, email.

userName $\subseteq$ User and userName $\rightarrow$ User

email $\subseteq$ User. email $\rightarrow$ User

We defined userName as primary key since email can be changed by a user.

Functional Dependencies:

F = {userName $\rightarrow$ firstName, userName $\rightarrow$ lastName, userName $\rightarrow$ email, userName $\rightarrow$ contactNumber, userName $\rightarrow$ password}

Canonical Cover:

$\Rightarrow$ Step 1: Making right hand side (RHS) a single attribute.

RHS is already a single attribute in all FDs.

$\Rightarrow$ Step 2: Having Left Hand Side in simple form.

userName $\rightarrow$ firstName (No left redundancy)

userName $\rightarrow$ lastName (No left redundancy)

userName $\rightarrow$ email (No left redundancy)

userName $\rightarrow$ contactNumber (No left redundancy)

userName $\rightarrow$ password (No left redundancy)

$\Rightarrow$ Step 3: Remove redundant FDs.

- For: userName $\rightarrow$ firstName

Let G = F – {userName $\rightarrow$ firstName}

G = {userName $\rightarrow$ lastName, userName $\rightarrow$ email, userName $\rightarrow$ contactNumber, userName $\rightarrow$ password}

$userName^{+}_{G}$ = {lastName, email, contactNumber, password}

Since firstName $\notin$ $userName^{+}_{G}$ so userName $\rightarrow$ firstName is not redundant.

- For: userName $\rightarrow$ lastName

Let G = F – {userName $\rightarrow$ lastName}

G = {userName $\rightarrow$ firstName, userName $\rightarrow$ email, userName $\rightarrow$ contactNumber, userName $\rightarrow$ password}

$userName^{+}_{G}$ = {firstName, email, contactNumber, password}

Since lastName $\notin$ $userName^{+}_{G}$ so userName $\rightarrow$ lastName is not redundant.

- For: userName $\rightarrow$ email

Let G = F – {userName $\rightarrow$ email}

G = {userName $\rightarrow$ firstName, userName $\rightarrow$ lastName, userName $\rightarrow$ contactNumber, userName $\rightarrow$ password}

$userName^{+}_{G}$ = {firstName, lastName, contactNumber, password}

Since email $\notin$ $userName^{+}_{G}$ so userName $\rightarrow$ email is not redundant.

- For: userName $\rightarrow$ contactNumber

Let G = F – {userName $\rightarrow$ contactNumber}

G = {userName $\rightarrow$ firstName, userName $\rightarrow$ lastName, userName $\rightarrow$ email, userName $\rightarrow$ password}

$userName^{+}_{G}$ = {firstName, lastName, email, password}

Since contactNumber $\notin$ $userName^{+}_{G}$ so userName $\rightarrow$ contactNumber is not redundant.

- For: userName $\rightarrow$ password

Let G = F – {userName $\rightarrow$ password}

G = {userName $\rightarrow$ firstName, userName $\rightarrow$ lastName, userName $\rightarrow$ email, userName $\rightarrow$ contactNumber}

$userName^{+}_{G}$ = {firstName, lastName, email, contactNumber}

Since password $\notin$ $userName^{+}_{G}$ so userName $\rightarrow$ password is not redundant.

Since there is no redundant Functional Dependencies in F so F is a Canonical Cover of itself.

Normalization

The primary key of User is "userName".

F = {userName $\rightarrow$ firstName, userName $\rightarrow$ lastName, userName $\rightarrow$ email, userName $\rightarrow$ contactNumber, userName $\rightarrow$ password}

Since LHS of all FDs in F is the primary key so User is in BCNF.

Summary

Primary Key: userName

Functional Dependencies: {userName $\rightarrow$ firstName, userName $\rightarrow$ lastName, userName $\rightarrow$ email, userName $\rightarrow$ contactNumber, userName $\rightarrow$ password}

Canonical Cover: {userName $\rightarrow$ firstName, userName $\rightarrow$ lastName, userName $\rightarrow$ email, userName $\rightarrow$ contactNumber, userName $\rightarrow$ password}

Normalization: BCNF

- **Employer** (<u>userName</u>, employerName, accStatus, category, balance)

Candidate key:

userName

userName $\subseteq$ Employer and userName $\rightarrow$ Employer

Functional Dependencies:

F = {userName $\rightarrow$ employerName, userName $\rightarrow$ accStatus, username $\rightarrow$ category, userName $\rightarrow$ balance}

Canonical Cover:

$\Rightarrow$ Step 1: Making RHS single attribute.

All the FDs have single attribute in the RHS.

$\Rightarrow$ Step 2: Having LHS in simple form.

{userName $\rightarrow$ employerName} (No left redundancy)

{userName $\rightarrow$ accStatus} (No left redundancy)

{userName $\rightarrow$ category} (No left redundancy)

{userName $\rightarrow$ balance} (No left redundancy)

$\Rightarrow$ Step 3: Removing redundant FDs

- For: userName $\rightarrow$ employerName

Let G = F – {userName $\rightarrow$ employerName}

G = {userName $\rightarrow$ accStatus, userName $\rightarrow$ category, userName $\rightarrow$ balance}

$userName^{+}_{G}$ = {accStatus, category, balance}

Since employerName $\notin$ $userName^{+}_{G}$ so userName $\rightarrow$ employer is not redundant.

- For: userName $\rightarrow$ accStatus

Let G = F – {userName $\rightarrow$ accStatus}

G = {userName $\rightarrow$ employerName, userName $\rightarrow$ category, userName $\rightarrow$ balance}

$userName^{+}_{G}$ = {employerName, category, balance}

Since accStatus $\notin$ $userName^{+}_{G}$ so userName $\rightarrow$ accStatus is not redundant.

- For: userName $\rightarrow$ category

Let G = F – {userName $\rightarrow$ category}

G = {userName $\rightarrow$ employerName, userName $\rightarrow$ accStatus, userName $\rightarrow$ balance}

$userName^{+}_{G}$ = {employerName, accStatus, balance}

Since category $\notin$ $userName^{+}_{G}$ so userName $\rightarrow$ category is not redundant.

- For: userName $\rightarrow$ balance

Let G = F – {userName $\rightarrow$ balance}

G = {userName $\rightarrow$ employerName, userName $\rightarrow$ accStatus}

$userName^{+}_{G}$ = {employerName, accStatus}

Since balance $\notin$ $userName^{+}_{G}$ so userName $\rightarrow$ balance is not redundant.

There are no redundant functional dependencies in F. So F is a canonical cover of itself.

Normalization

Primary Key: userName

F = {userName $\rightarrow$ employerName, userName $\rightarrow$ accStatus, userName $\rightarrow$ category, userName $\rightarrow$ balance}

Since LHS of all FD's in F is the primary key so Employer is in BCNF.

Summary

Primary Key: userName

Functional Dependencies: {userName $\rightarrow$ employerName, userName $\rightarrow$ accStatus, userName $\rightarrow$ category, userName $\rightarrow$ balance}

Canonical Cover: {userName $\rightarrow$ employerName, userName $\rightarrow$ accStatus, userName $\rightarrow$ category, userName $\rightarrow$ balance}

Normalization: BCNF

- **Applicant** (userName, category, accStatus, balance)

Candidate key:

userName

userName $\subseteq$ Applicant and userName $\rightarrow$ Applicant

Functional Dependencies:

F = {userName $\rightarrow$ category, userName $\rightarrow$ accStatus, userName $\rightarrow$ balance}

Canonical Cover:

$\Rightarrow$ Step 1: Making RHS single attribute.

All the FDs have single attribute in the RHS.

$\Rightarrow$ Step 2: Having LHS in simple form.

{userName $\rightarrow$ category} (No left redundancy)

{userName $\rightarrow$ accStatus} (No left redundancy)

{userName $\rightarrow$ balance} (No left redundancy)

$\Rightarrow$ Step 3: Removing redundant FDs

- For: userName $\rightarrow$ category

Let G = F – {userName $\rightarrow$ category}

G = {userName $\rightarrow$ accStatus, userName $\rightarrow$ balance}

$userName^{+}_{G}$ = {accStatus, balance}

Since category $\notin$ $userName^{+}_{G}$ so userName $\rightarrow$ category is not redundant.

- For: userName $\rightarrow$ accStatus

Let G = F – {userName $\rightarrow$ accStatus}

G = {userName $\rightarrow$ category, userName $\rightarrow$ balance}

$userName^{+}_{G}$ = {category, balance}

Since accStatus $\notin$ $userName^{+}_{G}$ so userName $\rightarrow$ accStatus is not redundant

- For: userName $\rightarrow$ balance

Let G = F – {userName $\rightarrow$ balance}

G = {userName $\rightarrow$ category, userName $\rightarrow$ accStatus}

$userName^{+}_{G}$ = {category, accStatus}

Since balance $\notin$ $userName^{+}_{G}$ so userName $\rightarrow$ balance is not redundant

There are no redundant functional dependencies in F. So F is a canonical cover of itself.

Normalization

Primary Key: userName

F = {userName $\rightarrow$ category , userName $\rightarrow$ accStatus}

Since LHS of all FD's in F is the primary key so Applicant is in BCNF.

Summary

Primary Key: userName

Functional Dependencies: {userName $\rightarrow$ category, userName $\rightarrow$ accStatus}

Canonical Cover: {userName $\rightarrow$ category, userName $\rightarrow$ accStatus}

Normalization: BCNF

- **Admin** (userName)

Candidate Key: userName.

userName $\subseteq$ Admin and userName $\rightarrow$ Admin

Functional Dependencies: F = {}

Canonical Cover: F is a canonical cover of itself.

Normalization: BCNF

Summary

Primary Key: userName

Functional Dependencies: {}

Canonical Cover: {}

Normalization: BCNF

- **Job** (jobID, employerUserName, title, datePosted, description, category, jobStatus, empNeeded)

Candidate key:

jobID

jobID $\subseteq$ Job and jobID $\rightarrow$ Job

Functional Dependencies:

F = {jobID $\rightarrow$ employerUserName, jobID $\rightarrow$ title, jobID $\rightarrow$ datePosted, jobID $\rightarrow$ description, jobID $\rightarrow$ category, jobID $\rightarrow$ jobStatus, jobID $\rightarrow$ empNeeded}

Canonical Cover:

$\Rightarrow$ Step 1: Making RHS single attribute.

All the FDs have single attribute in the RHS.

$\Rightarrow$ Step 2: Having LHS in simple form.

{jobID $\rightarrow$ employerUserName} (No left redundancy)

{jobID $\rightarrow$ title} (No left redundancy)

jobID $\rightarrow$ datePosted} (No left redundancy)

{jobID $\rightarrow$ description} (No left redundancy)

{jobID $\rightarrow$ category} (No left redundancy)

{jobID → jobStatus} (No left redundancy)

{jobID → empNeeded} (No left redundancy)

⇒ Step 3: Removing redundant FDs

- For: jobID → employerUserName

Let G = F – {jobID → employerUserName}

G = {jobID → title, jobID → datePosted, jobID → description, jobID → category, jobID → jobStatus, jobID → empNeeded}

$jobID^+_G$ = {title, datePosted, description, category, jobStatus, empNeeded}

Since employerUserName ∉ $jobID^+_G$ so jobID → employerUserName is not redundant.

- For: jobID → title

Let G = F – {jobID → title}

G = {jobID → employerUserName, jobID → datePosted, jobID → description, jobID → category, jobID → jobStatus, jobID → empNeeded}

$jobID^+_G$ = {employerUserName, datePosted, description, category, jobStatus, empNeeded}

Since title ∉ $jobID^+_G$ so jobID → title is not redundant.

- For: jobID → datePosted

Let G = F – {jobID → datePosted}

G = {jobID → employerUserName, jobID → title, jobID → description, jobID → category, jobID → jobStatus, jobID → empNeeded}

$jobID^+_G$ = {employerUserName, title, description, category, jobStatus, empNeeded}

Since datePosted ∉ $jobID^+_G$ so jobID → datePosted is not redundant.

- For: jobID → description

Let G = F – {jobID → description}

G = {jobID → employerUserName, jobID → title, jobID → datePosted, jobID → category, jobID → jobStatus, jobID → empNeeded}

$jobID^+_G$ = {employerUserName, title, datePosted, category, jobStatus, empNeeded}

Since description $\notin$ jobID^+_G so jobID $\rightarrow$ description is not redundant.

- For: jobID $\rightarrow$ category

Let G = F – {jobID $\rightarrow$ category}

G = {jobID $\rightarrow$ employerUserName, jobID $\rightarrow$ title, jobID $\rightarrow$ datePosted, jobID $\rightarrow$ description, jobID $\rightarrow$ jobStatus, jobID $\rightarrow$ empNeeded}

jobID^+_G = {employerUserName, title, datePosted, description, jobStatus, empNeeded}

Since category $\notin$ jobID^+_G so jobID $\rightarrow$ category is not redundant.

- For: jobID $\rightarrow$ jobStatus

Let G = F – {jobID $\rightarrow$ jobStatus}

G = {jobID $\rightarrow$ employerUserName, jobID $\rightarrow$ title, jobID $\rightarrow$ datePosted, jobID $\rightarrow$ description, jobID $\rightarrow$ category, jobID $\rightarrow$ empNeeded}

jobID^+_G = {employerUserName, title, datePosted, description, category, empNeeded}

Since jobStatus $\notin$ jobID^+_G so jobID $\rightarrow$ jobStatus is not redundant.

- For: jobID $\rightarrow$ empNeeded

Let G = F – {jobID $\rightarrow$ empNeeded}

G = {jobID $\rightarrow$ employerUserName, jobID $\rightarrow$ title, jobID $\rightarrow$ datePosted, jobID $\rightarrow$ description, jobID $\rightarrow$ category, jobID $\rightarrow$ jobStatus}

jobID^+_G = {employerUserName, title, datePosted, description, category, jobStatus}

Since empNeeded $\notin$ jobID^+_G so jobID $\rightarrow$ empNeeded is not redundant.

There are no redundant functional dependencies in F. So F is a canonical cover of itself.

Normalization

Primary Key: jobID

F = {jobID $\rightarrow$ employerUserName, jobID $\rightarrow$ title, jobID $\rightarrow$ datePosted, jobID $\rightarrow$ description, jobID $\rightarrow$ category, jobID $\rightarrow$ jobStatus, jobID $\rightarrow$ empNeeded}

Since LHS of all FD's in F is the primary key so Job is in BCNF.

Summary

Primary Key: jobID

Functional Dependencies: {jobID → employerUserName, jobID → title, jobID → datePosted, jobID → description, jobID → category, jobID → jobStatus, jobID → empNeeded}

Canonical Cover: {jobID → employerUserName, jobID → title, jobID → datePosted, jobID → description, jobID → category, jobID → jobStatus, jobID → empNeeded}

Normalization: BCNF

- **CreditCardInfo** (CCNumber, expireDate, userUserName, CCBNumber, defaultCard, auto_manual)

Candidate key:

{CCNumber, expireDate}

{CCNumber, expireDate} $\subseteq$ CreditCardInfo and {CCNumber, expireDate} → CreditCardInfo

Functional Dependencies:

F = {CCNumber, expireDate → userUserName, CCNumber, expireDate → CCBNumber, expireDate → defaultCard, expireDate → auto_manual}

Canonical Cover:

$\Rightarrow$ Step 1: Making RHS single attribute.

All the FDs have single attribute in the RHS.

$\Rightarrow$ Step 2: Having LHS in simple form.

$CCNumber^{+}$ = CCNumber.

$expireDate^{+}$ = expireDate.

CCNumber, expireDate → userUserName (No left redundancy)

CCNumber, expireDate → CCBNumber (No left redundancy)

CCNumber, expireDate → defaultCard (No left redundancy)

CCNumber, expireDate → auto_manual (No left redundancy)

⇒ Step 3: Removing redundant FDs

- For: CCNumber, expireDate → userUserName

Let G = F – {CCNumber, expireDate → userUserName}

G = {CCNumber, expireDate → CCBNumber, CCNumber, expireDate → defaultCard, CCNumber, expireDate → auto_manual}

$\{CCNumber, expireDate\}^{+}_{G}$ = {CCBNumber, defaultCard, auto_manual}

Since userUserName ∉ $\{CCNumber, expireDate\}^{+}_{G}$ so CCNumber, expireDate → userUserName is not redundant.

- For: CCNumber, expireDate → CCBNumber

Let G = F – {CCNumber, expireDate → CCBNumber}

G = {CCNumber, expireDate → userUserName, CCNumber, expireDate → defaultCard, CCNumber, expireDate → auto_manual}

$\{CCNumber, expireDate\}^{+}_{G}$ = {userUserName, defaultCard, auto_manual}

Since CCBNumber ∉ $\{CCNumber, expireDate\}^{+}_{G}$ so CCNumber, expireDate → CCBNumber is not redundant.

- For: CCNumber, expireDate → defaultCard

Let G = F – {CCNumber, expireDate → defaultCard}

G = {CCNumber, expireDate → userUserName, CCNumber, expireDate → CCBNumber, CCNumber, expireDate → auto_manual}

$\{CCNumber, expireDate\}^{+}_{G}$ = {userUserName, CCBNumber, auto_manual}

Since defaultCard ∉ $\{CCNumber, expireDate\}^{+}_{G}$ so CCNumber, expireDate → defaultCard is not redundant.

- For: CCNumber, expireDate → auto_manual

Let G = F – {CCNumber, expireDate → auto_manual}

G = {CCNumber, expireDate → userUserName, CCNumber, expireDate → CCBNumber, CCNumber, expireDate → defaultCard}

$\{CCNumber, expireDate\}^{+}_{G}$ = {userUserName, CCBNumber, defaultCard}

Since auto_manual $\notin$ {CCNumber, expireDate}$^{+}_{G}$ so CCNumber, expireDate $\rightarrow$ auto_manual is not redundant.

There are no redundant functional dependencies in F. So, F is a canonical cover of itself.

Normalization

Primary Key: CCNumber, expireDate

F = {CCNumber, expireDate $\rightarrow$ userUserName, CCNumber, expireDate $\rightarrow$ CCBNumber, expireDate $\rightarrow$ defaultCard, expireDate $\rightarrow$ auto_manual}

Since LHS of all FD's in F is the primary key so CreditCardInfo is in BCNF.

Summary

Primary Key: CCNumber, expireDate

Functional Dependencies: {CCNumber, expireDate $\rightarrow$ userUserName, CCNumber, expireDate $\rightarrow$ CCBNumber}

Canonical Cover: {CCNumber, expireDate $\rightarrow$ userUserName, CCNumber, expireDate $\rightarrow$ CCBNumber}

Normalization: BCNF

- **PADInfo** (accountNumber, instituteNumber, branchNumber, defaultAccount, auto_manual)

Candidate key:

accountNumber

accountNumber $\subseteq$ PADInfo and accountNumber $\rightarrow$ PADInfo

Functional Dependencies:

F = {accountNumber $\rightarrow$ instituteNumber, accountNumber $\rightarrow$ branchNumber, accountNumber $\rightarrow$ defaultAccount, accountNumber $\rightarrow$ auto_manual}

Canonical Cover:

$\Rightarrow$ Step 1: Making RHS single attribute.

All the FDs have single attribute in the RHS.

$\Rightarrow$ Step 2: Having LHS in simple form.

{accountNumber $\rightarrow$ instituteNumber} (No left redundancy)

{accountNumber $\rightarrow$ branchNumber} (No left redundancy)

{accountNumber $\rightarrow$ defaultAccount} (No left redundancy)

{accountNumber $\rightarrow$ auto_manual} (No left redundancy)

$\Rightarrow$ Step 3: Removing redundant FDs

- For: accountNumber $\rightarrow$ instituteNumber

Let G = F – {accountNumber $\rightarrow$ instituteNumber}

G = {accountNumber $\rightarrow$ userUserName, accountNumber $\rightarrow$ branchNumber, accountNumber $\rightarrow$ defaultAccount, accountNumber $\rightarrow$ auto_manual}

$accountNumber^{+}_{G}$ = {branchNumber, defaultAccount, auto_manual}

Since instituteNumber $\notin$ $accountNumber^{+}_{G}$ so accountNumber $\rightarrow$ instituteNumber is not redundant.

- For: accountNumber $\rightarrow$ branchNumber

Let G = F – {accountNumber $\rightarrow$ branchNumber}

G = accountNumber $\rightarrow$ instituteNumber, accountNumber $\rightarrow$ defaultAccount, accountNumber $\rightarrow$ auto_manual}

$accountNumber^{+}_{G}$ = {instituteNumber, defaultAccount, auto_manual}

Since branchNumber $\notin$ $accountNumber^{+}_{G}$ so accountNumber $\rightarrow$ branchNumber is not redundant.

- For: accountNumber $\rightarrow$ defaultAccount

Let G = F – {accountNumber → defaultAccount }

G = {accountNumber → instituteNumber, accountNumber → branchNumber, accountNumber → auto_manual}

$accountNumber^{+}_{G}$ = {instituteNumber, brachNumber, auto_manual}

Since defaultAccount ∉ $accountNumber^{+}_{G}$ so accountNumber → defaultAccount is not redundant.

- For: accountNumber → auto_manual

Let G = F – {accountNumber → auto_manual}

G = {accountNumber → instituteNumber, accountNumber → branchNumber, accountNumber → defaultAccount}

$accountNumber^{+}_{G}$ = {instituteNumber, brachNumber, defaultAccount}

Since auto_manual ∉ $accountNumber^{+}_{G}$ so accountNumber → auto_manual is not redundant.

There are no redundant FD. So, F is a canonical cover of itself.

Normalization

Primary Key: accountNumber

F = {accountNumber → instituteNumber, accountNumber → branchNumber, accountNumber → defaultAccount, accountNumber → auto_manual}

Since LHS of all FD's in F is the primary key so PADInfo is in BCNF.

Summary

Primary Key: accountNumber

Functional Dependencies: {accountNumber → instituteNumber, accountNumber → branchNumber, accountNumber → defaultAccount, accountNumber → auto_manual}

Canonical Cover: {accountNumber → instituteNumber, accountNumber → branchNumber, accountNumber → defaultAccount, accountNumber → auto_manual}

Normalization: BCNF

- **Application** (<u>applicantUserName</u>, <u>jobID</u>, applicationStatus, applicationDate)

Candidate key:

{applicantUserName, jobID}

{applicantUserName, jobID} $\subseteq$ Apply and {applicantUserName, jobID} $\rightarrow$ Apply

Functional Dependencies:

F = {applicantUserName, jobID $\rightarrow$ applicationStatus, applicantUserName, jobID $\rightarrow$ applicationDate}

Canonical Cover:

$\Rightarrow$ Step 1: Making RHS single attribute.

All the FDs have single attribute in the RHS.

$\Rightarrow$ Step 2: Having LHS in simple form.

applicantUserName $^{+}$ = applicantUserName.

jobID^{+} = jobID.

{applicantUserName, jobID} $\rightarrow$ applicationStatus (No left redundancy)

{applicantUserName, jobID} $\rightarrow$ applicationDate (No left redundancy)

$\Rightarrow$ Step 3: Removing redundant FDs

- For: {applicantUserName, jobID} $\rightarrow$ applicationStatus

Let G = F – {applicantUserName, jobID $\rightarrow$ applicationStatus}

G = {applicantUserName, jobID $\rightarrow$ applicationDate}

{applicantUserName, jobID}$^{+}_{G}$ = {applicationDate}

Since applicationStatus $\notin$ {applicantUserName, jobID}$^{+}_{G}$ so applicantUserName, jobID $\rightarrow$ applicationStatus is not redundant.

- For: {applicantUserName, jobID} $\rightarrow$ applicationDate

Let G = F – {applicantUserName, jobID $\rightarrow$ applicationDate}

G = {applicantUserName, jobID → applicationStatus}

$\{applicantUserName, jobID\}^+_G$ = {applicationStatus}

Since applicationDate ∉ $\{applicantUserName, jobID\}^+_G$ so applicantUserName, jobID → applicationDate is not redundant.

There are no redundant functional dependencies in F. So, F is a canonical cover of itself.

Normalization

Primary Key: applicantUserName, jobID

F = {applicantUserName, jobID → applicationStatus, applicantUserName, jobID → applicationDate}

Since LHS of all FD's in F is the primary key so Apply is in BCNF.

Summary

Primary Key: applicantUserName, jobID

Functional Dependencies: {applicantUserName, jobID → applicationStatus, applicantUserName, jobID → applicationDate}

Canonical Cover: {applicantUserName, jobID → applicationStatus, applicantUserName, jobID → applicationDate}

Normalization: BCNF

- **EmployerCC** (employerUserName**,** CCNumber**)**

Candidate key:

{CCNumber}

{CCNumber} ⊆ EmployerCC and {CCNumber} → EmployerCC

Functional Dependencies:

F = {CCNumber → employerUserName}

Canonical Cover:

$\Rightarrow$ Step 1: Making RHS single attribute.

All the FDs have single attribute in the RHS.

$\Rightarrow$ Step 2: Having LHS in simple form.

LHS of all FDs are in simple form.

$\Rightarrow$ Step 3: Removing redundant FDs

- For: {CCNumber $\rightarrow$ employerUserName}

Let G = F – {CCNumber $\rightarrow$ employerUserName}

G = {}

Since employerUserName $\notin$ {CCNumber}$^{+}_{G}$ so CCNumber $\rightarrow$ employerUserName is not redundant.

Normalization

Primary Key: CCNumber

F = {CCNumber $\rightarrow$ employerUserName}

Since LHS of all FD's in F is the primary key so Apply is in BCNF.

Summary

Primary Key: CCNumber

Functional Dependencies: {CCNumber $\rightarrow$ employerUserName}

Canonical Cover: {CCNumber $\rightarrow$ employerUserName}

Normalization: BCNF

- **EmployerPAD** (employerUserName**,** accountNumber**)**

Candidate key:

{accountNumber}

{accountNumber} $\subseteq$ EmployerPAD and {accountNumber} $\rightarrow$ EmployerPAD

Functional Dependencies:

F = {accountNumber → employerUserName}

Canonical Cover:

⇒ Step 1: Making RHS single attribute.

All the FDs have single attribute in the RHS.

⇒ Step 2: Having LHS in simple form.

LHS of all FDs are in simple form.

⇒ Step 3: Removing redundant FDs

- For: {accountNumber → employerUserName}

Let G = F – {accountNumber → employerUserName}

G = {}

Since employerUserName ∉ $\{accountNumber\}^{+}_{G}$ so accountNumber → employerUserName is not redundant.

Normalization

Primary Key: accountNumber

F = {accountNumber → employerUserName}

Since LHS of all FD's in F is the primary key so Apply is in BCNF.

Summary

Primary Key: accountNumber

Functional Dependencies: {accountNumber → employerUserName}

Canonical Cover: {accountNumber → employerUserName}

Normalization: BCNF

- **ApplicantCC** (applicantUserName**,** CCNumber**)**

Candidate key:

{CCNumber}

{CCNumber} $\subseteq$ ApplicantCC and {CCNumber} $\rightarrow$ ApplicantCC

Functional Dependencies:

F = {CCNumber $\rightarrow$ applicantUserName}

Canonical Cover:

$\Rightarrow$ Step 1: Making RHS single attribute.

All the FDs have single attribute in the RHS.

$\Rightarrow$ Step 2: Having LHS in simple form.

LHS of all FDs are in simple form.

$\Rightarrow$ Step 3: Removing redundant FDs

- For: {CCNumber $\rightarrow$ applicantUserName}

Let G = F – {CCNumber $\rightarrow$ applicantUserName}

G = {}

Since applicantUserName $\notin$ $\{CCNumber\}^{+}_{G}$ so CCNumber $\rightarrow$ applicantUserName is not redundant.

Normalization

Primary Key: CCNumber

F = {CCNumber $\rightarrow$ applicantUserName}

Since LHS of all FD's in F is the primary key so Apply is in BCNF.

Summary

Primary Key: CCNumber

Functional Dependencies: {CCNumber $\rightarrow$ applicantUserName}

Canonical Cover: {CCNumber $\rightarrow$ applicantUserName}

Normalization: BCNF

- **EmployerPAD** (applicantUserName**,** accountNumber**)**

Candidate key:

{accountNumber}

{accountNumber} $\subseteq$ ApplicantPAD and {accountNumber} $\rightarrow$ ApplicantPAD

Functional Dependencies:

F = {accountNumber $\rightarrow$ applicantUserName}

Canonical Cover:

$\Rightarrow$ Step 1: Making RHS single attribute.

All the FDs have single attribute in the RHS.

$\Rightarrow$ Step 2: Having LHS in simple form.

LHS of all FDs are in simple form.

$\Rightarrow$ Step 3: Removing redundant FDs

- For: {accountNumber $\rightarrow$ applicantUserName}

Let G = F – {accountNumber $\rightarrow$ applicantUserName}

G = {}

Since applicantUserName $\notin$ {accountNumber}$^{+}_{G}$ so accountNumber $\rightarrow$ applicantUserName is not redundant.

Normalization

Primary Key: accountNumber

F = {accountNumber $\rightarrow$ applicantUserName}

Since LHS of all FD's in F is the primary key so Apply is in BCNF.

Summary

Primary Key: accountNumber

Functional Dependencies: {accountNumber $\rightarrow$ applicantUserName}

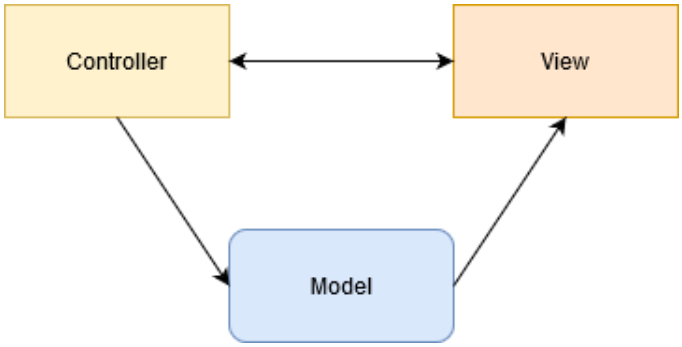
Canonical Cover: {accountNumber → applicantUserName}

Normalization: BCNF

3 Web Portal Functionalities

Technical languages used:
Programming languages: JavaScript, PHP, Python
Markup and Stylesheet Languages: HTML and CSS
Database Management System: MySQL

3.1 Software Design & Architecture



The architectural software design pattern used in the construction of this website is known as Model-View-Controller (MVC). The three interlocking components that perform the core functionality of the website work together to display the interface to the user, gather input from the user, and manage the interaction between the user and the database. The view component displays a graphical interface to the user and receives user input. The controller receives this input and sometimes updates the view directly or manages/manipulates the data requested by the inputs and provides it to the model component. Where the model component, in turn, relays information directly to the view.

3.2 Front-End Development

Bootstrap, a pivotal framework providing JavaScript and CSS templates was used in the constructions of the front-end. The layout of the website and other visual elements were able to be abstracted away with the Bootstrap framework, making working with the markup and style sheet languages much easier. The functionality for the way users interact with the website was programmed with both JavaScript and PHP.

3.2.1 Landing Page

The first page greeting the user is the landing page. Visually, a briefcase image is displayed as a foreground graphic for the website. On the landing page a login request is found. A user can proceed to login, create an account, or request a password if they have forgotten their password.

When choosing to create a new account the user will be shown a form to enter various information. All of the fields shown are required. Personal data such as name, email, address, payment information, etc... Will be used to create the new account. If a field is left with an invalid input, the user will be notified to correct any missing information. There are slightly different fields depending on which account type (employer, job seeker) the user chooses to create.

The missing password functionality simply sends a password to the email address that is stored on the account.

After entering an appropriate username and password the user will be able to login to the website and progress to the dashboard. The dashboard is different for each type of user: job seeker, employer, and administrator (admin).

3.2.2 Dashboards

The two main dashboards with the most users are the employer and job seeker dashboards. They each provide unique functionalities for the way the user interacts with the website. The third user type, administrator, has unique privileges and access to modify the other user accounts directly. An important feature is that if an account has a negative balance then the account becomes frozen and loses access to most features of the website until they provide a payment to obtain a positive balance.

The employer dashboard provides the ability to post jobs, update payment information, hire individuals who applied to job postings, delete/deny the applications of job seekers, upgrade the user category (prime, gold),

view status of jobs that have been applied to, view posted jobs, update their contact information, and modify payment features.

The job seeker dashboard provides similar features as the employer dashboard such as updating payment information, personal information, and upgrading or downgrading user categories (basic, prime, gold). There is the ability to search for all of the jobs posted in the database. Once a desired job is found, the job seeker user can apply to the job. All of the jobs that have been applied to will be able to be viewed for status updates and whether to accept or reject (withdraw from) job offers. Lastly, the user may delete their account if desired.

The last type of account is the administrator. The admins can activate or deactivate any other user (that is not also an admin). Every job post is also able to be viewed by the admin account.

3.3 MySQL Database Implementation

The database management system used is MySQL. A program written in Python was used to generate data to populate the database with. SQL scripts generated by the Python code were then run to create tables and insert the data into the database.

3.3.1 Generating Data

A python program was written to generate the data required to fill the database. Everything from phone numbers and email addresses to user names, company names, and credit card numbers were generated using a series of algorithms. Some of the algorithms draw existing data from text files consisting of names, nouns, adjectives, etc... Various forms of primitive data were taken to be used in the algorithms to be combined into data needed to satisfy the attributes belonging to the relations of the database.

3.3.2 Creating Tables & Inserting Data

The files 'create_tables.sql' and 'insert_data.sql' create the database tables and insert the data using SQL queries. The insert data file is generated by the python program. The create table file was written manually based on the database schema provided by the theoretical design work. The primary and foreign keys along with other constraints are clearly implemented in this file.

3.3.3 SQL Queries

The file that contains the requested queries is appropriately named 'queries.sql'. Some of the queries merely request the manipulation of one record. Such as deleting a user, inserting a user, or inserting one job position. All eighteen queries demanded in the main project requirements documented are listed here and labelled with appropriate comments.

The following queries from the requirements document were hand picked to show exactly five tuples as requested. They were chosen because only these four queries from the document actually demand for a plurality (more than one) tuple.

vi. Report of posted jobs by an employer during a specific period of time (Job title, date posted, short description, of the job up to 50 characters, number of needed employees to the post, number of applied jobs to the post, number of accepted offers).

```
SELECT SoJobs, DatePosted, description, EmpNeeded, EmployerName, NumberHired,
COUNT(SoJobs) as NumberOfApplicants
FROM (SELECT Title as SoJobs, DatePosted, description, EmpNeeded, EmployerName
    FROM employer e join job j on e.UserName = j.EmployerUserName join application
    WHERE EmployerName = 'Ultimate Software' AND (DatePosted BETWEEN '2020-01-26'
    AND '2020-11-08') and
          application.JobID = j.JobID) as
UltimateSoftwareJobs
join (SELECT a.Title as HireTitle, COUNT(ApplicationStatus) AS NumberHired
FROM (SELECT Title FROM
    employer e join job j on e.UserName = j.EmployerUserName join application
    WHERE EmployerName = 'Ultimate Software' AND (DatePosted BETWEEN '2020-01-26'
    AND '2020-11-08') and
          application.JobID = j.JobID GROUP BY Title) as a
LEFT JOIN (SELECT Title, ApplicationStatus FROM employer e join job j on
e.UserName = j.EmployerUserName join application
    WHERE EmployerName = 'Ultimate Software' AND (DatePosted BETWEEN '2020-01-26'
    AND '2020-11-08') and
          application.JobID = j.JobID) as b on a.Title = b.Title and
          b.ApplicationStatus = 'hired' GROUP BY
          a.Title) as
numberOfHires where UltimateSoftwareJobs.SoJobs = numberOfHires.HireTitle
GROUP BY SoJobs;
```

Job_Title	DatePosted	EmpNeeded	EmployerName		
Electronic Wirer	2020-03-27	2	Ultimate Software	0	4
Political Research Scientist	2020-08-13	4	Ultimate Software	0	2
Tile-Molder, Hand	2020-08-08	3	Ultimate Software	3	7
Jet-Piercer Operator	2020-07-13	5	Ultimate Software	2	4
Head of Data	2020-06-17	5	Ultimate Software	1	6

*Note: The job description column is missing from the above table due to not being able to fit on the page. The last two columns are named 'NumberHired' and 'NumberOfApplicants'. The column names were removed due to space restrictions.

xiii. Report of applied jobs by an employee during a specific period of time (Job title, date applied, short description of the job up to 50 characters, status of the application).

```
SELECT job.Title, application.ApplicationDate, job.description,
application.ApplicationStatus
FROM application, job
WHERE (ApplicationDate BETWEEN '2020-01-20' AND '2020-10-15') AND
job.JobID = application.JobID AND
      applicantUserName = 'Bethany_Delena72';
```

Job_Title	DateApplied	ApplicationStatus
Machine Joint Cutter	2020-06-20	denied
Green-Chain Offbearer	2020-08-25	review
Womens Health Care Nurse Practitioner	2020-05-18	sent
Plan Manager	2020-06-19	hired
Fish Skinning Machine Feeder	2020-06-03	review

*Note: The job description column is missing from the above table due to not being able to fit on the page.

xvii. Report of all users by the administrator for employers or employees (Name, email, category, status, balance).

```
SELECT FirstName, LastName, Email, Category, Balance
FROM user natural join applicant
UNION
SELECT FirstName, LastName, Email, Category, Balance
FROM user natural join employer;
```

FirstName	LastName	Email	Category	Balance
Abu	Llewellyn	AbuLlewellyn26@coldmail.com	prime	264.33
Adolfo	Shamus	AdolfoShamus91@hmail.ca	gold	148.68
Aften	Emmaline	AftenEmmaline62@hmail.ca	prime	-55.40
Agatha	Perfecto	AgathaPerfecto64@coldmail.com	prime	189.96
Ajeenah	Niclole	AjeenahNiclole39@coldmail.com	prime	47.13

xviii. Report of all outstanding balance accounts (User name, email, balance, since when the account is suffering).

```
SELECT UserName, email, balance
FROM user
natural join applicant
WHERE balance < 0
UNION
SELECT UserName, email, Balance
from user
natural join employer
WHERE Balance < 0;
```

UserName	Email	Balance
Daymond_Zola41	DaymondZola13@hmail.ca	-60.39
Jolena_Collins97	JolenaCollins79@coldmail.com	-82.09
Lysander_Rasean38	LysanderRasean18@coldmail.com	-65.59
Naquan_Madeleine55	NaquanMadeleine43@coldmail.com	-90.71
Catherina_Darrol69	CatherinaDarrol66@hmail.ca	-62.16

3.4 Back-End Development

The back back-end provides the interlocking mechanism to link the user interface portion of the web portal with the database. The input provided by the users of the website is acquired and used by the functional capabilities of the back end software to issue queries that retrieve, modify, or insert data into the database. The back-end portion of the website is coded in PHP.

3.4.1 Sign Up & Login

The sign up allows someone to create an employer or job seeker type account. Every field needs to be entered. No empty or inappropriately entered fields will be allowed. Invalid data will not be processed and entered into the database when creating a new account.

The login process retrieves the user input from the login fields and verifies the user name and password combination exists in the database. Based on the three different accounts (employer, job seeker, administrator), the appropriate tables are looked at and verified to log into the correct dashboard interface.

3.4.2 Employer Dashboard

This dashboard provides features for an employer type account. Jobs that have been posted may be viewed. The employer may also post new jobs for positions they are seeking to fill. The database is asked to retrieve the list of jobs the employer has posted by associating each job with an employer user name (which is a unique identifier for employers). The employer can post many jobs. Therefore, a job identification number and user name combination may be used to retrieve the appropriate list of jobs for each employer user. The employer may also update payment information and the necessary tables are updated in the database when the employer makes this request.

An employer that is marked as a 'prime' category may post up to five jobs. A 'gold' category has no limit on the amount of jobs they may post. The database keeps track of these values for each account and updates are made accordingly when an employer type account changes their category. When an employer makes a payment the appropriate amount is deducted from their overall balance, which is also kept track of in the database.

If the balance of an account becomes negative then the account is frozen. The database keeps track of every dollar amount in each account and automatically limits the website functionality for negative balance accounts.

These frozen accounts will not have access to any features of the website until enough payments are made to achieve a positive balance.

3.4.3 Job Seeker Dashboard

After verifying the login credentials for a job seeker type account based on the user input the user is greeted with the job seeker dashboard. The main features of this dashboard include a searchable database of all jobs that have been posted and the ability to send an application to these jobs. The user can search by job title and job category. The connection with the database allows the appropriate tables to be retrieved based on similarities with the search terms and what is inside the job table attributes.

The job seeker user may also see a list of jobs they have applied for as when they apply for a job a new job an 'application' is created and inserted into a table containing all jobs that have been applied to. If the user decides to withdraw their application, a deletion query is sent to the database system to remove the application from the table. When a job application is accepted then the user is hired and the application is marked as 'hired' in the database.

There is an option to upgrade a job seeker account category (basic, gold, or prime). The record associated with the user in question is updated as needed. A basic is a free account and may view jobs but not send an application. The prime account may send up to five job applications with a ten dollar monthly charge. The gold account has no limit to the amount of jobs they may send applications to and costs twenty dollars a month. When a payment is processed a query is sent to update the database with a deduction in the account balance.

3.4.4 Administrator Dashboard

The administrator type account has access to a unique dashboard that has the ability to effect other user type accounts. The administrator can also see every job, employer, and job seeker related data available in the database. The administrator can directly deactivate or activate user accounts and retrieve the data associated with accounts. User accounts may also be deleted by the administrator.

When an account is deactivated it can no longer login to the web portal and a warning message will be displayed to the user.

4 CONTRIBUTIONS

The entire project was a group collaboration. Weekly group meetings were held with every group member in attendance. We presented and discussed our work to the group and everyone provided feedback, suggestions, and ideas to implement. The entire main project underwent constant incremental improvement with effort provided from all members.

Database Design
Entity-Relation Diagram: Md Tanveer Alamgir
ER Diagram to Schema Conversion: Md Tanveer Alamgir
Normalization of the Schema: Md Tanveer Alamgir

Front-End Web Portal
Land Page Design & Interaction: Osman Momoh
Dashboard Designs & Interaction: Osman Momoh

MySQL Database Implementation
Data Generation Script:
Design and Construction of Algorithms: Craig Boucher
Code Organization with Object-Oriented Design: Osman Momoh
Table Creation, Data Insertion, and Queries: Craig Boucher

Back-End Web Portal
Sign Up & Login Functionality: Fan Zou
Employer & Job Seeker Dashboard Functionality: Fan Zou
Administrator Dashboard Functionality: Fan Zou

Project Report
Writing, Editing, and Organizing the Report: Craig Boucher