Bilkent University

Department of Computer Engineering

# Design Report

*CS353 DATABASE SYSTEMS*

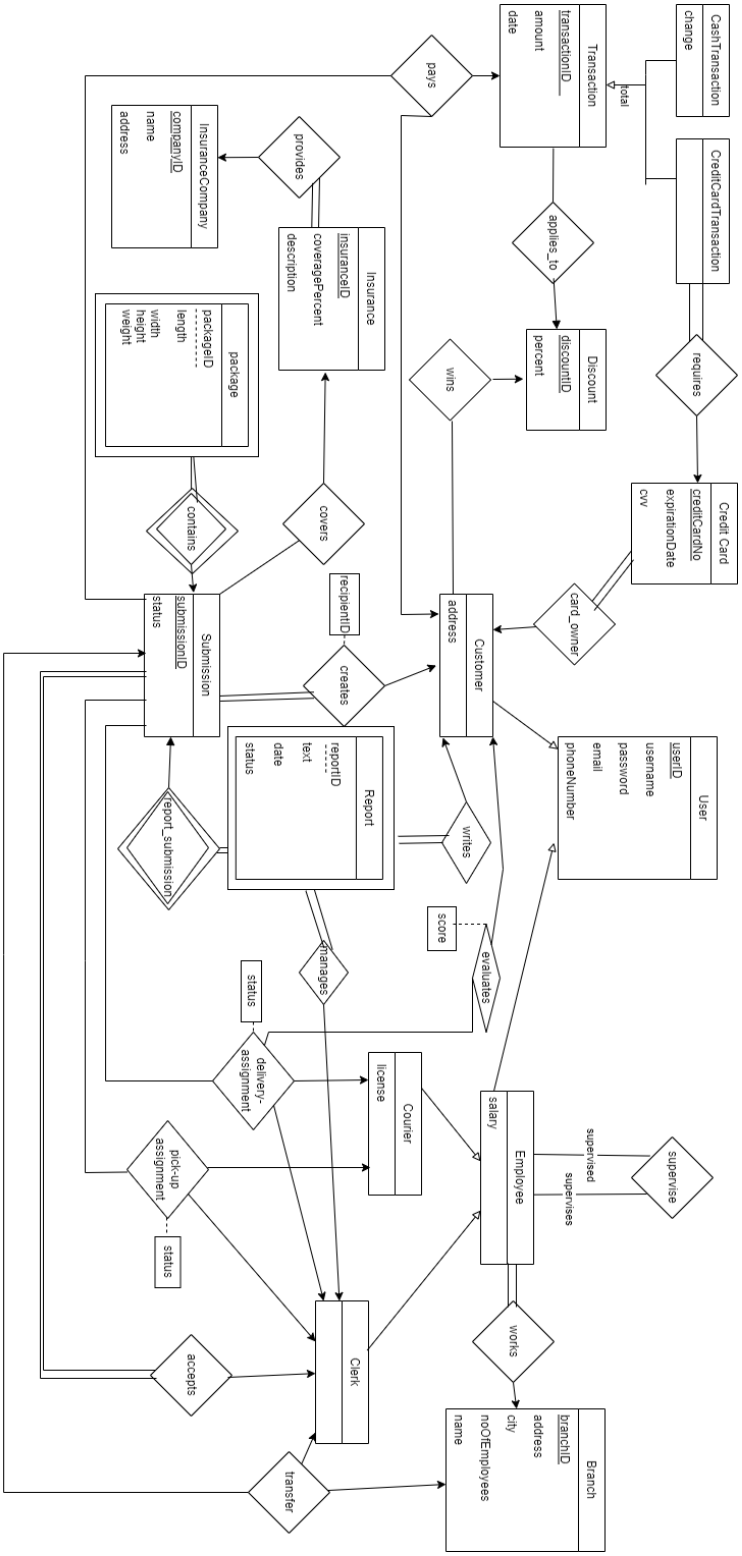
Onat Korkmaz
Muhammed Maruf Şatır
Ümit Çivi
Erdem Ege Eroğlu

November 29, 2021

http://onat.korkmaz.ug.bilkent.edu.tr/courses/cs353.html

# Contents

# 1. Revised ER Diagram



# 2. Relation Schemas

## 2.1 User

**Relational Model:**

user(<u>userID</u>, username, password, email, phone_number)

**Functional Dependencies:**

<u>userID</u> → username, password, email, phone_number

**Candidate Keys:**

{(userID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE user(
        userID int auto_increment,
        username varchar(64) NOT NULL,
        password varchar(64) NOT NULL,
        email       varchar(64) NOT NULL,
        phone_number int,
        primary key(userID)
        );
```

## 2.2 Employee

**Relational Model:**

employee(<u>userID</u>, salary,branchID,managerID)

**Functional Dependencies:**

<u>userID</u> → salary, branchID,managerID

**Candidate Keys:**

{(userID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE user(
        userID int,
        salary INT,
        branchID INT,
        managerID INT,
        primary key(userID),
        foreign key (branchID) references branch
        );
```

## 2.3 Supervise

**Relational Model:**

supervise(<u>supervisorID, superviseID</u>)

**Functional Dependencies:**

**Candidate Keys:**

{(supervisorID, superviseID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE supervise(
        supervisorID int,
        superviseID int,
        foreign key (supervisorID, superviseID ) references user
        );
```

## 2.4 Branch

**Relational Model:**

branch(<u>branchID</u>, name, address, city, noOfEmployees)

**Functional Dependencies:**

branchID → address,city, noOfEmployees, name

address,city → branchID, noOfEmployees, name

**Candidate Keys:**

{(branchID), (address, city)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE branch(
        branchID int auto_increment,
        name varchar(32),
        address varchar(128),
        city varchar(32),
        noOfEmployees int,
        primary key(branchID)
        );
```

## 2.5 Customer

**Relational Model:**

customer(<u>userID</u>, address, discountID)

**Functional Dependencies:**

userID → address, discountID

**Candidate Keys:**

{(userID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE customer(
        userID int,
        address varchar(128),
        discountID int,
        primary key(userID),
        foreign key (discountID) references discount
        );
```

## 2.6 Cash Transaction

**Relational Model:**

cashTransaction(<u>transactionID</u>, change)

**Functional Dependencies:**

transactionID → change

**Candidate Keys:**

{(transactionID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE cashTransaction(
        transactionID int,
        change float,
        primary key(transactionID),
        foreign key (transactionID) references transaction
        );
```

## 2.7 Credit Card Transaction

**Relational Model:**

creditCardTransaction(<u>transactionID</u>, creditCardNo)

**Functional Dependencies:**

<u>transactionID</u> → creditCardNo

**Candidate Keys:**

{(<u>transactionID</u>)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE creditCardTransaction(
        transactionID int,
        creditCardNo int NOT NULL,
        primary key(transactionID),
        foreign key (transactionID) references transaction,
        foreign key (creditCardNo) references creditCard,
        );
```

## 2.8 Credit Card

**Relational Model:**

creditCard(<u>creditCardNo</u>, expirationDate,cvv,customerID)

**Functional Dependencies:**

creditCard → expirationDate,cvv,customerID

**Candidate Keys:**

{(creditCardNo)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE creditCard(
        creditCardNo int,
        expirationDate date NOT NULL,
        cvv int NOT NULL,
        customerID int NOT NULL,
        primary key(creditCardNo),
        foreign key (customerID) references customer
        );
```

## 2.9 Transaction

**Relational Model:**

transaction(<u>transactionID</u>, amount, date, discountID)

**Functional Dependencies:**

transactionID → amount,date,discountID

discountID → transactionID

**Candidate Keys:**

{(transactionID),(discountID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE transaction(
        transactionID int auto_increment,
        amount float,
        date datetime,
        discountID int,
        primary key(transactionID),
        foreign key (discountID) references discount
        );
```

## 2.10 Discount

**Relational Model:**

discount(<u>discountID</u>,percent)

**Functional Dependencies:**

discountID → percent

**Candidate Keys:**

{(discountID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE discount(
        discountID int auto_increment,
        percent float,
        primary key(discountID)
        );
```

## 2.11 Pays

**Relational Model:**

pays(<u>submissionID,transactionID</u>, customerID)

**Functional Dependencies:**

submissionID, transactionID → customerID

submissionID, customerID→ transactionID

**Candidate Keys:**

{(submissionID, transactionID),(submissionID, customerID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE pays(
        transactionID int,
        customerID int,
        submissionID int,
        primary key(submissionID,transactionID),
        foreign key (customerID) references customer
        foreign key (transactionID) references transaction,
        foreign key (submissionID) references submission
        );
```

## 2.12 Submission

**Relational Model:**

Submission(<u>submissionID</u>, senderID, recipientID, insuranceID, clerkID,status)

**Functional Dependencies:**

submissionID → senderID, recipientID, insuranceID, clerkID,status

insuranceID → submissionID

**Candidate Keys:**

{(submissionID),(insuranceID)}

**Normal Form:**

**Table Definition:**

```
CREATE TABLE submission(
        submissionID int auto_increment,
        senderID int,
        recipientID int,
        insuranceID int,
        clerkID int,
        status varchar(128),
        primary key(submissionID),
        foreign key (senderID, recipientID) references customer,
        foreign key (insuranceID) references insurance,
        foreign key (clerkID) references clerk
        );
```

## 2.13 Package

**Relational Model:**

package(<u>packageID</u>, weight, width, length, height, <u>submissionID</u>)

**Functional Dependencies:**

packageID, submissionID → weight, width, length, height

**Candidate Keys:**

{packageID, submissionID}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE package(
        submissionID int,
        packageID int,
        weight float,
        width float,
        length float,
        height float,
        primary key(packageID, submissionID),
        foreign key (submissionID) references submission
        );
```

## 2.14 Insurance

**Relational Model:**

insurance(<u>insuranceID</u>, coveragePercent, description, companyID)

**Functional Dependencies:**

insuranceID → coveragePercent, description, companyID

**Candidate Keys:**

{insuranceID}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE insurance(
        insuranceID int auto_increment,
        coveragePercent float,
        description varchar(128),
        companyID int,
        primary key(insuranceID),
        foreign key (companyID) references insuranceCompany
        );
```

## 2.15 InsuranceCompany

**Relational Model:**

insuranceCompany(c<u>ompanyID</u>, name, address)

**Functional Dependencies:**

companyID → name, address

**Candidate Keys:**

{companyID}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE insuranceCompany(
        companyID int,
        name varchar(64),
        address varchar(128),
        primary key(companyID),
        );
```

## 2.16 Report

**Relational Model:**

report(<u>reportID, submissionID</u>, customerID, clerkID, text, date, status)

**Functional Dependencies:**

reportID, submissionID → customerID, clerkID, text, date, status

**Candidate Keys:**

{reportID, submissionID}

**Normal Form**:

BCNF

**Table Definition:**

```
CREATE TABLE report(
        reportID int auto_increment,
        submissionID int,
        customerID int,
        clerkID int,
        text varchar(1024),
        date datetime,
        status int,
        primary key(reportID, submissionID),
        foreign key (submissionID) references submission,
        foreign key (customerID) references customer,
        foreign key (clerkID) references clerk
        );
```

## 2.17 Transfer

**Relational Model:**

transfer(<u>submissionID</u>, clerkID, <u>branchID</u>)

**Functional Dependencies:**

submissionID, clerkID → branchID

clerkID, branchID → submissionID

submissionID, branchID → clerkID

**Candidate Keys:**

{{submissionID, clerkID}, {submissionID, branchID}, {branchID, clerkID} }

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE transfer(
    submissionID int,
    branchID int,
    clerkID int,
    primary key(branchID, submissionID),
    foreign key (submissionID) references submission,
    foreign key (branchID) references branch,
    foreign key (clerkID) references clerk
    );
```

## 2.18 Evaluates

**Relational Model:**

evaluates(<u>customerID, submissionID,</u> courierID<u>, clerkID,</u> score)

**Functional Dependencies:**

customerID, submissionID, clerkID → score, courierID

**Candidate Keys:**

{(customerID, submissionID, clerkID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE evaluates(
        customerID int,
        submissionID int,
        clerkID int,
        courierID int,
        score int,
        primary key(customerID, submissionID, clerkID),
        foreign key (customerID) references customer,
        foreign key (submissionID) references submission,
        foreign key (clerkID) references clerk,
        foreign key (courierID) references courier
        );
```

## 2.19 Courier

**Relational Model:**

courier(<u>userID</u>, license)

**Functional Dependencies:**

userID → license

**Candidate Keys:**

{(userID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE courier(
        userID int,
        license varbinary(max),
        primary key (userID)
        foreign key (userID) references employee
        );
```

## 2.20 Clerk

**Relational Model:**

clerk(<u>userID</u>)

**Functional Dependencies:**

**Candidate Keys:**

{(userID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE clerk(
        userID int
        primary key (userID)
        foreign key (userID) references employee
        );
```

## 2.21 Pick-up Assignment

**Relational Model:**

pickupassignment(<u>submissionID</u>, clerkID, <u>courierID</u>,status)

**Functional Dependencies:**

courierID, submissionID → clerkID,status

clerkID, submissionID → courierID, status

**Candidate Keys:**

{(courierID, submissionID), (clerkID, submissionID)}

**Normal Form:**

BCNF

**Table Definition:**

```
CREATE TABLE pickupassignment(
        submissionID int,
        clerkID int,
        courierID int,
        status varchar(128),
        primary key (submissionID, courierID),
        foreign key (submissionID) references submission,
        foreign key (clerkID) references clerk,
        foreign key (courierID) references courier
        );
```

## 2.22 Delivery Assignment

**Relational Model:**

deliveryAssignment(<u>clerkID</u>, courierID, <u>submissionID</u>, status)

**Functional Dependencies:**

clerkID,submissionID → courierID,status

submissionID, courierID → clerkID,status

**Candidate Keys:**

{(clerkID,submissionID),(submissionID, courierID)}

**Normal Form:**

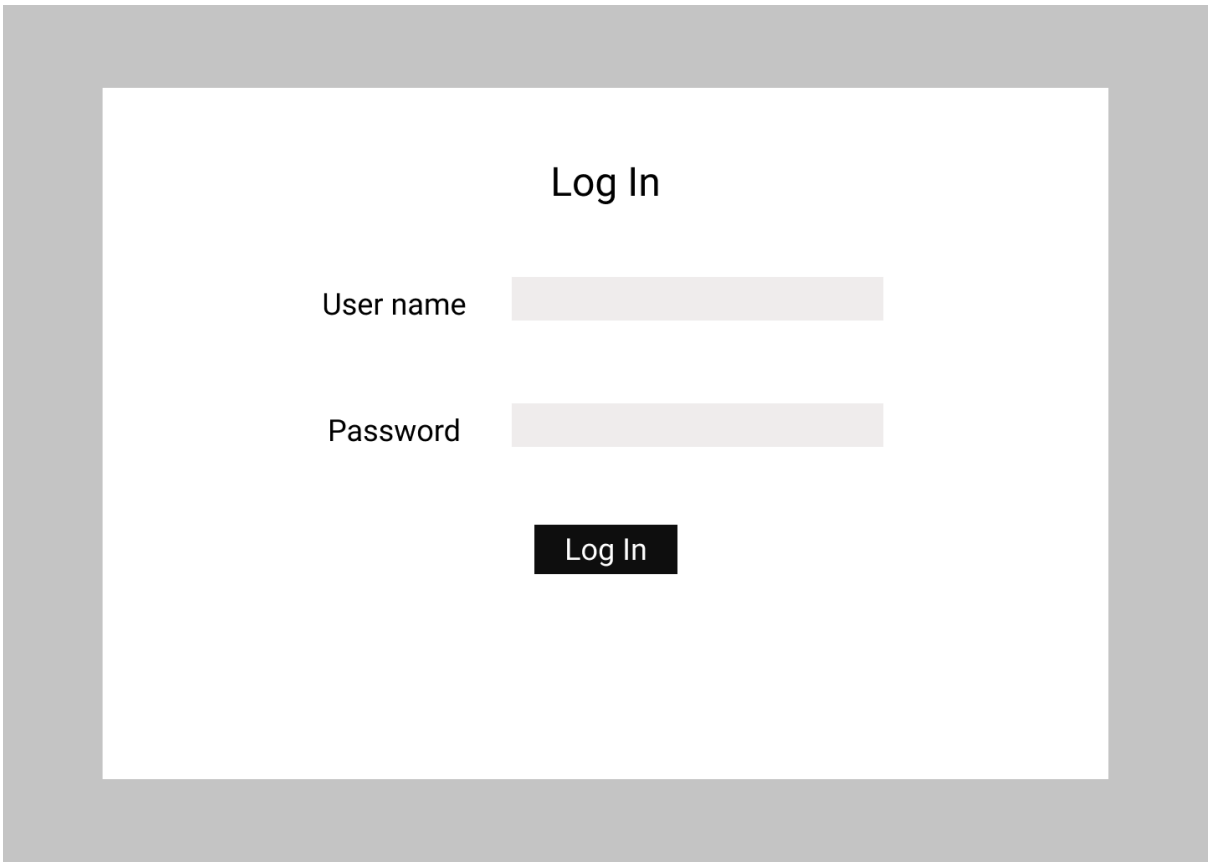
BCNF

**Table Definition:**

```
CREATE TABLE deliveryAssignment(
    clerkID int,
    courierID int,
    submissionID int,
    status varchar(128),
    primary key(submissionID, clerkID),
    foreign key (submissionID) references submission,
    foreign key (clerkID) references clerk,
    foreign key (courierID) references courier
    );
```

# 3. User Interface and Corresponding SQL Statements

## 3.1 Login




**Inputs:** @username, @password

**Process:** Every time a user enters the system, this page shows up. Users can login via this page.

**Sql Statements:**

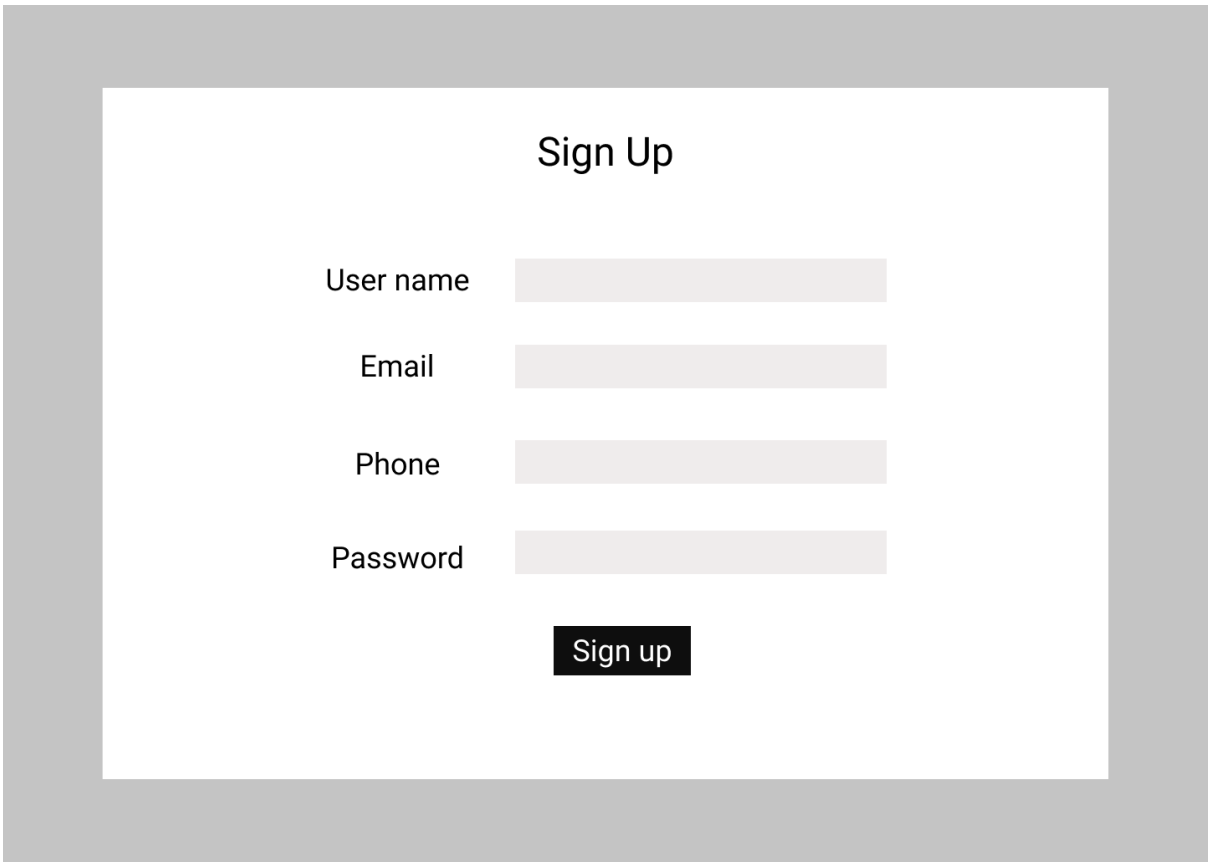
**Start Session:**

Select *

From Customer natural join User

Where username = @username And password = @password

## 3.2 Sign up




**Inputs:** @userName, @email, @phone, @password

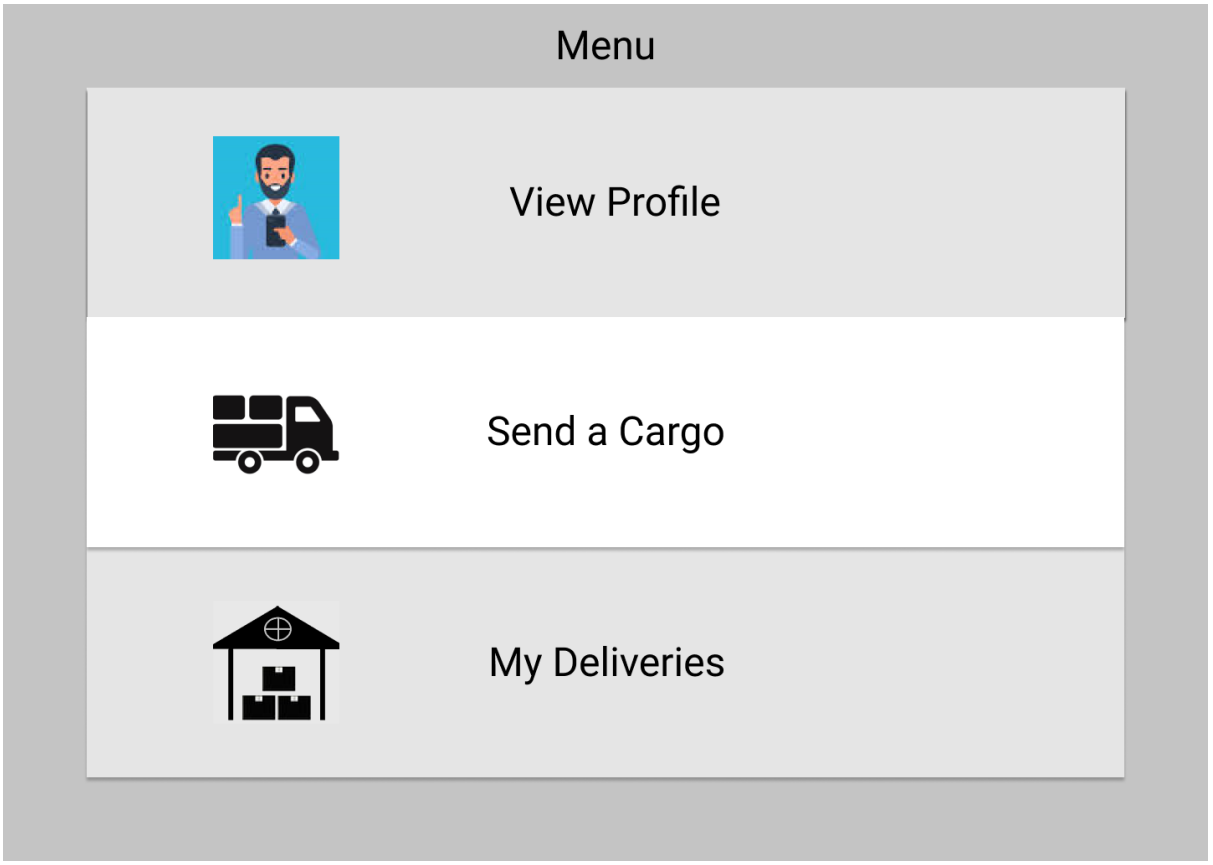
**Process:** This page accepts new users as customers automatically. Because we do not want random people to have access to employee privileges. UserID is an auto-generated variable.

**SQL Statements:**

**Sign up:**

INSERT INTO User(name, password, email, phoneNumber)
VALUES (@userName, @password, @email, @phoneNumber)
INSERT INTO Customer(userID,address,discountID)
VALUES (LAST_INSERT_ID(),null, null)

## 3.3 Menu Page




**Process:** Users can navigate from this page by clicking those options.

## 3.4 View Profile


**Process:** Users must add an address if they are newly registered. Otherwise, users can change the old address with a new one. We already know userID, thus there is no need to ask for an id again.

**SQL Statements:**

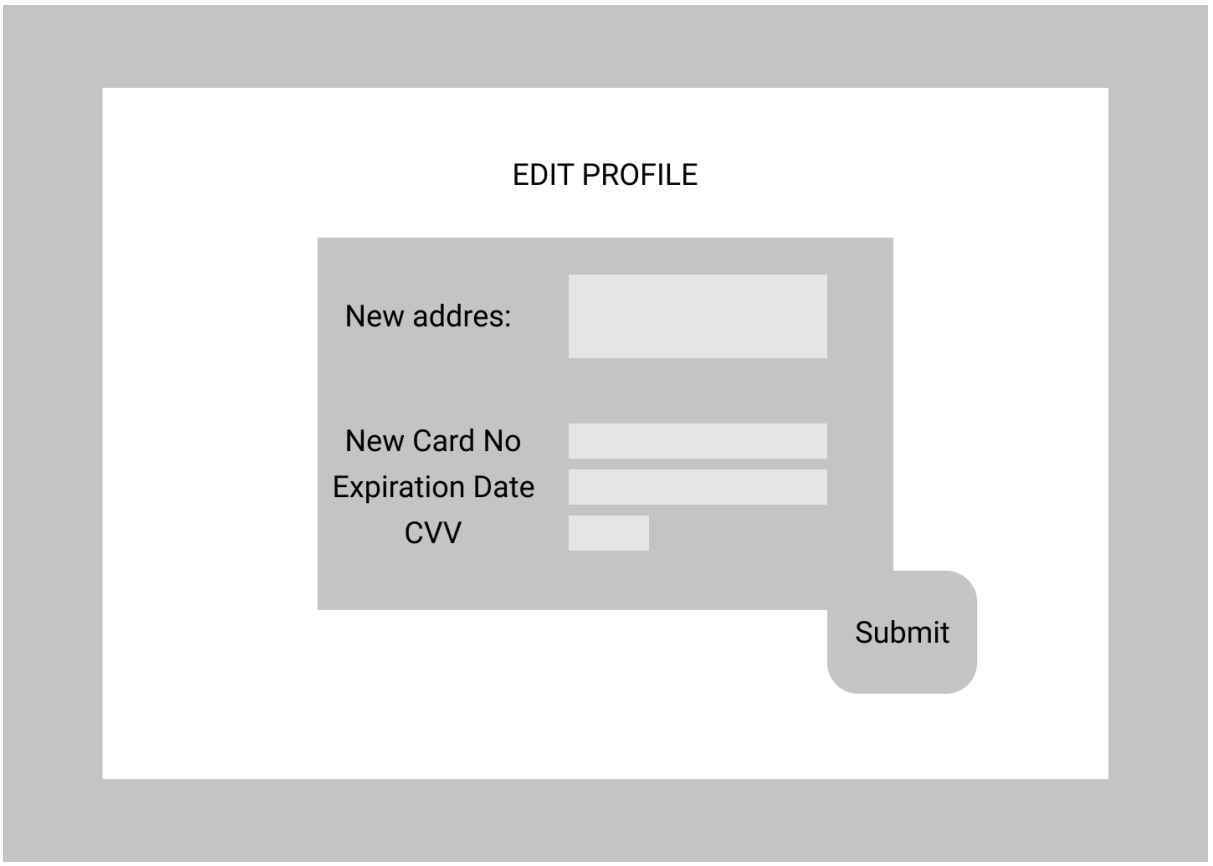
**View the current info:**

Select address, creditCardNo

From Customer natural join CreditCard

Where userID = @UserID

## 3.5 Edit profile




**Inputs:** @address, @creditCardNo, @expirationDate, @cvv

**Process:** Users can update old info or add if there is no record.

**SQL Statements:**

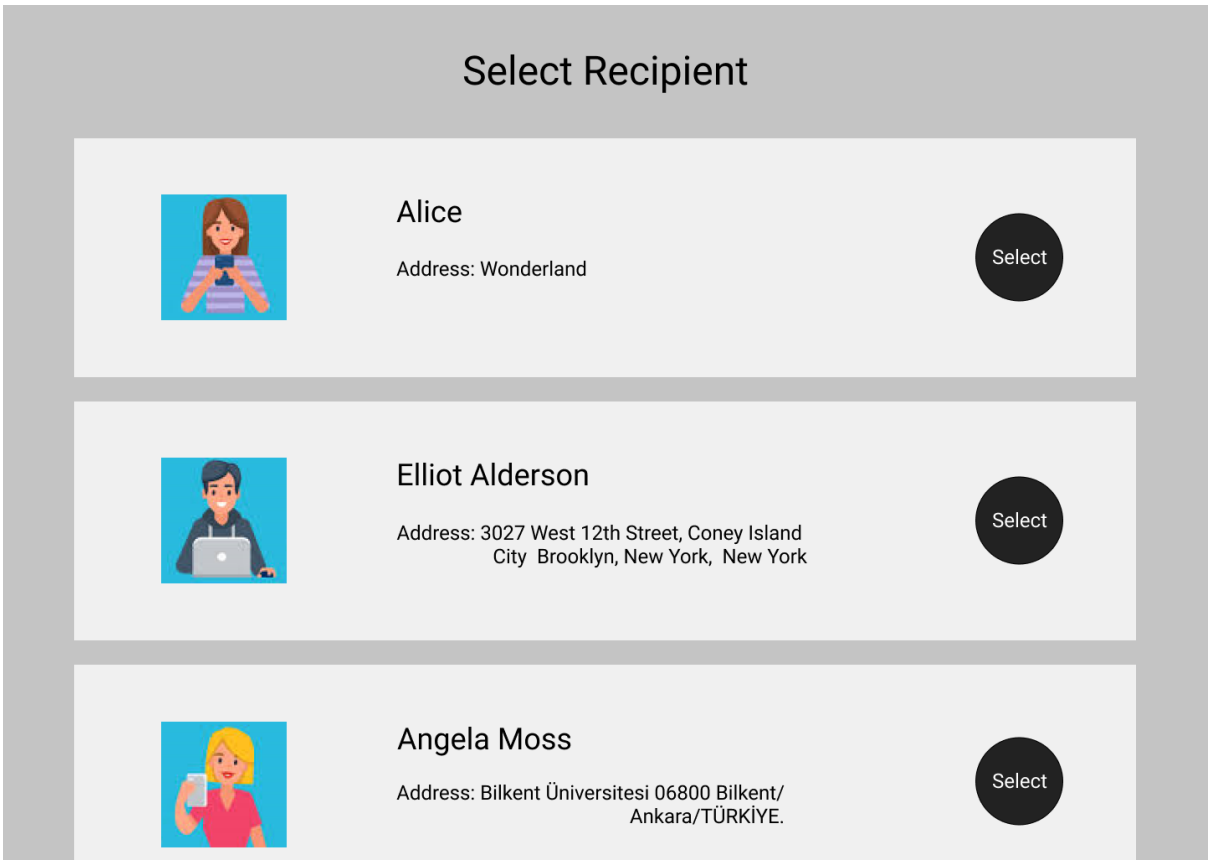
**Change Address:**

Update Customer SET address =@address Where userID=@userID

**Change Card:**

Update CreditCard SET creditCardNo=@creditCardNo AND expirationDate=@expirationDate AND cvv=@cvv Where customerID=@userID

## 3.6 Select Recipient




**Input:** userIDR (for the recipient)

**Process**: In this page users can see all other customers as potential recipients. Then, the user can select a recipient from this list.

**SQL Statements:**

**Selecting:**

Select *

From User natural join Customer

Where userID=@userIDR

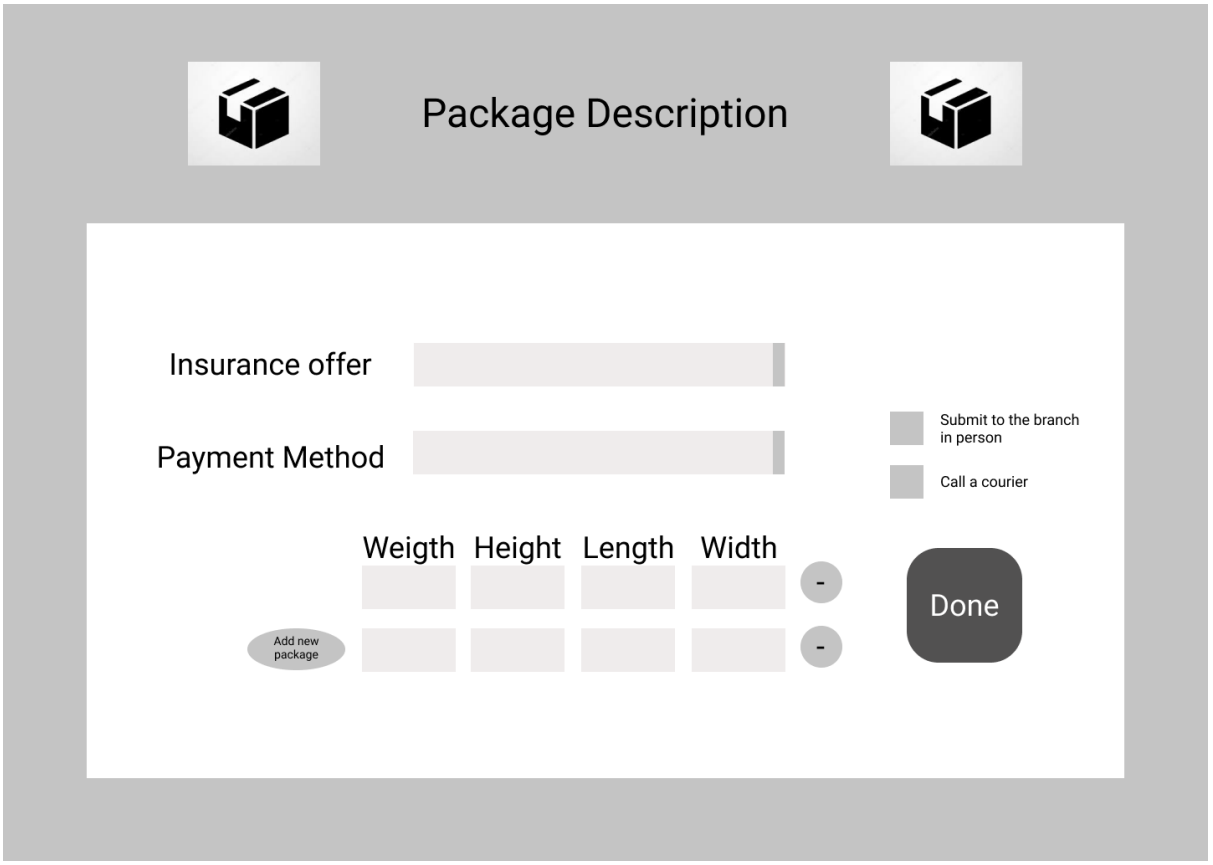
**List of all customers:**

Select userID, name, address

From User natural join Customer

Where userID<>@userID

## 3.7 Package Description




**Input:** @weigth, @length, @width, @insuranceID

**Process:** User gives information about the package. We already have userID and userIDR. The user can select a payment method via combo box. Eventually, the user can call a courier or submit by himself/herself. PackageID is a variable in PHP that is incremented by one after each package insertion. Each package submission is done after the insertion submission.

**SQL Statements:**

**Selecting Insurance:**

Select coveragePercent, description, companyID as insuranceCompany
From Insurance

**Submit for both method:**

INSERT INTO Submission(userID, recipientID, insuranceID, clerkID,status)
VALUES (@userID, @userIDR, @insuranceCompany, null,
'waiting_to_be_approved')
INSERT INTO Package(submissionID, packageID, weight, width, length, height, submissionID)

VALUES (LAST_INSERT_ID(), packageID,@weight, @width, @length, @height)

**Transaction with Cash:**

INSERT INTO Transaction(amount, date, discountID)

VALUES (null, null, null)

INSERT INTO CashTransaction(transactionID, change)

VALUES (@transactionID,null)

**Transaction with Card:**

Select creditCartNo

From CreditCart natural join Customer

INSERT INTO Transaction(amount, date, discountID)

VALUES (null, null, null)

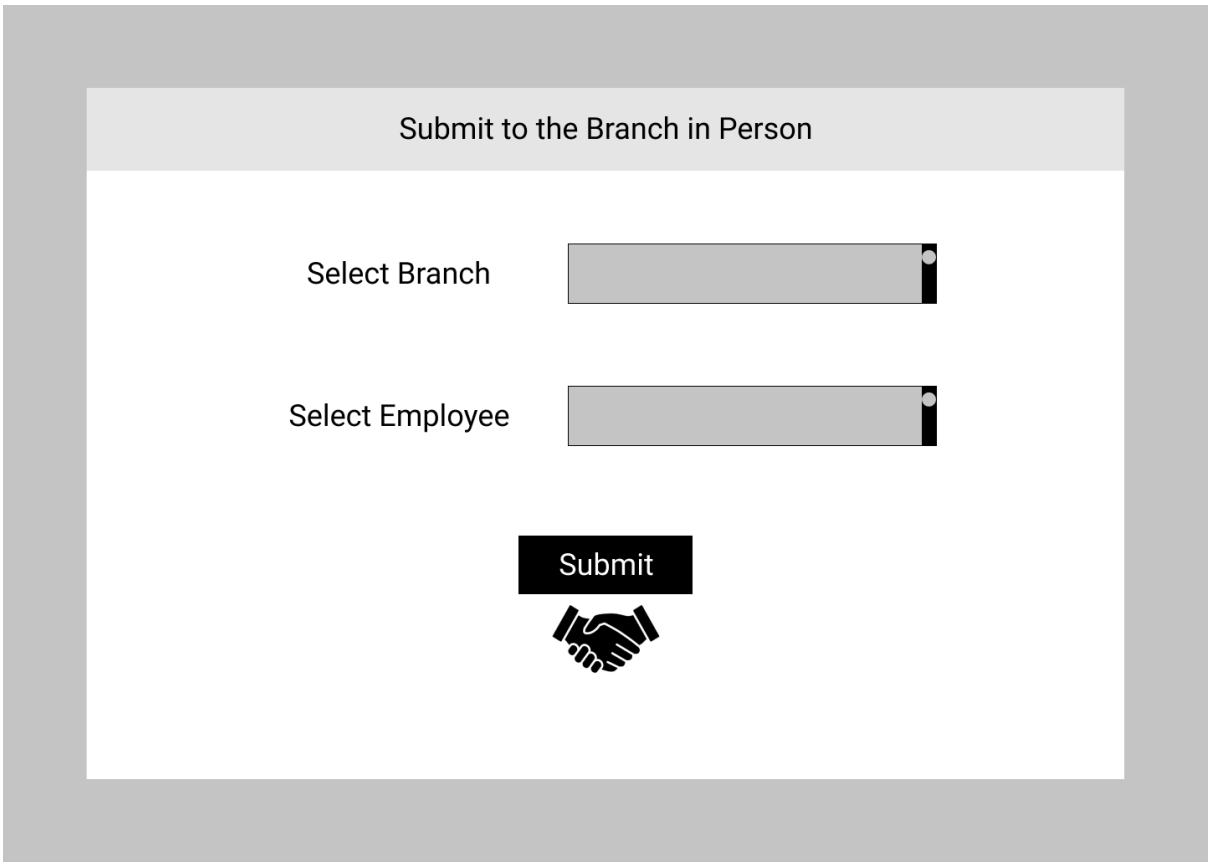
INSERT INTO creditCardTransaction(transactionID, creditCardNo)

VALUES (@transactionID,@creditCartNo)

**Call a courier:** (works only if call a courier check box is filled)

INSERT INTO pickupassignment(submissionID, clerkID, courierID, status) VALUES (@submissionID, null, null, null)

## 3.8 Submit to the Branch in Person




**Input:** @branchID, @name, @userIDC

**Process:** The user selects the desired branch and the clerk from the combo boxes**.**

Desired branch is shown with branchID and name next to each other.

**SQL Statements:**

**Branch Listing:**

Select branchID, name

From branch

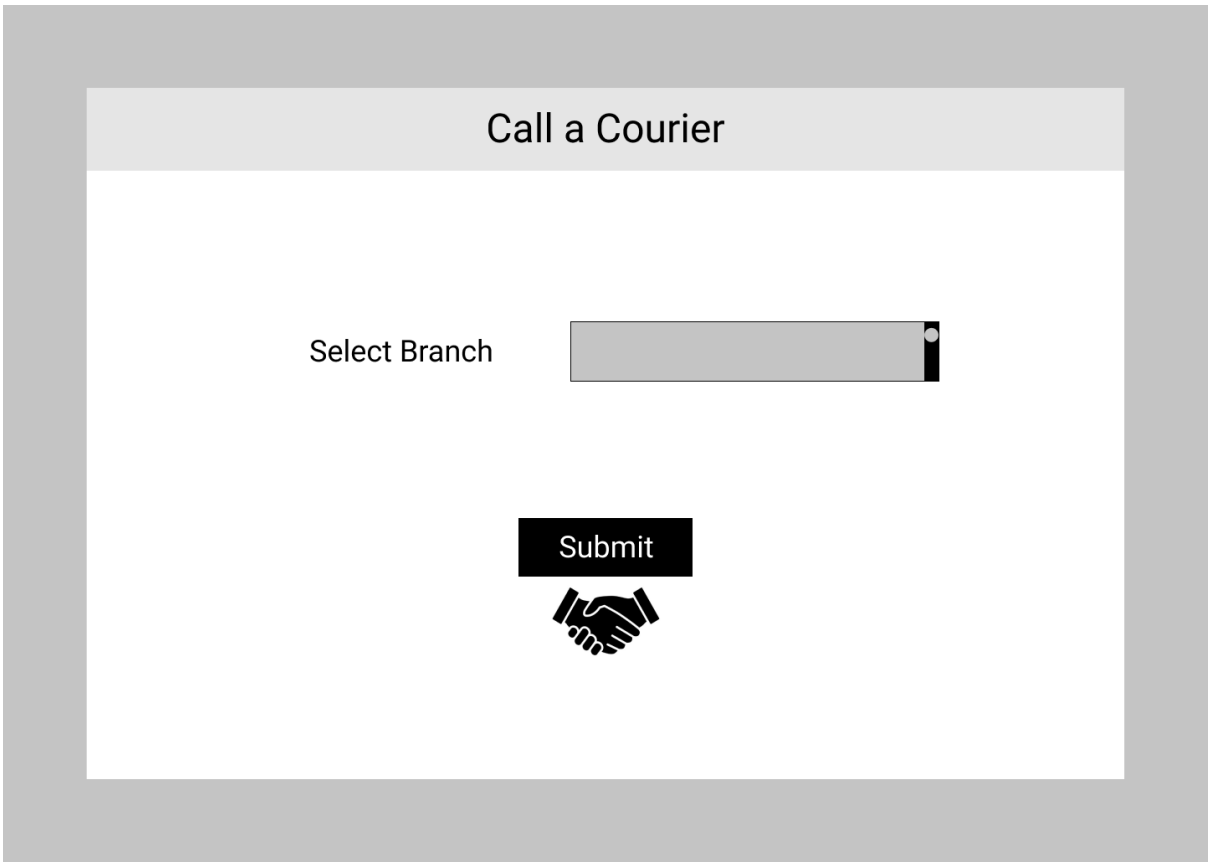
**Employee Listing:**

```
SELECT name, userID as userIDC
FROM user, (SELECT clerk.userID AS userID
            FROM clerk, (Select employee.userID AS userID
                         FROM employee
                         Where employee.branchID = @branchID) AS emp
            WHERE clerk.userID = emp.userID) AS cle
Where  user.userID = cle.userID
```

**Assign package to clerk:**

```
Update submission
SET clerkID=@userIDC AND status= ‘on the branch’
Where submissionID=@submissionID
```

## 3.9 Call a Courier




**Input:** @branchID

**Process:** The user selects a branch. Random clerk is assigned to pickup assignment table but since every clerk in specified branch can update table, which means assigning a specific courier to that submission, this will not affect the situation.

**SQL Statements:**

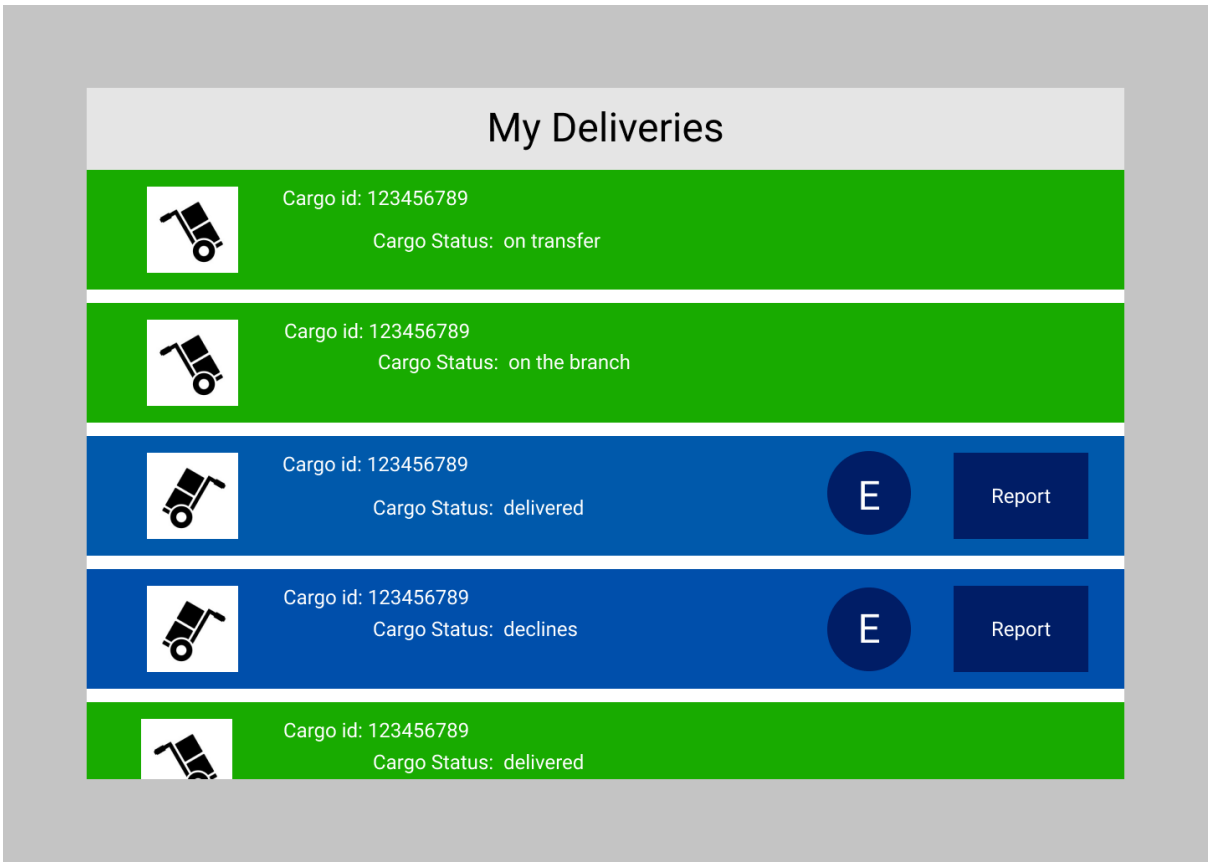
**Branch Listing:**

```
Select name
From branch
```

**Create request:**

```
insert into pickupassignment(submissionID, clerkID, courierID,status) values
(LAST_INSERT_ID(), (select userID
                    from employee natural join clerk
                    where branchid = @branchID
                    limit 1), null, 'request')
```

## 3.10 My Deliveries




**Process:** The user can see the list of all packages as sender or receipt.

**SQL Statements:**

**Display as sender:**

select submissionID, status

from submission, customer

where senderID = @userID

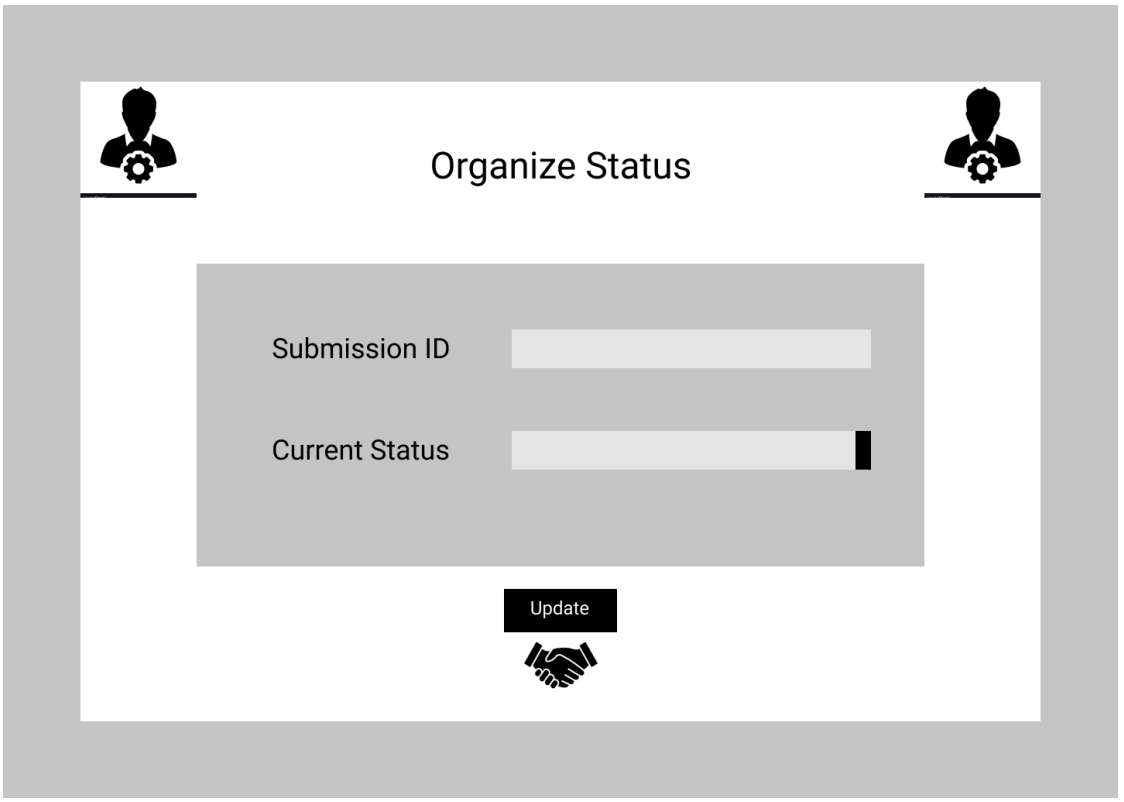
**Display as receipt:**

select submissionID, status

from submission, customer

where recipientID = @userID

## 3.11 Organize Status of Submission



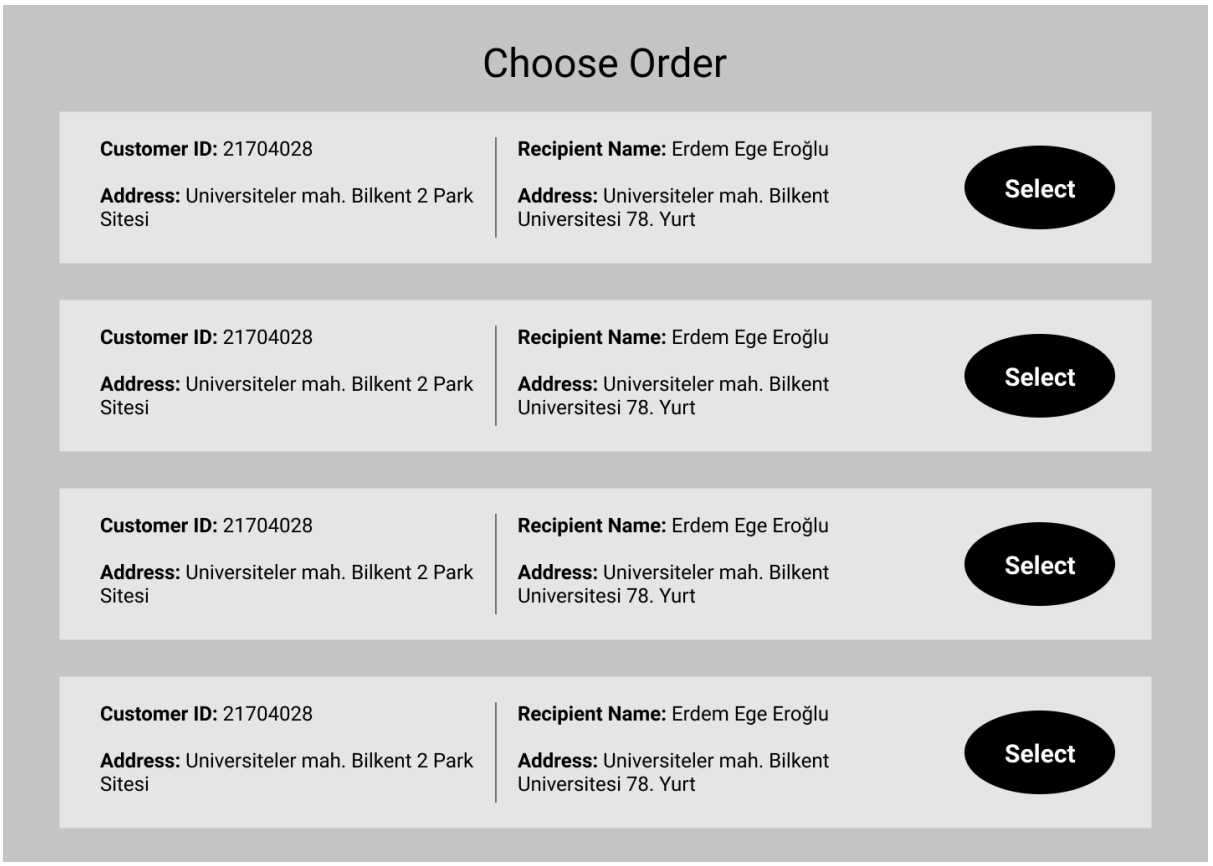

**Input:** @submissionID, @status
**Process:** The clerk can modify the status of the submission
**SQL Statements:**

**Update status:**
Update submission

SET status = @status

Where submissionID=@submissionID

## 3.11 Choose Order



Choose Order

**Customer ID:** 21704028

**Address:** Universiteler mah. Bilkent 2 Park Sitesi

**Recipient Name:** Erdem Ege Eroğlu

**Address:** Universiteler mah. Bilkent Universitesi 78. Yurt

**Customer ID:** 21704028

**Address:** Universiteler mah. Bilkent 2 Park Sitesi

**Recipient Name:** Erdem Ege Eroğlu

**Address:** Universiteler mah. Bilkent Universitesi 78. Yurt

**Customer ID:** 21704028

**Address:** Universiteler mah. Bilkent 2 Park Sitesi

**Recipient Name:** Erdem Ege Eroğlu

**Address:** Universiteler mah. Bilkent Universitesi 78. Yurt

**Customer ID:** 21704028

**Address:** Universiteler mah. Bilkent 2 Park Sitesi

**Recipient Name:** Erdem Ege Eroğlu

**Address:** Universiteler mah. Bilkent Universitesi 78. Yurt

**Input:** @submissionID

**Process:** The employee can choose an order to assign to a courier.

**SQL Statements:**

**Display order requests:**

select S.senderID, sender.address, U.username, recipient.address

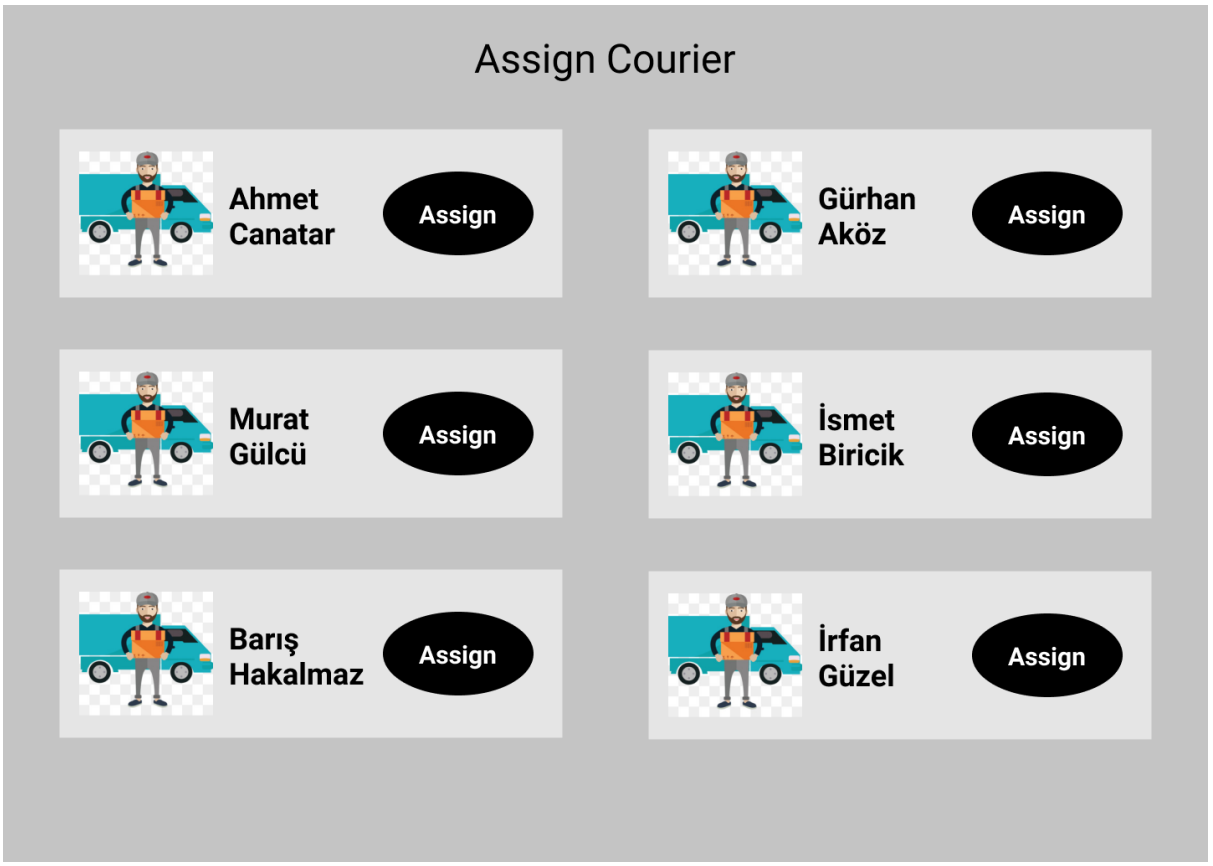
from submission S, customer sender, customer recipient, user U

where S.senderID = sender.userID, S.recipientID = recipient.userID, recipient.userID = U.userID

**Choosing an order:**

insert into pickupassignment(submissionID, clerkID, courierID,status) values (@submissionID, @userID, null, null)

## 3.12 Assign Courier




**Input:** @courierID, @submissionID(taken from the Choose Order page)

**Process:** The employee can assign a courier to the submission that is chosen in the previous page.

**SQL Statements:**

**Display available couriers:**

select sender.userID, sender.address, U.username, recipient.address

from submission S, customer sender, customer recipient, user U

where S.submissionID= @submissionID, S.senderID = sender.userID, S.recipientID = recipient.userID, recipient.userID = U.userID

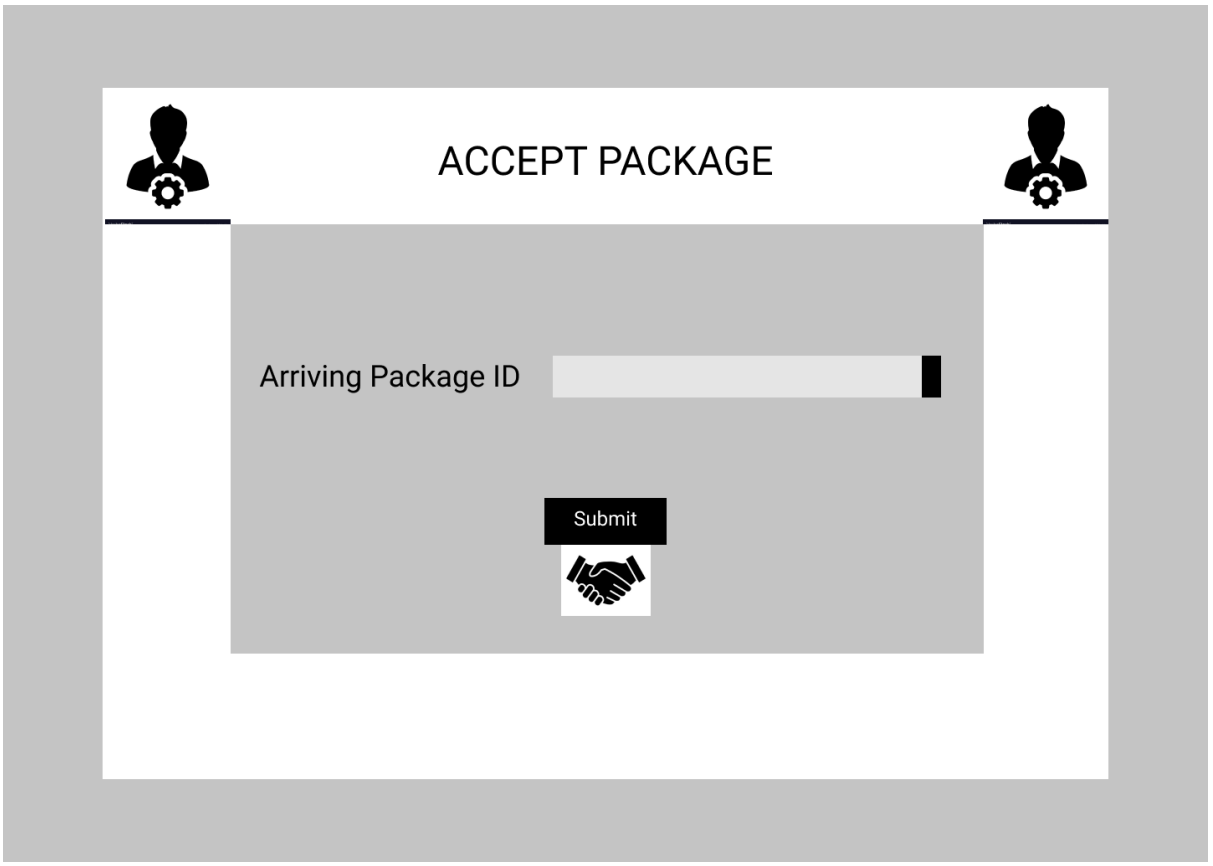
**Choosing a courier for assignment:**

update pickupassignment

set courierID = @courierID AND status = 1

where submissionID = @submissionID

## 3.13 Accept Package




**Input:** @submissionID,@userIDC

**Process:** The employee who is a clerk can accept packages that came to the branch by the customer himself/herself. userIDC is the id of the clerk and we already know the id in this session.

**SQL Statements:**

**Select a submission to be accepted:**

Select submissionID

from submission

where status = ‘waiting_to_be_approved’

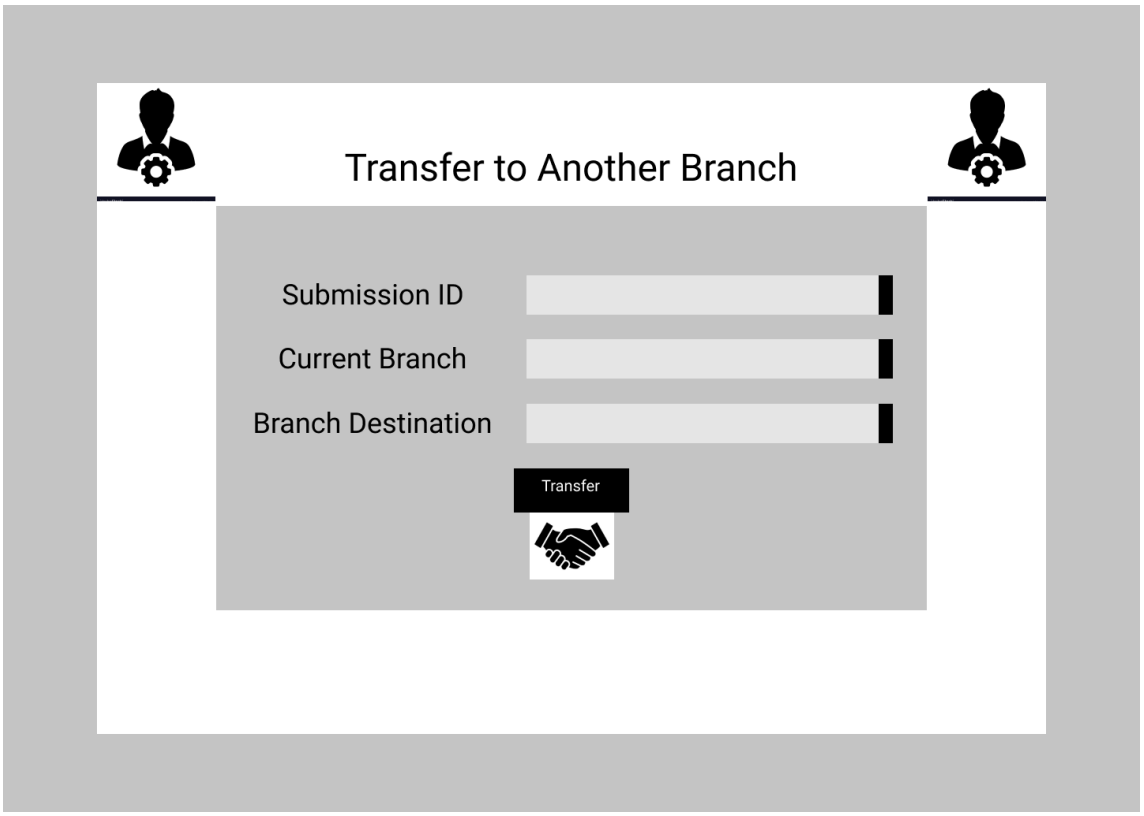
**Assign clerk:**

Update Submission

SET clerkID=@userIDC AND status= ‘on the branch’

Where submissionID=@submissionID

## 3.14 Transfer Submission to Another Branch




**Input:** @branchID, @submissionID, @branchIDD

**Process:** The clerk selects the package to send to the desired branch. clerkID is a session variable.

**SQL Statements:**

**Listing packages:**

Select submissionID From submission where clerkID = @clerkID

**Current branch:**

Select name, branchID as currentBranch

From branch

Where branchID = (select branchID from clerk natural join employee where userID = clerkID)

**Listing destination branches:**

Select branchName, branchID as branchDestination

From branch Where branchID<>(select branchID from clerk natural join employee where userID = clerkID)

**Transferring the packages for a particular submission**

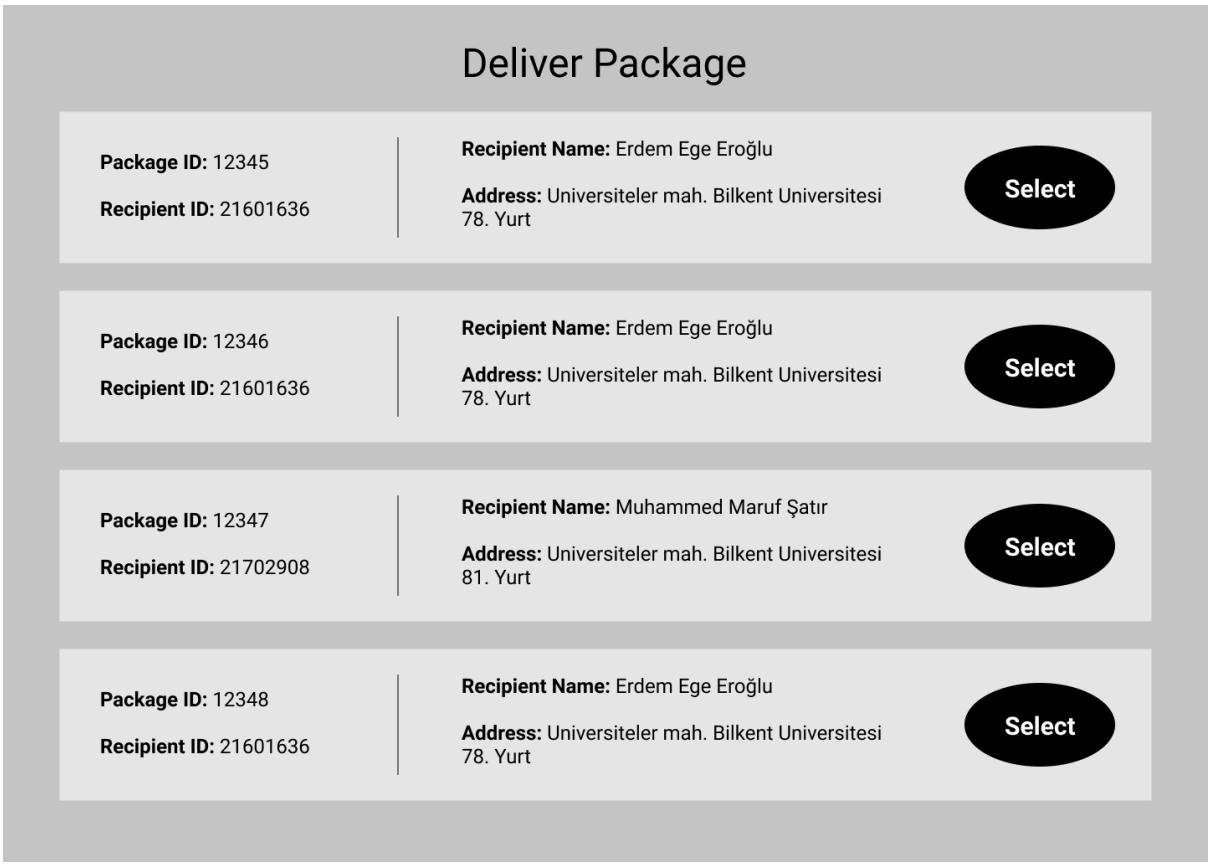
if there is no transfer that is made for that submission:

```
Insert into transfer(clerkID, branchID, submissionID)
Values(@userIDC, @branchName, @submissionID)

if there is transfer already made in that submission:
Update transfer
set branch = @branchname
where clerkID = @userIDC AND submissionID = @submissionID


update submission
set status='waiting_to_be_accepted'
where submissionID = @submissionID
```

## 3.15 Deliver Package




**Input:** @submissionID

**Process:** The employee can choose an order to assign to a courier, this time for the delivery to the recipient.

**SQL Statements:**

**Display deliveries:**

select S.submissionID, A.address, U.username, B.address

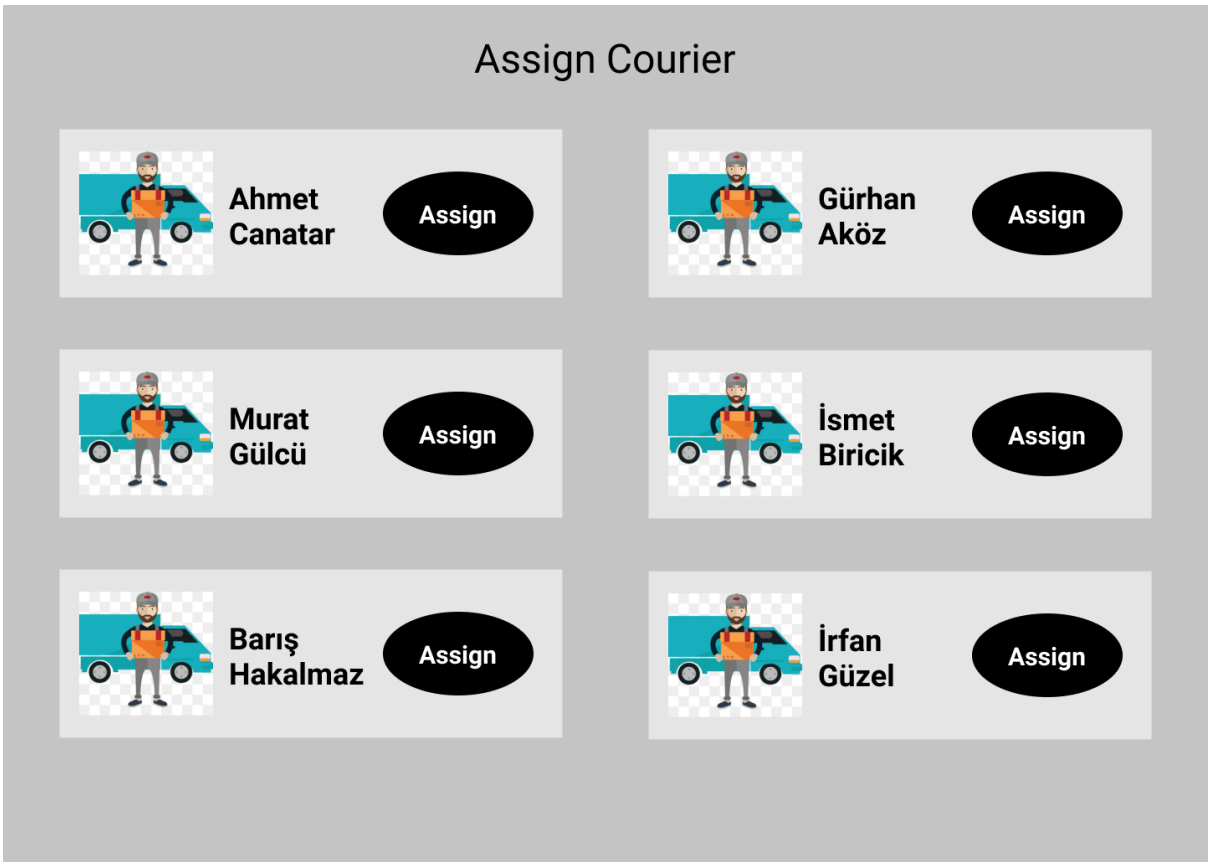
from submission S, customer A, customer B, user U

where S.submissionID= @submissionID AND S.senderID = A.userID, S.recipientID = B.userID, B.userID = U.userID

**Choosing an order:**

insert into deliveryassignment(submissionID, clerkID, courierID,status) values (@submissionID, @clerkID, null, 0)

## 3.16 Assign Courier (for delivery)




**Input:** @courierID, @submissionID(taken from the Deliver Package page)

**Process:** The employee can assign a courier to complete the delivery to the recipient.

**SQL Statements:**

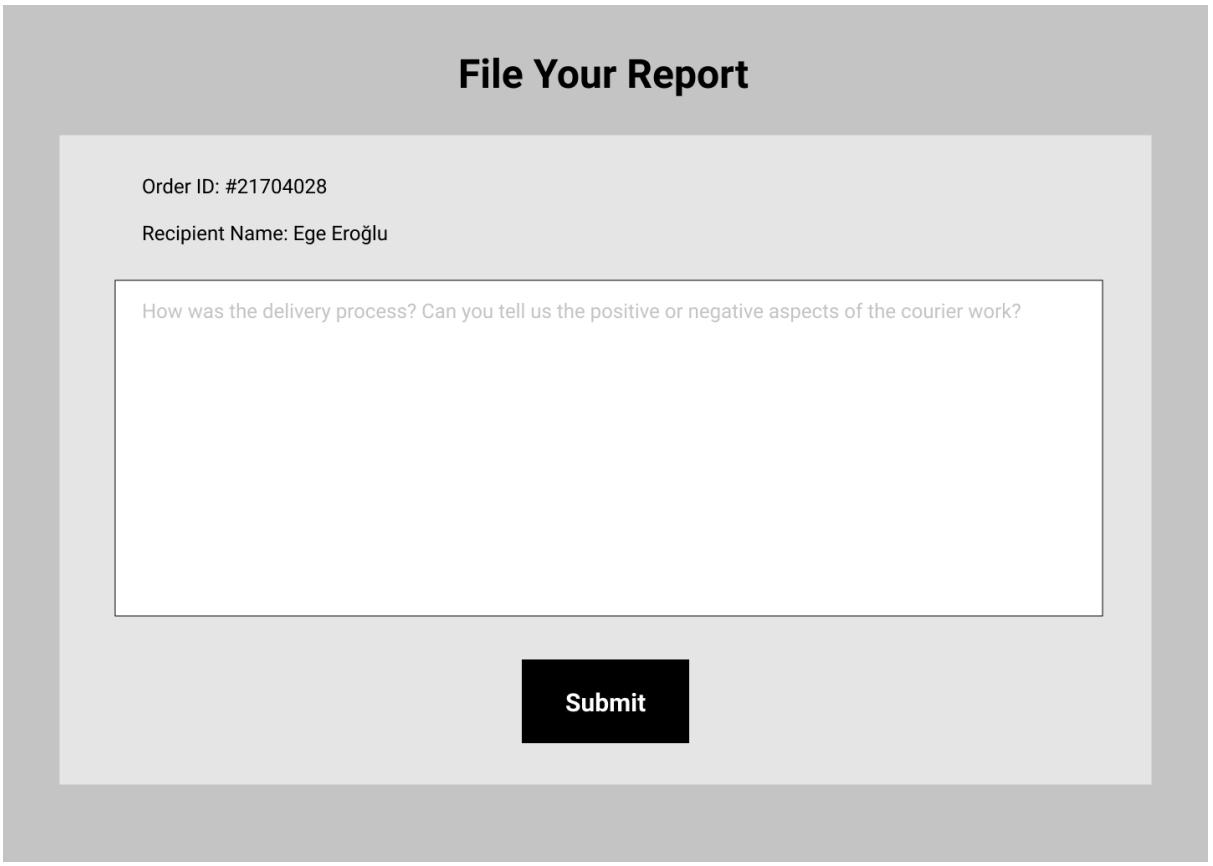
**Display available couriers:**

select sender.userID, sender.address, U.username, recipient.address
from submission S, customer sender, customer recipient, user U
where S.submissionID= @submissionID, S.senderID = sender.userID, S.recipientID = recipient.userID, recipient.userID = U.userID

**Assign courier for delivery:**

Update deliveryassignment
SET courierID=@courierID AND status = ‘on transfer’
Where submissionID=@submissionID

## 3.17 File Your Report




**Inputs:** @submissionID, @userID, @text, @date

**Process:** Recipients can file a report about the delivery just made to them. @submissionID and @userID are session variables.

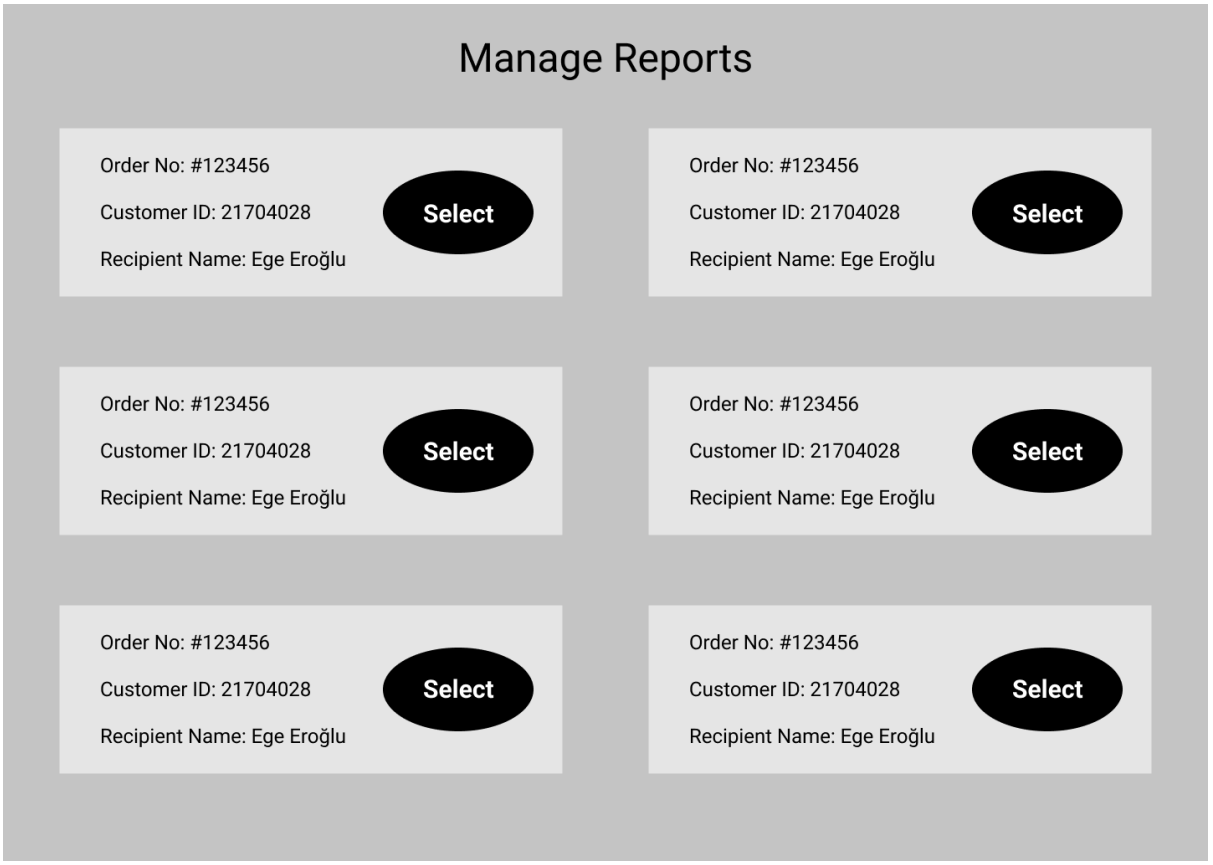
**SQL Statements:**

**Filing Reports:**

insert into report(submissionID, customerID, clerkID, text, date, status)

values (@submissionID, @userID, null, @text, @date, null)

## 3.18 Manage Reports




**Inputs:** @reportID

**Process:** Clerks can choose the reports filed to evaluate them to record.

**SQL Statements:**

**Display list of reports:**

select S.submissionID, A.userID, U.username

from report R, submission S, customer A, customer B, user U

where R.submissionID = S.submissionID, S.senderID = A.userID,

S.recipientID = B.userID, B.userID = U.userID

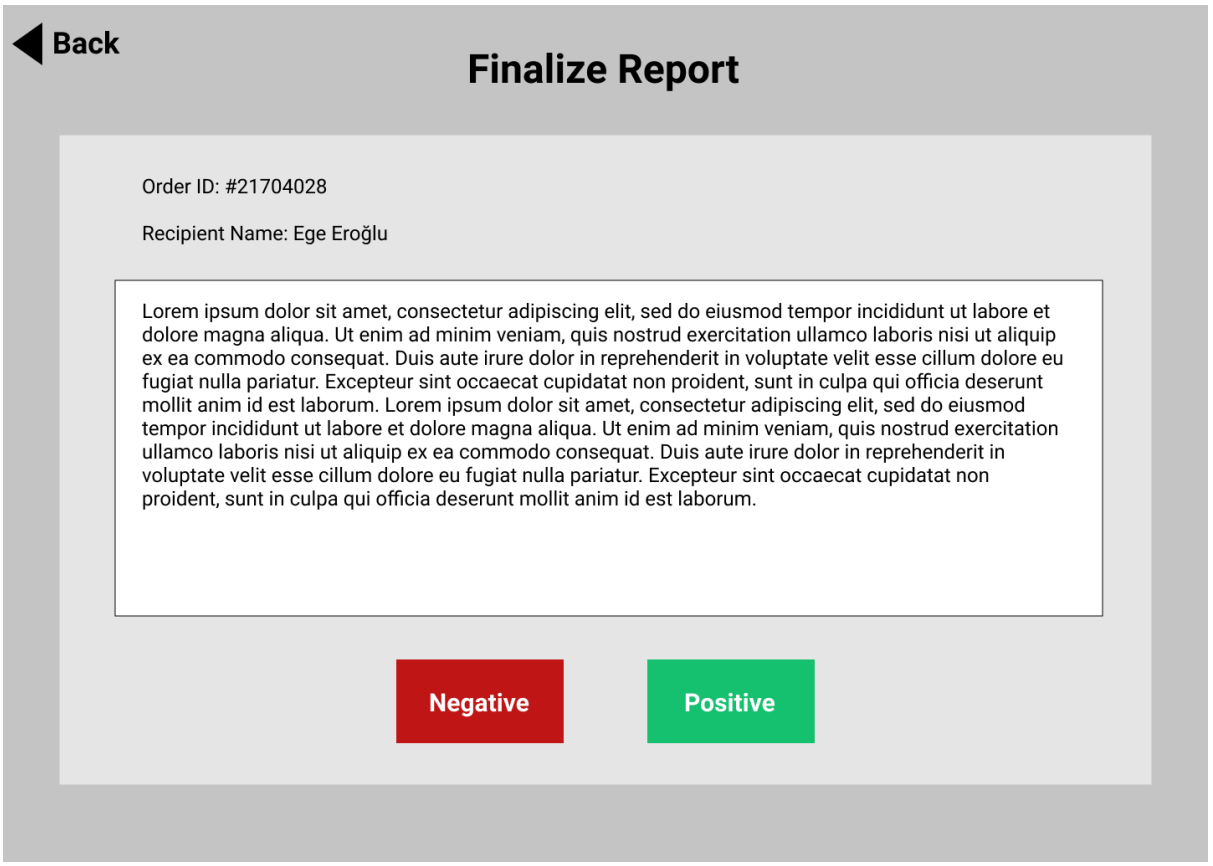
**Choosing a report to evaluate:**

select *

from report

where reportID = @reportID

## 3.19 Finalize Report




**Inputs:** @status, @reportID, @userID

**Process:** Clerks can evaluate and finalize the reports using this page.

**SQL Statements:**

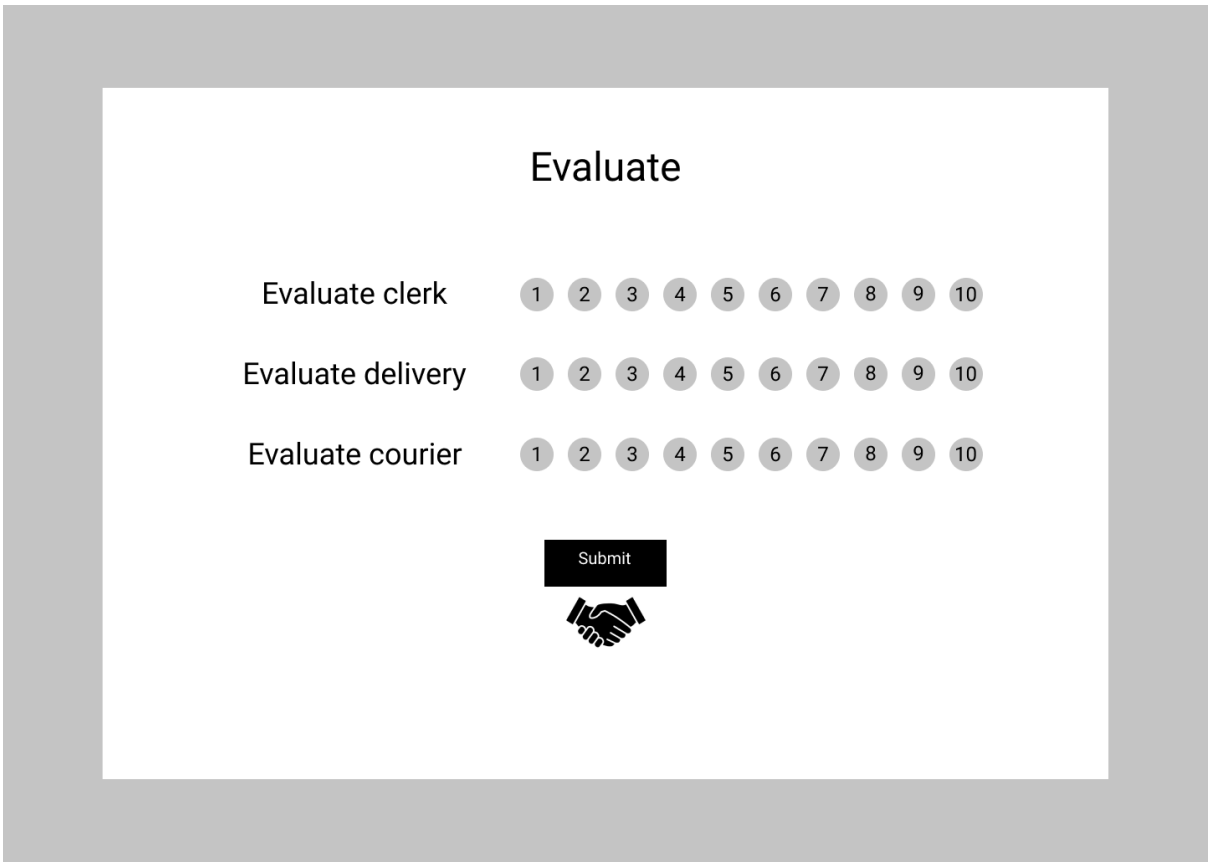
**Finalizing Reports:**

update report(reportID, submissionID, customerID, clerkID, text, date, status)

set clerkID = @userID AND status = @status

where reportID = @reportID

## 3.20 Evaluate




**Inputs:@submissionID, @clerkID, @courierID, @customerID, @score**

**Process:** Recipient can evaluate the process. Score is obtained by taking the average of all.

**SQL Statements:**

```
insert INTO evaluates VALUES( @customerID,
                              @submissionID,
                              select clerkID from submission where submissionID =
                              @submissionID,
                              select courierID from deliveryassignment
                              where submissionID = @submissionID, score)
```

# 4. Implementation Details

- We will use MySQL to implement our database.
- We will use PHP, HTML, CSS to implement user interface and application logic.