CS 353

Project Design Report
Shipping Company Data Management System

Group 14
Okan Şen 21202377

CS 353 **1**

1.Revised E-R Diagram **4**

2. Relation Schemas **6**

2.1 User 6
2.2 Customer 6
2.3 Employee 7
2.4 Courier 7
2.5 Message 7
2.6 Package 8
2.7 Orders 9
2.8 Receives 9
2.9 Report 10
2.10 Files 11
2.11 Handle 11
2.12 AssociatedTo 12
2.13 Has 12
2.14 Branch 13
2.15 Transportation 13
2.16 WorksAt 14
2.17 DeliversToLocation 14
2.18 SentOut 15

3. Functional Dependencies and Normalisation of Tables **16**

4. Functional Components **16**

4.1 Functional Requirements 16
4.1.1 User 16
4.1.2 Customer 16
4.1.3 Employee 16
4.1.4 Courier 17
4.2 Use Cases 17
4.3 Algorithms 17

5. UI Design and SQL Statements **18**

5.1 Login/Signup View 18
5.2 Customer View 20
5.3 Employee View 22

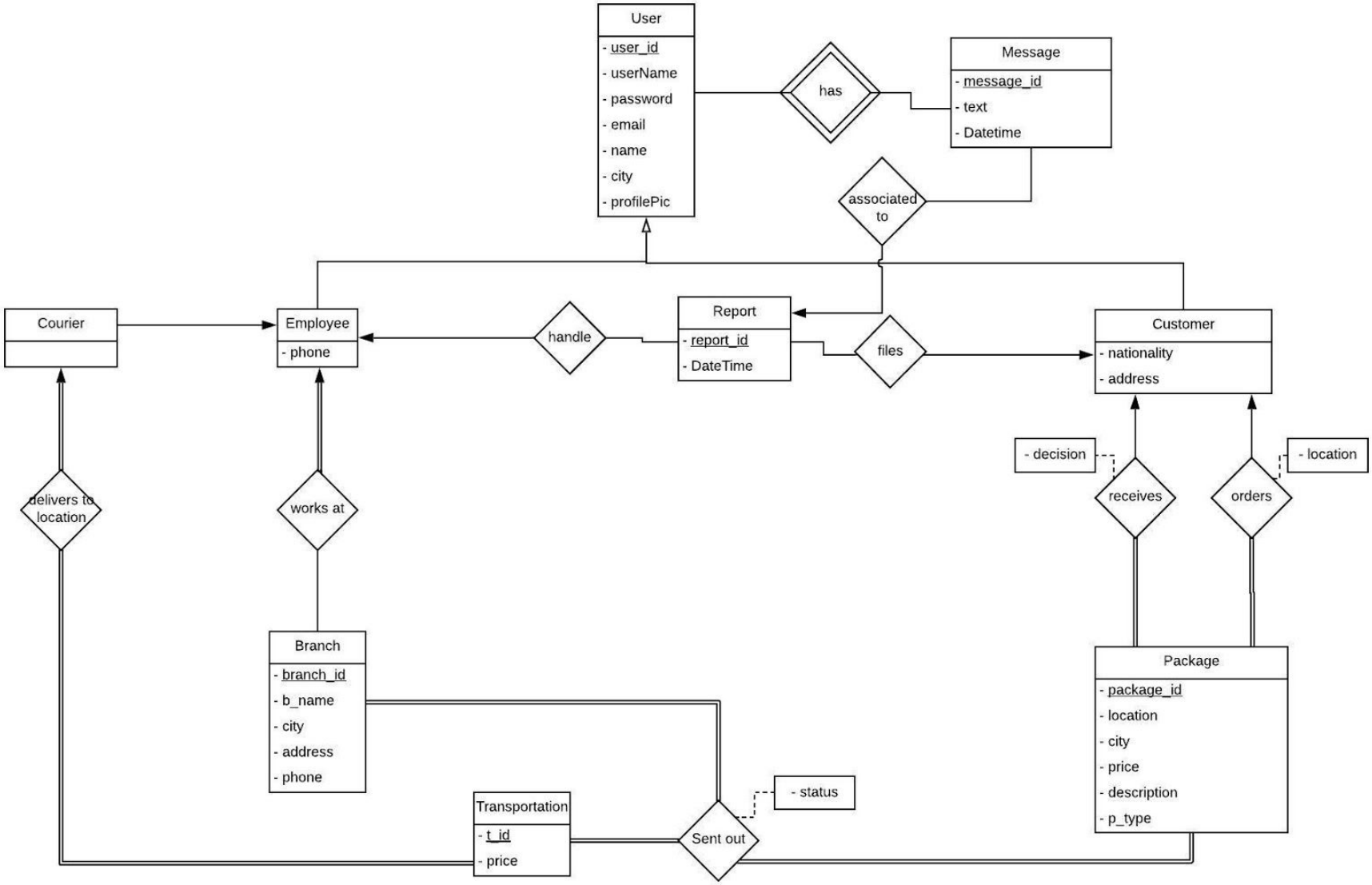
6. Advanced Database Components **24**

6.1 Views 24

6.1.1 Employee's Customer View 24
6.1.2 Employee's Employee View 24
6.1.3 Customer's Employee View 24
6.1.4 Customer's Customer View 24
6.2 Stored Procedures 25
6.3 Reports 25
6.3.1 Total Sum of Price Received by Customer 25
6.4 Triggers 25
6.5 Constraints 25
7. Implementation 26
8. Website 26

1.Revised E-R Diagram

- Message relation added with weak relation.
- Courier is now a subtype of employee which is a subtype of user.
- Shipment and payment types are removed from the diagram.
- Delivery changed to Transportation, which got connected with branch and package with a ternary relationship called “Sent out” that keeps the status of an ongoing delivery/shipment/transportation.
- Customers can now both send and receive packages, from other users.
- Added decision attribute to receive relationship. Acceptance will end the transportation, while declination will make the customer to file a report which then will be handled by an employee. The handling will be associated with message entity. The employee will be able to contact the specific user via the chat view.
- Order relation is now a strong relation.
- Employee entity now works at a branch.


User
- user_id
- userName
- password
- email
- name
- city
- profilePic
has
Message
- message_id
- text
- Datetime
associated to
Courier
Employee
- phone
handle
Report
- report_id
- DateTime
files
Customer
- nationality
- address
- decision
receives
orders
- location
delivers to location
works at
Branch
- branch_id
- b_name
- city
- address
- phone
Package
- package_id
- location
- city
- price
- description
- p_type
Transportation
- t_id
- price
Sent out
- status

2. Relation Schemas

2.1 User

Relational Model:
User(<u>user_id</u>, userName, password, email, name, city, profilepic)

Functional Dependencies:
user_id → userName, password, email, name, city, profilepic

Primary Key:
user_id

Normal Form:
BCNF

Table Definition:

```
Create Table User(
        user_id         int not null auto increment primary key,
        userName        varchar(20) not null primary key,
        password        varchar(20) not null,
        email           varchar(320) not null,
        name            varchar(50) not null,
        City            varchar(20) not null,
        profilepic      varchar(260) not null default '****',     // to be determined
) Engine=InnoDB;
```

2.2 Customer

Relational Model:
Customer(<u>user_id</u>, nationality, address)

Primary Key:
user_id

Normal Form:
BCNF

Table Definition:

```
Create Table Customer(
                user_id         int not null auto increment primary key,
                nationality     varchar(20),
                address         varchar(30),
```

```
        foreign key (user_id)        references User(user_id)
                                     on cascade delete
                                     on cascade update
) Engine=InnoDb;
```

2.3 Employee

Relational Model:
Employee(<u>user_id</u>, phone)

Primary Key:
user_id

Normal Form:
BCNF

Table Definition:

```
Create Table Employee(
                     user_id      int not null auto increment primary key,
                     phone        varchar(20) not null,
        foreign key (user_id)     references User(user_id)
                                  on cascade delete
                                  on cascade update
) Engine=InnoDb;
```

2.4 Courier

Relational Model:
Courier(<u>user_id</u>)

Primary Key:
user_id

Normal Form:
BCNF

Table Definition:

```
Create Table Courier(
                     user_id      int not null auto increment primary key,
        foreign key (user_id)     references Employee(user_id)
                                  on cascade delete
                                  on cascade update
) Engine=InnoDb;
```

2.5 Message

Relational Model:
Message(Sender, Recipient, MessageID, Text, DateTime)

Primary Key:
Sender, Recipient, MessageID

Normal Form:
BCNF

Table Definition:

```
Create Table Message(
                    Sender          varchar(20),
                    Recipient       varchar(20),
                    MessageID       int not null auto increment,
                    Text            varchar(1024) not null,
                    DateTime        DateTime not null default current_timestamp,
    primary key(Sender, Recipient, MessageID),
    foreign key (Sender)            references Employee(userName)
                                    on cascade delete
                                    on cascade update,
    foreign key (Recipient)         references Customer(userName)
                                    on cascade delete
                                    on cascade update
) Engine=InnoDb;
```

2.6 Package

Relational Model:
Package(package_id, city, price, description, p_type, delivery_type, payment_type)

Functional Dependencies:
package_id → city, price, description, p_type, delivery_type, payment_type
Primary Key:
package_id

Normal Form:
BCNF

Table Definition:

```
Create Table Package(
            package_id          int not null auto increment primary key,
            city                varchar(20),
```

```
                price                   int not null,
                description             varchar(20),
                p_type                  varchar(20),
                delivery_type           varchar(20),
                payment_type            varchar(20)
) Engine=InnoDB;
```

2.7 Orders

Relational Model:
Orders(<u>user_one_id, user_two_id, package_id</u>, location)

Primary Key:
User_one_id,user_two_id, package_id

Functional Dependencies:
user_one_id, user_two_id, package_id→ location

Normal Form:
BCNF

Table Definition:

```
Create Table Orders(
                user_one_id   int not null,
                user_two_id   int not null,
                location      varchar(20),
        primary key (user_one_id, user_two_id, package_id),
        foreign key (user_one_id)     references Customer(user_id)
                                      on cascade delete
                                      on cascade update,
        foreign key (user_two_id)     references Customer(user_id)
                                      on cascade delete
                                      on cascade update,
        foreign key (package_id)      references Package(package_id)
                                      on cascade delete
                                      on cascade update

) Engine=InnoDb;
```

2.8 Receives

Relational Model:
Receives(<u>user_one_id , user_two_id, package_id</u>, decision)

Primary Key:

user_one_id, user_two_id, package_id

Functional Dependencies:
user_one_id, user_two_id, package_id → decision

Normal Form:
BCNF

Table Definition:

```
Create Table Receives(
        user_one_id   int not null,
        user_two_id   int not null,
        decision      bit default 1 not null,
    primary key(user_one_id, user_two_id, package_id),
    foreign key (user_one_id)     references Customer(user_id)
                                  on cascade delete
                                  on cascade update,
    foreign key (user_two_id)     references Customer(user_id)
                                  on cascade delete
                                  on cascade update,
    foreign key (package_id)      references Package(package_id)
                                  on cascade delete
                                  on cascade update

) Engine=InnoDb;
```

2.9 Report

Relational Model:
Report(report_id, DateTime)

Functional Dependencies:
report_id → DateTime

Primary Key:
report_id

Normal Form:
BCNF

Table Definition:

```
Create Table Report(
    report_id        int not null auto increment primary key,
    DateTime         DateTime not null default current_timestamp
```

```
) Engine=InnoDB;
```

2.10 Files

Relational Model:
Files(<u>user_one_id, user_two_id, package_id, cour_id</u>)

Primary Key:
user_one_id, user_two_id, package_id, cour_id

Normal Form:
BCNF

Table Definition:

```
Create Table Files(
        user_one_id   int not null,
        user_two_id   int not null,
        package_id    int not null,
        cour_id       int not null,
    primary key(user_one_id, user_two_id, cour_id),
    foreign key (user_one_id)     references Customer(user_id)
                                  on cascade delete
                                  on cascade update,
    foreign key (user_two_id)     references Customer(user_id)
                                  on cascade delete
                                  on cascade update,
    foreign key (package_id)      references Package(package_id)
                                  on cascade delete
                                  on cascade update,
    Foreign key (cour_id)         references Courier(user_id)
                                  on cascade delete
                                  on cascade update
) Engine=InnoDb;
```

2.11 Handle

Relational Model:
Handle(<u>user_one_id, report_id, package_id</u>)

Primary Key:
user_one_id, report_id, package_id

Normal Form:
BCNF

Table Definition:

```
Create Table Handle(
        user_one_id   int not null,
        report_id     int not null,
        package_id    int not null,
    primary key(user_one_id, report_id, package_id),
    foreign key (user_one_id)     references Employee(user_id)
                                  on cascade delete
                                  on cascade update,
    foreign key (report_id)       references Report(report_id)
                                  on cascade delete
                                  on cascade update,
    foreign key (package_id)      references Package(package_id)
                                  on cascade delete
                                  on cascade update
) Engine=InnoDb;
```

2.12 AssociatedTo

Relational Model:

AssociatedTo(<u>report_id, message_id</u>)

Primary Key:

report_id, message_id

Normal Form:

BCNF

Table Definition:

```
Create Table AssociatedTo(
        report_id     int not null,
        message_id    int not null,
    primary key(report_id, message_id),
    foreign key (report_id)       references Report(report_id)
                                  on cascade delete
                                  on cascade update,
    foreign key (message_id)      references Message(message_id)
                                  on cascade delete
                                  on cascade update
) Engine=InnoDb;
```

2.13 Has

Relational Model:
Has(<u>user_one_id, user_two_id, message_id</u>)

Primary Key:
user_one_id, user_two_id, message_id

Normal Form:
BCNF

Table Definition:

```
Create Table Has(
                user_one_id   int not null,
                user_two_id   int not null,
                message_id    int not null
        primary key(user_one_id, user_two_id, message_id),
        foreign key (user_one_id)     references Customer(user_id)
                                      on cascade delete
                                      on cascade update,
        foreign key (user_two_id)     references Employee(user_id)
                                      on cascade delete
                                      on cascade update,
        foreign key (message_id)      references Message(message_id)
                                      on cascade delete
                                      on cascade update

) Engine=InnoDb;
```

2.14 Branch

Relational Model:
Branch(<u>branch_id</u>, b_name, city, address, phone)

Functional Dependencies:
branch_id → b_name, city, address, phone
Primary Key:
branch_id

Normal Form:
BCNF

Table Definition:
Create Table Branch(

```
        branch_id       int not null auto increment primary key,
        b_name          varchar(20),
        city            varchar(20),
        address         varchar(30),
        phone           varchar(20)
) Engine=InnoDB;
```

2.15 Transportation

Relational Model:
Transportation(t_id, price)

Functional Dependencies:
t_id → price
Primary Key:
t_id

Normal Form:
BCNF

Table Definition:

```
Create Table Transportation(
        t_id            int not null auto increment primary key,
        price           int not null
) Engine=InnoDB;
```

2.16 WorksAt

Relational Model:
WorksAt(user_id, branch_id)

Primary Key:
user_id, branch_id

Normal Form:
BCNF

Table Definition:

```
Create Table WorksAt(
        user_id         int not null,
        branch_id       int not null,
    primary key(user_id, branch_id),
    foreign key (user_id)       references Employee(user_id)
```

```
                                        on cascade delete
                                        on cascade update,
        foreign key (branch_id)         references Branch(branch_id)
                                        on cascade delete
                                        on cascade update
) Engine=InnoDb;
```

2.17 DeliversToLocation

Relational Model:
DeliversToLocation(<u>user_id</u>)

Primary Key:
user_id

Normal Form:
BCNF

Table Definition:

```
Create Table DeliversToLocation(
                user_id         int not null,
                primary key(user_id),
        foreign key (user_id)           references Courier(user_id)
                                        on cascade delete
                                        on cascade update
) Engine=InnoDb;
```

2.18 SentOut

Relational Model:
SentOut(<u>package_id, t_id, branch_id</u>, status)

Functional Dependencies:
package_id, t_id, branch_id → status

Primary Key:
package_id, t_id, branch_id

Normal Form:
BCNF

Table Definition:

```
Create Table SentOut(
                package_id      int not null,
                t_id            int not null,
```

```
        branch_id       int not null,
        status          varchar(20),
    primary key(package_id, t_id, branch_id),
    foreign key (package_id)    references Package(package_id)
                                on cascade delete
                                on cascade update,
    foreign key (t_id)          references Transportation(t_id)
                                on cascade delete
                                on cascade update,
    foreign key (branch_id)     references Branch(branch_id)
                                on cascade delete
                                on cascade update
) Engine=InnoDb;
```

3. Functional Dependencies and Normalisation of Tables

All tables are in Boyce-Codd normal form.

4. Functional Components

4.1 Functional Requirements

There are four types of users: user, customer, employee, and courier. User has all the common features of the other user types. Other three has specific abilities that will be listed below.

4.1.1 User

- User can login with their username and password.
- User can add a profile picture or change it if they already have one.
- Their cities are kept as well.

4.1.2 Customer

- Customer can order package to a location.
- Customer can receive package that is sent from another user.
- Customer can customize payment and shipment types.
- Customer can accept or decline a delivery. If the package is damaged, the customer should decline it, and file a report about it.
- Customer can list their past and present package deliveries.
- Customer can view their current deliveries and check their status: justSent, onWay, ondelivery.

● Customer can message with an employee if they have filed a report recently, in which case the employee will contact them first.

● Receiving customer will pay for package and delivery costs.

4.1.3 Employee

● Employee works at only one branch.

● Employee can list all reports; handled, ongoing, open, on others.

● Employee can handle a report that is filed by a customer.

● Employee can choose which report to handle.

● The handled report will either be returned as positive or negative.

● In case of positive handling, the employee will message the customer and notify them about the situation. In case of negative handling, the employee will message the customer again.

4.1.4 Courier

● Courier works at only one branch.

● Courier can deliver packages from location to location.

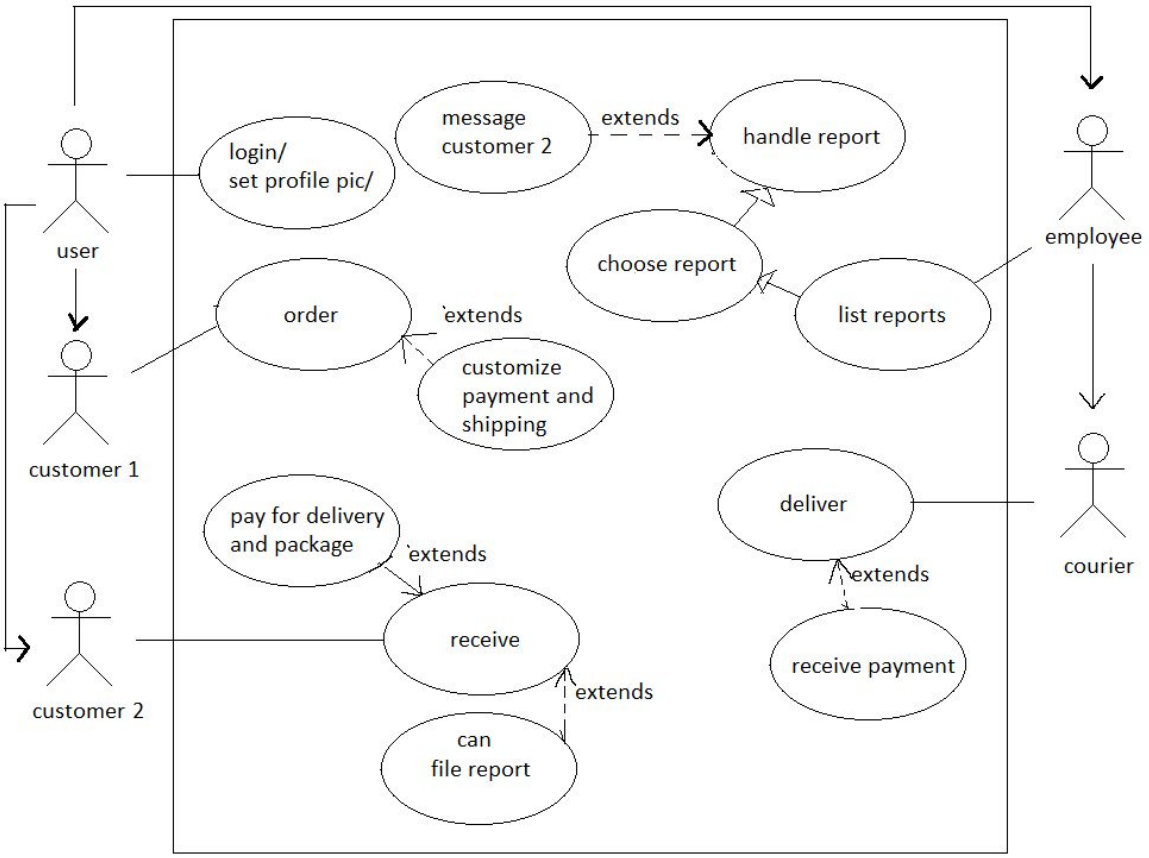
● In case of reports, courier will have to return the package back to the closest branch, until the report is handled.

● When the report is handled as positive, courier will have to bring the new package to the branch and then the package will be delivered from that branch to the customer's location.

● If the report is handled negatively, the package will be sent back to the user.

● Courier receives payment from customer.

4.2 Use Cases



4.3 Algorithms

Privacy Setting Algorithm

Since there is no Admin type user all the users can only view public information of other users, with the exception of customers not being able to view employee's phone number. Other than this feature, the system does not have much use for additional algorithms. However, I am pleased to add an idea here(it will probably stay as an idea unless I can add more user types);

Many Reports in a Row

If many transportations are reported with the same courier appearing frequently, the courier will be contacted by a manager and questioned.

5. UI Design and SQL Statements

5.1 Login/Signup View

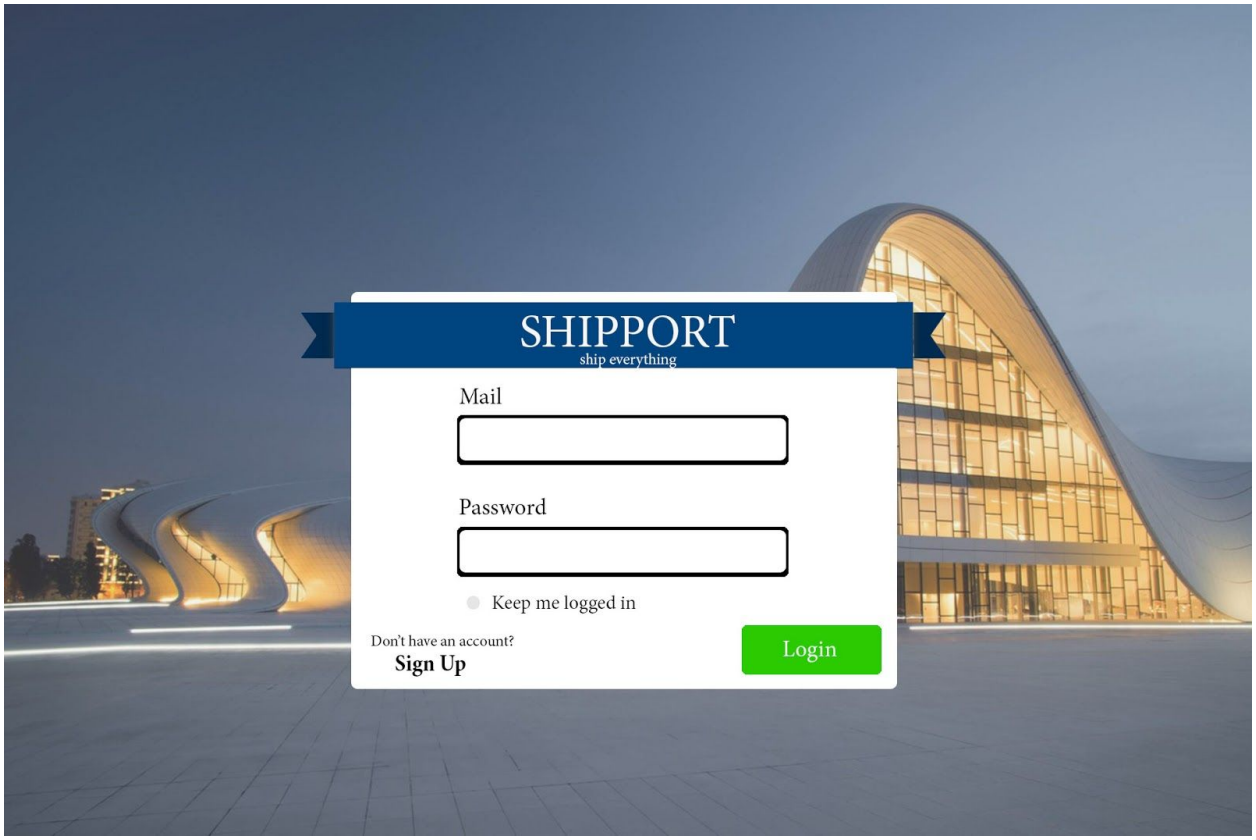


Figure 5.1.1: Login View

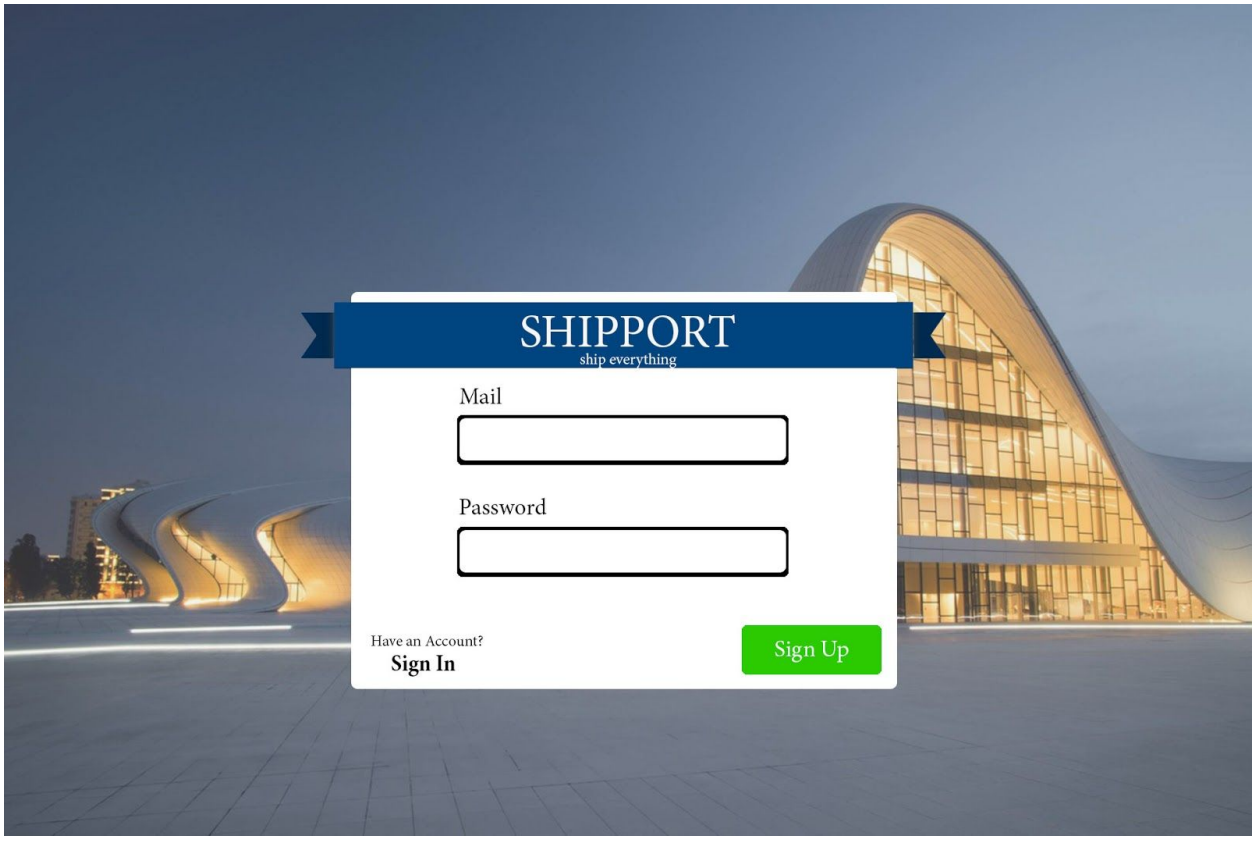


Figure 5.1.2: Signup View

Inputs: @email, @password

Process: In Figure 5.1.1, the user can login to an existing account using their email and password. If they don't have an account when faced with this view, they can go into sign up view instead with a click, which is shown in Fİgure 5.1.2. We want the account creation to be fast and easy, so entering valid email and password would suffice. Other attributes, such as, userName, city, nationality, etc… can be changed in user settings.

Login:
SELECT *
FROM User
WHERE email = @email AND password = @password

Sign Up:
INSERT INTO User(password, email)
VALUES(@password, @email)

5.2 Customer View

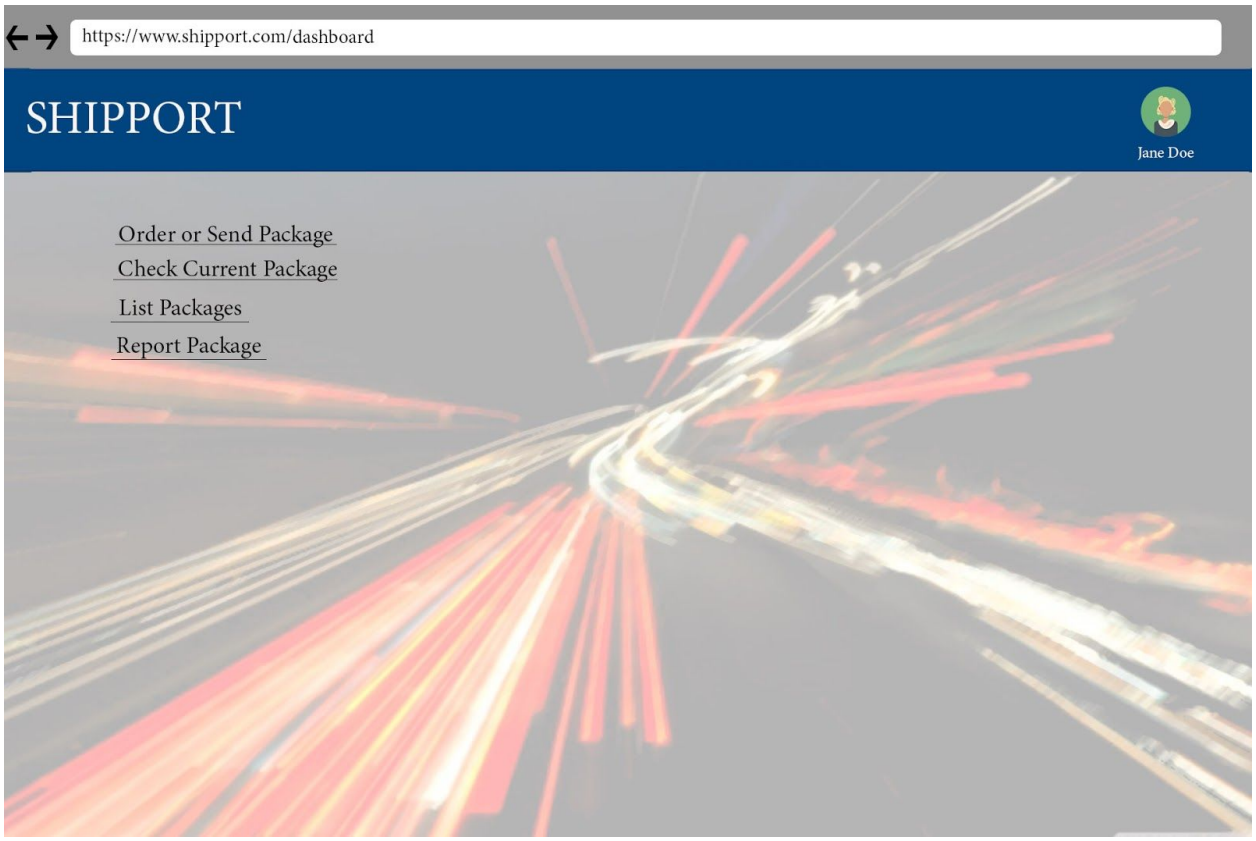


Figure 5.2.1: Customer View

inputs: @user_one_id, @user_two_id, @package_id, @location, @branch_id, @t_id, @status, @delivery_type, @payment_type, @p_type, @cour_id

Process: Customer can choose to view their current ordered(sent) package's delivery status, if they have any current orders. Additionally, they can view their past and current orders, received packages, from the "List Packages" button. They can order/send a package as well from here. When they choose to order a package, they will be able to customize package type, payment type and shipment type. If they have an active delivery, that is received, and the customer has found a damage on the package, they can report it, adding in sender, receiver, package and courier id's.

Order/Send Package:
INSERT INTO Orders
VALUES (@user_one_id, user_two_id, package_id, location)

UPDATE Package

SET p_type = @ p_type AND delivery_type = @delivery_type AND payment_type = @payment_type
WHERE package_id = @package_id

Check Current Package:
SELECT *
FROM Package NATURAL JOIN Transportation NATURAL JOIN Branch
WHERE package_id = @package_id AND t_id = @t_id AND branch_id = @branch_id

List Packages:
SELECT *
FROM Customer NATURAL JOIN TRANSPORTATION
WHERE t_id = @t_id AND user_id = @user_id
GROUP BY @status

Report Package:
INSERT INTO Files
VALUES (@user_one_id, @user_two_id, @package_id, @cour_id)

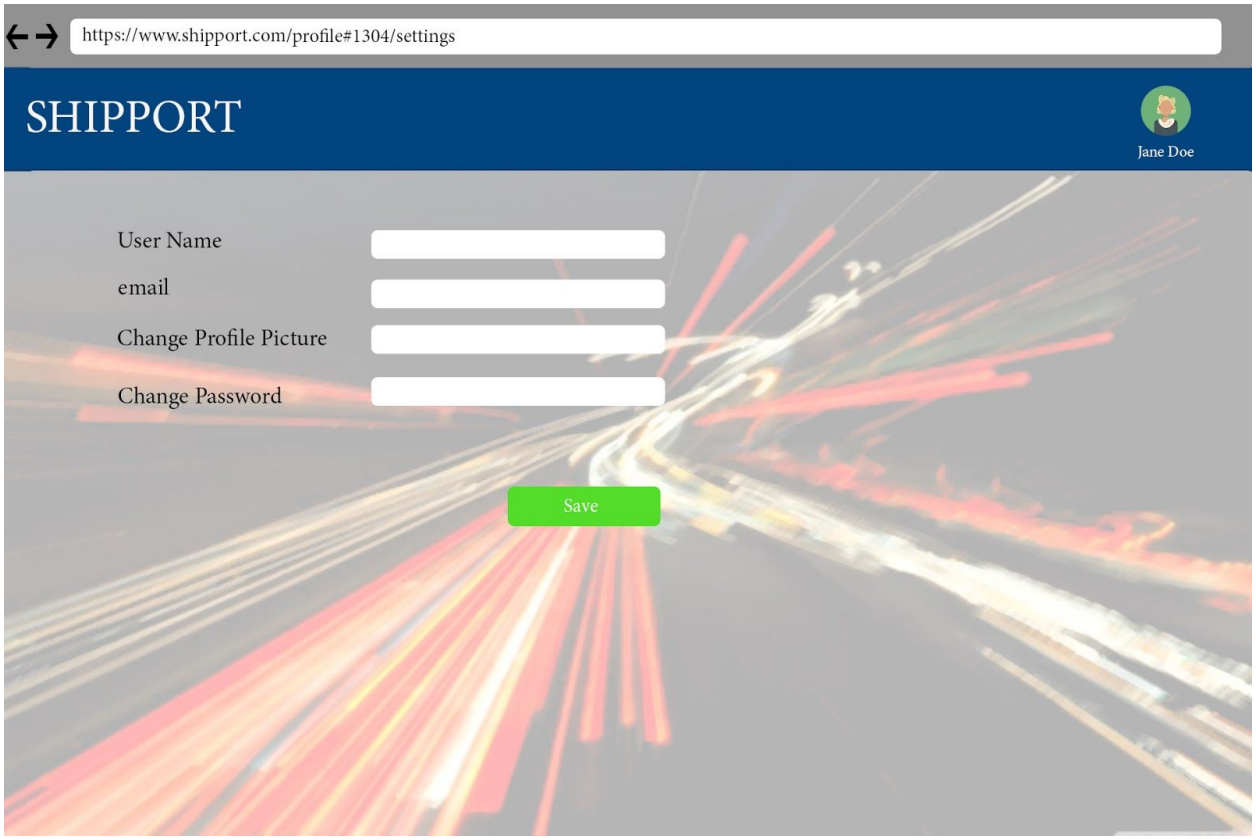


Figure 5.2.2: Customer Settings View

inputs: @user_id, @profilepic, @userName, @password

Process: The customer can also change their settings by clicking on their profile picture and name on the top right. Here, they can update their information.

Change Username:
UPDATE Customer
SET userName = @userName
WHERE user_id = @user_id

Change email:
UPDATE Customer
SET email = @email
WHERE user_id = @user_id

Change Profile Pic:
UPDATE Customer
SET profilepic = @profilepic
WHERE user_id = @user_id

Change password:
UPDATE Customer
SET password = @password
WHERE user_id = @user_id

5.3 Employee View

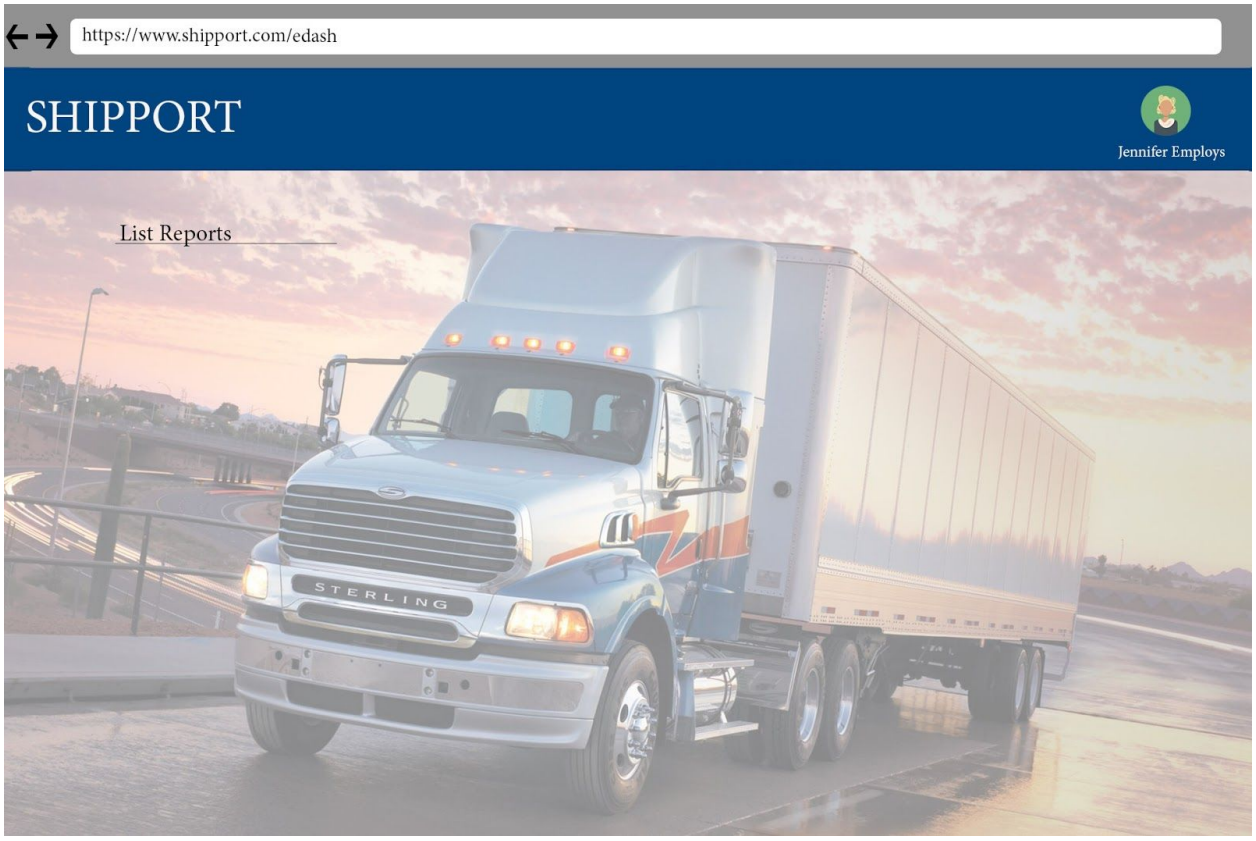
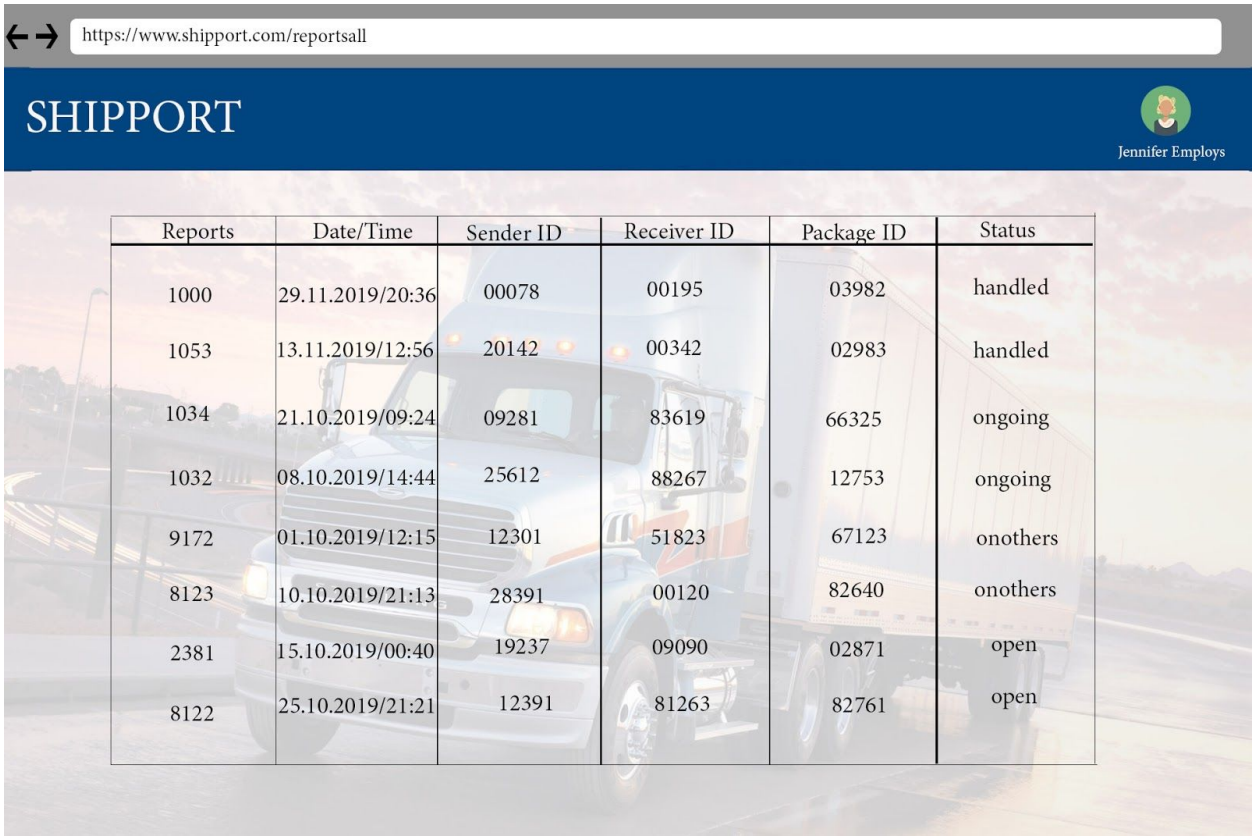


Figure 5.3.1: Employee View



Reports	Date/Time	Sender ID	Receiver ID	Package ID	Status
1000	29.11.2019/20:36	00078	00195	03982	handled
1053	13.11.2019/12:56	20142	00342	02983	handled
1034	21.10.2019/09:24	09281	83619	66325	ongoing
1032	08.10.2019/14:44	25612	88267	12753	ongoing
9172	01.10.2019/12:15	12301	51823	67123	onothers
8123	10.10.2019/21:13	28391	00120	82640	onothers
2381	15.10.2019/00:40	19237	09090	02871	open
8122	25.10.2019/21:21	12391	81263	82761	open

Figure 5.3.2: Employee List Reports View

inputs: @user_one_id, @user_two_id, @user_id, @package_id, @report_id, @DateTime, @rep_stat

Process: Employee will see Fİgure 5.3.1 when logged on to their account. Upon clicking on list reports, they will be directed to Figure 5.3.2, where all the information needed about a report is displayed. They are grouped by their status, and the employee can choose to handle an open report which is not yet taken by any employee. From there, the employee can finalize a report(positive or negative), and message the related customer(s).

List Reports:
SELECT *
FROM Reports NATURAL JOIN Customer NATURAL JOIN Package
WHERE report_id = @report_id AND package_id = @package_id AND user_one_id = @user_one_id AND user_two_id = @user_two_id, rep_stat = @rep_stat
GROUP BY @rep_stat

Choose Report:
SELECT *
FROM Reports NATURAL JOIN Employee
WHERE report_id = @report_id AND user_id = @user_id

Finalize Report:
UPDATE Report
SET rep_stat = @rep_stat
WHERE report_id = @report_id

6. Advanced Database Components

6.1 Views

6.1.1 Employee's Customer View

Employees can not see or change customer's information. Customers' addresses are also hidden from employees.

```
create view useremployeeview as
        Select userName, email, city, profilepic, nationality
        From User
        Where user_id in Employee
```

6.1.2 Employee's Employee View

Employees can view other employees and see their contact info and the branch they work at as well, in case it is needed. Passwords are hidden again.

```
Create view employeeemployeeview as
        Select userName, email, city, profilepic, branch_id
        From Employee
        Where user_id in Employee
```

6.1.3 Customer's Employee View

A customer can view an employee but they cannot see their password or their phones, since phone number is considered private with regard to customers. However, they can check the branch an employee are assigned to in case of contacting the branch directly.

```
Create view customeremployeeview as
        Select userName, email, city, profilepic, branch_id
        From Employee
        Where user_id in Customer
```

6.1.4 Customer's Customer View

A customer can view another customer with only limited information such as, common attributes. Profilepic, username, and email.

```
Create view customercustomerview as
        Select userName, email, profilepic, city
        From Customer
        Where user_id in Customer
```

6.2 Stored Procedures

- A procedure will be used to remove reports that have not been handled in 30 days, and to notify the filing customer that their report has been removed.

6.3 Reports

6.3.1 Total Sum of Price Received by Customer

```
Select t1.price + p1.price As 'total price'
From (Select t_id, price
        From Transportation t1) UNION
        (Select package_id, price
        From Package p1) ) NATURAL JOIN          (Select package_id
                                                 From Orders o1
                                                 Where p1.package_id = o1.package_id)
```

6.4 Triggers

- When a package has reached different stages of a shipment, update the status of it. onWay, Delivered, justSent, onBranch, etc…

- When a transportation is done, add the package price with the transportation price to return the final price.

- Remove a report once it is handled, either positively or negatively.

6.5 Constraints

- The website cannot be used without having an account.
- user_id's are the identity key of any customer and employee.
- Only customers can order or receive packages. Employees are out of this equation.
- Reports cannot be removed by an employee unless it is handled first, in which case, the report will be deleted automatically.
- Only employees can handle a report.
- Only employees can update status of package.

7. Implementation

MySQL will be used for database procedures, and PHP will be used for implementing the web page. Additionally, to integrate these two components Java will be used.

8. Website

Updates for the project will be up-to-date available on:
https://github.com/OkanSen/ship-comp-dat