



Bilkent University

CS 353 - Database Systems

Shipping Company Data Management System

Project Design Report

Group 15

Ahmet Burak Şahin

21301755

Arda Gültekin

21401142

Doruk Altan

21401362

Assigned Teaching Asistant

Arif Usta

Deadline: Nov 28, 2019

1. Introduction	5
1.1. Design Pattern Analysis for IS-A Relationships	6
1.2. Class Table Inheritance	7
2. Revised E/R Diagram	8
2.1 General Changes	8
2.2 Changes in Shipment	9
2.3 Changes in Customer	10
2.4 Changes in Person	10
2.5 Changes in Employee	10
2.6 Changes in Branch	10
2.7 Changes in Asset	10
2.8 Changes in Warehouse	10
2.9 Revised E/R Diagram	11
3. Relational Schemas	12
3.1 Entity Sets	12
3.1.1 Person	12
3.1.2 Customer	13
3.1.3 Employee	14
3.1.4 UserCustomer	15
3.1.5 Courier	16
3.1.6 Asset	17
3.1.7 Warehouse	18
3.1.8 Branch	19
3.1.9 Shipment	20
3.1.10 Report	21
3.2 Relationship Sets	22
3.2.1 work_at	22
3.2.2 stored	23
3.2.3 path_taken	24
3.2.4 sent_to	25

3.2.5 delivers	26
3.2.6 orders	27
3.2.7 files	28
3.2.8 evaluates	29
3.3 Multi-valued Attribute Tables	30
3.3.1 user_credit_cards	30
4. Functional Components	31
4.1 Use Cases	31
4.2 Algorithms	35
5. User Interface Design and Corresponding SQL Statements	37
5.1 Login	37
5.2 Register	38
5.3 View Shipment, View Shipment Status and Approve a Package, Decline a Package	39
5.4 Order Shipment	40
5.5 File a Report	41
5.6 Add/Remove Credit Card	42
5.7 Change Address, Phone Number, Email Address and Password	44
5.8 View and Manage Reports	45
6. Advanced Database Components	46
6.1 Views	46
6.1.1 Employee's Customer View	46
6.1.2 Customer's Shipment View	46
6.1.3 Courier's Shipment View	47
6.1.4 Employee's Report View	47
6.2 Reports	48
6.2.1 Total Number of Shipments and Total Money Spent by Customers	48
6.3 Triggers	48
6.4 Stored Procedures	48
6.5 Constraints	50
7. Implementation Plan	51
8. References	51

1. Introduction

In this design phase of the project, we revised our E/R model from the feedback received from the TA, and improved upon the design so that our database design meets the requirements that were established in the proposal phase. Additionally, this report provides relational schemas acquired from the corresponding E/R model counterparts, along with the SQL DDL statements which will be implemented in the database. Functional requirements are defined and demonstrated via use cases. The scenarios are used to depict the inner functional workings of our project. SQL modification (add/delete/update) statements and query statements are provided along with the corresponding GUI component to show which component supply which functionality in our database. This report finishes with the advanced design components that will be implemented in the project.

1.1. Design Pattern Analysis for IS-A Relationships

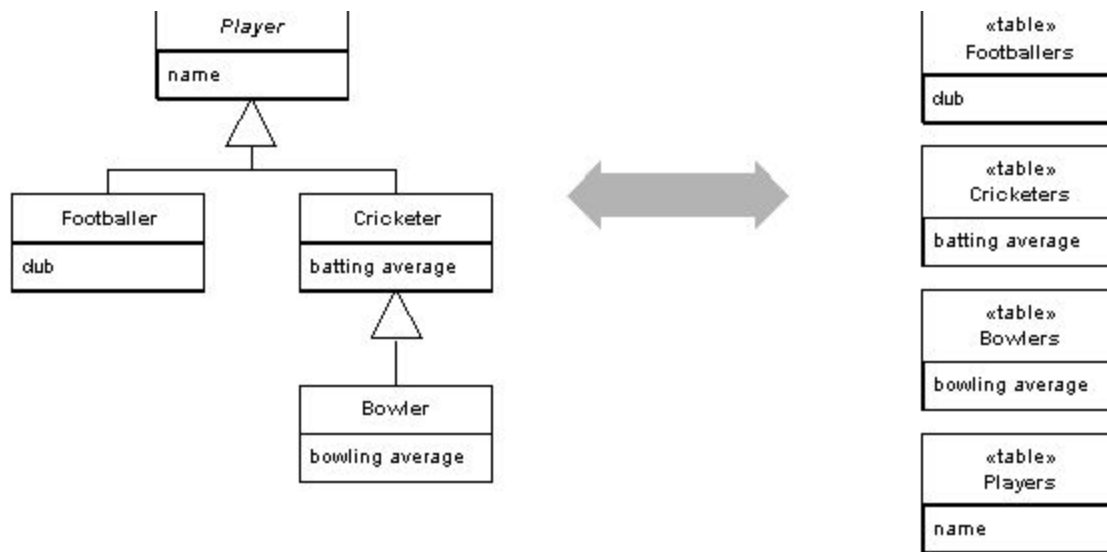
Our database implements Class Table Inheritance^[1] design pattern as a primary way of implementing IS-A relationships between entity sets. It is implemented in order to avoid issues that appear with the implementation of other designs, namely Single Table Inheritance^[2] and Concrete Table Inheritance^[3]. Such undesirable issues are

- **Possible null values** for some attributes which stem from the joining of the parent and child tables. *Simple Table Inheritance*^[2] method proposes a solution to avoid speed degradation over multiple join operations, therefore, it suggests the merging of all the attributes as in single, more general table. In such case, since the existence of one child tuple is not going to directly imply the existence of other child tuples, we are forced to put null values for the other child attributes, which is undesirable for consistent database implementation.
- **High redundancy** for each child tuple. *Concrete Table Inheritance*^[3] suggests the creation of child tables having all of the attributes of its corresponding parent table(s). This method forces the database queries to have multiple join operations in order to find the result of even the simplest queries. Therefore, for

such reasons our implementation will use *Class Table Inheritance*^[1] to achieve the best trade-off between speed and space.

1.2. Class Table Inheritance

Class Table Inheritance, in simplest terms, creates a table for each entity set involved in the IS-A relationship and links the child table with the primary key of its parent table, as shown below:



Class Table Inheritance creates tables for each child and links with the parent via a primary key^[1].

With this approach, we can access the inherited attributes via a natural join of the parent table and the child table. For modification (add/update/delete operation) of any of the tables involved in IS-A relationship, chain of triggers will be facilitated to keep the consistency throughout the database. Child table will hold the primary key of its parent as its primary key and constraint it as a foreign key so that database will remain consistent throughout. Another reason why Concrete Class Inheritance is being avoided is because each abstract (a.k.a. parent table) is involved in some queries so we need to create instances of these tables in order to be able to use them. Following E/R diagram will show what kind of relationships we have with the parent and other tables.

2. Revised E/R Diagram

According to the feedback we have received from our TA, Arif Usta, we have made the following changes.

2.1. General Changes

- Discarded entity sets:
 - International
 - Domestic
 - Admin
 - Task
 - TimeSlot
 - Vehicle
 - Truck
 - Plane
 - Location
 - Recipient
 - Complaint
- Added entity sets:
 - Report
 - UserCustomer
- Recipient is no longer a separate entity, merged with Customer. Now Recipients can benefit from the attributes inherited from Person which simplified our E/R diagram. Our application will not enforce people to sign up to our system in order to be able to send a package. Although we will not enforce membership for shipping operations we will still keep the ID, name, address and phone number information for possible future shippings involving the same person on one end. However, in order for a Customer to be able to file a Complaint or Report (both are called Report in our system), he/she must register in our database and thus turn into a *UserCustomer* so that he/she can use the different functionalities of our application which are restricted for *UserCustomer* use.

Finally, by providing an email address our shipping company can keep updated with the situation about the filed Report. *UserCustomers* logs in to their account via ID and password.

- Relation between *Courier* and *Shipment* is established via *delivers*.
- Relation between *Report* and *Shipment* is established via *belong_to*.
- Relation between *Employee* and *Branch* is established via *work_at*.
- Relation between *Shipment* and *Asset* is established via *path_taken*. Now we can track which *Warehouses* or *Branches* our *Shipment* gone through when we wanted to check the departure of the package.

2.2. Changes in Shipment

Following changes were made to Shipment entity set:

- *price* is added.
- *weight* is added.
- *package_content* is renamed to *package_description*. -> Short string value for explaining what is inside the package.
- *is_paid_by_sender* is renamed to *payment_side*. → Takes values: `{"SENDER", "RECIPIENT"}`.
- *payment_type* is added. → Takes values: `{"CREDIT_CARD", "CASH"}`.
- *shipment_date* is added.
- *delivery_type* is added. → Takes values: `{"COURIER", "SELF_PICK_UP"}`.
- *shipment_status* is added. → Takes values: `{"EN_ROUTE", "DELIVERED", "DECLINED"}`.

2.3. Changes in Customer

All the attributes of customer are removed and inserted into a new child entity set *UserCustomer* . Since, in real life, a non-*UserCustomer* customer can place a shipment order, we removed the possibility of a random customer, by database design, be able to save his/her credit card credentials. If a customer wants to save his/her credit card, he/she must sign up first.

2.4. Changes in Person

password attribute is removed from the *Person* because of the changes described in Section 2.3.

2.5. Changes in Employee

password attribute is added due to the changes described in Section 2.3.

2.6. Changes in Branch

service_start and *service_end* attributes are added to indicate the daily operational time interval of each individual *Branch*.

2.7. Changes in Asset

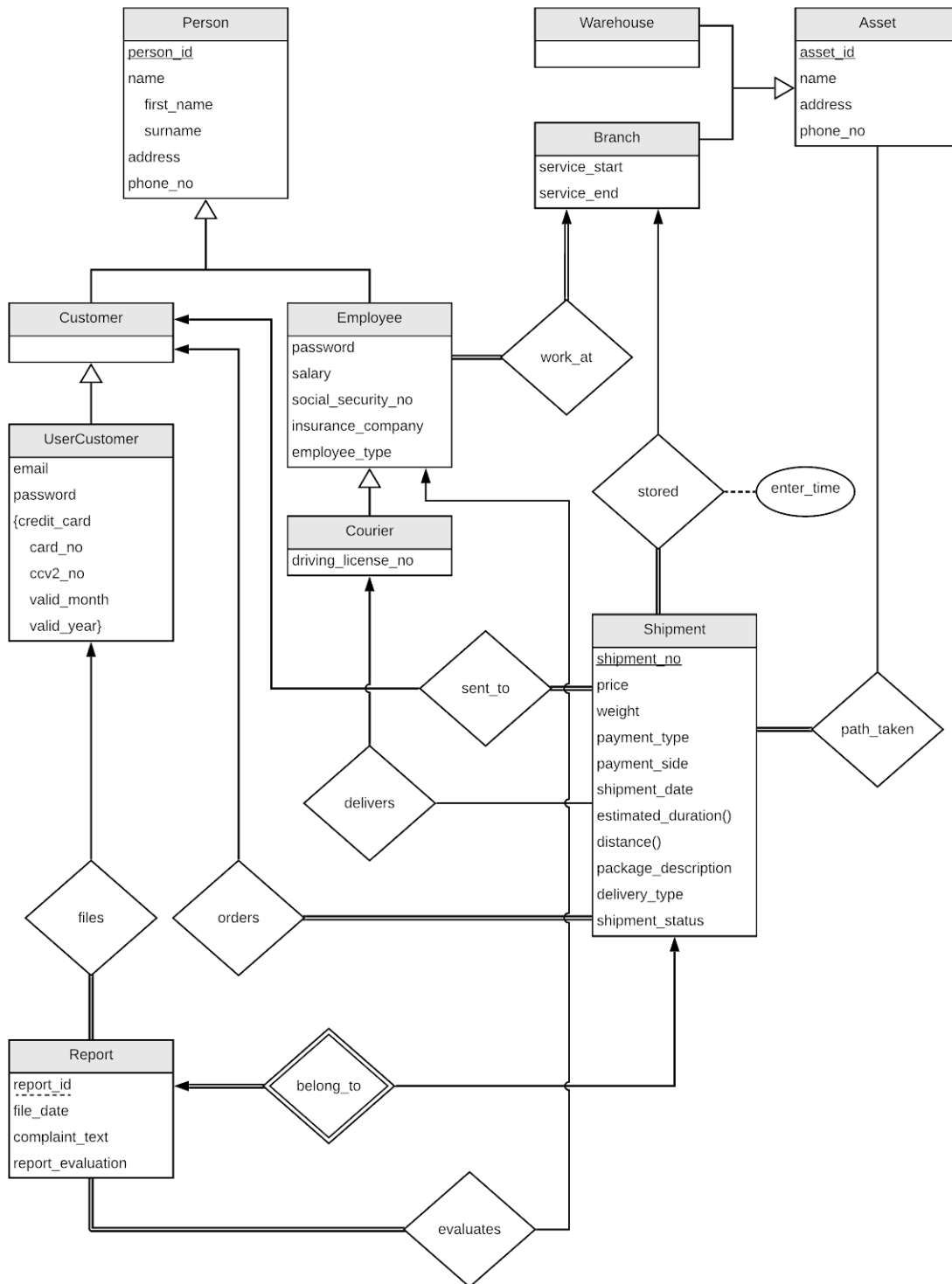
Following changes were made to *Asset* entity set:

- *name* is added.
- *address* is added.
- *phone_no* is added.

2.8. Changes in Warehouse

available_space is removed

2.9. Revised E/R Diagram



3. Relational Schemas

In this section, conversion from E/R model to Relational Model was performed. The corresponding Relational Schemas are:

3.1. Entity Sets

3.1.1. Person

Relational Schema:

`Person(person_id, first_name, surname, address, phone_no)`

Functional Dependencies:

`person_id → (first_name, surname, address, phone_no)`

Candidate Keys:

`{person_id}`

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE Person(  
    person_id    CHAR(11),  
    first_name   VARCHAR(20) NOT NULL,  
    surname      VARCHAR(20) NOT NULL,  
    address      VARCHAR(300),  
    phone_no     INT UNSIGNED,  
  
    PRIMARY KEY (person_id)  
) ENGINE=InnoDB;
```

3.1.2. Customer

Relational Schema:

Customer (person_id)

Functional Dependencies: \emptyset

Candidate Keys:

{person_id}

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE Customer(  
    person_id CHAR(11),  
  
    PRIMARY KEY (person_id),  
    FOREIGN KEY (person_id)  
        REFERENCES Person (person_id)  
        ON DELETE CASCADE  
) ENGINE=InnoDB;
```

3.1.3. Employee

Relational Schema:

Employee(person_id, password, salary, social_security_no, insurance_company, employee_type)

Functional Dependencies:

$\text{person_id} \rightarrow (\text{password}, \text{salary}, \text{social_security_no}, \text{insurance_company}, \text{employee_type})$

Candidate Keys:

{person_id}

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE Employee(  
    person_id          CHAR(11),  
    password           VARCHAR(20) NOT NULL,  
    salary             INT UNSIGNED NOT NULL,  
    social_security_no INT UNSIGNED NOT NULL,  
    insurance_company   VARCHAR(10) NOT NULL,  
    employee_type       ENUM('MANAGER', 'STAFF') NOT NULL,  
  
    PRIMARY KEY (person_id),  
    FOREIGN KEY (person_id)  
        REFERENCES Person (person_id)  
        ON DELETE CASCADE  
) ENGINE=InnoDB;
```

3.1.4. UserCustomer

Relational Schema:

UserCustomer(person_id, email, password)

Functional Dependencies:

person_id \rightarrow (email, password)

Candidate Keys:

{person_id}

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE UserCustomer(  
    person_id    CHAR(11),  
    email        VARCHAR(50) NOT NULL,  
    password     VARCHAR(20) NOT NULL,  
  
    PRIMARY KEY (person_id),  
    FOREIGN KEY (person_id)  
        REFERENCES Customer (person_id)  
        ON DELETE CASCADE,  
    UNIQUE(email)  
) ENGINE=InnoDB;
```

3.1.5. Courier

Relational Schema:

Courier(person_id, driving_license_no)

Functional Dependencies:

person_id \rightarrow (driving_license_no)

Candidate Keys:

{person_id}

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE Courier(  
    person_id          CHAR(11),  
    driving_license_no INT NOT NULL,  
  
    PRIMARY KEY (person_id),  
    FOREIGN KEY (person_id)  
        REFERENCES Employee (person_id)  
        ON DELETE CASCADE  
) ENGINE=InnoDB;
```

3.1.6. Asset

Relational Schema:

Asset(asset_id, name, address, phone_no)

Functional Dependencies:

asset_id \rightarrow (name, address, phone_no)

Candidate Keys:

{asset_id}

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE Asset (  
    asset_id    CHAR(11),  
    name        VARCHAR(50) NOT NULL,  
    address     VARCHAR(300) NOT NULL,  
    phone_no    INT UNSIGNED NOT NULL,  
  
    PRIMARY KEY (asset_id)  
) ENGINE=InnoDB;
```


3.1.7. Warehouse

Relational Schema:

Warehouse (asset_id)

Functional Dependencies: \emptyset

Candidate Keys:

{asset_id}

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE Warehouse(  
    asset_id    CHAR(11),  
  
    PRIMARY KEY (asset_id),  
    FOREIGN KEY (asset_id)  
        REFERENCES Asset (asset_id)  
        ON DELETE CASCADE  
) ENGINE=InnoDB;
```

3.1.8. Branch

Relational Schema:

Branch(asset_id, service_start, service_end)

Functional Dependencies:

asset_id \rightarrow (service_start, service_end)

Candidate Keys:

{asset_id}

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE Branch(  
    asset_id      CHAR(11),  
    service_start TINYINT NOT NULL,  
    service_end   TINYINT NOT NULL,  
  
    PRIMARY KEY (asset_id),  
    FOREIGN KEY (asset_id)  
        REFERENCES Asset (asset_id)  
        ON DELETE CASCADE  
) ENGINE=InnoDB;
```

3.1.9. Shipment

Relational Schema:

Shipment(shipment_no, price, weight, payment_type, payment_side, shipment_date, package_description, delivery_type)

Functional Dependencies:

shipment_no \rightarrow (price, weight, payment_type, payment_side, shipment_date, package_description, delivery_type)

Candidate Keys:

{shipment_no}

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE Shipment(  
    shipment_no          VARCHAR(36),  
    price                INT NOT NULL,  
    weight               INT NOT NULL,  
    payment_type         ENUM('CREDIT_CARD', 'CASH') NOT NULL,  
    payment_side         ENUM('SENDER', 'RECIPIENT') NOT NULL,  
    shipment_date        DATETIME NOT NULL,  
    package_description   VARCHAR(100),  
    delivery_type        ENUM('COURIER', 'SELF_PICK_UP') NOT NULL,  
    shipment_status      ENUM('EN_ROUTE', 'ON_BRANCH', 'DELIVERED', 'DECLINED') NOT NULL,  
  
    PRIMARY KEY (shipment_no)  
) ENGINE=InnoDB;
```

3.1.10. Report

Relational Schema:

Report(shipment_no, report_id, file_date, complaint_text, report_evaluation)

Functional Dependencies:

(shipment_no, report_id) → (file_date, complaint_text, report_evaluation)

Candidate Keys:

{(shipment_no, report_id)}

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE Report(  
    shipment_no      CHAR(20),  
    report_id        CHAR(20),  
    file_date        DATETIME NOT NULL,  
    content_text      TEXT NOT NULL,  
    report_evaluation TINYINT(1),  
  
    PRIMARY KEY (shipment_no, report_id),  
    FOREIGN KEY (shipment_no)  
        REFERENCES Shipment (shipment_no)  
) ENGINE=InnoDB;
```

3.2. Relationship Sets

3.2.1. work_at

Relational Schema:

`work_at(person_id, asset_id)`

Functional Dependencies:

`person_id → (asset_id)`

Candidate Keys:

`{person_id}`

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE work_at(  
    person_id    char(11),  
    asset_id     char(11) NOT NULL,  
  
    PRIMARY KEY (person_id),  
    FOREIGN KEY (person_id)  
        REFERENCES Employee (person_id)  
        ON DELETE CASCADE,  
    FOREIGN KEY (asset_id)  
        REFERENCES Branch (asset_id)  
        ON DELETE CASCADE  
) ENGINE=InnoDB;
```

3.2.2. stored

Relational Schema:

stored(shipment_no, asset_id, enter_time)

Functional Dependencies:

shipment_no \rightarrow (asset_id, enter_time)

Candidate Keys:

{shipment_no}

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE stored(  
    shipment_no CHAR(20),  
    asset_id CHAR(11) NOT NULL,  
    enter_time DATETIME NOT NULL,  
  
    PRIMARY KEY (shipment_no),  
    FOREIGN KEY (shipment_no)  
        REFERENCES Shipment (shipment_no)  
        ON DELETE CASCADE,  
    FOREIGN KEY (asset_id)  
        REFERENCES Branch (asset_id)  
        ON DELETE CASCADE  
)  
ENGINE=InnoDB;
```

3.2.3. path_taken

Relational Schema:

path_taken(shipment_no, asset_id)

Functional Dependencies: \emptyset

Candidate Keys:

{(shipment_no, asset_id)}

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE path_taken(  
    shipment_no CHAR(20),  
    asset_id CHAR(11) NOT NULL,  
  
    PRIMARY KEY (shipment_no, asset_id),  
    FOREIGN KEY (shipment_no)  
        REFERENCES Shipment (shipment_no)  
        ON DELETE CASCADE,  
    FOREIGN KEY (asset_id)  
        REFERENCES Branch (asset_id)  
        ON DELETE CASCADE  
) ENGINE=InnoDB;
```

3.2.4. sent_to

Relational Schema:

`sent_to(shipment_no, person_id)`

Functional Dependencies:

`shipment_no → (person_id)`

Candidate Keys:

`{shipment_no}`

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE sent_to(  
    shipment_no CHAR(20),  
    person_id CHAR(11) NOT NULL,  
  
    PRIMARY KEY (shipment_no),  
    FOREIGN KEY (shipment_no)  
        REFERENCES Shipment (shipment_no)  
        ON DELETE CASCADE,  
    FOREIGN KEY (person_id)  
        REFERENCES Customer (person_id)  
        ON DELETE CASCADE  
) ENGINE=InnoDB;
```


3.2.5. delivers

Relational Schema:

`delivers(shipment_no, person_id)`

Functional Dependencies:

`shipment_no → (person_id)`

Candidate Keys:

`{shipment_no}`

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE delivers(  
    shipment_no CHAR(20),  
    person_id CHAR(11) NOT NULL,  
  
    PRIMARY KEY (shipment_no),  
    FOREIGN KEY (shipment_no)  
        REFERENCES Shipment (shipment_no)  
        ON DELETE CASCADE,  
    FOREIGN KEY (person_id)  
        REFERENCES Courier (person_id)  
        ON DELETE CASCADE  
) ENGINE=InnoDB;
```

3.2.6. orders

Relational Schema:

orders(shipment_no, person_id)

Functional Dependencies:

shipment_no \rightarrow (person_id)

Candidate Keys:

{shipment_no}

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE orders(  
    shipment_no CHAR(20),  
    person_id CHAR(11) NOT NULL,  
  
    PRIMARY KEY (shipment_no),  
    FOREIGN KEY (shipment_no)  
        REFERENCES Shipment (shipment_no)  
        ON DELETE CASCADE,  
    FOREIGN KEY (person_id)  
        REFERENCES Customer (person_id)  
        ON DELETE CASCADE  
) ENGINE=InnoDB;
```

3.2.7. files

Relational Schema:

files(shipment_no, report_id, person_id)

Functional Dependencies:

(shipment_no, report_id) → (person_id)

Candidate Keys:

{(shipment_no, report_id)}

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE files(  
    shipment_no CHAR(20),  
    report_id CHAR(20) NOT NULL,  
    person_id CHAR(11) NOT NULL,  
  
    PRIMARY KEY (shipment_no, report_id),  
    FOREIGN KEY (shipment_no)  
        REFERENCES Shipment (shipment_no)  
        ON DELETE CASCADE,  
    FOREIGN KEY (report_id)  
        REFERENCES Report (report_id)  
        ON DELETE CASCADE,  
    FOREIGN KEY (person_id)  
        REFERENCES UserCustomer (person_id)  
        ON DELETE CASCADE  
) ENGINE=InnoDB;
```

3.2.8. evaluates

Relational Schema:

`evaluates(shipment_no, report_id, person_id)`

Functional Dependencies:

$(shipment_no, report_id) \rightarrow (person_id)$

Candidate Keys:

$\{(shipment_no, report_id)\}$

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE evaluates(  
    shipment_no CHAR(20),  
    report_id CHAR(20),  
    person_id char(11) NOT NULL,  
  
    PRIMARY KEY (shipment_no, report_id),  
    FOREIGN KEY (shipment_no)  
        REFERENCES Shipment (shipment_no)  
        ON DELETE CASCADE,  
    FOREIGN KEY (person_id)  
        REFERENCES Employee (person_id)  
        ON DELETE CASCADE,  
    FOREIGN KEY (report_id)  
        REFERENCES Report (report_id)  
        ON DELETE CASCADE  
) ENGINE=InnoDB;
```

3.3. Multi-valued Attribute Tables

3.3.1. user_credit_cards

Relational Schema:

user_credit_cards(person_id, card_no, ccv2_no, valid_month, valid_year)

Functional Dependencies:

$(\text{person_id}, \text{credit_card}) \rightarrow (\text{ccv2_no}, \text{valid_month}, \text{valid_year})$

Candidate Keys:

{(person_id, credit_card)}

Normal Form:

BCNF

SQL DDL Statement:

```
CREATE TABLE user_credit_cards(  
    person_id      CHAR(11),  
    card_no        CHAR(16),  
    ccv2_no        CHAR(3) NOT NULL,  
    valid_month    CHAR(2) NOT NULL,  
    valid_year     CHAR(4) NOT NULL,  
  
    PRIMARY KEY (person_id, card_no),  
    FOREIGN KEY (person_id)  
        REFERENCES UserCustomer (person_id)  
        ON DELETE CASCADE  
) ENGINE=InnoDB;
```

4. Functional Components

4.1. Use Cases

Shipping Company Data Management System has three types of users. These users are *Customer*, *UserCustomer*, *Employee* and *Managers*. These roles have similarities but they also have specific attributes that are distinguishing them from each other. All *UserCustomers* and *Employees* have to have an account and login with that account in order to use the system. However, regular *Customers* cannot perform login operation, they must first sign up. System will provide different functionalities depending on the type of the user.

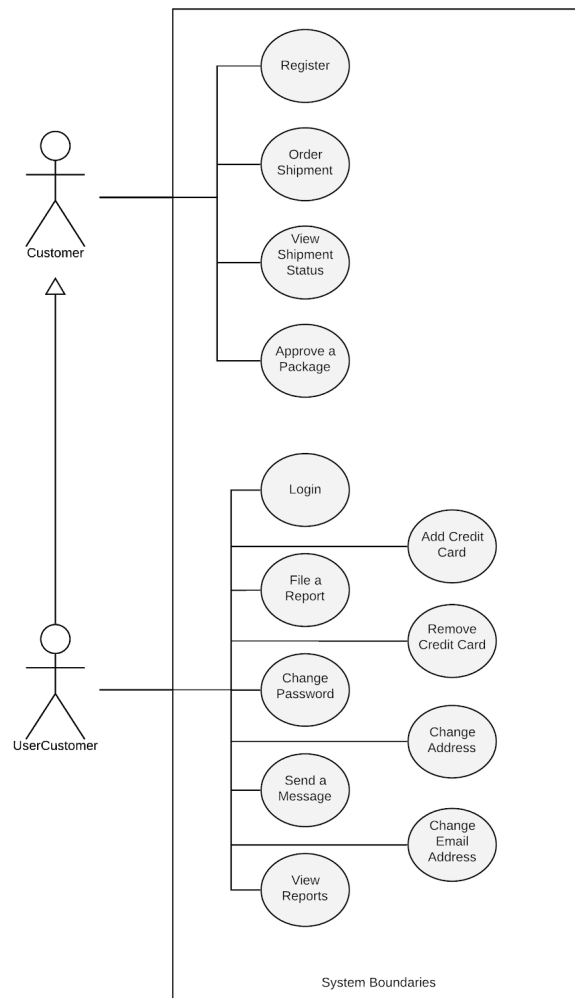
Customer:

- *Customer* can register to the system. They need to enter their *ID*, *email* and *password* upon first registration in order to be able to login.
- *Customers* can place a *shipment* order, select *payment side* as either sender or recipient and *payment type* as either a credit card or cash. Also *Customers* can specify *delivery types* to be either by *Courier* or self pick up.
- *Customers* can view transportation process and status. They can access such information via their *shipment no*. Package's distance to recipient, and estimated duration left can be seen by the *Customer*.
- *Customers* can approve a package when they receive it.
- *Customers* can view additional information about package, these are *package description*, *payment type*, *payment side*, *delivery type* via *shipment no*.
- *Shipment* status will be shown as: EN ROUTE or ON BRANCH or DELIVERED or DECLINED.

UserCustomer:

UserCustomers are *Customers* by inheritance, however, they have additional functionalities such as:

- *UserCustomers* can log in to their accounts with their ID and password.
- *UserCustomers* can change their email addresses.
- *UserCustomers* can also add and remove their *credit cards*.
- *UserCustomers* can change their *password* from their profile page.
- *UserCustomers* can change their address.
- *UserCustomers* can file a *Report* for broken or damaged packages.
- *UserCustomers* can view previous *Reports*.

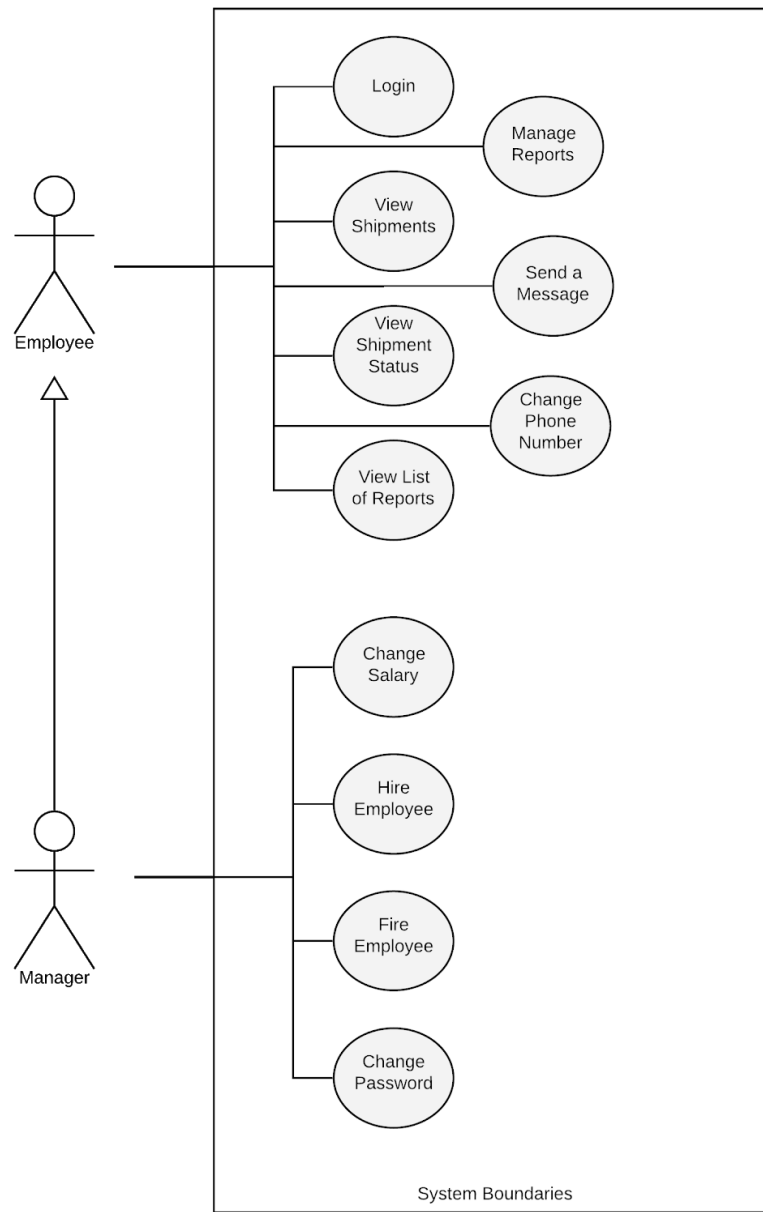


Employees:

- *Employees* can login to the system with their *ID* and *password*.
- *Employees* can view list of *Shipments* that are on the *Branch* they are working at.
- *Employees* can view the list of *Reports* which are filed by *Customers*. *Employees* can view these reports as: ON GOING or FINISHED.
- *Employees* can manage any *Reports* in the pool.
- *Employees* can communicate with *Customers* via messages. *Employees* can send and receive messages through a chat.
- *Employees* can evaluate *Reports* as positive or negative.

Managers:

- *Managers* can hire/fire *Employees* (a.k.a. add or delete employee tuples).
- *Managers* can view salaries of *Employees*. Also *Managers* can alter the value of the salaries.
- *Managers* can change passwords.



4.2. Algorithms

Customer – Related Algorithms

Customers will be kept tracked in the data management system. Data regarding the customer will be kept in the *Customer* table. When a customer wants to send a package, their *Customer* ID and package information (delivery type, delivery destination, recipient name) will be taken from the *Customer*.

Customers will be greeted with an index page where statistics will be shown. In terms of statistics, the number of shipments with respect to time and the number of reports filed by the current user will be shown. *Customers* who filed a report about a broken or damaged packages should be tracked so that their problems can be solved in order to maintain customer satisfaction.

Shipment – Related Algorithms

Every package has its own *shipment id* kept in Shipping Company Data Management System. Keeping information about the package description, delivery type, payment type, payment side, and shipment status

Packages who got complaints from the *UserCustomer* should be identified so that *Employees* can resolve the issue about the package. Every time a *Customer* places a new *Shipment* order, that *Shipment* will be shown at the index page. After each delivery and customer report *Shipment* table should be updated. Even if after *Employees* help a *UserCustomer* wants to decline a *Shipment*, then the *Shipment* should be deleted and related package should be rolled back.

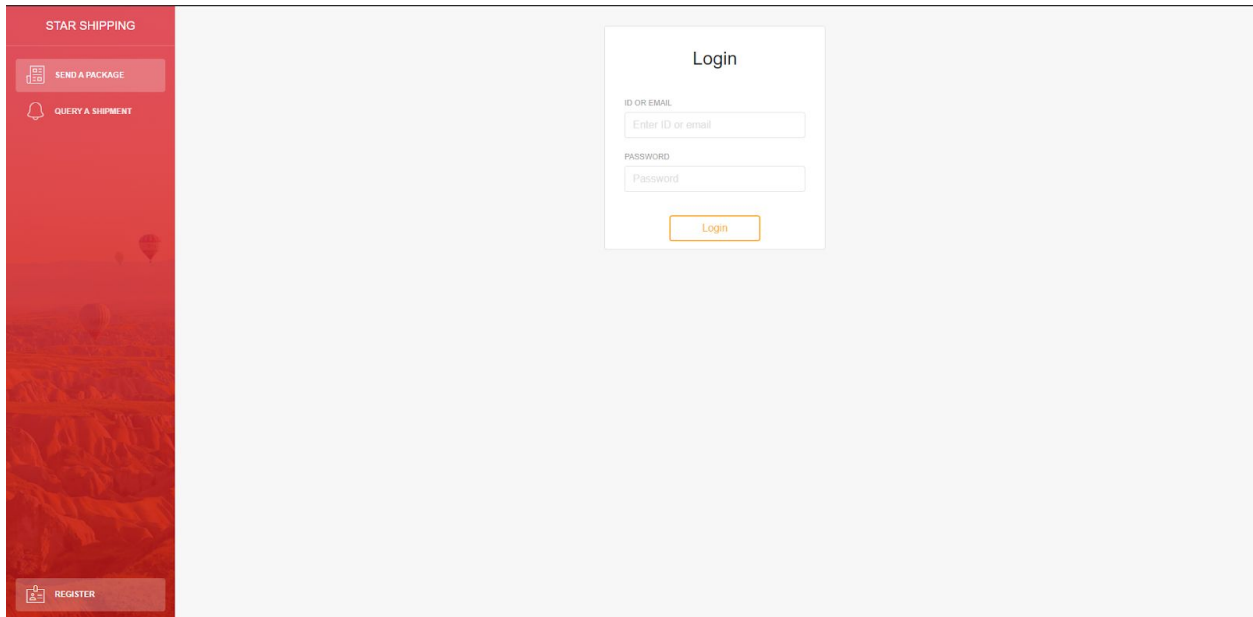
Algorithms to Meet Miscellaneous Logical Requirements

To prevent logical errors, system should be careful about the propagating insert statement from child to parent table via triggers. One example potential error case occurs when a *UserCustomer* is inserted, corresponding *Person* tuple should be inserted to *Person* table as well.

Date of complaint reports are also important. *Customer* service should be cautious, answering to these complaints back on time. Logically, order date and time of a package should be earlier than the arrival date and time of a package. These kind of restrictions should prevent some of the possibilities of mistakes, especially date and time related logical errors.

5. User Interface Design and Corresponding SQL Statements

5.1. Login



The image shows a user interface for 'STAR SHIPPING'. On the left is a red sidebar with the company name at the top. Below it are three buttons: 'SEND A PACKAGE' with a box icon, 'QUERY A SHIPMENT' with a bell icon, and 'REGISTER' with a person icon. The main area is light gray and contains a white 'Login' form. The form has two input fields: 'ID OR EMAIL' with placeholder text 'Enter ID or email', and 'PASSWORD' with placeholder text 'Password'. A yellow 'Login' button is at the bottom of the form.

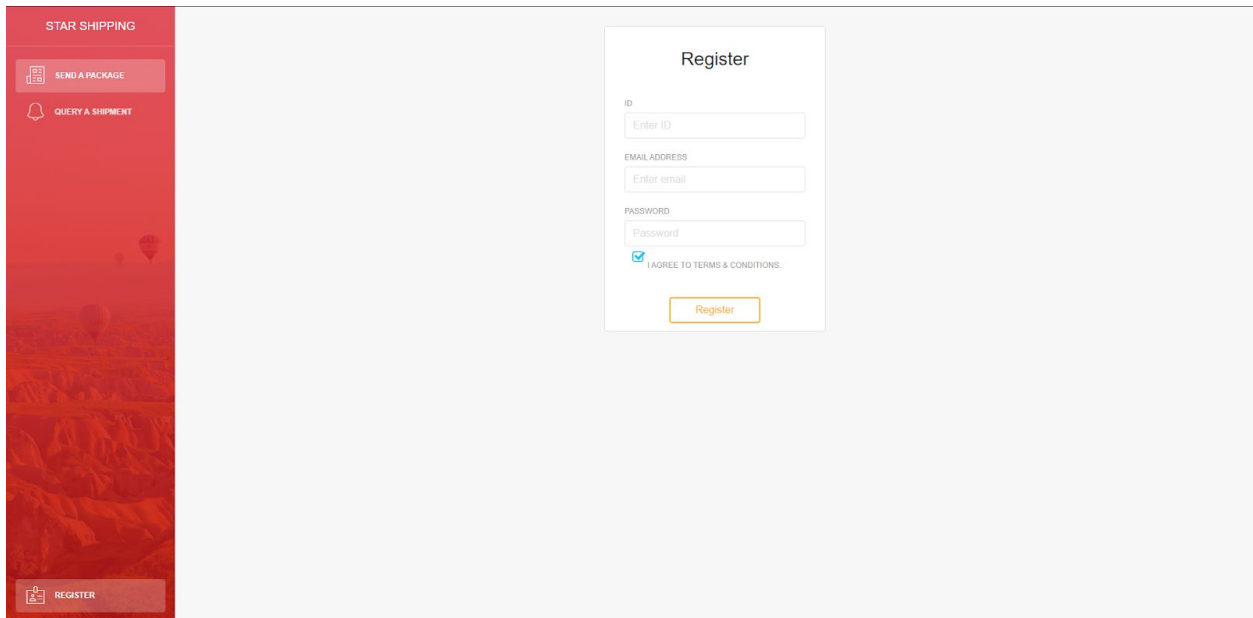
Inputs: @IDField, @passwordField

Login: Used by a either *UserCustomer* or *Employee*, upon providing appropriate ID and password user logs in to STAR Shipping DBMS.

SQL Statement:

```
SELECT person_id
FROM Person
WHERE person_id = @IDField AND password = @PasswordField
```

5.2. Register



The screenshot shows a web application interface for STAR SHIPPING. On the left is a red sidebar with navigation links: 'SEND A PACKAGE' (with a package icon), 'QUERY A SHIPMENT' (with a bell icon), and 'REGISTER' (with a user icon). The main content area is light gray and features a white 'Register' form. The form includes three input fields: 'ID' (placeholder 'Enter ID'), 'EMAIL ADDRESS' (placeholder 'Enter email'), and 'PASSWORD' (placeholder 'Password'). Below these fields is a checked checkbox labeled 'I AGREE TO TERMS & CONDITIONS'. At the bottom of the form is an orange 'Register' button. The background of the sidebar features a faint image of a hot air balloon over a landscape.

Inputs: @IDField, @emailField, @passwordField

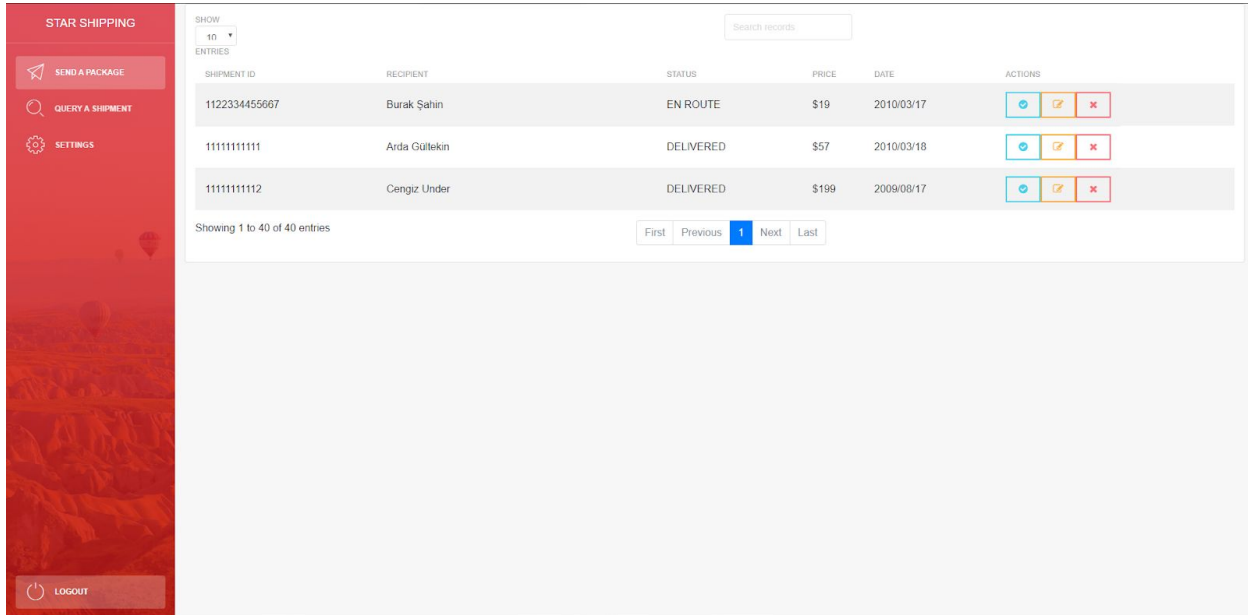
Register: Any outsider with ID number can register to the STAR Shipping DBMS.

Upon providing appropriate ID, email and password user logs in to STAR Shipping DBMS.










SQL Statement:

```
INSERT INTO UserCustomer VALUES (  
    @IDField,  
    @emailField,  
    @passwordField  
);
```

5.3. View Shipment, View Shipment Status and Approve a Package, Decline a Package



The screenshot shows the STAR SHIPPING application interface. On the left is a red sidebar with navigation options: SEND A PACKAGE, QUERY A SHIPMENT, SETTINGS, and a LOGOUT button at the bottom. The main content area displays a table of shipments. At the top of the table, there is a 'SHOW 10 ENTRIES' dropdown and a 'Search records' input field. The table has columns for SHIPMENT ID, RECIPIENT, STATUS, PRICE, DATE, and ACTIONS. There are three rows of data. The first row has a blue tick icon in the ACTIONS column. The second and third rows have yellow checkmark icons. Below the table, it says 'Showing 1 to 40 of 40 entries' and has pagination buttons: First, Previous, 1 (selected), Next, Last.

SHIPMENT ID	RECIPIENT	STATUS	PRICE	DATE	ACTIONS
1122334455667	Burak Şahin	EN ROUTE	\$19	2010/03/17	  
11111111111	Arda Gülekin	DELIVERED	\$57	2010/03/18	  
11111111112	Cengiz Under	DELIVERED	\$199	2009/08/17	  

View Shipment: When user arrives this page the following query runs and displays the unapproved (either EN ROUTE or ON BRANCH), approved (DELIVERED) or declined (DECLINED) shipment entities present in the database.

SQL Statement:

```
SELECT shipment_no, first_name, surname, price, shipment_date, shipment_status  
FROM Shipment NATURAL JOIN sent_to NATURAL JOIN Person
```

View Shipment Status: Same as the above process, we can observe the Shipment Status within the result of the above query.

Approve a Package: This process involves the update of a shipment tuple. To approve a shipment, user must click the blue tick button under the actions thread. The click event will trigger the execution of the following query.

SQL Statement:

```
UPDATE Shipment
SET shipment_status = 'DELIVERED'
WHERE shipment_no = @shipment_no
```

Decline a Package: User declines a package in a similar fashion. User now clicks the red cross button and then Shipment status will be set to DECLINED.

```
UPDATE Shipment
SET shipment_status = 'DECLINED'
WHERE shipment_no = @shipment_no
```

5.4. Order Shipment

The screenshot shows a web application interface for 'STAR SHIPPING'. On the left is a red sidebar with navigation links: 'SEND A PACKAGE', 'QUERY A SHIPMENT', 'SETTINGS', and 'LOGOUT'. The main content area is titled 'Place a Shipment Order' and contains a form with the following fields and controls:

- FIRST NAME:** Text input with 'Burak' entered.
- LAST NAME:** Text input with 'Şahin' entered.
- ADDRESS:** Text input with 'Bilkent University Central Campus B building' entered.
- CITY:** Text input with 'Ankara' entered.
- COUNTRY:** Text input with 'Turkey' entered.
- POSTAL CODE:** Text input with 'ZIP Code' entered.
- PACKAGE DESCRIPTION:** Text input with 'Computer drive' entered.
- WEIGHT:** Text input with '5 kg' entered.
- PAYMENT TYPE:** Dropdown menu with 'Select Payment Type' selected.
- PAYMENT SIDE:** Dropdown menu with 'Select Payment Side' selected.
- DELIVERY TYPE:** Dropdown menu with 'Select Delivery Type' selected.
- Price:** Displayed as '\$90'.
- Proceed:** A blue button to submit the order.

Inputs: @firstNameField, @lastNameField, @addressField, @cityField, @countryField, @postalCodeField, @packageDescriptionField, @weightField,

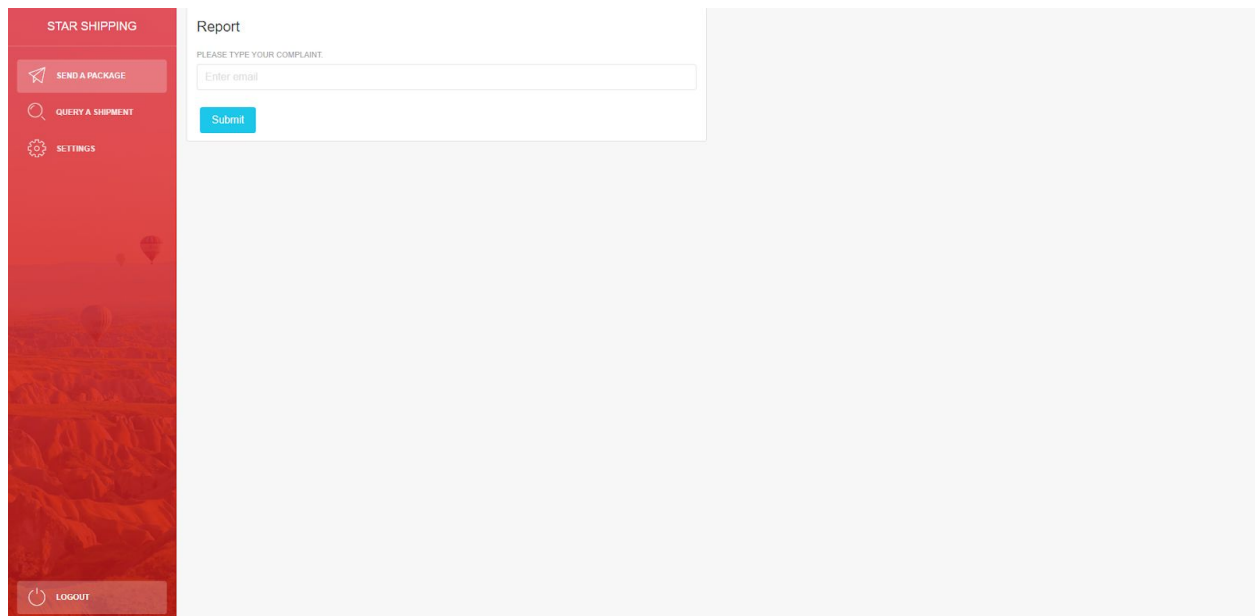
@paymentTypeMenuItem, @paymentSideMenuItem, @deliveryTypeMenuItem,
@priceField

Order Shipment : The ordering of shipment is pretty straightforward, we insert shipment tuple into Shipment table.

SQL Statement:

```
INSERT INTO Shipment VALUES (  
    UUID() ,  
    @priceField,  
    @weightField,  
    @paymentTypeMenuItem,  
    @paymentSideMenuItem,  
    NOW() ,  
    @packageDescriptionField,  
    @deliveryTypeMenuItem,  
    'ON_BRANCH'  
);
```

5.5. File a Report



The screenshot displays the 'STAR SHIPPING' web application interface. On the left is a red sidebar with navigation links: 'SEND A PACKAGE' (with a paper plane icon), 'QUERY A SHIPMENT' (with a magnifying glass icon), 'SETTINGS' (with a gear icon), and 'LOGOUT' (with a power icon). The main content area is titled 'Report' and contains a form with the instruction 'PLEASE TYPE YOUR COMPLAINT.' Below this is a text input field with the placeholder 'Enter email' and a blue 'Submit' button. The background of the main area is a light gray with a faint, abstract pattern.

Inputs: @textField

File a Report : Middle action button, shown in Section 5.4, is used to file a report about the Shipment tuple of interest. User provides only the complaint/report as a text value.

SQL Statement:

```
INSERT INTO Report(  
    @shipmentID,  
    UUID(),  
    NOW(),  
    @contentTextField,  
    FALSE  
);
```

5.6. Add/Remove Credit Card

STAR SHIPPING

SEND A PACKAGE

QUERY A SHIPMENT

SETTINGS

LOGOUT

Add/Remove Credit Card

FIRST NAME ON THE CARD: Burak

LAST NAME ON THE CARD: Şahin

CARD NO: 1111 2222 3333 4444

CCV2: 111

EXP. MONTH: 06

EXP. YEAR: 2020

Add

My Credit Cards

#	CARD ENDING WITH	DATE
1	0963	06/2020
2	1313	08/2021

inputs: @nameField, @surnameField, @cardNoField, @expMonthField,
@expYearField, @ccv2Field

Add/Remove Credit Card: A UserCustomer can enter and save his/her card for the future ease of use. He/she must fill the necessary information fields in order to add a card. To remove it, current user simply selects the cross sign next to the corresponding row for the credit card table. the person_id is already known, which is the current UserCustomer.

SQL Statement:

```
INSERT INTO user_credit_cards(  
    @person_id,  
    @cardNoField,  
    @ccv2Field,  
    @expMonthField,  
    @expYearField  
);
```

```
DELETE FROM user_credit_cards  
WHERE person_id = @person_id AND card_no = @card_no;
```

5.7. Change Address, Phone Number, Email Address and Password

inputs: @passwordField, @emailField, @phoneNumberField

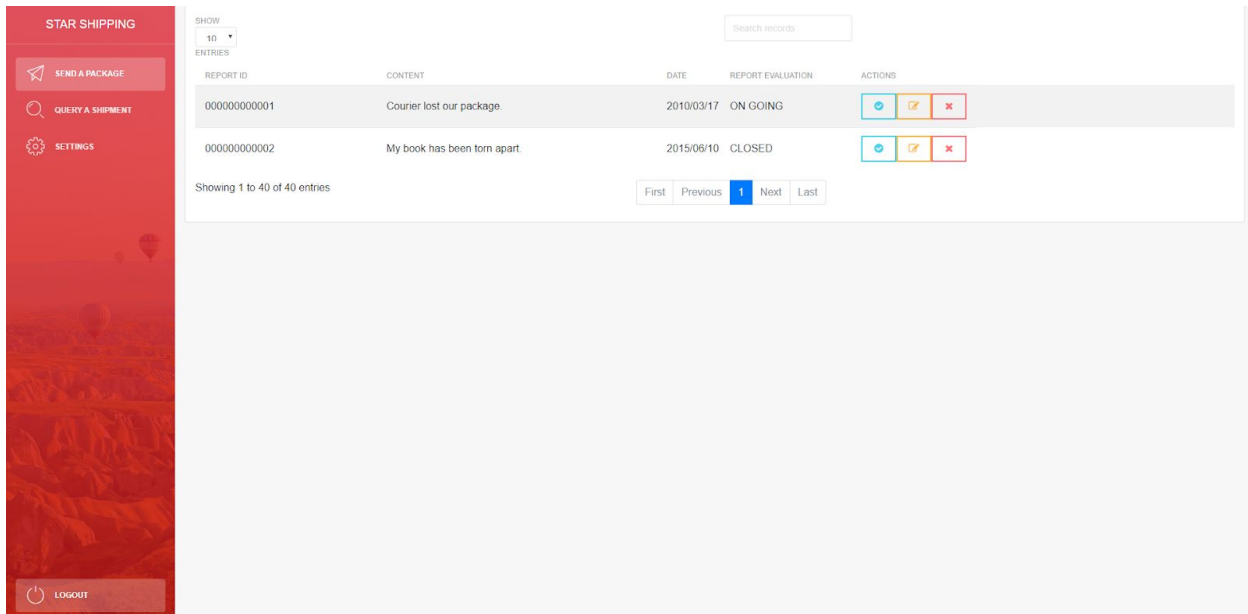
Change Address, Phone Number, Email Address and Password: In the settings page there are separate fields for each type we are changing the value of. User simply

SQL Statement:

```
UPDATE UserCustomer
SET email = @emailField, password = @passwordField
WHERE person_id = @person_id

UPDATE Person
SET address = @addressField, phone_no = @phoneNumberField
WHERE person_id = @person_id
```

5.8. View and Manage Reports



The screenshot shows a web application interface for 'STAR SHIPPING'. On the left is a red sidebar with navigation links: 'SEND A PACKAGE', 'QUERY A SHIPMENT', 'SETTINGS', and 'LOGOUT'. The main content area displays a table of reports. At the top of the main area, there is a 'SHOW' dropdown set to '10 ENTRIES' and a 'SEARCH RECORDS' input field. The table has columns for 'REPORT ID', 'CONTENT', 'DATE', 'REPORT EVALUATION', and 'ACTIONS'. It contains two rows of data. Below the table, it says 'Showing 1 to 40 of 40 entries' and has pagination controls: 'First', 'Previous', '1' (selected), 'Next', and 'Last'.

REPORT ID	CONTENT	DATE	REPORT EVALUATION	ACTIONS
000000000001	Courier lost our package.	2010/03/17	ON GOING	🔍 📝 ✖
000000000002	My book has been torn apart.	2015/06/10	CLOSED	🔍 📝 ✖

View/Manage Reports: Employee users have the right to view all of the reports in the database. We can change the status of the reports by setting it to either TRUE or FALSE meaning that the case remains open or closed. @shipment_no is the shipment_foreign key which is already stored in Report tuple.

SQL Statement:

```
UPDATE Report
SET report_evaluation = TRUE
WHERE shipment_no = @shipment_no
```

6. Advanced Database Components

6.1. Views

6.1.1. Employee's Customer View

Employee can see information about any customer. However, Employee should not be able to change information of any customer. When an Employee member logs in to our system, this view will be used to list customers.

```
CREATE VIEW employee_userCustomer AS
SELECT person_id, email
FROM UserCustomer T
WHERE T.person_id = @person_id
```

6.1.2. Customer's Shipment View

Customers can only see the shipments they have ordered. They cannot view other customer's shipments. In order to accomplish this we have the following view.

```
CREATE VIEW customer_shipment AS
SELECT *
FROM Shipment
WHERE shipment_no = (SELECT shipment_no
                     FROM Shipment
                     NATURAL JOIN sent_to
                     WHERE person_id = @person_id)
```

6.1.3. Courier's Shipment View

Couriers can only see the shipments they are delivering. They cannot view other shipment information. In order to accomplish this, we have the following view.

```
CREATE VIEW courier_shipment AS
  SELECT *
  FROM Shipment
  WHERE shipment_no = (SELECT shipment_no
                      FROM Shipment NATURAL JOIN delivers
                      WHERE person_id = @person_id)
```

6.1.4. Employee's Report View

Employees are limited by the amount of reports that they can view. This is done due in order to prevent from one Employee having huge workload. So we have the following view.

```
CREATE VIEW employee_report AS
  SELECT *
  FROM Report
  WHERE report_id IN (SELECT report_id
                     FROM Report NATURAL JOIN evaluates
                     WHERE person_id = @person_id)
```

6.2. Reports

6.2.1. Total Number of Shipments and Total Money Spent by Customers

```
WITH customers_and_shipments(person_id, name, totalAmount) AS
    (SELECT c.person_id, c.name, s.totalAmount
     FROM Shipment s NATURAL JOIN sent_to NATURAL JOIN Person c)
SELECT person_id, name, COUNT(*) AS shipCount, SUM(price) as totalAmount
FROM customers_and_shipments
GROUP BY person_id
```

6.3. Triggers

- For every insertion of Shipment there must be a corresponding recipient Customer tuple.
- When a UserCustomer gets altered, corresponding Person tuple should be modified by a trigger as well.
- When a Courier gets altered, corresponding Person tuple and Employee should be modified as well.
- When a shipment is added corresponding total number of shipments statistical data must be updated

6.4. Stored Procedures

- A procedure to select all customers with their names. This procedure is created because this piece of code will be used often when we need to deal with the customer table.

```
CREATE PROCEDURE selectAllCustomers
    @name NVARCHAR(30) AS
    SELECT *
    FROM Customer
    WHERE name = @name
GO;
```

- Similar to the first procedure, second procedure selects all employees with their names. This procedure is created because this piece of code will be used often when we need to deal with the employee table.

```
CREATE PROCEDURE selectAllEmployees
    @name NVARCHAR(30) AS
    SELECT *
    FROM Employee
    WHERE name = @name
GO;
```

- Another procedure will be used when person table is modified. This procedure will update the rows in the person table.


```

ALTER PROCEDURE modifyPersonTable(
    @person_id      INTEGER,
    @first_name     VARCHAR(20),
    @surname        VARCHAR(20),
    @address        VARCHAR(50),
    @phone_no       INTEGER,
    @StatementType  NVARCHAR(20)=''
) AS BEGIN
    IF @StatementType = 'INSERT' BEGIN
        INSERT INTO PERSON(person_id, first_name,
                           surname, address, phone_no)
        VALUES(@person_id, @first_name, @surname,
               @address, @phone_no)
    END

    IF @StatementType = 'SELECT' BEGIN
        SELECT * FROM Person
    END

    IF @StatementType = 'UPDATE' BEGIN
        UPDATE Person
        SET     first_name = @first_name, surname = @surname,
               address = @address, phone_no = @phone_no
        WHERE person_id = @person_id
    END
    ELSE IF @StatementType = 'DELETE' BEGIN
        DELETE FROM Person WHERE person_id = @person_id
    END
END

```

6.5. Constraints

- The system cannot be used without logging in.
- The usernames of the employee are their personal identity numbers.
- A customer service employee can be responsible from three customers at once but no more.
- Returned shipments cannot be count as delivered.
- Arrival date of a shipment cannot be earlier than the date shipment was

- made on.
- A shipment's courier cannot be changed when the current courier is on
- his/her way.
- Solved complaints cannot be reserved in the reports pool.
- Password must be alphanumeric.

7. Implementation Plan

MySQL will be the database implementation of our choice. For application layer we will use JavaScript along with React framework.

8. References

- [1] M. Fowler, "Class Table Inheritance," *P of EAA: Class Table Inheritance*. [Online]. Available: <https://martinfowler.com/eaCatalog/classTableInheritance.html>. [Accessed: 18-Nov-2019].
- [2] M. Fowler, "Single Table Inheritance," *P of EAA: Single Table Inheritance*. [Online]. Available: <https://martinfowler.com/eaCatalog/singleTableInheritance.html>. [Accessed: 18-Nov-2019].
- [3] M. Fowler, "Concrete Table Inheritance," *P of EAA: Concrete Table Inheritance*. [Online]. Available: <https://martinfowler.com/eaCatalog/concreteTableInheritance.html>. [Accessed: 18-Nov-2019].