COMP 306 - Term Project Report - Group 5

Ahmet Can Turgut - 40616
Damla Ovek - 31587
Cevat Mert Dokumcu - 27864

https://github.com/acanturgut/Database_Design_for_Cargo_Company

Instructor: Attila Gürsoy
Responsible TA: Ali Tuğrul Balcı

Introduction

This project aims to the achieve learning database management systems under creating cargo company web page. In this case we are starting to design our ER diagram initially. Then we start to create our company website and database into provided server (istavrit.eng.ku.edu.tr). This project requires both frontend and backend development process. So while implementing web page, both employees and customers needs were considered. For the easy implementing design part twitter Bootstrap is used. Other cases handled with PHP and SQL procedures, triggers and queries.

Design

In this project we are responsible to achieve below tasks.

Customers:

- (DONE) Customers should be able to authenticate themselves as users of the system through a login page, asking for their username and password.
- (DONE) Customers should be able to query information about their packages via a unique tracking number (e.g. whether it is delivered, the estimated date of arrival, date of shipment, cost, shipment details).
- (DONE) Customers should be able to view their personal account information (their recorded name –first, middle, and last-, address, phone number, e-mail, shipment history, and shipments that have not delivered yet).
- (DONE) Customers should be able to view their bills.

Employee:

- (DONE) Employees should be able to authenticate themselves as users of the system through a login page, asking for their username and password.
- (DONE) Employees should be able to enter the shipment information via a web-page.
- (DONE) Employees can only see the shipments for which they are the source or the destination of.
- (DONE) Employees should be able to create accounts for the customers via a web interface.
- (DONE) Employees can change the status of a shipment by marking them as delivered.
- (DONE) Employees should be able to generate a bill for each customer.
- (DONE) The billing procedure for each customer depends on whether they have a contract or not (contracted: 1 month / regular: current shipment).

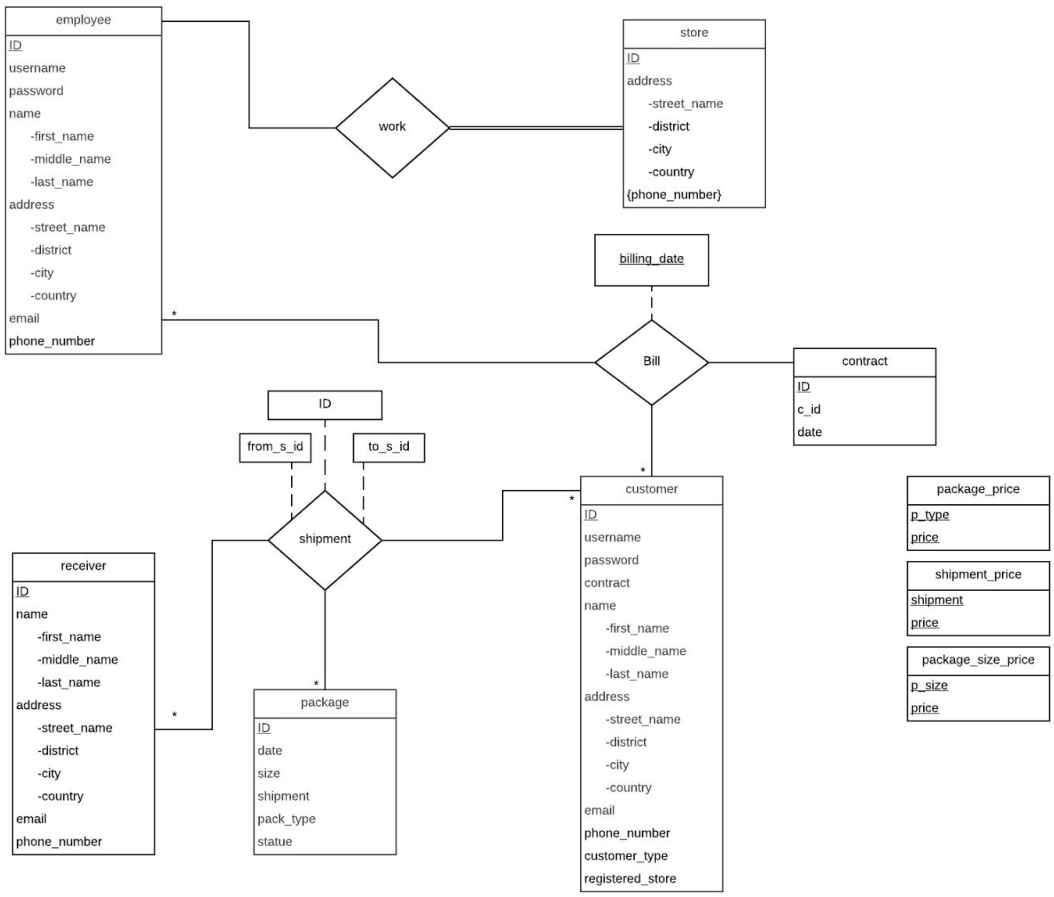
System:

- (DONE) You should create employees and stores manually, i.e. not through a web-page.
- (DONE) The system should automatically calculate the cost of a shipment depending on the package type, weight, content and mode of delivery for each package in the shipment.
- (DONE) The system should automatically calculate the estimated time of delivery depending on the shipment date and the mode of delivery.
- (DONE) System should assign a unique tracking number for each shipment, consisting of 9 numbers starting with a nonzero number.
- (DONE) System should assign a unique ID for each store, employee, and customer.

Constraints:

- (DONE) Both customers and employees will use a common login page. You should differentiate user types through their login information.
- (DONE) Each user (customer or employee) should have a distinct username.
- (DONE) Each user should have a valid password which consists of at least 6 alphanumeric characters.
- (DONE) Each customer is associated with a unique customer ID.
- (DONE) Some customers can have a contract with the company. In this case, they are billed monthly. Otherwise, customer will pay in the store at the time of shipment.
- (DONE) Each store has a unique ID.
- (DONE) Each store has only one employee but each employee is not necessarily associated with a store.
- (DONE) Each store has one address and may have one or more phone numbers.
- (DONE) Each employee has a unique ID.
- (DONE) Each user and employee should have a home address, an e-mail address and a phone number, which cannot be left blank.
- (DONE) The system should store the addresses as street name, district, city, and country.
- (DONE) Each shipment/order can contain one or more packages; however, each package belongs to a single shipment.
- (DONE) Each package has a type (flat, envelope, small box, mid-sized box, large box), content (hazardous, international and normal) and weight. The cost of the package depends on the package type and package content, and the cost of the shipment is the sum of the cost of each package.
- (DONE) Each shipment has a unique tracking number.
- (DONE) Each shipment must have the sender and receiver information with source and destination store. Receivers are identified by their name and cell phone number.
- (DONE) Each sender must be a customer. However, receivers are not necessarily customers.

According to tasks above our initial ER diagram is as follows. This diagram is our roadmap to implement our web page and database. Database structure changed in some cases.



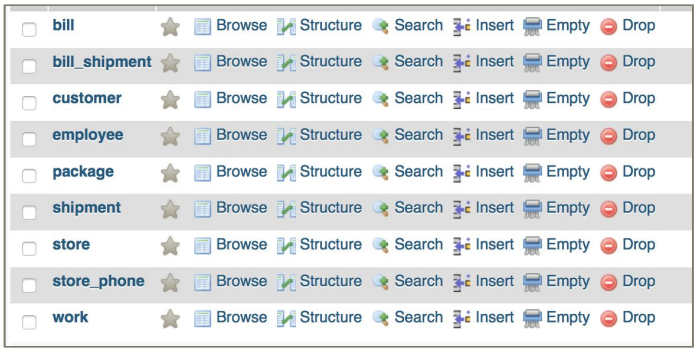
In our design, there are three relations named as “work”, “shipment”, and “bill”. Work relation is a binary relation between employee and store. Participation of store to this relation is total whereas the participation of employee is partial. Shipment is a ternary relation among customer, package, and receiver. They all have many-to-many connections. Shipment also includes from and to attributes related with store. Bill relation is also ternary -if customer is a frequent user- among contract, customer, and employee. Connection between contract and employee is one-to-many. If customer does not have contract, then relation is binary between employee and customer and there is many-to- many connection. Name and address attributes of customer, store, receiver, and employee are composite. Therefore, we design name and address tables which have first_name, middle_name, last_name, and street_name, district, city, country attributes respectively. Furthermore, store has a multi-valued attribute named as phone_number. Hence, we design phone_number table. All attributes of all tables are atomic.

Thus, they are all in first normal form. We normalise package table in order to eliminate repetition of information about package price, based on size and content/type, and shipment price and package table becomes in third normal form. Also, we use Boyce-Codd normal form for bill table. It only has c_id attribute to reach customer. With this attribute, we can reach all customer related information, the corresponding package, and package related information.

Implementation

Initially we are constructed our tables as follows.

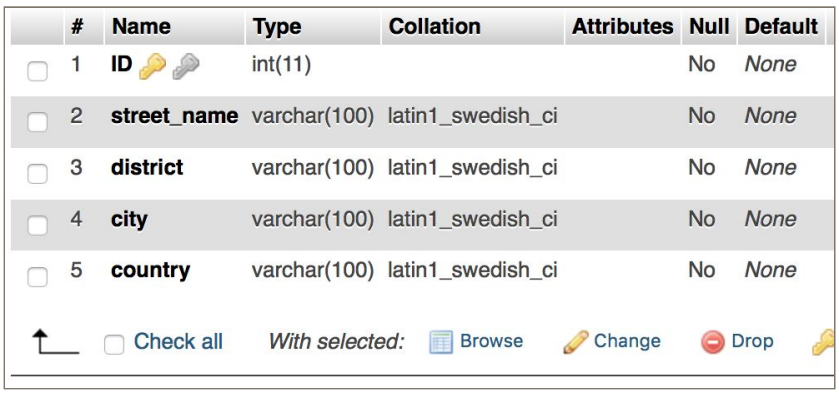
Tables lists are given below.



This tables arranged according to ER diagram since some changes needed while implementations. With implementing procedures some tables removed while implementation.

Then, we populated our tables with columns.

Store:



#	Name	Type	Collation	Attributes	Null	Default
1	ID	int(11)			No	*None*
2	street_name	varchar(100)	latin1_swedish_ci		No	*None*
3	district	varchar(100)	latin1_swedish_ci		No	*None*
4	city	varchar(100)	latin1_swedish_ci		No	*None*
5	country	varchar(100)	latin1_swedish_ci		No	*None*

Check all *With selected:* Browse Change Drop

In store part according to requirements its columns given left hand side figure.

Store phone numbers hold in Store Phone table with store ID.

Store ID’s are both foreign and private for making them unique.

Store Phone:

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	ID	int(11)			No	*None*	
2	phone_number	int(11)			No	*None*	

Store Phone table holds store ID and phone numbers of stores according to one to many relation.

Employee:

#	Name	Type	Collation	Attributes	Null	Default	Extra
1	password	varchar(100)	latin1_swedish_ci		No	*None*	
2	username	varchar(100)	latin1_swedish_ci		No	*None*	
3	email	varchar(50)	latin1_swedish_ci		No	*None*	
4	phone_number	varchar(50)	latin1_swedish_ci		No	*None*	
5	ID	int(11)			No	*None*	AUTO_INCREMENT
6	first_name	varchar(100)	latin1_swedish_ci		No	*None*	
7	middle_name	varchar(100)	latin1_swedish_ci		No	*None*	
8	last_name	varchar(100)	latin1_swedish_ci		No	*None*	
9	street_name	varchar(100)	latin1_swedish_ci		No	*None*	
10	district	varchar(100)	latin1_swedish_ci		No	*None*	
11	city	varchar(100)	latin1_swedish_ci		No	*None*	
12	country	varchar(100)	latin1_swedish_ci		No	*None*	

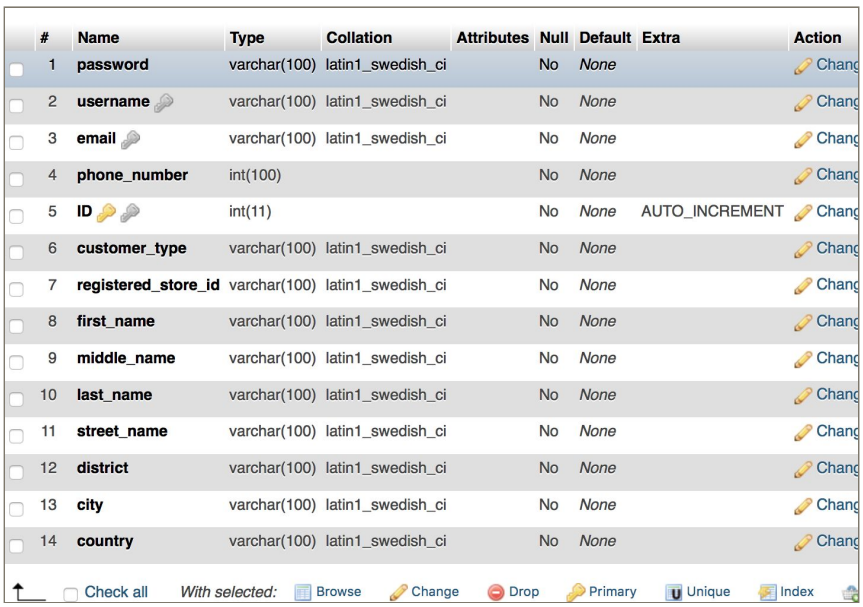
Employe table structured according to requirements. username, email and ID's are unique columns. Since employees have unique ID and usernames.

Work

#	Name	Type	Collation	Attributes	Null	Default
1	s_id	int(11)			No	*None*
2	e_id	int(11)			No	*None*

Work table exist since we need to assign employees to stores. Both s_id and e_id are foreign keys.

Customer:

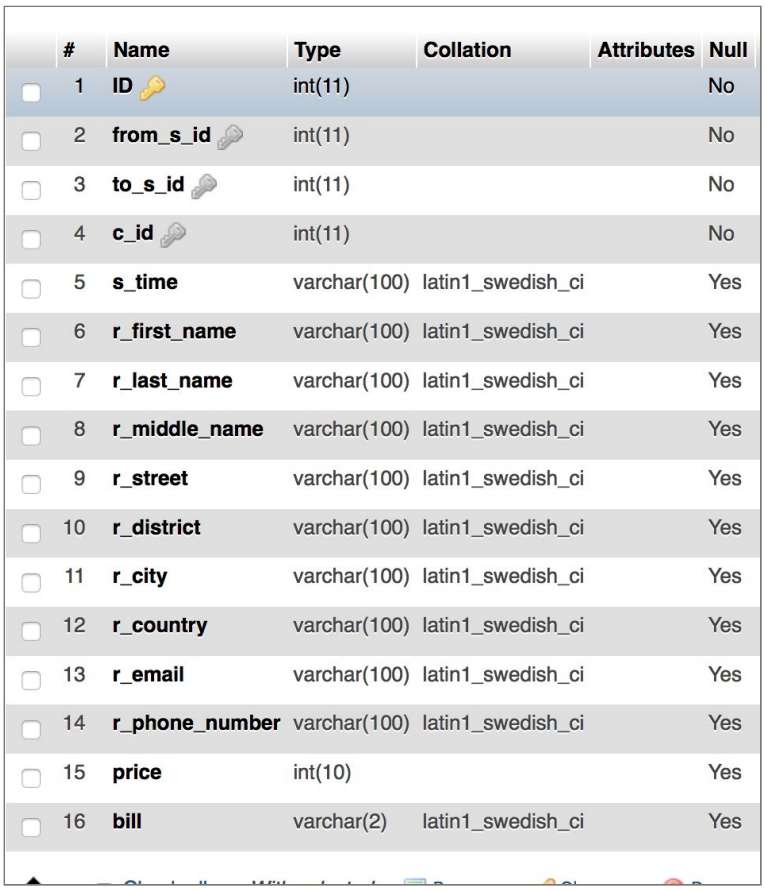


#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	password	varchar(100)	latin1_swedish_ci		No	None		Chang
2	username	varchar(100)	latin1_swedish_ci		No	None		Chang
3	email	varchar(100)	latin1_swedish_ci		No	None		Chang
4	phone_number	int(100)			No	None		Chang
5	ID	int(11)			No	None	AUTO_INCREMENT	Chang
6	customer_type	varchar(100)	latin1_swedish_ci		No	None		Chang
7	registered_store_id	varchar(100)	latin1_swedish_ci		No	None		Chang
8	first_name	varchar(100)	latin1_swedish_ci		No	None		Chang
9	middle_name	varchar(100)	latin1_swedish_ci		No	None		Chang
10	last_name	varchar(100)	latin1_swedish_ci		No	None		Chang
11	street_name	varchar(100)	latin1_swedish_ci		No	None		Chang
12	district	varchar(100)	latin1_swedish_ci		No	None		Chang
13	city	varchar(100)	latin1_swedish_ci		No	None		Chang
14	country	varchar(100)	latin1_swedish_ci		No	None		Chang

Check all With selected: Browse Change Drop Primary Unique Index

Customer table created according to requirements. Same as employee part ID and username's are choose as unique columns.

Shipment

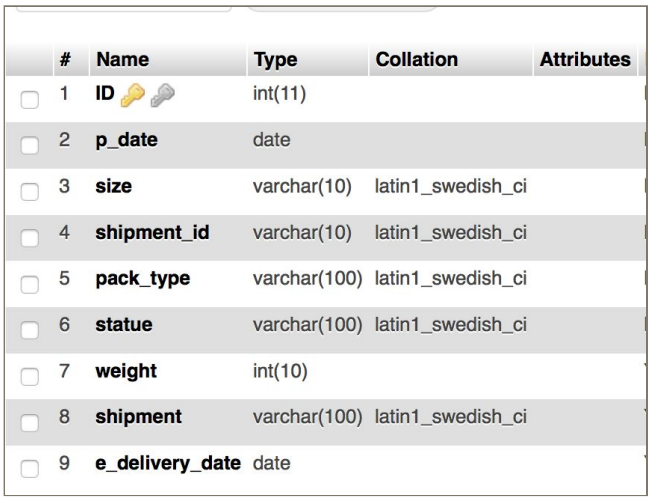


#	Name	Type	Collation	Attributes	Null
1	ID	int(11)			No
2	from_s_id	int(11)			No
3	to_s_id	int(11)			No
4	c_id	int(11)			No
5	s_time	varchar(100)	latin1_swedish_ci		Yes
6	r_first_name	varchar(100)	latin1_swedish_ci		Yes
7	r_last_name	varchar(100)	latin1_swedish_ci		Yes
8	r_middle_name	varchar(100)	latin1_swedish_ci		Yes
9	r_street	varchar(100)	latin1_swedish_ci		Yes
10	r_district	varchar(100)	latin1_swedish_ci		Yes
11	r_city	varchar(100)	latin1_swedish_ci		Yes
12	r_country	varchar(100)	latin1_swedish_ci		Yes
13	r_email	varchar(100)	latin1_swedish_ci		Yes
14	r_phone_number	varchar(100)	latin1_swedish_ci		Yes
15	price	int(10)			Yes
16	bill	varchar(2)	latin1_swedish_ci		Yes

Shipment table created according to requirements. We hold the receiver informations inside this table, since every shipment has only one receiver. We set ID as private key for make it unique. Auto Incrementation also applied on ID column, and it starts from 100000000. Since we need to create unique 9 digit number which is start with non zero character.

Shipments associate with stores so we represent store id's as a receiver and sender store. Also we store customers ID for identifying which package belongs to who.

Package

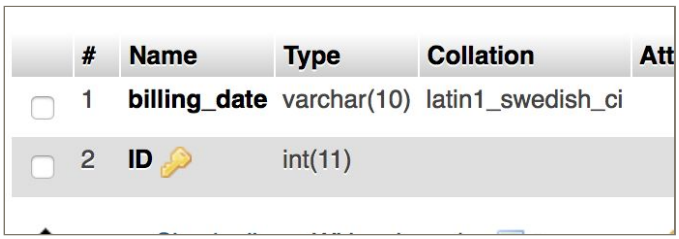


#	Name	Type	Collation	Attributes
1	ID	int(11)		
2	p_date	date		
3	size	varchar(10)	latin1_swedish_ci	
4	shipment_id	varchar(10)	latin1_swedish_ci	
5	pack_type	varchar(100)	latin1_swedish_ci	
6	statue	varchar(100)	latin1_swedish_ci	
7	weight	int(10)		
8	shipment	varchar(100)	latin1_swedish_ci	
9	e_delivery_date	date		

Every shipments populated with packages. So there was a one to many relations can observers between shipment and package.

Reason of that we represent shipment ID's in this table.

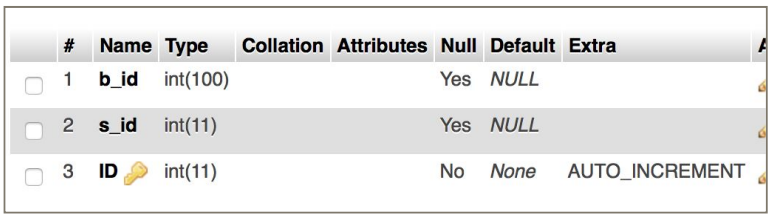
Bill



#	Name	Type	Collation	Att
1	billing_date	varchar(10)	latin1_swedish_ci	
2	ID	int(11)		

In requirements billing_date should be represented on bill. Bill table also holds ID of it self with auto incrementation for helper purposes.

Bill Shipment



#	Name	Type	Collation	Attributes	Null	Default	Extra
1	b_id	int(100)			Yes	*NULL*	
2	s_id	int(11)			Yes	*NULL*	
3	ID	int(11)			No	*None*	AUTO_INCREMENT

Bill shipment is a relation between bills and shipments. For created for providing one to many relation between bill and shipments.

After creating tables we are going to start filling tables with fake stores and employees. Since these fields not filled from web-interface.

Web interface is implemented with PHP and HTML, PHP handles the server side jobs and HTML is handle GUI part.

For that purpose, for employee and customer two different web-interface implemented with HTML and PHP.

Login Page

We started to build the interface from login page. This page is allowed to both employee and customer authenticate themselves to system from same form.

This form is associated with the *auth.php* file. Source code given below.

```php
<?php

require 'connect.php';

$username = $_POST["username"];
$password = $_POST["password"];

echo $username;
echo "<br><br>";
echo $password;

if($username == "admin" && $password == "admin"){

  require 'admin.php';

}else{

  $sql = 'SELECT password, username FROM employee';
  $result = $con -> query($sql);

  if ($result->num_rows > 0) {

    while($row = $result->fetch_assoc()) {

      if($username == $row['username'] && $password == $row['password']){

        header('Location: employee_account.php?username='.$username);

        return;
      }
    }
  }

  $sql = 'SELECT password, username FROM customer';

  $result = $con -> query($sql);

  if ($result->num_rows > 0) {

    while($row = $result->fetch_assoc()) {

      if($username == $row['username'] && $password == $row['password']){

        header('Location: customer_welcome_page.php?username='.$username);

        return;
      }
    }
  }

  header('Location: index.html');
}

?>
```

In *"Auth.php"*, when query information from our database about username and passwords are checked with entered password and username. It starts with employee if it fails it continue with customer table. If both of them fails it directs to initial page (*index.html*).

After implementing login page with authentication system, we continue with employee interface.

Employee Interface

This interface starts with employee welcome page. This page takes the username token from *auth.php*.

Employee interface allows employee to create customers. Create shipment, package. Employees can create bills via web interface as well.

Create Customer via Employee Interface

Let's continue with employee customer creation part. This part is associated with "*op_emloyee_create_shipment.php*" This php file communicates with the server with sending INSERT query. It also fills the customer part.

Customer Insert Query is as follows:

```
"$sql = "insert into customer values(
'$c_password',
'$c_user_name',
'$c_email',
'$c_phone_number',
NULL,
'$c_customer_type',
'$c_store_id',
'$c_first_name',
'$c_mid_name',
'$c_last_name',
'$c_addr_street',
'$c_addr_district',
'$c_addr_city',
'$c_addr_country');";
```

This query triggers SQL trigger function below which does not allow less than 6-digit passwords.

This trigger given below. You can find this trigger "*from check_password_length*".

```
BEGIN
  DECLARE decision varchar(1);
  CALL check_password(new.password,decision);
  IF decision = 'F' THEN
    SIGNAL sqlstate '45000' SET message_text = 'Password must have at least 6 characters!';
    END IF;
END
```

Create New Shipment via Employee Interface

Our PHP files organization allows employees adding shipments with giving customer ID.

First employee create shipment with web interface. This web page directs to "*op_emloyee_create_shipment.php*" file. This file runs INSERT queries to add shipments according to filled creating shipment form from web interface.

This insert query given below.

```
$sql ="INSERT INTO shipment
(ID,s_time,c_id,from_s_id,to_s_id,r_first_name,r_middle_name,r_last_name,r_street,r_district,r_
city,r_country,r_email,r_phone_number,price) VALUES (
NULL,
'$s_shipment_time',
'$s_customer_ID',
'$s_source_location_id',
'$s_r_destination',
'$s_r_first_name',
'$s_r_mid_name',
'$s_r_last_name',
'$s_r_address_street',
'$s_r_address_district',
'$s_r_address_city',
'$s_r_address_country',
'$s_r_email',
'$s_r_phone_number',
NULL
);";
```

The query above triggers the SQL trigger below "*check_customer*" which is from server side.

```
BEGIN
  DECLARE decision varchar(1);
  CALL check_customer_exists(new.c_id,decision);
  IF decision = 'F' THEN
    SIGNAL sqlstate '45000' SET message_text = 'Sender must be a customer!';
    END IF;
END
```

Above trigger code allows to disable to send non-exist customers to assign shipments.

After employee creates shipment, the created shipment is populated with packages via interface. This interface is supported with "*emloyee_create_package.php*". This PHP document gets Shipment ID and shipment time from previously created shipment with the query below.

```
$sql1 = "SELECT max(ID) as maximum FROM shipment";
$result = $con->query($sql1);
$row = mysqli_fetch_assoc($result);
$myVar = $row['maximum'];

$sql3 = "SELECT s_time as timem FROM shipment WHERE ID=(select max(ID) FROM shipment);";
$result = $con->query($sql3);
$row = mysqli_fetch_assoc($result);
$timem = $row['timem'];
```

After retrieving time and shipment ID from server below INSERT query gets sent to server.

```
$sql2 ="INSERT INTO package (ID,shipment,p_date,size,pack_type,shipment_id,statue,weight)
VALUES (
NULL,
'$timem',
'$p_date',
'0',
'$p_type',
'$myVar',
'$p_status',
'$p_weight'
);";
```

Above query triggers the SQL trigger which calculates price, estimated delivery date and size according to the given weight.

"update_price"

```
BEGIN
  DECLARE myPrice int(10);
  CALL calculate_price(new.weight,new.shipment,new.size,myPrice);
  UPDATE shipment SET price=myPrice where s_time=new.shipment;
END
```

The trigger above calls the SQL procedure below "*calculate_price*".

```
BEGIN
  IF size<=1 THEN
    SET price:=1;
  ELSEIF size<=5 THEN
    SET price:=5;
  ELSEIF size<=7 THEN
    SET price:=10;
  ELSEIF size<=10 THEN
    SET price:=20;
  ELSE
    SET price:=20+(size-10)*1;
  END IF;
  IF shipment LIKE 'Overnight' THEN
    SET price:=price+25;
  ELSEIF shipment LIKE 'Two-day' THEN
    SET price:=price+10;
  ELSE
    SET price:=price+0;
  END IF;
  IF type LIKE 'Hazardous' THEN
    SET price:=price+10;
  ELSEIF type LIKE 'International' THEN
    SET price:=price+(price*0.1);
  ELSE
    SET price:=price+0;
  END IF;
END
```

"update_size"

```
BEGIN
  DECLARE p_size varchar(1);
  CALL find_size(new.weight,p_size);
  SET new.size=p_size;
END
```

The trigger above calls the procedure below "*find_size*".

```
BEGIN
  IF weight<= 1 THEN
    SET size:= 'E';
  ELSEIF weight<= 5 THEN
    SET size:= 'S';
  ELSEIF weight<= 7 THEN
    SET size:= 'M';
  ELSE
    SET size:= 'L';
  END IF;
END
```

"set_estimated_delivery_date"

```
BEGIN
  DECLARE my_e_delivery_date date;
  CALL estimate_delivery_Date(new.p_date,new.shipment,my_e_delivery_date);
  SET new.e_delivery_date=my_e_delivery_date;
END
```

Above trigger calls below SQL procedure which is "estimated_delivery_date"

```
BEGIN
  IF shipment LIKE 'Overnight' THEN
    SET e_delivery_date:= DATE_ADD(p_date, INTERVAL 1 DAY);
  ELSEIF shipment LIKE 'Regular' THEN
    SET e_delivery_date:= DATE_ADD(p_date, INTERVAL 7 DAY);
  ELSE
    SET e_delivery_date:= DATE_ADD(p_date, INTERVAL 2 DAY);
  END IF;
END
```

Creating Bill via Employee Web Interface

Employees able to generate bill for both contracted and uncontracted users. Uncontracted users bill generated automatically after shipment created. With below INSERT queries from "gen_bill.php"

```
$sql3 = "INSERT INTO bill (billing_date,ID) VALUES ('".$todaysDate."',NULL);";
```

```
$sql4 = "INSERT INTO bill_shipment(ID,b_id,s_id) VALUES (NULL,(SELECT max(ID) FROM
bill),".$shipmentID.");";
```

Above queries populate both relational bill - shipment table and bill table with corresponded columns.

Contracted customer's bill generated monthly. Employees generate bills every month from generate bill web interface. This web interface connected with "op_emloyee_generate_bill.php" file. This php file runs below queries for generating monthly bill for contracted customers.

```
$sql = "SELECT ID FROM shipment WHERE c_id=".$p_id." AND bill='F';";
```

The query above makes current shipment false to prevent confusion while creating next month's bills.

```
$sql3 = "INSERT INTO bill (billing_date,ID) VALUES ('".$todaysDate."',NULL);";
```

```
$sql4 = "INSERT INTO bill_shipment(ID,b_id,s_id) VALUES (NULL,(SELECT max(ID) FROM bill),
".$shipment_ids[$i].");";
```

```
$sql5 = "UPDATE shipment SET bill='T' WHERE ID=".$shipment_ids[$i];
```

The three queries above populate current month's bills and bill-shipment relation tables.

Tracking Store via Employee Web Page

Employees can track their stores coming and ongoing shipments with web interface. This interface connected with "emloyee_check_shipment.php". This PHP file communicates with server with SELECT SQL queries.

Coming Shipment Queries

```
$sql2 = "SELECT * FROM package WHERE shipment_id IN (SELECT ID FROM shipment WHERE
from_s_id=(SELECT ID FROM store WHERE ID = (SELECT s_id FROM work WHERE e_id=(SELECT ID FROM
employee WHERE username='".$username."'))))";
```

On-Going Shipment Queries

```
$sql3 = "SELECT * FROM package WHERE shipment_id IN (SELECT ID FROM shipment WHERE
to_s_id=(SELECT ID FROM store WHERE ID = (SELECT s_id FROM work WHERE e_id=(SELECT ID FROM
employee WHERE username='".$username."'))))";
```

The queries above retrieve information from server with SELECT statement. This queries retrieve information from shipment and employee tables.

Marking Packages Delivered via Employee Web Interface

Employees can mark packages as delivered which is employee associated with corresponded store. Below UPDATE query allow to update packages mark as delivered.

```
$sql = "SELECT ID FROM package WHERE shipment_id IN (SELECT ID FROM shipment WHERE to_s_id =
(SELECT s_id FROM work WHERE e_id = (SELECT ID FROM employee WHERE
username='".$username."')));";
```

Employee ID allows us to find employees associated store from work relation table.

Customer Interface

This interface starts with employee welcome page. This page takes the username token from *auth.php*.

Customer interface allows customers to see their bills, personal info and details about packages.

Personal Information Page via Customer Web Interface:

Customers can see their personal information. We hold customer usernames for every page of the customer web interface. With this data we are retrive data from database with below query.

```
$sql2 = "SELECT * FROM package WHERE shipment_id IN (SELECT ID FROM shipment WHERE c_id=(SELECT
ID FROM customer WHERE username= '".$username."' )) and statue='Not Delivered';";
```

The query above communicate with customer table and retreive data from there.

Finding Package From Shipment Unique ID via Customer Web Interface

Customers can find their packages via unique shipment ID. In web interface customers can use the search bar to browse and track their packages. This search bar is an HTML form. This form communicates with "*customer_result_show.php*" This php file takes username of the user and tracking ID of the shipment and runs SELECT query to display shipment and package information.

```
$sql = "SELECT store.street_name as to_street,store.district as to_district, store.city as
to_city,store.country as to_country FROM package join shipment join store where
shipment.ID=".$trackID." AND package.shipment_id=".$trackID." AND store.ID=shipment.to_s_id;";
```

The query above retrieves data for shipments and stores it into an array.

```
$sql = "SELECT * FROM package join shipment join store where shipment.ID=".$trackID." AND
package.shipment_id=".$trackID." AND store.ID=shipment.from_s_id";
```

The query above shows the search result.

Customer Bill Information

Customers can see their bills via web interface. When "my bills" tab is clicked all bills up to current date is visible. The queries below assists us to display information about customer bills.

```
include 'connect.php';

$sql3 = "SELECT ID,first_name as fn, middle_name as mn, last_name as lan, street_name as s,
district as d, city as c, country as co, phone_number as p FROM customer WHERE
username='".$username."';";
```

```
$sql1 = "SELECT customer_type FROM customer WHERE username='".$username."';";
```

```
$sql = "SELECT ID FROM shipment WHERE c_id=(SELECT ID FROM customer WHERE
username='".$username."');";
```

```
$sql3 = "SELECT distinct b_id FROM bill_shipment WHERE s_id=".$shipment_ids[$i];
```

```
$sql = "SELECT s_id FROM bill_shipment WHERE b_id=".$bill_ids[$j]
```

```
$sql3 = 'SELECT billing_date FROM bill WHERE ID='.$bill_ids[$j];
```

```
$sql2 = "SELECT customer.first_name as c_f_n,customer.middle_name as c_m_n,customer.last_name
as c_l_n,customer.street_name as c_s,customer.district as c_d,customer.city as
c_c,customer.country as c_co,customer.phone_number as c_p,shipment.ID as
tracking_number,count(package.shipment_id) as num_pack,store.street_name as s_s,store.district
as s_d, store.city as s_c, store.country as s_co,shipment.price, package.p_date from customer
join package join shipment join store where customer.ID=shipment.c_id and
package.shipment_id=shipment.ID and store.ID=shipment.from_s_id AND
shipment.ID=".$shipment_ids2[$i]." group by package.p_date";
```

Thequerie above communicate with almost every table from database and retrieve data to generating both contracted and non contracted customers.

In this project we need to run queries above to show data.

1. Find the customer who has shipped the most packages in the last year.
2. Find the customer who has spent the most money on shipping last year.
3. Sort the customers with a contract according to the money that they have spent on shipping last year.
4. Find the customer that has sent the most hazardous packages.
5. Find the store that has received the most hazardous packages.
6. Find how frequently each store is used by customers for sending packages. Sort these stores from most to least frequently used store.
7. Find the store that has earned the most money from shipping packages.
8. Find the cities that ships the most and least number of the packages between April 20, 2013 and May 15, 2013.
9. Find the city that has received the most packages.

The answer queries of the above are as follows:

1) Find the customer who has shipped the most packages in the last year.

```
SELECT * FROM customer WHERE ID IN (SELECT c_id FROM (SELECT c_id, count(c_id) AS num_ship FROM
shipment WHERE ID IN (SELECT shipment_id FROM package WHERE year(p_date)=2016) GROUP BY c_id)
AS T WHERE num_ship IN (SELECT max(number) FROM (SELECT count(c_id)  AS number FROM shipment
WHERE ID IN (SELECT shipment_id FROM package where year(p_date)=2016) GROUP BY c_id) as S));
```

2) Find the customer who has spent the most money on shipping last year.

```
CREATE VIEW V AS (select shipment.c_id AS ID,sum(shipment.price) AS sum_price from shipment
group by shipment.c_id);

SELECT * FROM customer WHERE ID IN (SELECT id FROM V WHERE sum_price = (SELECT max(sum_price)
FROM V));
```

3) Sort the customers with a contract according to the money that they have spent on shipping last year.

```
SELECT c_id as customer_id, sum(price) as sum_price FROM shipment WHERE c_id IN (SELECT ID FROM
customer WHERE customer_type='on') AND ID IN (SELECT shipment_id FROM package WHERE
year(p_date)=2016) GROUP BY c_id ORDER BY sum_price DESC;
```

4) Find the customer that has sent the most hazardous packages.

```
SELECT * FROM customer WHERE ID IN (SELECT c_id as ID FROM shipment WHERE ID IN (SELECT
shipment_id FROM (SELECT shipment_id, count(pack_type) AS c FROM package WHERE
pack_type="hazardous" GROUP BY shipment_id) AS T WHERE c IN (SELECT max(c) FROM (select
shipment_id, count(pack_type) AS c FROM package WHERE pack_type="hazardous" GROUP BY
shipment_id) AS T)))
```

5) Find the store that has received the most hazardous packages.

```
SELECT * FROM store WHERE ID IN (SELECT to_s_id AS ID FROM shipment WHERE ID IN (SELECT
shipment_id FROM (SELECT shipment_id, count(pack_type) AS c FROM package WHERE
pack_type="hazardous" GROUP BY shipment_id) AS T WHERE c IN (SELECT max(c) FROM (SELECT
shipment_id, count(pack_type) AS c FROM package WHERE pack_type="hazardous" GROUP BY
shipment_id) AS T)))
```

6) Find how frequently each store is used by customers for sending packages. Sort these stores from most to least frequently used store.

```
SELECT from_s_id, count(*) AS frequency FROM shipment GROUP BY from_s_id ORDER BY count(*) DESC
```

7) Find the store that has earned the most money from shipping packages.

```
SELECT * FROM store WHERE ID IN (SELECT S.from_s_id FROM (SELECT from_s_id, sum(price) AS
sum_price FROM shipment GROUP BY from_s_id) AS S WHERE S.sum_price=(SELECT max(sum_price) as
max_price FROM (SELECT sum(price) AS sum_price FROM shipment GROUP BY from_s_id) AS S))
```

8) Find the cities that ships the most and least number of the packages between April 20, 2013 and May 15, 2013.

```
most: SELECT city FROM store WHERE ID IN (SELECT from_s_id FROM shipment WHERE ID IN (SELECT
shipment_id FROM (SELECT shipment_id, count(shipment_id) as num_pack FROM package WHERE p_date
between "2013-04-20" and "2013-05-15" GROUP BY shipment_id) AS S WHERE S.num_pack=(SELECT
max(S.num_pack) FROM (SELECT shipment_id, count(shipment_id) as num_pack FROM package WHERE
p_date between "2013-04-20" and "2013-05-15" GROUP BY shipment_id) AS S)))

least: SELECT city FROM store WHERE ID IN (SELECT from_s_id FROM shipment WHERE ID IN (SELECT
shipment_id FROM (SELECT shipment_id, count(shipment_id) as num_pack FROM package WHERE p_date
between "2013-04-20" and "2013-05-15" GROUP BY shipment_id) AS S WHERE S.num_pack=(SELECT
min(S.num_pack) FROM (SELECT shipment_id, count(shipment_id) as num_pack FROM package WHERE
p_date between "2013-04-20" and "2013-05-15" GROUP BY shipment_id) AS S)))
```

9) Find the city that has received the most packages.

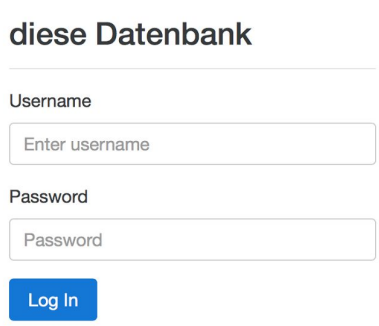
```
CREATE VIEW A AS (select T.r_city AS city,sum(T.num_pack) AS total_pack from (select shipment.r_city AS r_city,S.shipment_id AS shipment_id, S.num_pack AS num_pack from (shipment join (select package.shipment_id AS shipment_id,count(package.shipment_id) AS num_pack from package group by package.shipment_id) S) where (shipment.ID = S.shipment_id)) AS T group by T.r_city)

SELECT city FROM A WHERE total_pack = (SELECT max(total_pack) FROM A)
```

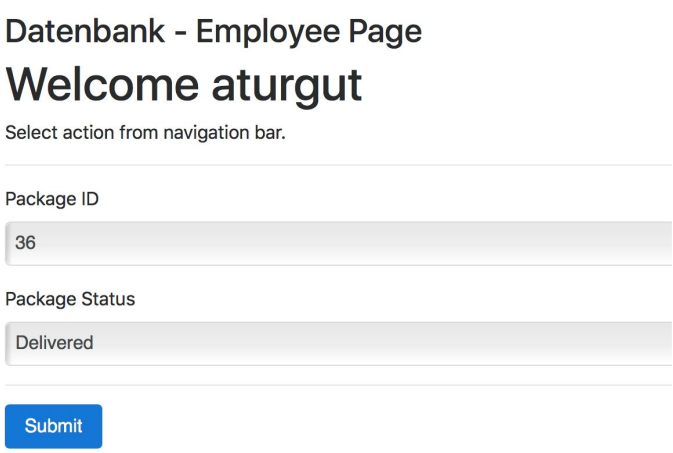
Result

In this project we completely design and develop cargo-company database. Web interface Results given below.

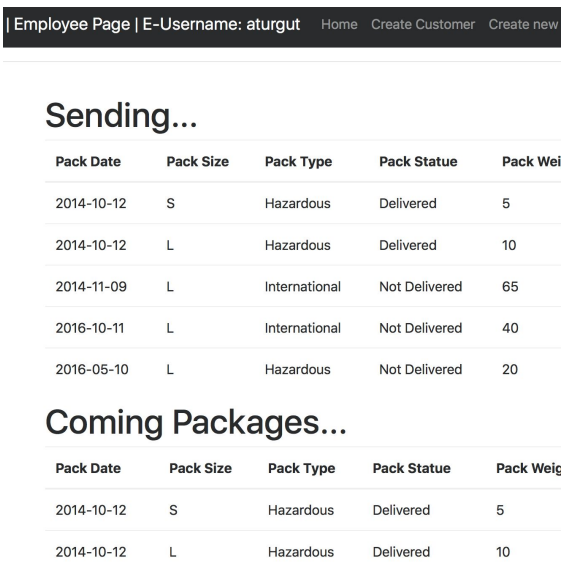
Login page:



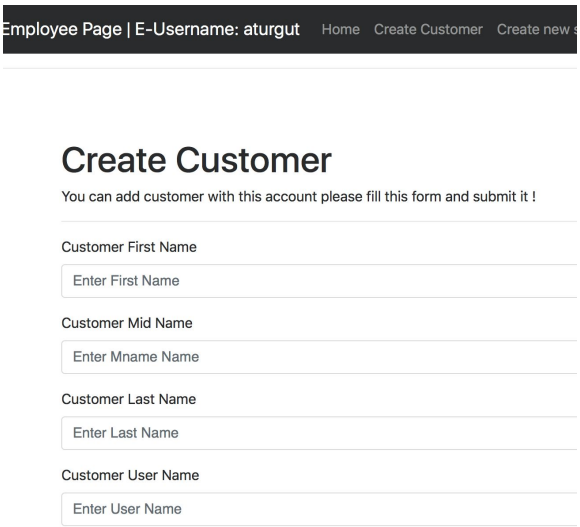
Employee page:



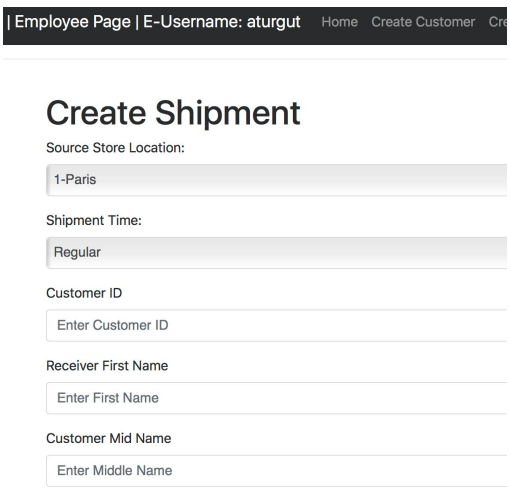
Employee page package edit and welcome page.



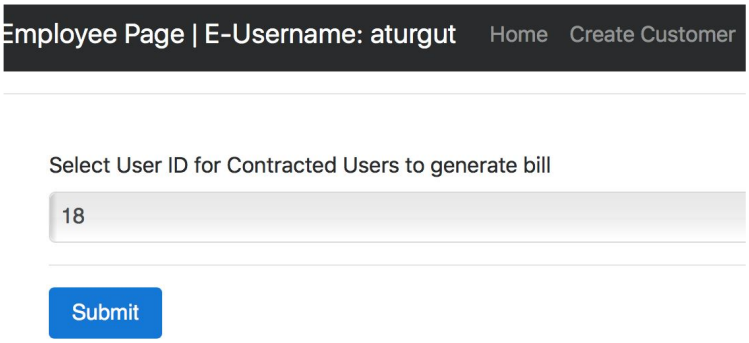
Employee page, store tracking.



Employee page, create customer.

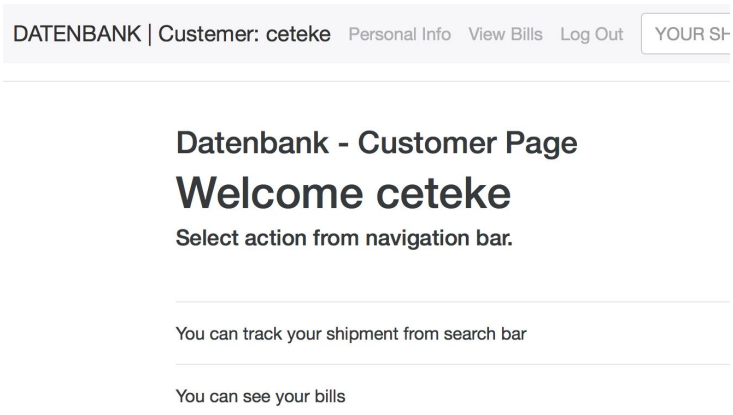


Employee page, create shipment.

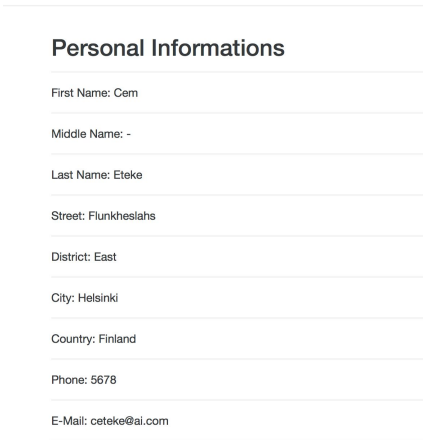


Bill generation for contracted users.

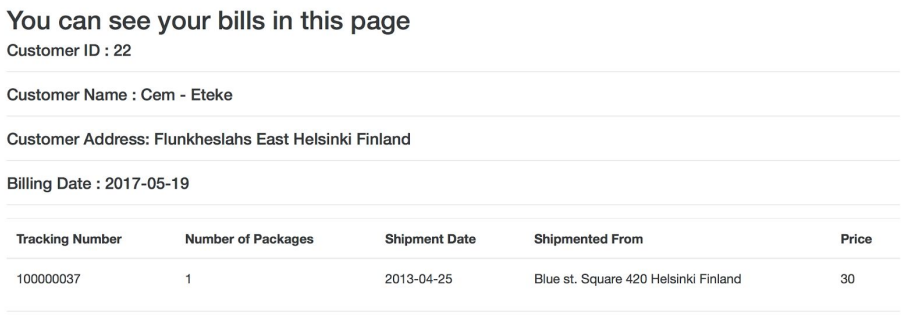
Customer page:



Customer Page, welcome page



Customer Page, personal information



You can see your bills in this page

Customer ID : 22

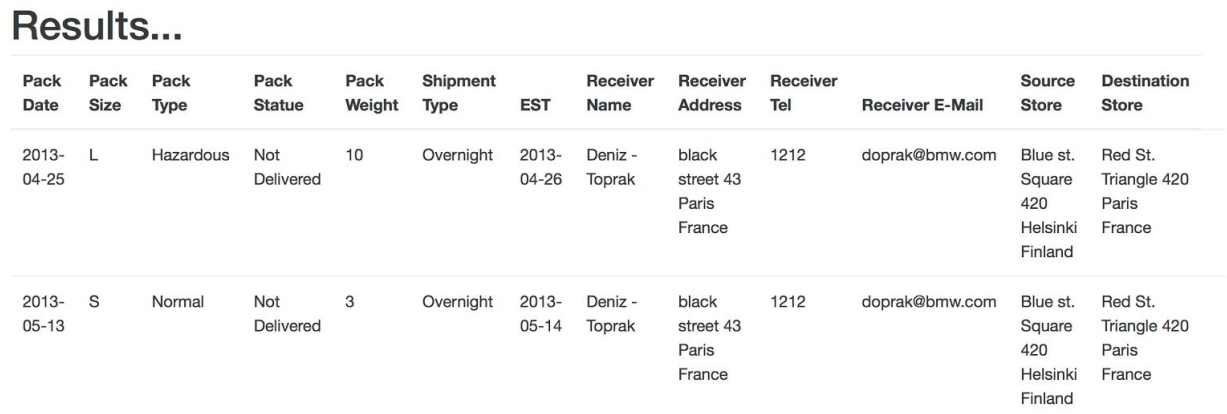
Customer Name : Cem - Eteke

Customer Address: Flunkheslahs East Helsinki Finland

Billing Date : 2017-05-19

Tracking Number	Number of Packages	Shipment Date	Shipmented From	Price
100000037	1	2013-04-25	Blue st. Square 420 Helsinki Finland	30

Customer Page, bill page



Results...

Pack Date	Pack Size	Pack Type	Pack Statue	Pack Weight	Shipment Type	EST	Receiver Name	Receiver Address	Receiver Tel	Receiver E-Mail	Source Store	Destination Store
2013-04-25	L	Hazardous	Not Delivered	10	Overnight	2013-04-26	Deniz - Toprak	black street 43 Paris France	1212	doprak@bmw.com	Blue st. Square 420 Helsinki Finland	Red St. Triangle 420 Paris France
2013-05-13	S	Normal	Not Delivered	3	Overnight	2013-05-14	Deniz - Toprak	black street 43 Paris France	1212	doprak@bmw.com	Blue st. Square 420 Helsinki Finland	Red St. Triangle 420 Paris France

Customer Page, Shipment tracking

Conclusion

In this project we have experienced what it's like to actually build a database from scratch. We have learned in lectur*es* design phase is crucial and we planned our roadmap in accordance with that. Small mistakes or flaws may have caused huge mess so we have spent considerable amount of time to come up with a solid ER diagram and design.

Despite doing our best for the design part we have had moments where we had to go back to the design and fix / change small details. This project was a good challenge for us to test our PHP, and SQL skills and have a deeper understanding about building blocks of web UI design such as HTML, CSS, etc.

What was enjoyable the most?
Using bootstrap.

What was challenging the most?
Every single detail of the bill part. Especially the query writing phase was more difficult than we've anticipated.

References

In this project twitter bootstrap framework was used.

You can access source code after 19 May 12:00 from this github repo:
https://github.com/acanturgut/Database_Design_for_Cargo_Company