Project Team 7- Shipping Management Systems

Team Members: Shubham Amilkanthwar
Aditya Rathi
Upasana Pattnaik
Vrushali Mahuli

PROJECT PROPOSAL

Context:
A shipping company's primary operations include order management, vessel information and consignment allocation. We use a Shipping Management System to aid in seamless functioning of key operations. The database of this system is highly reliable, maintains data integrity, manages redundancy and is secure.

Scope: The database system of a shipping management company aims to manage fleet information, bookings, ship and container tracking and voyage estimation.

Objectives:

The overall objective of the database system is to maintain the data integrity, consistency, security and avoid data redundancy within the shipping system.

The objectives for the Shipping Management System:

- To provide a quote and voyage estimation of the consignment to customers.
- Successfully book consignments to vessels without any conflict.
- To provide customers with consignment tracking
- Maintain vessel records

PROJECT ENVIRONMENT

Software Environment-

Database Client	MySQL Workbench 8.0	8.0.12
Database Server	MySQL Instance: AWS RDS server	5.7.23
Application Server	Apache	2.4.35
Programming Languages	HTML, CSS, PHP, Javascript	
Web Browser	Google Chrome	69.0.3497.100
Operating System	Windows 10	10.0.17134

Hardware Environment-

Ram- 512 mb
Processor- Intel Pentium 500MHZ
Hard disk - 20GB

HIGH LEVEL REQUIREMENTS

Initial user roles

User Role	**Description**
Customer	A Customer places an order with the company and is able to track the order's progress.
Employee	An Employee views, manages, modifies and updates the orders and shipping fleet.

Initial user story descriptions

Story ID	Story description
US1	As a Customer, I want to create an account so that I can access the portal
US2	As a Customer, I want to get quotation for the order
US3	As a Customer, I want to place an order
US4	As a Customer, I want to track my order
US5	As an Employee, I want to add a new vessel information
US6	As an Employee, I want to update a vessel information
US7	As an Employee, I want to update port information
US8	As an Customer, I want to cancel the order

HIGH LEVEL CONCEPTUAL DESIGN

Entities:

1. Customer
2. Employee
3. Order
4. Vessel
5. Port

Relationships:

Customer creates Account
Customer gets quotation for an Order
Customer places an Order
Customer cancels an Order
Employee adds a new Vessel information
Employee updates Vessel Information
Employee updates Port Information

Sprint 1

UPDATED REQUIREMENTS

Story ID	Story description
US1	As a Customer, I want to create an account so that I can access the portal
US2	As a Customer, I want to get quotation for the order
US3	As a Customer, I want to place an order
US4	As a Customer, I want to track my order
US5	As an Employee, I want to add a new vessel information
US6	As an Employee, I want to update a vessel information
US7	As an Employee, I want to update port information
US8	As an Customer, I want to cancel the order
US9	As a Customer, I want to be able to view vessel schedule
US10	As an Employee, I want to add port information
US11	As a Customer, I would like to save my credit-card information
US12	As a Customer, I would like to split my order if my order is large

CONCEPTUAL DESIGN

Entity: **Customer**
Attributes:-

- customer_id (primary key)
- name[composite]
  - first_name
  - last_name

email_id
contact_no
password

Entity: **Vessel**
Attribute-
vessel_id
vessel_name
vessel_capacity

Entity: **Port**
Attribute-
port_id
port_name

Relationship: **Customer** gets information about **Vessel**
Attributes:
departing_time
departing_date
arriving_time
arriving_date
Cardinality: Many to Many
Participation: Customer has partial participation
Vessel has partial participation

Relationship: **Vessel** gets information from **Port**
Cardinality: Many to Many
Participation: Vessel has partial participation
Port has partial participation

LOGICAL DESIGN

Table: **Customer**
Columns:
customer_id (primary key)
first_name
last_name

email_id
contact_no
password

Entity: **Vessel**

Attribute-

vessel_id(primary key)
vessel_name
vessel_capacity
current_port_id(foreign key references **port_id** of **Port** table)
destination_port_id(foreign key references **port_id Port** table)
rate

Foreign key approach to mapping relation as it suited our requirements better than creating a new table which cross-references Vessel and Port.

Entity: **Port**

Attribute-

port_id(primary key)
port_name

Table: **Schedule**

Columns:

schedule_id(primary key)
vessel_id(foreign key references **vessel_id** of **Vessel** Table)
departing_time
departing_date
arriving_time
arriving_date

Foreign key approach to mapping relation as it suited our requirements better than creating a new table which cross-references Customer and Vessel.

SQL QUERIES

1. **Customer views schedule**

>select s.schedule_id as 'Schedule Number', v.vessel_name Vessel, s.vessel_id as 'Vessel ID', v.vessel_capacity as 'Vessel Capacity', p.port_name as 'Departing From',s.departing_time as 'Departing Time', s.departing_date as 'Departing Date', pd.port_name as 'Arriving',s.arriving_time as 'Arriving Time', s.arriving_date as 'Arriving Date',rate
from Schedule s
 inner join Vessel v on s.vessel_id=v.vessel_id
inner join Port p on v.current_port_id=p.port_id
inner join Port pd on v.destination_port_id=pd.port_id;

localhost

Schedule Number	Vessel	Vessel ID	Vessel Capacity	Departing From	Departing Time	Departing Date	Arriving	Arriving Time	Arriving Date	Rate
1	Elizabeth	1	5000	Los Angeles	04:00:00	2018-11-02	Seattle	04:00:00	2018-11-03	10
2	Elizabeth	12	5000	Seattle	15:00:00	2018-11-03	Los Angeles	15:00:00	2018-11-04	10
3	Elizabeth	1	5000	Los Angeles	04:00:00	2018-11-05	Seattle	04:00:00	2018-11-06	10
4	Elizabeth	12	5000	Seattle	15:00:00	2018-11-06	Los Angeles	15:00:00	2018-11-07	10
5	Elizabeth	1	5000	Los Angeles	04:00:00	2018-11-08	Seattle	04:00:00	2018-11-09	10
6	Elizabeth	12	5000	Seattle	15:00:00	2018-11-09	Los Angeles	15:00:00	2018-11-10	10
7	Elizabeth	1	5000	Los Angeles	04:00:00	2018-11-11	Seattle	04:00:00	2018-11-12	10
8	Elizabeth	12	5000	Seattle	15:00:00	2018-11-12	Los Angeles	15:00:00	2018-11-13	10
13	Titanic	4	150000	London	12:00:00	2018-11-01	New York	09:30:00	2018-11-03	30
14	Titanic	6	150000	New York	17:00:00	2018-11-03	London	05:00:00	2018-11-07	30
15	Titanic	4	150000	London	12:00:00	2018-11-07	New York	09:30:00	2018-11-10	30
16	Titanic	6	150000	New York	17:00:00	2018-11-10	London	05:00:00	2018-11-13	30
17	Titanic	4	150000	London	12:00:00	2018-11-13	New York	09:30:00	2018-11-16	30
18	Titanic	6	150000	New York	17:00:00	2018-11-16	London	05:00:00	2018-11-19	30
19	Titanic	4	150000	London	12:00:00	2018-11-19	New York	09:30:00	2018-11-22	30
20	Titanic	6	150000	New York	17:00:00	2018-11-22	London	05:00:00	2018-11-25	30
29	Queenmary	2	2000	Los Angeles	10:00:00	2018-11-01	Long beach	01:30:00	2018-11-01	9
30	Queenmary	13	2000	Long beach	17:30:00	2018-11-01	Los Angeles	21:00:00	2018-11-01	20
31	Queenmary	2	2000	Los Angeles	10:00:00	2018-11-03	Long beach	01:30:00	2018-11-03	9
32	Queenmary	13	2000	Long beach	17:30:00	2018-11-03	Los Angeles	21:00:00	2018-11-03	20
33	Queenmary	2	2000	Los Angeles	10:00:00	2018-11-05	Long beach	01:30:00	2018-11-05	9
34	Queenmary	13	2000	Long beach	17:30:00	2018-11-05	Los Angeles	21:00:00	2018-11-05	20
35	Queenmary	2	2000	Los Angeles	10:00:00	2018-11-07	Long beach	01:30:00	2018-11-07	9
36	Queenmary	13	2000	Long beach	17:30:00	2018-11-07	Los Angeles	21:00:00	2018-11-07	20
37	American Dream	8	7500	Houston	08:30:00	2018-11-10	New York	23:30:00	2018-11-10	20
38	American Dream	9	7500	New York	07:30:00	2018-11-11	Houston	22:30:00	2018-11-11	20
39	American Dream	8	7500	Houston	08:30:00	2018-11-12	New York	23:30:00	2018-11-12	20
40	American Dream	9	7500	New York	07:30:00	2018-11-13	Houston	22:30:00	2018-11-13	20
41	American Dream	8	7500	Houston	08:30:00	2018-11-14	New York	23:30:00	2018-11-14	20
42	American Dream	9	7500	New York	07:30:00	2018-11-15	Houston	22:30:00	2018-11-15	20
43	Eclipse	3	20000	New York	07:30:00	2018-11-07	Seattle	23:30:00	2018-11-08	25
44	Eclipse	14	20000	Seattle	07:30:00	2018-11-09	New York	22:30:00	2018-11-10	25
45	Eclipse	3	20000	New York	07:30:00	2018-11-10	Seattle	23:30:00	2018-11-11	25
46	Eclipse	14	20000	Seattle	07:30:00	2018-11-12	New York	22:30:00	2018-11-13	25

>SELECT s.schedule_id as 'Schedule Number', v.vessel_name Vessel, s.vessel_id as 'Vessel ID',
v.vessel_capacity as 'Vessel Capacity', p.port_name as 'Departing From',s.departing_time
as 'Departing Time', s.departing_date as 'Departing Date', pd.port_name as
 'Arriving',s.arriving_time as 'Arriving Time', s.arriving_date as 'Arriving Date',rate
FROM

```
Schedule s
        inner join Vessel v on s.vessel_id=v.vessel_id
inner join Port p on v.current_port_id=p.port_id
inner join Port pd on v.destination_port_id=pd.port_id
WHERE
s.departing_date between '2018-11-01' and
'2018-12-01' and p.port_name='New York' and pd.port_name='London';
```

localhost

Schedule Number	Vessel	Vessel ID	Vessel Capacity	Departing From	Departing Time	Departing Date	Arriving	Arriving Time	Arriving Date	Rate
14	Titanic 6	150000		New York	17:00:00	2018-11-03	London	05:00:00	2018-11-07	30
16	Titanic 6	150000		New York	17:00:00	2018-11-10	London	05:00:00	2018-11-13	30
18	Titanic 6	150000		New York	17:00:00	2018-11-16	London	05:00:00	2018-11-19	30
20	Titanic 6	150000		New York	17:00:00	2018-11-22	London	05:00:00	2018-11-25	30

2. **Customer gets quotation**

```
>SELECT
        s.schedule_id as 'Schedule Number', v.vessel_name Vessel, s.vessel_id as
'Vessel ID',
        v.vessel_capacity as 'Vessel Capacity', p.port_name as 'Departing
From',s.departing_time
        as 'Departing Time', s.departing_date as 'Departing Date', pd.port_name as
        'Arriving',s.arriving_time as 'Arriving Time', s.arriving_date as 'Arriving Date',
        DATEDIFF(s.arriving_date,s.departing_date) Days,rate Rate, (rate*5000)  Quote
FROM
        Schedule s
         inner join Vessel v on s.vessel_id=v.vessel_id
        inner join Port p on v.current_port_id=p.port_id
        inner join Port pd on v.destination_port_id=pd.port_id
```

```
WHERE
        s.departing_date between '2018-11-01' and '2018-12-01'
        and p.port_name='New York' and pd.port_name='London' and s.schedule_id=14;
```

localhost

Schedule Number	Vessel	Vessel ID	Vessel Capacity	Departing From	Departing Time	Departing Date	Arriving	Arriving Time	Arriving Date	Days	Rate	Quote
14	Titanic 6	150000		New York	17:00:00	2018-11-03	London	05:00:00	2018-11-07	4	30	150000

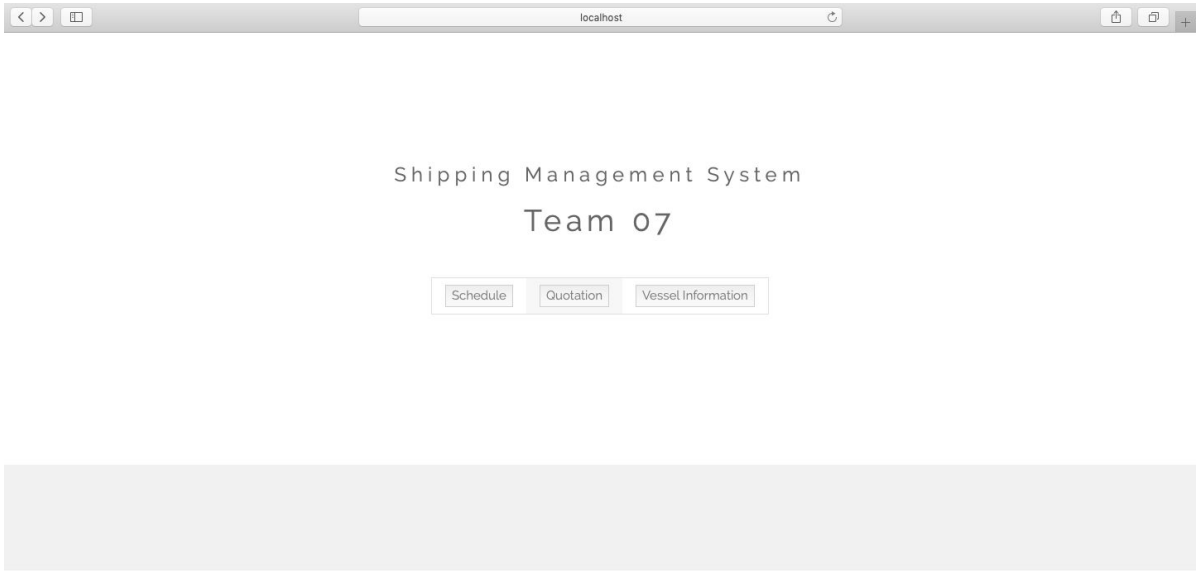
3. Customer views vessel information

```
SELECT
        v.vessel_id ID ,v.vessel_name Name,v.vessel_capacity Capacity,p.port_name
`From`,
        p1.port_name `To`,v.rate as Rate
FROM
        Vessel v
        inner join Port p on v.current_port_id=p.port_id
        inner join Port p1 on v.destination_port_id=p1.port_id;
```

localhost

Vessel ID	Vessel Name	Capacity	Originating	Destination	Rate
1	Elizabeth	5000	Los Angeles	Seattle	10
2	Queenmary	2000	Los Angeles	Long beach	9
3	Eclipse	20000	New York	Seattle	25
4	Titanic	150000	London	New York	30
5	Olympic	120000	London	New York	28
6	Titanic	150000	New York	London	30
7	Olympic	120000	New York	London	28
8	American Dream	7500	Houston	New York	20
9	American Dream	7500	New York	Houston	20
12	Elizabeth	5000	Seattle	Los Angeles	10
13	Queenmary	2000	Long beach	Los Angeles	20
14	Eclipse	20000	Seattle	New York	25
15	Emma Maresk	15000	London	New York	25
16	Emma Maresk	15000	New York	London	25

User Interface:



Sprint 2

UPDATED REQUIREMENTS

Story ID	Story description
US1	As a Customer, I want to create an account so that I can access the portal(partial implementation)
US2	As a Customer, I want to get quotation for the order
US3	As a Customer, I want to be able to view vessel schedule(updated)
US4	As a Customer, I want to place an order
US5	As a Customer, I want to track my order
US6	As an Employee, I want to add a new vessel information
US7	As an Employee, I want to update a vessel information
US8	As an Employee, I want to update port information
US9	As an Customer, I want to cancel the order
US10	As an Employee, I want to add port information
US11	As a Customer, I want to save my credit-card information
US12	As a Customer, I would like to split my order if my order is large

CONCEPTUAL DESIGN

Entity: **Customer**
Attributes:-

<u>customer_id</u> (primary key)
name[composite]
first_name

last_name
email_id
contact_no
password

Entity: **Vessel**

Attribute-

vessel_id(primary key)
vessel_name
vessel_capacity

Entity: **Port**

Attribute-

port_id(primary key)
port_name

Relationship: **Vessel** is scheduled to travel to a **Port**
Attributes: departing_time
departing_date
arriving_time
arriving_date
rate
Cardinality: Many to Many
Participation: Vessel has partial participation
Port has partial participation

Relationship: **Customer** places an order based on the **Vessel** schedule
Attributes:
order_capacity
order_value
order_billing
order_shipping_address
order_consignee_address

order_status

Cardinality: Many to Many
Participation: Customer has partial participation
Vessel has partial participation

LOGICAL DESIGN

Table: **Customer**

Columns:

customer_id (primary key)
first_name
last_name
email_id
contact_no
password

Highest Normalization Form: Table is in 4NF

Table: **Vessel**

Columns-

vessel_id(primary key)
vessel_name
vessel_capacity
~~current_port_id(foreign key references **port_id** of **Port** table)~~
~~destination_port_id(foreign key references **port_id Port** table)~~

Highest Normalization Form:Table is in 4NF

~~Foreign key approach to mapping relation as it suited our requirements better than creating a new table which cross-references Vessel and Port.~~

Table: **Port**

Columns-

port_id(primary key)

port_name

Highest Normalization Form: Table is in 4NF

Table: **Schedule**
Columns:

schedule_id(primary key)
vessel_id(foreign key references **vessel_id** from **Vessel** table)
departing_port_id(foreign key references **port_id** from **Port** table)
departing_time
departing_date
arrival_port_id(foreign key references **port_id** from **Port** table)
arriving_time
arriving_date
rate

Highest Normalization Form:Table is in 2NF
Justification: Vessel rate is dependent on distance between the destination and the arrival ports. To reduce the complexity, we decided to keep the table in 2NF instead of creating a new table for distance and rate. We are not calculating the distance.
We recognize a dependency but since our current increment does not deal with rate calculation based on geographical distance, we decided to keep the table in 2NF.

Table:**Order_details**
Columns:

order_id (primary key)
customer_id(foreign key references **customer_id** from **Customer** table)
schedule_id(foreign key references **schedule_id** from **Schedule** table)
order_capacity

order_value
shipping_address_line_1
shipping_address_line_2
shipping_city
shipping_state
shipping_country
shipping_zip_code
consignee_address_line_1
consignee_address_line_2
consignee_city
consignee_state
consignee_country
consignee_zip_code
order_billing
order_status

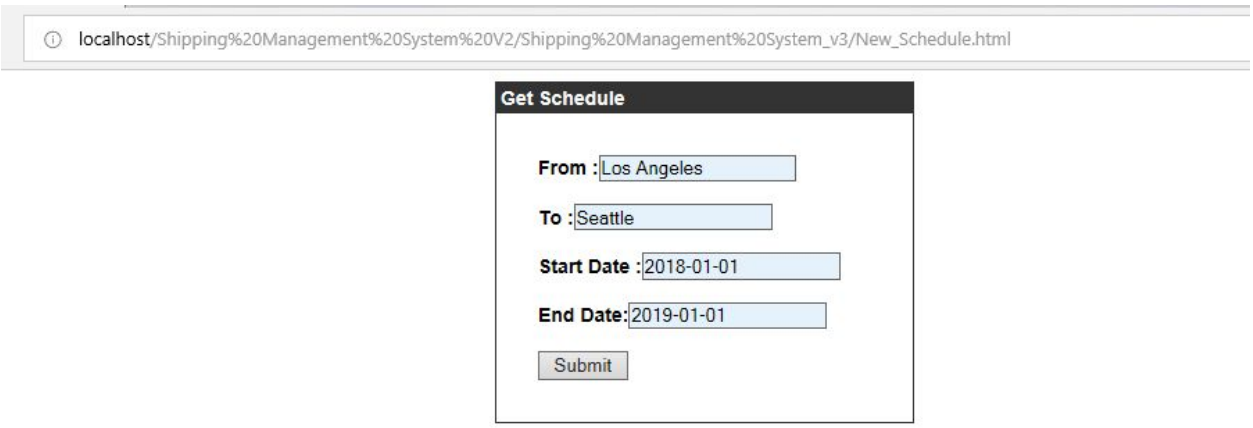
Highest Normalization Form: Table is in 2NF
Justification: Shipping_city, shipping_state, shipping_country can be derived from shipping_zip_code; but to avoid complexity we are not creating a new table for zip_code, city, state and country

SQL QUERIES-

1)Show Schedule procedure - customer gets availability of vessel with updated vessel capacity (updated from sprint 1)

```
DELIMITER $$
CREATE DEFINER=`DBProject2018`@`%` PROCEDURE `showschedule`(
 IN currentport VARCHAR(25),
 in destination varchar(25),
 in a_date date,
 in d_date date)
BEGIN
 select * from
(select s.schedule_id,v.vessel_name, s.departing_date,s.departing_time,
s.arriving_date,s.arriving_time,
v.vessel_capacity - ifnull(sum(od.order_capacity),0) as available_capacity
from `Schedule` s inner join Vessel v on s.vessel_id = v.vessel_id
```

```
left join Order_details od on s.schedule_id = od.schedule_id
where departing_date >=a_date
and arriving_date <=d_date
and current_port_id = (select port_id from `Port` where port_name =
currentport )
and destination_port_id = (select port_id from `Port` where port_name =
destination )
group by s.schedule_id) A
where A.available_capacity <>0;
END$$
DELIMITER ;
```



localhost/Shipping%20Management%20System%20V2/Shipping%20Management%20System_v3/schedule_results.php

Schedule

Schedule ID	Vessel Name	Departing Date	Departing Time	Arriving Date	Arriving Time	Available Capacity
67	Queenmary	2018-12-01	22:00:00	2018-12-02	10:00:00	2000
69	Queenmary	2018-12-03	22:00:00	2018-12-04	10:00:00	2000
75	Elizabeth	2018-03-05	13:00:00	2018-04-06	23:00:00	2585
76	Emma Maresk	2018-02-20	14:00:00	2018-03-04	21:00:00	15000
77	Olympic	2018-11-23	15:00:00	2018-10-15	22:00:00	120000

Place Order

2)Create order Procedure - Order is placed by customer based on schedule

```
DELIMITER $$
CREATE DEFINER=`DBProject2018`@`%` PROCEDURE `create_order`(in
cust_id int,in sch_id int, in capacity int,in val int, in shipaddress1
varchar(255),
in shipaddress2 varchar(255),in shipcity varchar(255),in shipstate
varchar(255),in shipcountry varchar(255),in zip1 int,
in conaddress1 varchar(255),in conaddress2 varchar(255),in concity
varchar(255), in constate varchar(255),in concount varchar(255),
in zip2 int)
BEGIN
INSERT INTO `shipping_management_system`.`Order_details`
(`customer_id`,
```

```
`schedule_id`,
`order_capacity`,
`order_value`,
`shipping_address_line_1`,
`shipping_address_line_2`,
`shipping_city`,
`shipping_state`,
`shipping_country`,
`shipping_zip_code`,
`consignee_address_line_1`,
`consignee_address_line_2`,
`consignee_city`,
`consignee_state`,
`consignee_country`,
`consignee_zip_code`
)
VALUES(cust_id,sch_id,capacity,val,shipaddress1,shipaddress2,shipcity,ships
tate,shipcountry,zip1,conaddress1,
conaddress2,concity,constate,concount,zip2);


END$$
DELIMITER ;
```

localhost/Shipping%20Management%20System%20V2/Shipping%20Management%20System_v3/schedule_results.php

Place Order

Email

upattnai@uncc.edu

Schedule ID

69

Order Capacity

300

Order Value

8000

Sender Address Line 1

80, Creek Road

Sender Address Line 2

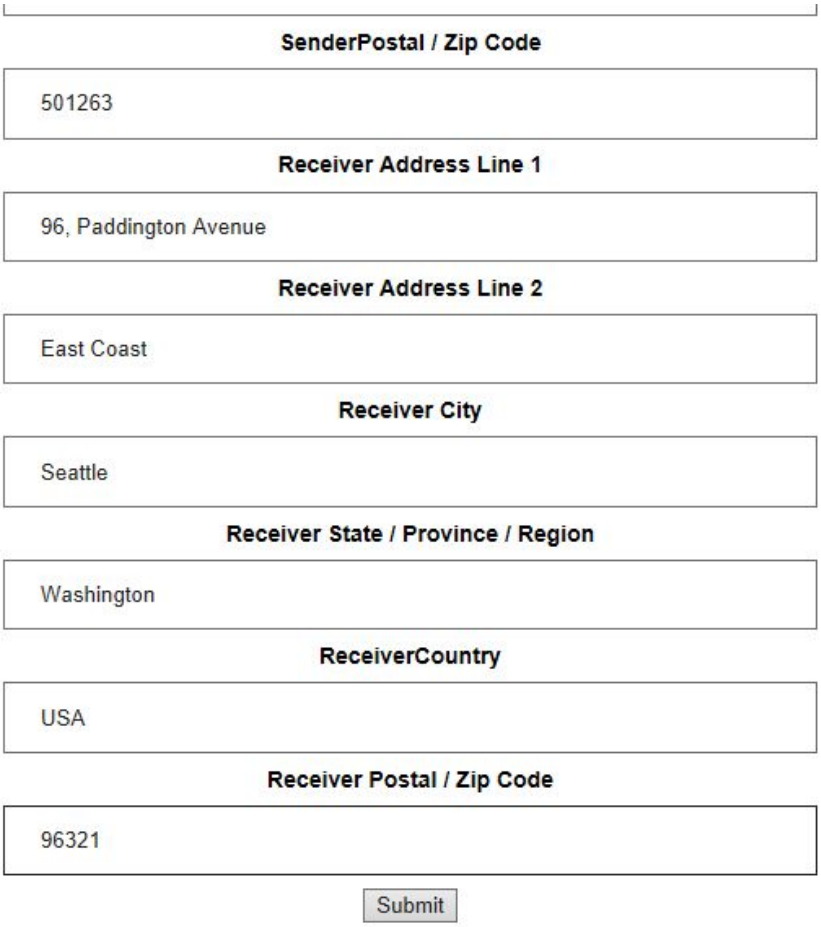
Wallstreet Park Avenue

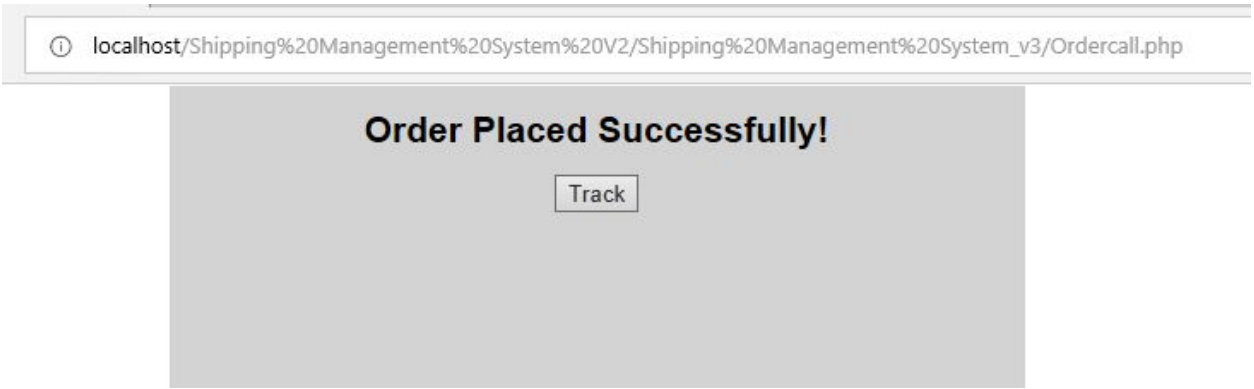
Sender City

Los Angeles

Sender State / Province / Region

California





3)Order view - Created a view where order status is updated

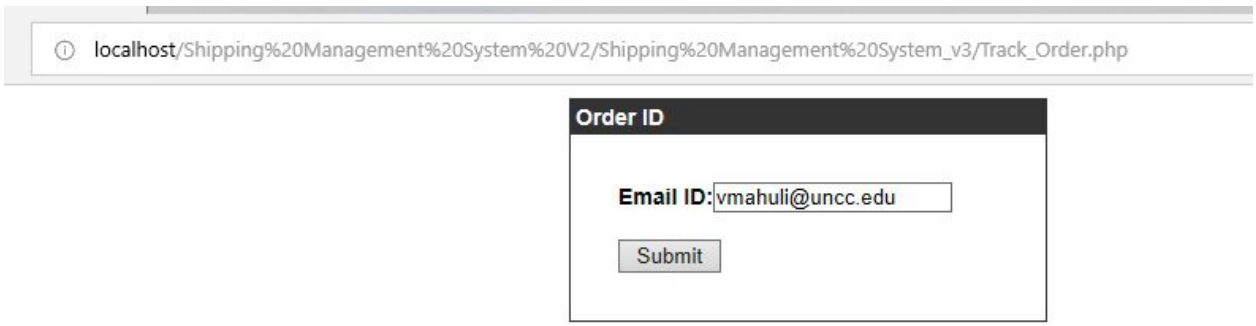
create view Order_tracker as

```
select
`order_id`,`customer_id`,`order_capacity`,`order_value`,`consignee_addr
ess_line_1`,
`consignee_address_line_2`,`consignee_city`,`consignee_state`,`consigne
e_country`,`consignee_zip_code`,
case
 when
curdate() >s.departing_date and curdate()>=s.arriving_date and
od.order_status !="Cancelled"
then
"Order successfully reached"
when
curdate()>s.departing_date and curdate()<s.arriving_date and
od.order_status !="Cancelled"
then
"Order in transit"
when curdate()<s.departing_date and curdate()<s.arriving_date and
od.order_status !="Cancelled"
then
"Placed"
when od.order_status = "Cancelled"
then
"Cancelled"
end as order_status
from Order_details od
inner join `Schedule` s on od.schedule_id = s.schedule_id
;
```

4)
Order Tracker procedure - Created a procedure to track a order

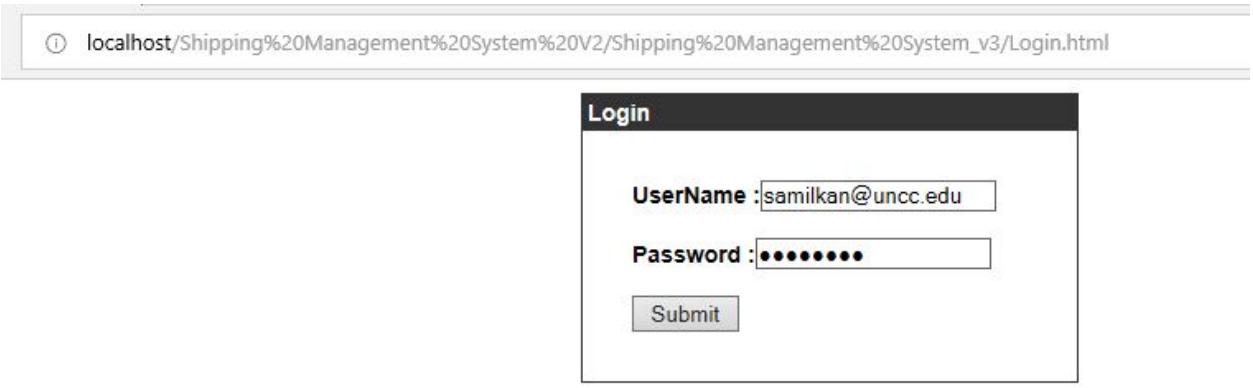
```
DELIMITER $$
CREATE DEFINER=`DBProject2018`@`%` PROCEDURE `order_tracker`(IN
custid int)
```

```
BEGIN
select * from Order_tracker where customer_id=custid  order by order_id
desc;
END$$
DELIMITER ;
```

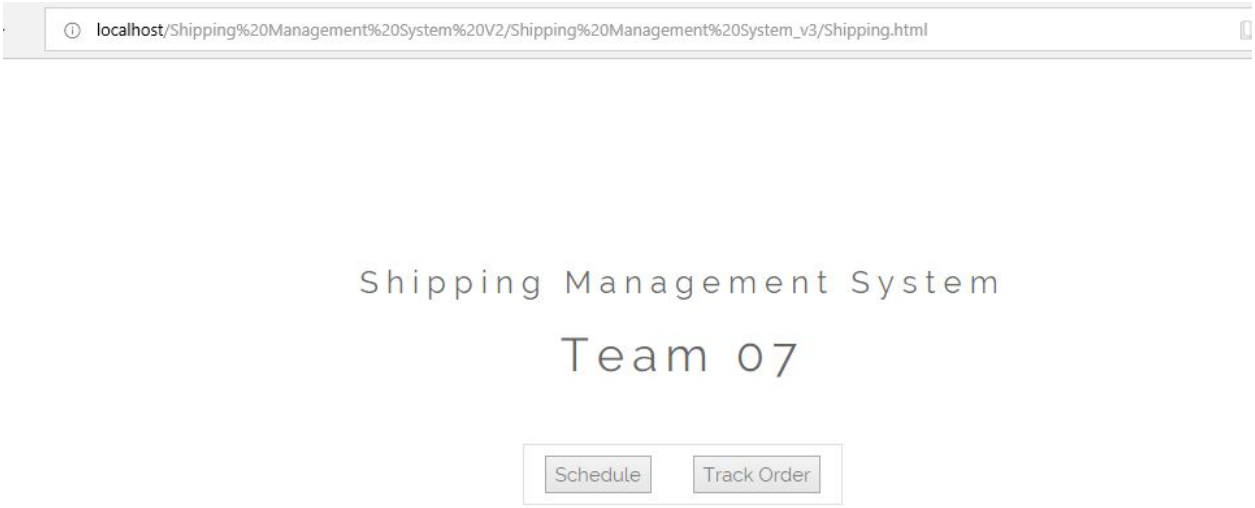


Order ID	Order Capacity	Order Value	Receiver Address Line 1	Receiver Address Line 2	Receiver City	Receiver State	Receiver Country	Receiver Zipcode	Order Status
25	720	3000	101, Lake View Apartments	Parkgreen drive	Seattle	Washington	USA	520146	Placed
22	805	10000	13B, Baker Street	Lexington avenue	Seattle	Washington	USA	100254	Order successfully reached

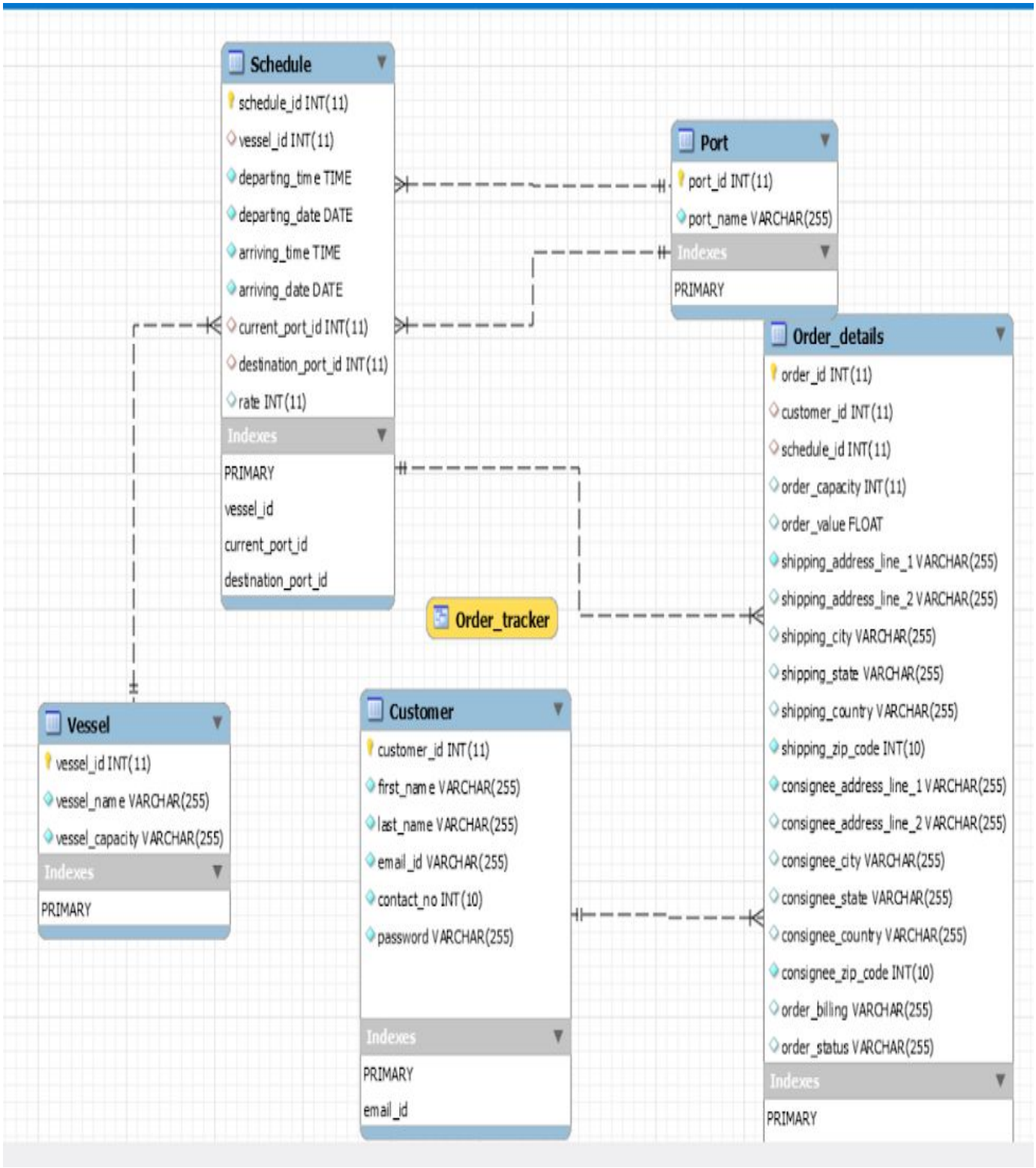
USER INTERFACE MAIN PAGE



User Interface (Customer Homepage)



ER Diagram


Schedule
schedule_id INT(11)
vessel_id INT(11)
departing_time TIME
departing_date DATE
arriving_time TIME
arriving_date DATE
current_port_id INT(11)
destination_port_id INT(11)
rate INT(11)
Indexes
PRIMARY
vessel_id
current_port_id
destination_port_id
Port
port_id INT(11)
port_name VARCHAR(255)
Indexes
PRIMARY
Order_details
order_id INT(11)
customer_id INT(11)
schedule_id INT(11)
order_capacity INT(11)
order_value FLOAT
shipping_address_line_1 VARCHAR(255)
shipping_address_line_2 VARCHAR(255)
shipping_city VARCHAR(255)
shipping_state VARCHAR(255)
shipping_country VARCHAR(255)
shipping_zip_code INT(10)
consignee_address_line_1 VARCHAR(255)
consignee_address_line_2 VARCHAR(255)
consignee_city VARCHAR(255)
consignee_state VARCHAR(255)
consignee_country VARCHAR(255)
consignee_zip_code INT(10)
order_billing VARCHAR(255)
order_status VARCHAR(255)
Indexes
PRIMARY
Order_tracker
Vessel
vessel_id INT(11)
vessel_name VARCHAR(255)
vessel_capacity VARCHAR(255)
Indexes
PRIMARY
Customer
customer_id INT(11)
first_name VARCHAR(255)
last_name VARCHAR(255)
email_id VARCHAR(255)
contact_no INT(10)
password VARCHAR(255)
Indexes
PRIMARY
email_id

Sprint 3

UPDATED REQUIREMENTS

Story ID	Story description
US1	As a Customer, I want to create an account so that I can access the portal(partial implementation)
US2	As a Customer, I want to get quotation for the order
US3	As a Customer, I want to be able to view vessel schedule(updated)
US4	As a Customer, I want to place an order
US5	As a Customer, I want to track my order
US6	As an Employee, I want to add port information
US7	As an Employee, I want to add a new vessel information
US8	As an Employee, I want to update a vessel information
US9	As an Employee, I want to update port information
US10	As an Customer, I want to cancel the order
US11	As a Customer, I would like to see the details of my billing. (updated)
US12	As a Customer, I would like to split my order if my order is large (Cancelled user story)

CONCEPTUAL DESIGN

Entity: **Customer**
Attributes:-

- customer_id (primary key)
- name[composite]
  - first_name
  - last_name

email_id
contact_no
password

Action:
Customer creates account
Customer cancels order
Customer gets quotation

Entity: **Vessel**
Attribute-
vessel_id(primary key)
vessel_name
vessel_capacity

Entity: **Port**
Attribute-
port_id(primary key)
port_name

Entity: **Employee**
Attribute:
employee_id
employee_password

Actions:
Employee adds new Vessel
Employee updates Vessel information
Employee adds new Port
Employee updates Port

Relationship: **Vessel** is scheduled to travel to a **Port**
Attributes: departing_time
departing_date
arriving_time
arriving_date
rate
Cardinality: Many to Many
Participation: Vessel has partial participation

Port has partial participation

Relationship: **Customer** places an order based on the **Vessel** schedule
Attributes:
order_capacity
order_value
order_billing
order_shipping_address
order_consignee_address
order_status
last_modified_at

Cardinality: Many to Many
Participation: Customer has partial participation
Vessel has partial participation

LOGICAL DESIGN

Table: **Customer**

Columns:

<u>customer_id</u> (primary key)
first_name
last_name
email_id
contact_no
password

Highest Normalization Form: Table is in 4NF

Index:

1. 1st Index- PRIMARY (default)
   Column: customer_id
   Classification: Clustered
   Justification: Frequently used in table joins and select statements.

2. 2st Index-
   Column:email_id
   Classification: Non-clustered
   Justification: It is a unique field and queried frequently in our user interface queries.

Table: **Vessel**

Columns-

vessel_id(primary key)
vessel_name
vessel_capacity

Highest Normalization Form:Table is in 4NF

1. 1st Index- PRIMARY (default)
   Column: vessel_id
   Classification: Clustered
   Justification: Frequently used in table joins and select statements.

Table: **Port**

Columns-

port_id(primary key)
port_name

Highest Normalization Form: Table is in 4NF

Index:

1st Index- PRIMARY (default)
Column: port_id
Classification: Clustered
Justification: Frequently used in table joins and select statements.

2nd Index:
Column: port_name
Classification: Non-clustered
Justification: Frequently used in table joins and select statements.

```
CREATE INDEX port_name ON Port (port_name);
```

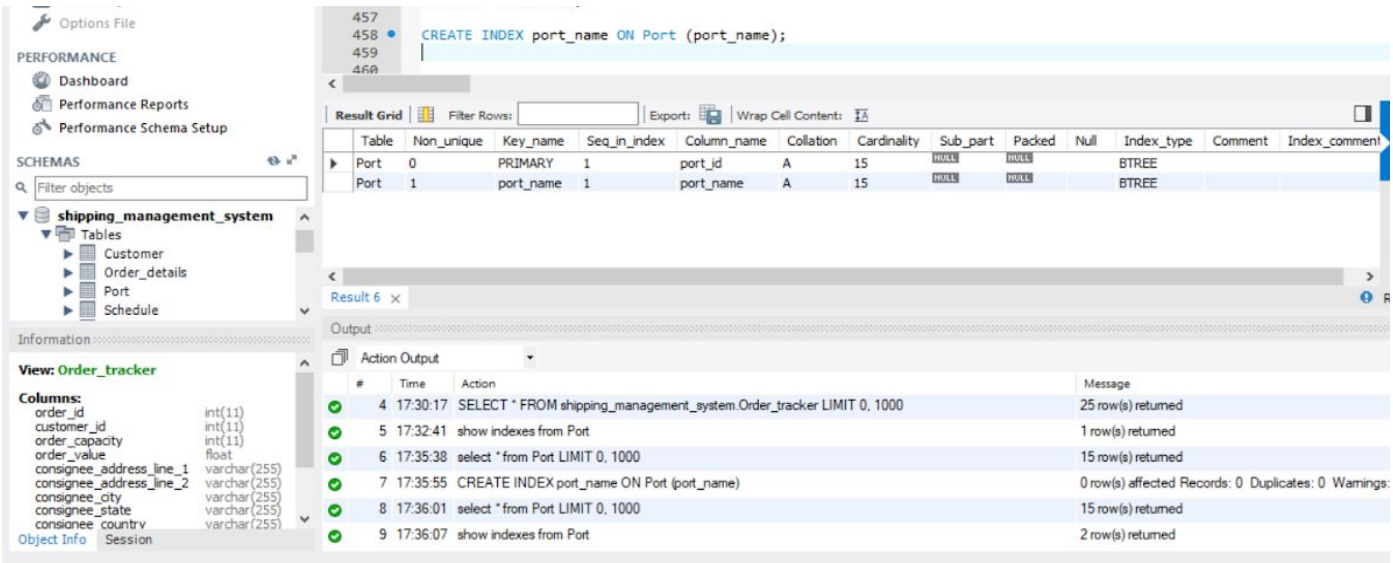


Table: **Schedule**

Columns:

- schedule_id(primary key)
- vessel_id(foreign key references **vessel_id** from **Vessel** table)
- departing_port_id(foreign key references **port_id** from **Port** table)
- departing_time
- departing_date
- arrival_port_id(foreign key references **port_id** from **Port** table)
- arriving_time
- arriving_date
- rate

Highest Normalization Form:Table is in 2NF

Justification: Vessel rate is dependent on distance between the destination and the arrival ports. To reduce the complexity, we decided to keep the table in 2NF instead of creating a new table for distance and rate. We are not calculating the distance.

We recognize a dependency but since our current increment does not deal with rate calculation based on geographical distance, we decided to keep the table in 2NF.

1st Index- PRIMARY (default)

Column: schedule_id

Classification: Clustered

Justification: Frequently used in table joins and select statements.

2nd Index:

Column: vessel_id

Classification: Non-clustered

Justification: Frequently used in table joins and select statements.

3. 3nd Index:

Column: current_port_id

Classification: Non-clustered

Justification: Frequently used in table joins and select statements.

4. 4nd Index:

Column: destination_port_id

Classification: Non-clustered

Justification: Frequently used in table joins and select statements.

Table:**Order_details**

Columns:

order_id (primary key)

customer_id(foreign key references **customer_id** from **Customer** table)

schedule_id(foreign key references **schedule_id** from **Schedule** table)

order_capacity

order_value

shipping_address_line_1

shipping_address_line_2

shipping_city

shipping_state

shipping_country

shipping_zip_code

consignee_address_line_1

consignee_address_line_2

consignee_city
consignee_state
consignee_country
consignee_zip_code
order_billing
order_status
last_modified_at

Highest Normalization Form: Table is in 2NF
Justification: Shipping_city, shipping_state, shipping_country can be derived from shipping_zip_code; but to avoid complexity we are not creating a new table for zip_code, city, state and country

1. 1st Index- PRIMARY (default)
   Column: order_id
   Classification: Clustered
   Justification: Frequently used in table joins and select statements.
2. 2nd Index:
   Column: customer_id
   Classification: Non-clustered
   Justification: Frequently used in table joins and select statements.
3. 3nd Index:
   Column: schedule_id
   Classification: Non-clustered
   Justification: Frequently used in table joins and select statements.

SQL QUERIES-

<u>Events:-</u>

1. Added a Transaction field in our Order_details table:

   alter table Order_details add order_transaction varchar(255);

CREATE EVENT IF NOT EXISTS time_event_02
ON SCHEDULE AT CURRENT_TIMESTAMP
DO

update Order_details o set
order_transaction=concat(order_id,customer_id,schedule_id,order_capacity,
order_billing);

2. Added last_modified_at attribute in Order_details table

alter table Order_details add last_modified_at datetime;

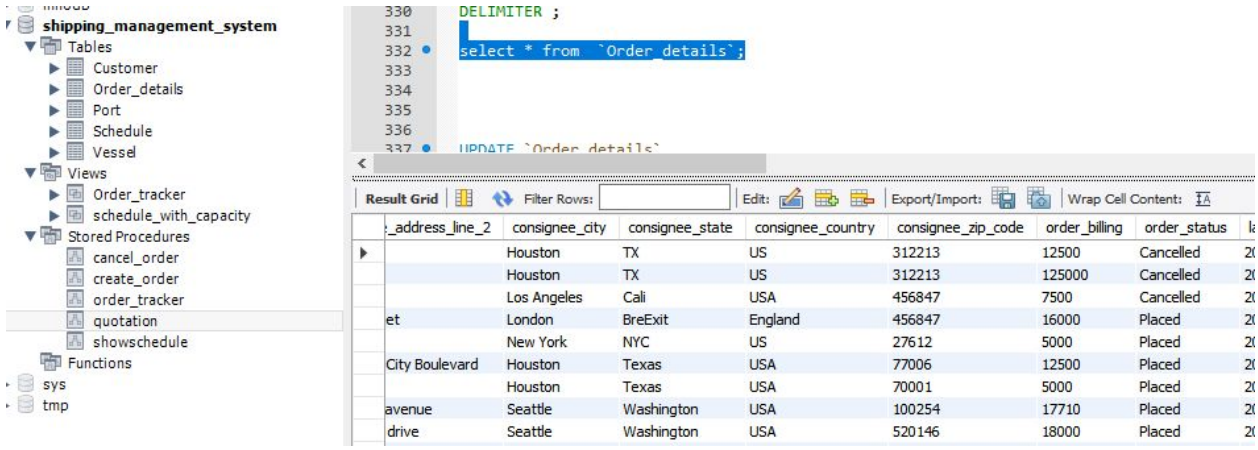
CREATE EVENT IF NOT EXISTS time_event_01
ON SCHEDULE AT CURRENT_TIMESTAMP
DO
update Order_details set last_modified_at=NOW();

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

onsignee_city	consignee_state	consignee_country	consignee_zip_code	order_billing	order_status	last_modified_at
ndon	BreExit	England	456847	0	Placed	2018-11-26 20:27:53
w York	NYC	US	27612	NULL	Placed	2018-11-26 20:27:53
uston	Texas	USA	77006	NULL	Placed	2018-11-26 20:27:53
uston	Texas	USA	70001	NULL	Placed	2018-11-26 20:27:53
attle	Washington	USA	100254	NULL	Placed	2018-11-26 20:27:53
attle	Washington	USA	520146	NULL	Placed	2018-11-26 20:27:53
uston	TX	USA	98101	NULL	Cancelled	2018-11-26 20:27:53

3. Billing values in our Order_details table

CREATE EVENT bill_update
ON SCHEDULE AT CURRENT_TIMESTAMP
DO
UPDATE `Order_details`
INNER JOIN
Schedule
ON Schedule.schedule_id = `Order_details`.schedule_id
SET `Order_details`.`order_billing` =
`Order_details`.`order_capacity`*Schedule.rate;



Trigger:-

1. Trigger to set newly inserted order to current time

```
DELIMITER $$
CREATE TRIGGER before_order_insert
before INSERT ON Order_details
FOR EACH ROW
BEGIN
set new.last_modified_at=now();
END$$
DELIMITER ;
```

2. Trigger to set last modification on order to current time

```
DELIMITER $$
CREATE TRIGGER before_order_update
BEFORE UPDATE ON Order_details
FOR EACH ROW
BEGIN
set new.last_modified_at=now();
END$$
DELIMITER ;
```

Successfully created Triggers

Trigger	Event	Table	Statement	Timing	Created	sql_mode
before_order_insert	INSERT	Order_details	BEGIN set new.last_modified_at=now(); END	BEFORE	2018-11-26 20:30:35.32	NO_ENGINE_SUBS
before_order_insert_transaction	INSERT	Order_details	BEGIN set new.order_transaction=concat((SEL...	BEFORE	2018-12-01 00:56:19.21	NO_ENGINE_SUBS
before_order_update	UPDATE	Order_details	BEGIN set new.last_modified_at=now(); END	BEFORE	2018-11-26 20:43:33.97	NO_ENGINE_SUBS

Before_order_insert

onsignee_city	consignee_state	consignee_country	consignee_zip_code	order_billing	order_status	last_modified_at
s Angeles	California	US	512211	NULL	Placed	2018-11-26 20:27:53
s Angeles	California	US	512211	NULL	Cancelled	2018-11-26 20:44:40
s Angeles	California	US	512211	NULL	Cancelled	2018-11-26 20:44:40
attle	WA	USA	90001	NULL	Placed	2018-11-26 20:46:19
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Before_order_update

onsignee_city	consignee_state	consignee_country	consignee_zip_code	order_billing	order_status	last_modified_at
s Angeles	California	US	512211	NULL	Cancelled	2018-11-26 20:44:40
s Angeles	California	US	512211	NULL	Cancelled	2018-11-26 20:44:40
attle	Wa	USA	90001	NULL	Placed	2018-11-26 21:30:03

3.

```
DELIMITER $$
CREATE TRIGGER before_order_insert_transaction
before INSERT ON Order_details
FOR EACH ROW
BEGIN
set new.order_transaction=concat((SELECT AUTO_INCREMENT
      FROM information_schema.TABLES
      WHERE TABLE_SCHEMA=DATABASE()
            AND
TABLE_NAME='Order_details'),new.customer_id,new.schedule_id,new.order
_capacity,new.order_billing);
END$$
DELIMITER ;
```

consignee_country	consignee_zip_code	order_billing	order_status	last_modified_at	order_transaction
US	312213	125000	Cancelled	2018-11-28 23:04:13	21655000125000
USA	456847	7500	Cancelled	2018-11-28 23:04:13	35703007500
England	456847	16000	Placed	2018-11-28 23:04:13	467280016000
US	27612	5000	Placed	2018-11-28 23:04:13	101662005000
USA	77006	12500	Placed	2018-11-28 23:04:13	1426350012500
USA	70001	5000	Placed	2018-11-28 23:04:13	162702005000
USA	100254	17710	Placed	2018-11-28 23:04:13	2247580517710

Views:-

```
CREATE VIEW
`schedule_with_capacity` AS
(select `A`.`schedule_id`  AS `schedule_id`,
`A`.`vessel_name` AS `vessel_name`,
`A`.`departing_date` AS `departing_date`,
`A`.`departing_time` AS `departing_time`,
`A`.`arriving_date` AS `arriving_date`,
`A`.`arriving_time` AS `arriving_time`,
`A`.`available_capacity` AS `available_capacity`
from
(select
        `s`.`schedule_id` AS `schedule_id`,
        `v`.`vessel_name` AS `vessel_name`,
        `s`.`departing_date` AS `departing_date`,
        `s`.`departing_time` AS `departing_time`,
        `s`.`arriving_date` AS `arriving_date`,
        `s`.`arriving_time` AS `arriving_time`,
        (`v`.`vessel_capacity` - ifnull(sum(`od`.`order_capacity`),0)) AS
         `available_capacity`
from
        ((`shipping_management_system`.`Schedule` `s` join
        `shipping_management_system`.`Vessel` `v`
        on((`s`.`vessel_id` = `v`.`vessel_id`))) left join
```

```
        `shipping_management_system`.`Order_details` `od`
        on((`s`.`schedule_id` = `od`.`schedule_id`)))
where
        (`od`.`order_status` <> 'Cancelled') group by `s`.`schedule_id`)
        `A` where (`A`.`available_capacity` > 0));
```

Stored Procedure:-

1. Cancel order: Called when a customer wants to cancel an order.

```
DELIMITER $$
CREATE PROCEDURE `cancel_order`(in cust_id int,in order_id1 int)
BEGIN
 update Order_details set order_status='Cancelled' where
order_id=order_id1 and customer_id=cust_id;
 select 'ORDER CANCELLED' as order_stat;
 END$$
DELIMITER ;
```

	onsignee_city	consignee_state	consignee_country	consignee_zip_code	order_billing	order_status	last_modified_at
▶	uston	TX	US	312213	0	Cancelled	2018-11-26 20:27:53
	uston	TX	US	312213	0	Cancelled	2018-11-26 20:27:53
	s Angeles	Cali	USA	456847	0	Cancelled	2018-11-26 21:15:05
	ndon	BreExit	England	456847	0	Placed	2018-11-26 20:27:53
	w York	NYC	US	27612	NULL	Placed	2018-11-26 20:27:53

2. Create Order: Is called when a customer places an order.

```
DELIMITER $$
CREATE DEFINER=`DBProject2018`@`%` PROCEDURE `create_order`(in
cust_id int,in sch_id int, in capacity int,in val int, in shipaddress1
varchar(255),
in shipaddress2 varchar(255),in shipcity varchar(255),in shipstate
varchar(255),in shipcountry varchar(255),in zip1 int,
in conaddress1 varchar(255),in conaddress2 varchar(255),in concity
varchar(255), in constate varchar(255),in concount varchar(255),
in zip2 int,out val1 varchar(255))
BEGIN
declare bill int;
case
```

```
when ( select available_capacity from schedule_with_capacity where
schedule_id=sch_id
)>capacity

then
set bill=(select (rate*capacity) from Schedule  where schedule_id=sch_id);


INSERT INTO `shipping_management_system`.`Order_details`
(`customer_id`,
`schedule_id`,
`order_capacity`,
`order_value`,
`shipping_address_line_1`,
`shipping_address_line_2`,
`shipping_city`,
`shipping_state`,
`shipping_country`,
`shipping_zip_code`,
`consignee_address_line_1`,
`consignee_address_line_2`,
`consignee_city`,
`consignee_state`,
`consignee_country`,
`consignee_zip_code`,`order_billing`
)
VALUES(cust_id,sch_id,capacity,val,shipaddress1,shipaddress2,shipcity,ships
tate,shipcountry,zip1,conaddress1,
conaddress2,concity,constate,concount,zip2,bill);
 set val1='Order Placed Successfully';
 SELECT 'Order Placed Successfully' as 'Order_Status';
 else
 set val1='Order Cannot be placed!!Capacity Exceed more than Available
capacity!!';
 SELECT 'Order Cannot be placed!!Capacity Exceed more than Available
capacity!!' as 'Order_Status';
  end case;
 END$$
DELIMITER ;
```

3.
Order tracker: Is called when a customer wants to track an order

```
DELIMITER $$
CREATE PROCEDURE `order_tracker`(IN custid int)
BEGIN
select * from Order_tracker where customer_id=custid  order by order_id
desc;
END$$
DELIMITER ;
```

4.
Quotation: Gets the expected quotation for the order.

```
DELIMITER $$
CREATE PROCEDURE `quotation`(IN capacity int, IN schedid int)
BEGIN
select schedule_id, capacity, concat(`rate`*capacity ,' $') as Quotation
from `Schedule` where schedule_id = schedid;
END$$
DELIMITER ;
```

5.
Show Schedule: Gets the schedule based on the departing and arriving ports.

```
DELIMITER $$
CREATE DEFINER=`DBProject2018`@`%` PROCEDURE `showschedule`(
 IN currentport VARCHAR(25),
 in destination varchar(25),
 in a_date date,
 in d_date date)
BEGIN
 select * from
(select s.schedule_id,v.vessel_name, s.departing_date,s.departing_time,
s.arriving_date,s.arriving_time,
v.vessel_capacity - ifnull(sum(od.order_capacity),0) as available_capacity
from `Schedule` s inner join Vessel v on s.vessel_id = v.vessel_id
left join Order_details od on s.schedule_id = od.schedule_id
where departing_date >=a_date
and arriving_date <=d_date
```

```
and current_port_id = (select port_id from `Port` where port_name =
currentport )
and destination_port_id = (select port_id from `Port` where port_name =
destination )
and od.order_status != 'Cancelled'
group by s.schedule_id) A
where
A.available_capacity <>0;
END$$
DELIMITER ;
```