Oil Transaction Management System

Sr. No.	Name	Net-ID
1.	Madhuri Abnave	mxa146230
2.	Ronak Shah	rxs144130
3.	Kanchan Waikar	kpw150030

STEP 1. Draw the ER/EER diagram, and then convert it to a relational schema. Indicate primary keys, foreign keys and any other constraints. Follow the notation used throughout the textbook. Clearly specify any assumptions you make and your rationale.

Please find ER Diagram in the Appendix #1

Constraints/Relationships Identified:

1. Based on the description of the OTS provided in the questions, system clearly has got 3 types of users, Manager, Trader and Client (a.k.a customer). System mandates that the address information must be stored for each client at detailed level. However, we do need this information for managers as well as traders, hence this information was abstracted into a table called "User" and ISA relationship was extracted. It would be only valid to assume that A Manager is also a trader with higher privileges. Hence User can be a Trader or a Client or a Trader with higher privileges.
2. Role of the trader was identified using Role_Id field that has been assigned to the Trader.
3. Every trader has to have a role - either manager or Trader role. Hence Full participation is applicable for Trader Has Role relationship. Also, a Trader can have exactly single role.
4. There are several requirements that are tied to the role "Capability of settling payment for orders", "Cancelling orders","Viewing reports", "Creating a Trader/Client/Manager" user, etc. In order to add support for them in scalable way, "Features" entity was introduced that was tied to a role through "Role has Features" relationship. Every feature has to belong to some role, hence full participation
5. Trader Accepts Payments from customer is a ternary relationship between trader, client and a payment.
6. Any user can place order but the order must be for a client. Hence Client_id has been introduced as the attribute for "User places order"
7. Only trader cancels the order hence, cancels table has been introduced along with cacellation_date as the attribute of the table.
8. A payment is accepted for selected orders is a one to many relationship between payment and orders
9. Every order has to have exactly one entry in places relationship, it cannot belong to multiple entries.
10. Orders table was identified for holding each of the oil transaction that has been made for the client. This way we are storing commission, quantity, total amount, oil for each transaction separately. Also, Shipping_status and date_fulfilled columns were identified for storing shipping information of the oil.

11. Client is allowed to pay for one or more orders, hence Full participation was identified between “Payment Belongs to Order” relationship and Payment entity.
12. Every user has to belong to exactly one company, hence the same constraint has been added in the ER diagram.
13. Level of client, along with timestamp at which the upgrade happened has been associated with the Client entity itself, also, for cancelled orders, we need to keep track of balance as well, hence balance oil and amount has been associated with client entity.
14. Commissions to be applied to user levels is the information that must be maintained with company, hence company entity has been introduced.
15. Oil prices is an entity that has been associated with the company. price maintained in the table can vary for each company(after applying state taxes, city taxes, etc) hence oil_prices are tied to Company through “Company has oil prices” relationship.

Assumptions and Rationale:

1. Any Trader/Manager can do transaction for any Client. This flexibility is required especially considering that employees can fall sick, and the same should not result in impact on business
2. Problem statement states that for each payment, trader ID must be recorded, This can translate to Client should not have privileges of doing the transaction all by himself/herself. Thus assumption has been made that only Traders/Managers are allowed to accept the payment on behalf of the user.
3. User is allowed to place an order but cannot cancel the order as system implicitly accepts the order
4. System assumes availability of infinite amount of oil and hence does not maintain or cap count of amount of oil available for sell/buy. Client is allowed to buy or sell whatever amount of oil he/she desires to buy or sell.

Please refer to relational model from Appendix #2

Rationale behind ERD to Relational model conversion

1. Not null constraint has been enforced wherever it was found to be necessary in order to ensure logical completeness of the schema
2. Implicit assumption considered : Oil and prices are float values.
3. one to many Relationships that do not have any attributes hence are not converted into tables
    a. “Payment belongs to orders” - Payment_id has been added to orders table
    b. “Company has oil Prices” - company_id has been added to oil_prices table
    c. “Company has users” - A user can belong to only one company hence company_id has been added to the users table.
    d. “Trader has Role” - role_id with non-null constraint has been added on trader table.
4. “Role has Features” is a many to many relationship hence has been converted into a separate table.
5. Ternary relationship “Trader accepts payment from client” has been incorporated into Payments table by adding client_id and trader id in payments table.
6. ISA relationship - Client is a user, Trader is a user, hence both trader as well as client table inherits the primary key of the user table.
7. Since commission fee and commission oil are exclusive and atleast one of them must be specified, a trigger was written to ensure the same.
8. Oil_prices table will have company_id and date as the composite primary key.

9. Conversion of client’s level is performed through a “ON INSERT” trigger that upgrades user on 30th transaction.
10. For each order, either commission_fee should be in oil or cash, hence one of the two has to be mandated. This has been achieved through an ON insert trigger on orders table that logs an error every time this constraint is violated.
11. Check constraints were added in client and trader table to ensure the Covering constraint of Users (user can be either client or trader and never both).
12. Trigger was defined for upgrading user’s level to GOLD on completion of their 30 orders in any month.

Brief Description

We have a implemented a Web Based Oil Transaction System which can serve three types users ie. client, trader, and manager. A client can directly buy/sell oil depending on his current oil reserves, or contact a trader to perform these transactions for him. Upon each transaction client pays commission to the company in form of either oil or cash. A client has a level which is initially SILVER and changes to GOLD after 30 barrels are purchased in a single month. A client’s commission is applied according to his level and his oil reserves or account balance is adjusted accordingly.

The company database keeps records for each transactions. Client can pay money to the company from time to time to settle their transactional balances. A Trader can cancel certain payments and oil transactions, but these cancellations are logged for audit purposes. Payments and cancellations must go through traders only.

A manager(administrator) is also a trader and has all functionalities available to a trader. Additionally, a manager can also view reports which gives him analysis of various trades of the system.

Assumptions

- Client cannot pay money directly and it has to be come through a trader only - since the requirement states that trader information must be recorded for each payment
- Clients are not supposed to make partial payments
- Total oil quantity - initial value is zero .
- Oil prices will be downloaded through an external downloader service (Enhancement) that would insert the data into the database by logging onto the system using user “oil_price_loader”
- Client cannot cancel order directly and must contact trader/administrator for cancelling the order
- Payment cannot be completely cancelled - Only orders can be cancelled.
- An administrator can add another administrator
- An Administrator is a Trader with higher privileges
- No cancellation fee is applied by the system.

Security Measures Taken:

- Password field has been saved in encrypted format - Even if anybody gets access to database, password field will be secure as it has been encrypted using an encryption key that is not stored in the database.
- For Login, Post form has been provided, so that credentials are not visible in the

- A different user id has been created for the utility that downloads the oil price data into the system (userName- oil_price_loader) . This user ID has not been given access to any tables apart from oil_prices.
- All IDs are stored as UUID instead of sequence number in order to avoid predictability of the ID. Some technical users tend to play with the Rest API calls and it is important that we make all Id column values unpredictable.
- Prepared statements have been used throughout the application in order to avoid SQL injection attacks.
- Instead of defining the new page for accepting credit card information, a third party which performs the credit card transaction through https, called "Stripe" was used. Stripe exposes an API that allows the integrator to charge users without storing the information on their end.

Database Authorization

For Access to Database, if user needs external access to database, separate db-users have been created. They have been granted access to only those tables that he/she is supposed to access.

- **Admin** - This database user has complete access to all functionalities on all tables.
- **oil_price loader** - This database user has restricted access.

  ```
  GRANT ALL PRIVILEGES ON ots.oil_prices TO 'oil_price_loader'@'localhost';
  ```
- **trader** - This database user has restricted access to only those functionalities that trader can perform.

  ```
  GRANT SELECT ON ots.oil_prices TO 'trader'@'localhost' ;
  GRANT select,update ON ots.users to 'trader@localhost' identified by 'trader@123';
  GRANT select ON ots.feature to 'trader@localhost' identified by 'trader@123';
  GRANT select ON ots.role to 'trader@localhost' identified by 'trader@123';
  GRANT select ON ots.role_has_features to 'trader@localhost' identified by
  'trader@123';
  GRANT select,update,insert ON ots.payments to 'trader@localhost' identified by
  'trader@123';
  GRANT select,INSERT,UPDATE ON ots.orders to 'trader@localhost' identified by
  'trader@123';
  GRANT select,INSERT,UPDATE ON ots.places to 'trader@localhost' identified by
  'trader@123';
  GRANT select,INSERT,UPDATE ON ots.cancels to 'trader@localhost' identified by
  'trader@123';
  GRANT select,update ON ots.client to 'trader@localhost' identified by 'trader@123';
  GRANT select,update ON ots.trader to 'trader@localhost' identified by 'trader@123';
  ```

- **client** - This database user has restricted access to only those Tables that can be accessed by users of this role.

  ```
   GRANT SELECT  ON ots.oil_prices TO 'client'@'localhost' ;
  GRANT select ON ots.role to 'client@localhost' identified by 'client@123';
  GRANT select ON ots.feature to 'client@localhost' identified by 'client@123';
  GRANT select ON ots.role_has_features to 'client@localhost' identified by 'client@123';
  GRANT select ON ots.payments to 'client@localhost' identified by 'client@123';
  GRANT SELECT,INSERT ON ots.orders to 'client@localhost' identified by 'client@123';
  GRANT select,INSERT ON ots.places to 'client@localhost' identified by 'client@123';
  GRANT select,update ON ots.users to 'client@localhost' identified by 'client@123';
  ```

GRANT select ON ots.client to 'client@localhost' identified by 'client@123';

Application Authorization
Role,Feature, Role_has_features tables have been created in order to implement Access control on all the features that application has.
Roles Added

- ADMIN
- CLIENT
- TRADER

Features/Authorization added:

- FEATURE_INSERT_USER(Admin)
- FEATURE_EDIT_PROFILE (admin,Trader,Client)
- FEATURE_CANCEL_ORDER(Trader,Admin)
- FEATURE_ACCEPT_PAYMENT(Trader,Admin)
- FEATURE_VIEW_REPORTS(Admin)

Performance:
Database is one of the most important tiers that is responsible for performance of the systems. Oil Transaction system is not a very a compute heavy application and hence Database would definitely be the primary resource on which application's performance will depend. Measures taken for Performance of the database are as described below.

Connection: Database connection takes time which is why dbcp connection pooling Module was used in order to not have to create a new connection for every query. DBCP connection pool of following configuration has been defined

a. initialSize="3"
b. maxIdle="5"
c. maxActive="100"

This means that connection pool will start with a size of three active connections and will grow as more and more requests start coming in. The system is anticipated to provide approximately 100 concurrent users - assuming the total user size of around 5000.

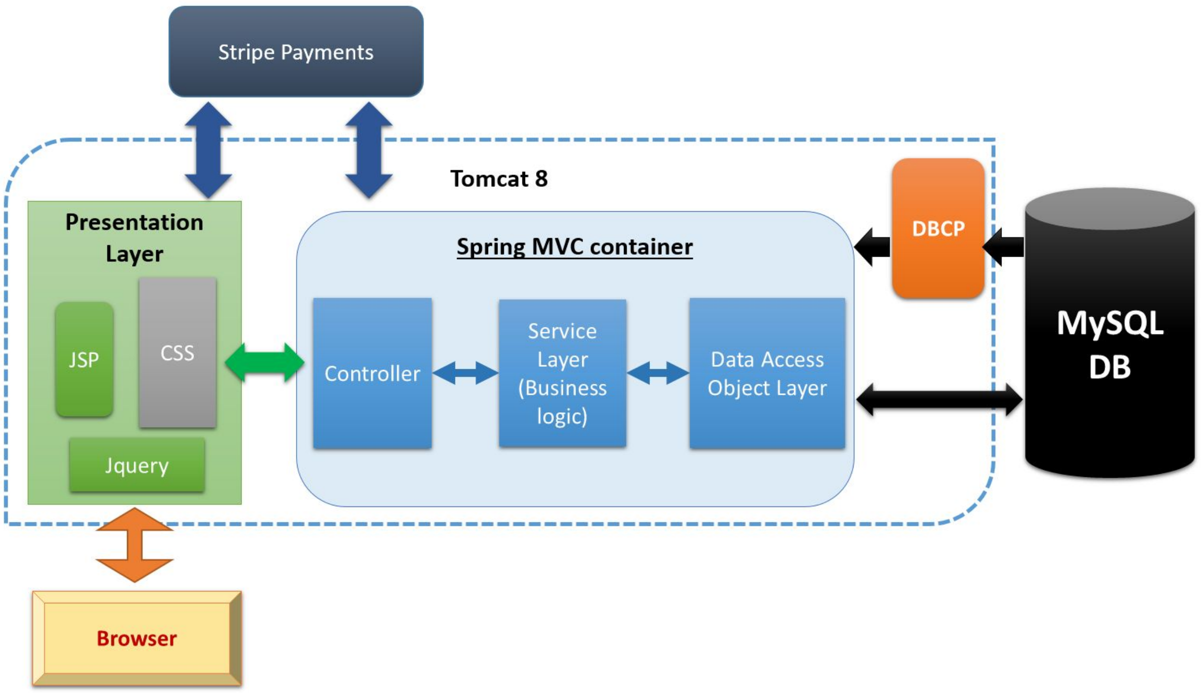
MySQL by default defines a BTREE index on all primary key, foreign key constraints defined for a table. After analyzing all the queries written in application, it was found that Search User query will be used by every trader and administrator for selecting a user, hence it was important to

Indexes:

- BTREE index has been defined on all table's primary Keys(Mysql's default behavior)
- Index for Select User query -Hash index because Search form does not support any search type except "Equal to" search.
    - create index search_client USING HASH on users (id,last_name,apt_no,street,city,zip_code,phone_no,cell_no,email);
- B tree index on id column of Orders table has been defined, since this index will be used every time we display the order summary page for a user.

- View reports functionality has queries on following columns, hence indexes corresponding to the same have been defined.
  - create index quantity_report USING HASH on orders(id, payment_id, quantity);
  - create index amount_report USING HASH on orders(id, payment_id, total_amt);
  - create index oil_commission_report USING HASH on orders(id, payment_id, oil_adjusted_quantity);
  - create index fees_report USING HASH on orders(id, payment_id, commission_fees);

Application architecture overview:



Architecture Diagram

Code Overview

- **Backend Technology Used:**
  - Spring MVC, : Spring Provides MVC Archietcture support thus enabling automatic transaction of data into ModelAttribute beans. Also, It enables us to use 3 layered archietcure of separating requirement into three layers
    - Controller Layer - For Accepting data from Frontend and returning data back.
    - Service Layer - For performing business logic
    - Data Access Object Layer - For Doing all Data access tasks.
  - Spring - JDBC templates,
    - Spring JDBC automatically creates a connection to the Datasource defined in ots-servlet.xml and executes preparedstatement as called
  - MySQL
    - Mysql being free as backend database
  - DBCP for connection pooling
    - Since this will be used by many customers, and traders, connection pooling was implemented in order to avoid connection overhead

- **Frontend Technology used** :
    - JSP,JQuery, HTML5, JSP EL
        - These were used in order to render the dynamic data received from backend and for sending data to backend
    - CSS, Bootstrap
        - Used for Look and Feel of the Applicaiton
    - Google Charts API
        - Used for displaying several reports that managers can analyze.
- **Server** : Tomcat 8
- **Development Environment** : Eclipse, Maven, MySQL, Chrome Debug Tools
- **Source control management** - GIT

As we can see in the above architecture diagram, JSP,JQuery and CSS are primarily used in order to render the frontend of the oil transaction system. The presentation layer communicates with the Spring Controller called "HomeController". Internally Spring uses Front-Controller pattern in order to map these requests to individual mappings. Controller calls Service layer for executing the business logic required. Service Layer calls Dao Layer for executing Queries and fetching the result from database. Request Mappings

Jquery Script for communicating with Backend:
Functions.js
This Javascript file has all AJAX calls that are made to the backend for sending/returning view and or data to be displayed.The code looks like following, with Different URLs, data and request types. Following call makes a POST backend request to "/placeOrder" Request mapping written in Home controller and displays the html received in the container.

```
function putOrder() {
        $.ajax({
                type : "POST",
                url : "placeOrder",
                data : $("#placeOrderForm").serialize(),
                success : function(data) {
                        $("#container").html(data);
                }});}
```

Modules from HomeController

- @RequestMapping(value = "/home", method = RequestMethod.GET)

This method is invoked eveytime when user launches the home page of the application. This method checks whether user is logged in and whether user is trader or not. It returns view of order summary for Clients where as it returns the search page for traders and managers. If user is not logged in, it simply presents user with the Login form.

- @RequestMapping(value = "/login", method = RequestMethod.POST)

This method accepts user Id and password and validates the credentials. It internally calls UserDaoImpl.getUserDetails and verifies email/Password combination by using aes_decrypt method for decrypting the password stored in the database.

- @RequestMapping(value = "/searchUser", method = RequestMethod.POST)

This method accepts search form From User and executes an OR search on all the fields. Please note that it executes Exact search and returns all records present.

- @RequestMapping(value = "/selectUser", method = RequestMethod.GET)

This method is called when Trader/Manager needs to perform certain functions like view order summary, balance, accept payment, Cancel order, etc. This method sets the user in the session and uses the same for performing all user actions

- @RequestMapping(value = "/insertOrUpdateUser", method = RequestMethod.POST)

This method is called when user clicks on submit button on create user screen. This method validates where both the passwords entered are same or not. If complete data is valid, it creates a user by calling UserDaoImpl.INSERT_USER method through service. Also, For trader, it creates an entry in trader table, and for client, it creates an entry in client table.

- @RequestMapping(value = "/cancelOrder", method = RequestMethod.GET)

This method is called when Trader/Manager cancels specific orders for a client. This method performs following logic

1. Creates an entry in cancels table. This entry helps track the trader who performed the cancellation
2. Updates Account information of the user - especially Total Oil quantity and Credit balance. For cancelled “Buy Oil” and sell oil transactions.

- @RequestMapping(value = "/payment", method = RequestMethod.GET)

This method is called when user select orders to pay for. The total amount is calculated based on orders selected and user is presented a Credit card payment screen. Here, user’s existing balance is taken into consideration while accepting payment. Stripe - third party payment gateway service has been used for accepting user’s credit card information. Once provided, the form gets redirected to Stripe form that is setup in Test mode. Accepted credit card number to be used is **4242 4242 4242 4242**

- @RequestMapping(value = "/paymentAccepted", method = RequestMethod.POST)

This method is called when used provides the credit card information. This information is then used and an entry is made to the payments table. Also, for corresponding orderIds, payment_id column is populated in the database. once done, it updates the user’s balance to zero.

- @RequestMapping(value = "/topMenu", method = RequestMethod.GET)

This method is called in order to determine and display the different features that user has access to. It is responsible for displaying “View Reports” Menu in the top right of the panel, once manager has logged on to the system.

- @RequestMapping(value = "/loadOrders", method = RequestMethod.GET)

This method fetches orders from the database by invoking OrderDaoImpl.getOrders() method. The Orders table is joined with Cancels table and the same is used for displaying the orders in the UI.

- @RequestMapping(value = "/placeOrder", method = RequestMethod.POST)

This method is used for creating order. It creates an entry in the order table by invoking OrderDaoImpl.createOrder method. It then creates an entry in the places table in order to store information related to the user who created the order. For Buy transactions, total oil owned by the user is increased, whereas for Sell transactions, the oil quantity is reduced.

- @RequestMapping(value = "/logout", method = RequestMethod.GET)

This method invalidates the session and logs user out.

- @RequestMapping(value = "/viewReports", method = RequestMethod.GET)

This method is responsible for displaying several results that are displayed to Managers. Reports are mainly displayed at aggregated level. Queries used for generating these reports are as listed below.

- select sum(o.quantity) as sums,(isnull(o.payment_id)!=true) as payment_avl, (isnull(c.client_id)!=true) as is_cancelled from orders o left join cancels c on o.id=c.order_id group by payment_avl,is_cancelled order by sums asc
- select sum(o.total_amt) as sums,(isnull(o.payment_id)!=true) as payment_avl, (isnull(c.client_id)!=true) as is_cancelled from orders o left join cancels c on o.id=c.order_id group by payment_avl,is_cancelled order by sums asc
- select sum(o.oil_adjusted_quantity) as sums,(isnull(o.payment_id)!=true) as payment_avl, (isnull(c.client_id)!=true) as is_cancelled from orders o left join cancels c on o.id=c.order_id group by payment_avl,is_cancelled order by sums asc
- select sum(o.commission_fees) as sums,(isnull(o.payment_id)!=true) as payment_avl, (isnull(c.client_id)!=true) as is_cancelled from orders o left join cancels c on o.id=c.order_id group by payment_avl,is_cancelled order by sums asc

These results are then aggregated and Google Json based charts API is used for displaying several charts to the manager. These reports gives manager insight into how much money/oil is pending or cancelled or is paid for by the users.

This application takes care of all the requirements that have been specified by the friend for Oil transaction management system. Possible future enhancements can be as listed as follows.

Possible future enhancements:

1. SSL Needs to be added to the website in order to make connection to the website secure and prevent Man in the Middle attack as well as Session hijacking attacks.
2. Support For Edit profile functionality
3. An external system will be in place that will populate the data into Oil_price table - They will be using oil_price_loader to do the same.

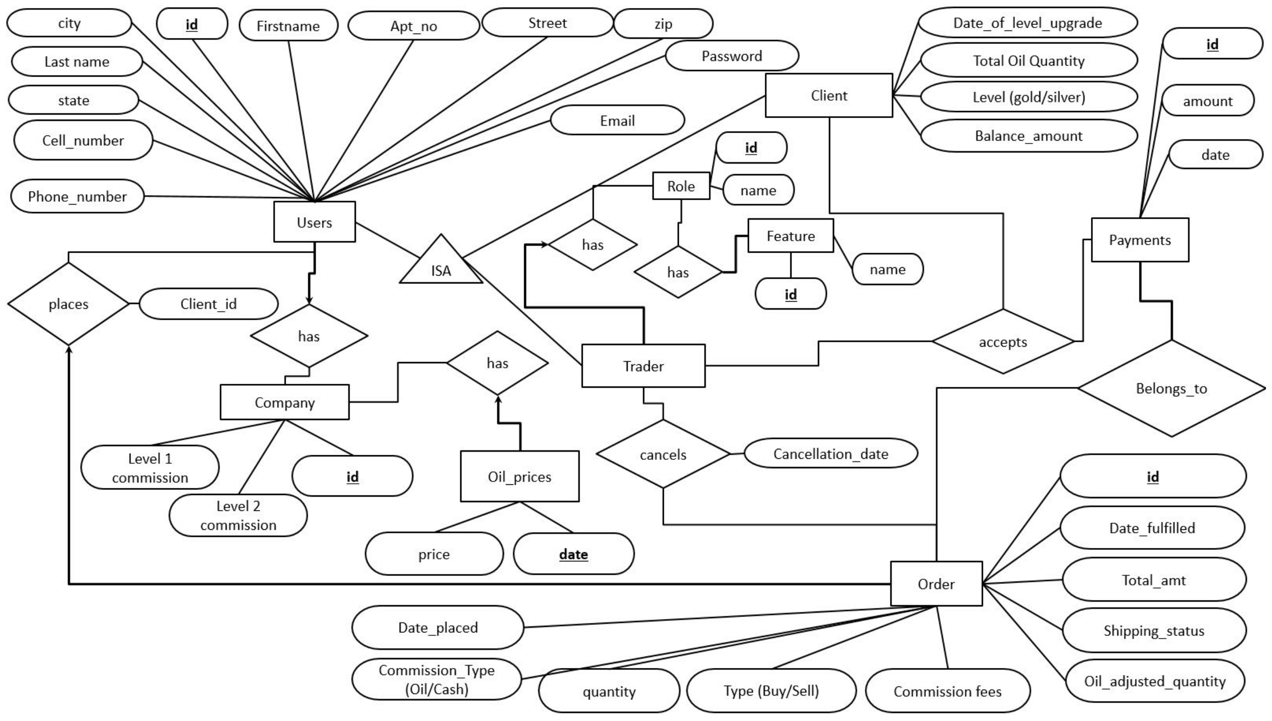
Application Deployment:

Please refer to OTS-DeploymentGuide.pdf for details on how to deploy the oil transaction system

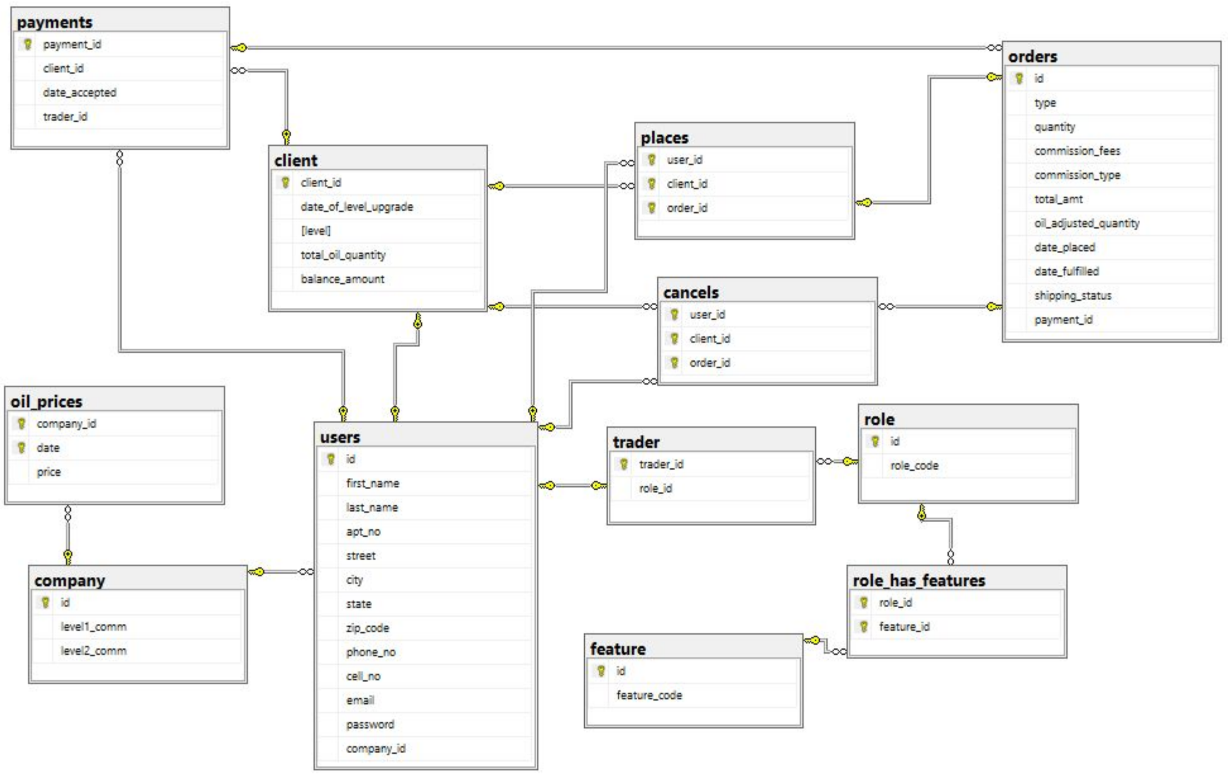
Appendix

1. ER Diagram of Oil Transaction System

<u>ER Diagram</u>

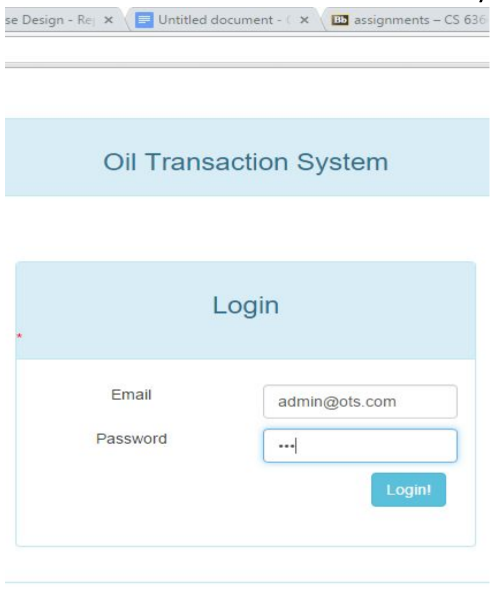


<u>2. Relational Model</u>: Shows Primary keys, foreign keys and relationship between relations.



3. Application Screenshots

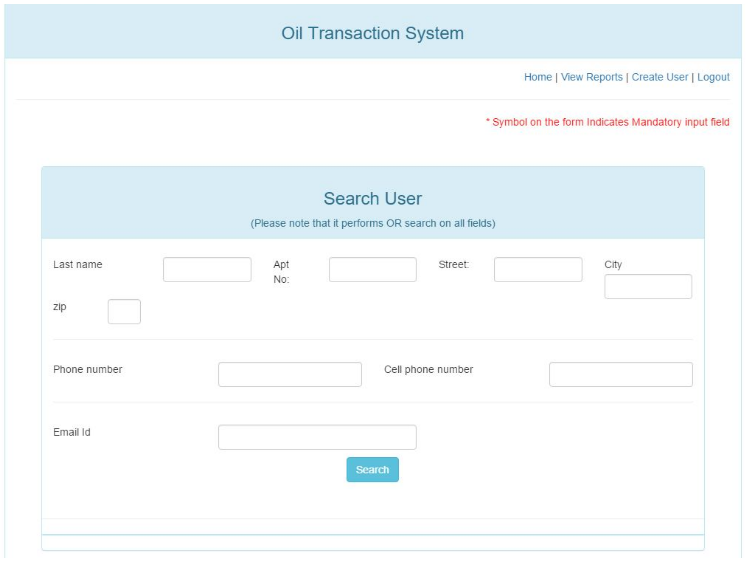
1. Login Screen.
2. Admin Home Page -Admin Have access to “View Reports”, “Create User” functionality along with “Search User” functionality.



Oil Transaction System

Login

Email	admin@ots.com
Password	•••



Oil Transaction System

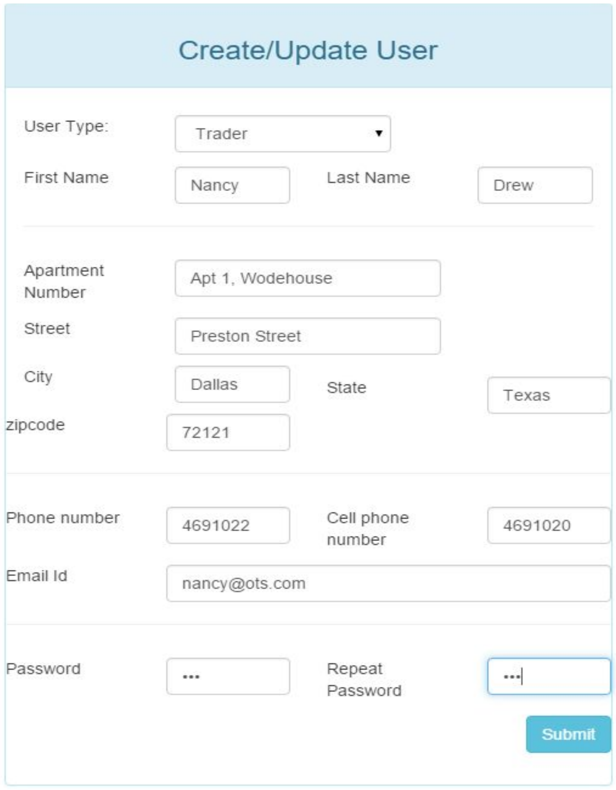
* Symbol on the form Indicates Mandatory input field

Search User

(Please note that it performs OR search on all fields)

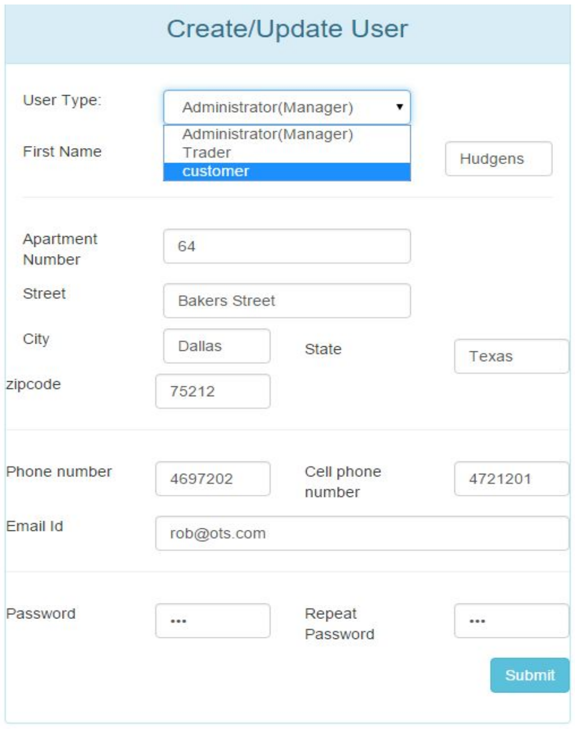
Last name		Apt No:		Street:		City	
zip							
Phone number				Cell phone number			
Email Id							

3. Admin can Create another Admin, Trader or a Client



Create/Update User

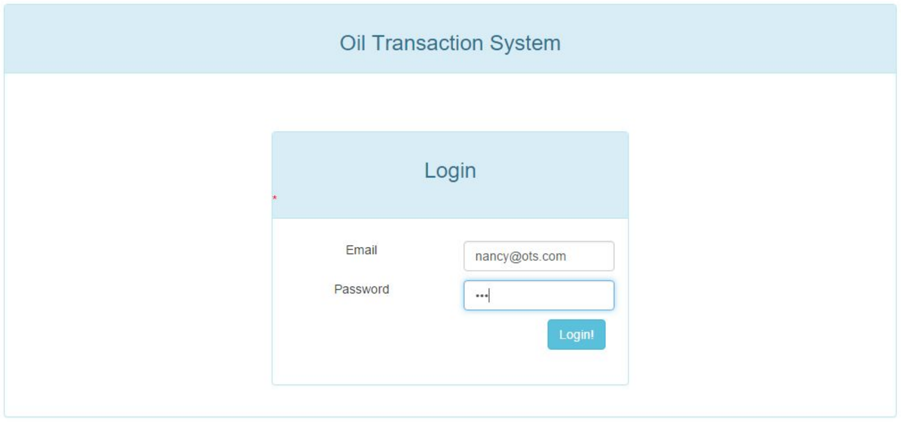
User Type:	Trader		
First Name	Nancy	Last Name	Drew
Apartment Number	Apt 1, Wodehouse		
Street	Preston Street		
City	Dallas	State	Texas
zipcode	72121		
Phone number	4691022	Cell phone number	4691020
Email Id	nancy@ots.com		
Password	•••	Repeat Password	•••



Create/Update User

User Type:	Administrator(Manager)		
First Name			Hudgens
Apartment Number	64		
Street	Bakers Street		
City	Dallas	State	Texas
zipcode	75212		
Phone number	4697202	Cell phone number	4721201
Email Id	rob@ots.com		
Password	•••	Repeat Password	•••

4. Trader Logs in

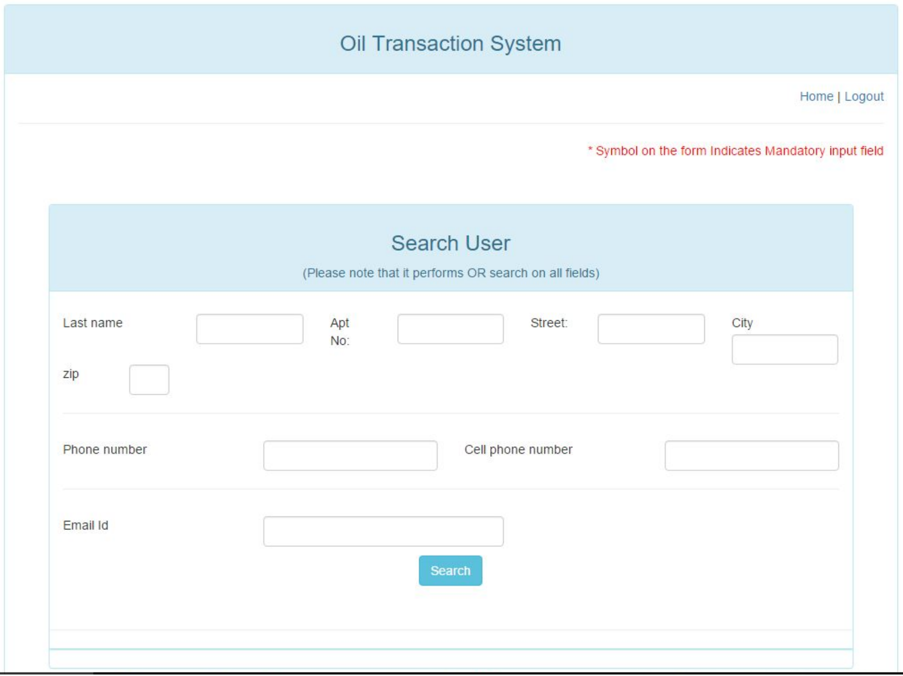


Oil Transaction System

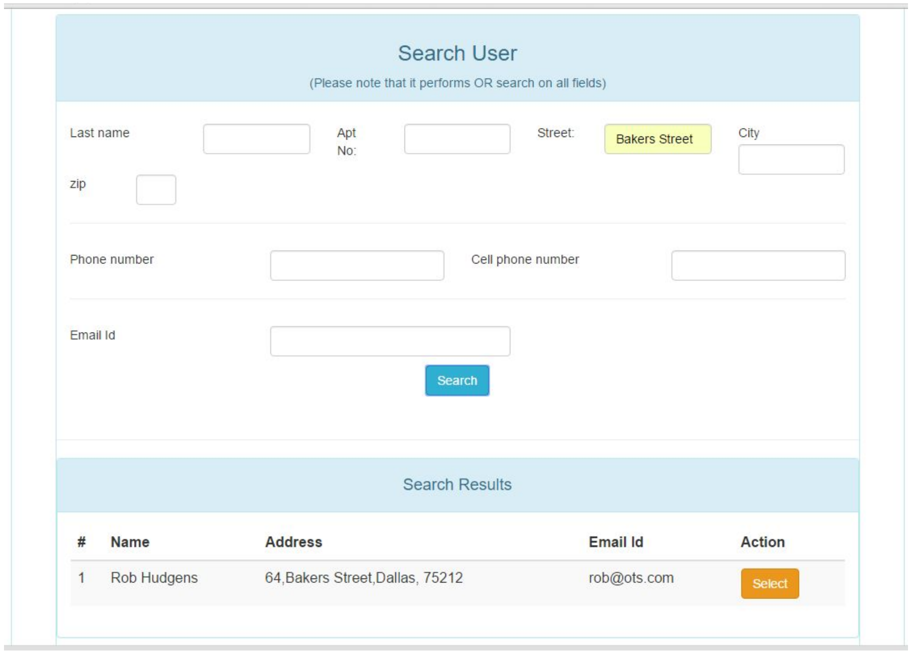
Login

Email	nancy@ots.com
Password	•••

5. Home Page for nancy@ots.com (trader) - Search User Screen.



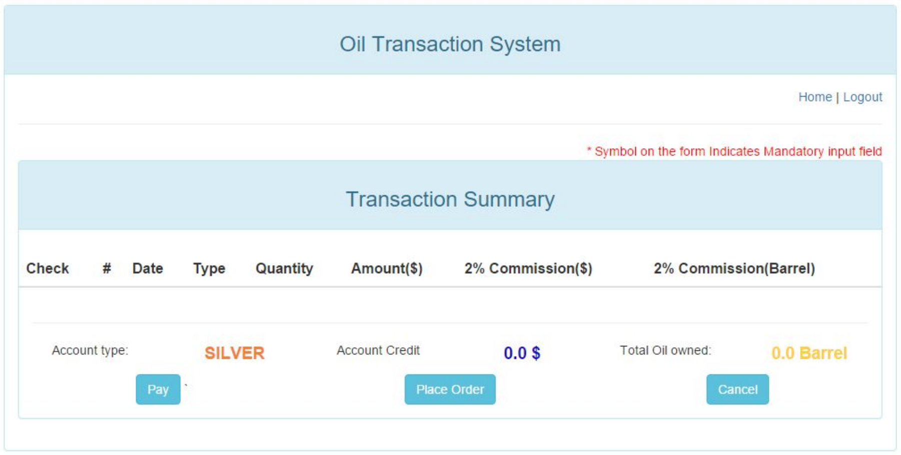
6. Trader searches user by Street



Street: Bakers Street

#	Name	Address	Email Id	Action
1	Rob Hudgens	64,Bakers Street,Dallas, 75212	rob@ots.com	Select

8.Nancy(trader) selects a user and sees Rob’s Order summary



Oil Transaction System

Home | Logout

* Symbol on the form Indicates Mandatory input field

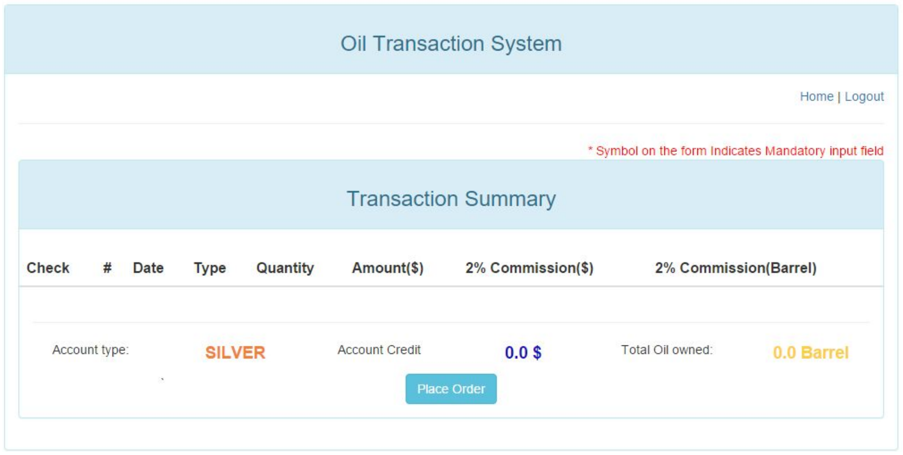
Transaction Summary

Check	#	Date	Type	Quantity	Amount($)	2% Commission($)	2% Commission(Barrel)

Account type:	SILVER	Account Credit	0.0 $	Total Oil owned:	0.0 Barrel

Pay Place Order Cancel

9. Home page for rob@ots.com (Client) when he logs in (No access to Cancel/Pay functionality)



Oil Transaction System

Home | Logout

* Symbol on the form Indicates Mandatory input field

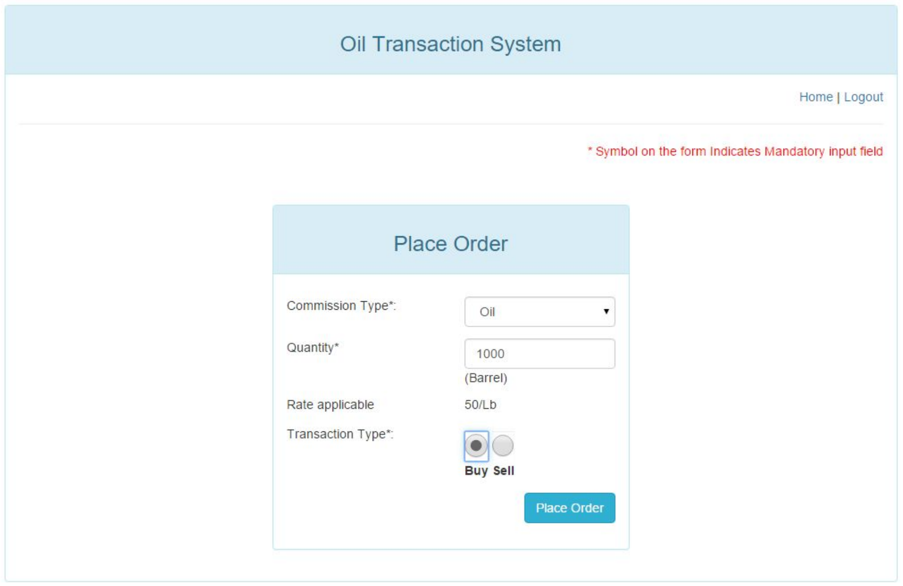
Transaction Summary

Check	#	Date	Type	Quantity	Amount($)	2% Commission($)	2% Commission(Barrel)

Account type:	SILVER	Account Credit	0.0 $	Total Oil owned:	0.0 Barrel

Place Order

10. Rob Buys 1000Lb of Oil



Oil Transaction System

Home | Logout

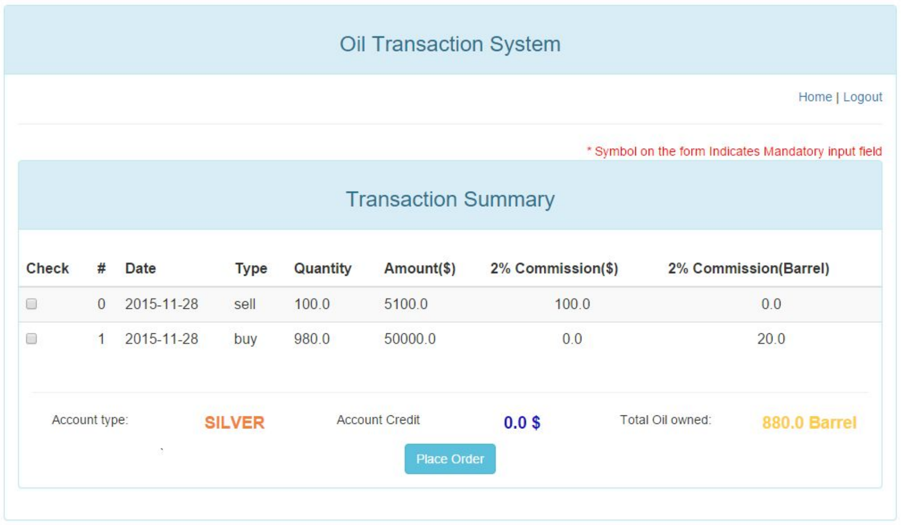
* Symbol on the form Indicates Mandatory input field

Place Order

Commission Type*:	Oil
Quantity*	1000 (Barrel)
Rate applicable	50/Lb
Transaction Type*:	[✓ Buy] Sell

Place Order

11. Robs transaction summary now also displays his recent transaction



Oil Transaction System

Home | Logout

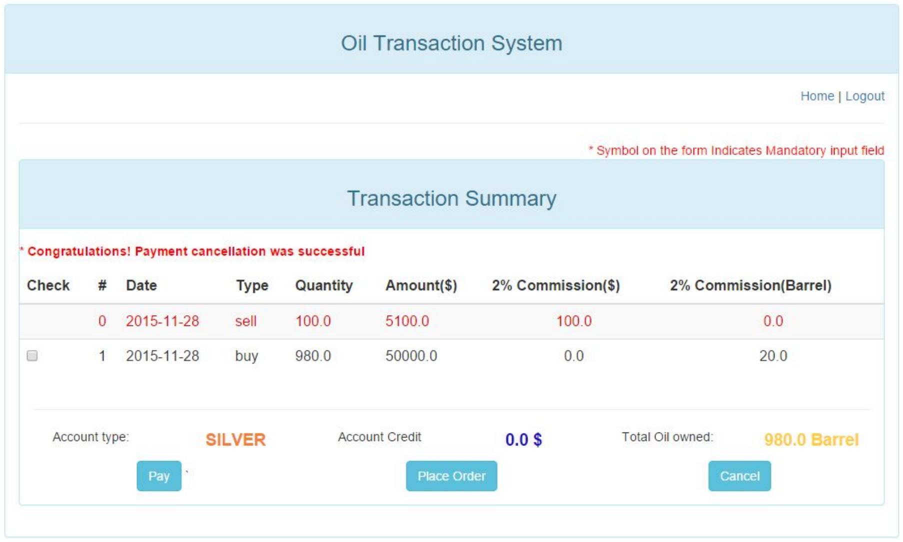
* Symbol on the form Indicates Mandatory input field

Transaction Summary

Check	#	Date	Type	Quantity	Amount($)	2% Commission($)	2% Commission(Barrel)
☐	0	2015-11-28	sell	100.0	5100.0	100.0	0.0
☐	1	2015-11-28	buy	980.0	50000.0	0.0	20.0

Account type: SILVER Account Credit 0.0 $ Total Oil owned: 880.0 Barrel

12. Once done, he intends to make payment for the oil bought and cancel the "sell". Order summary now displays cancelled record in red color.



Oil Transaction System

Home | Logout

* Symbol on the form Indicates Mandatory input field

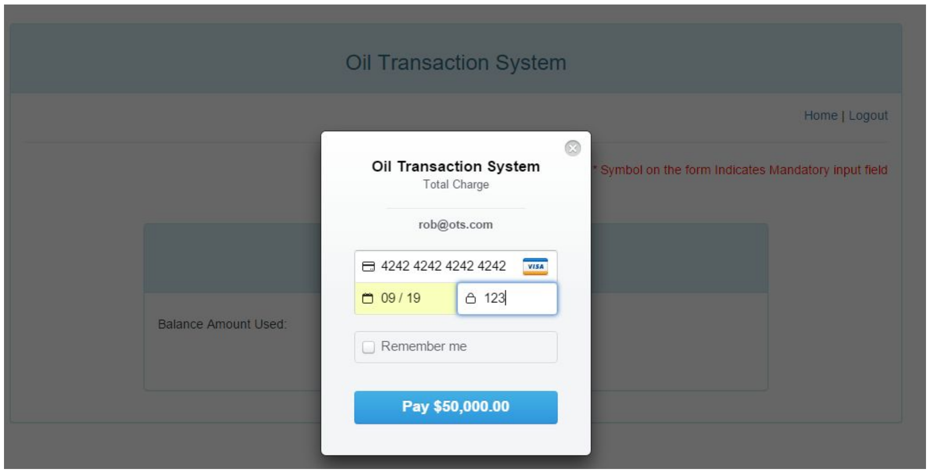
Transaction Summary

* Congratulations! Payment cancellation was successful

Check	#	Date	Type	Quantity	Amount($)	2% Commission($)	2% Commission(Barrel)
	0	2015-11-28	sell	100.0	5100.0	100.0	0.0
☐	1	2015-11-28	buy	980.0	50000.0	0.0	20.0

Account type: SILVER Account Credit 0.0 $ Total Oil owned: 980.0 Barrel

13.Nancy Accepts payment from user on phone, Please find below, Charge acceptance screen



Oil Transaction System

Home | Logout

* Symbol on the form Indicates Mandatory input field

Balance Amount Used:

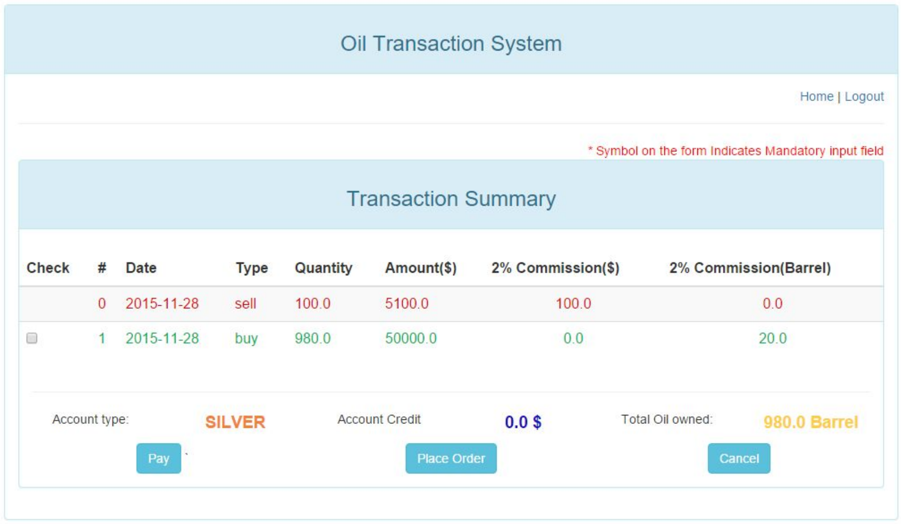
Oil Transaction System
Total Charge

rob@ots.com

4242 4242 4242 4242

09 / 19 123

14. Nancy can now see that payment went through, hence the order is now displayed in green color



Oil Transaction System

Home | Logout

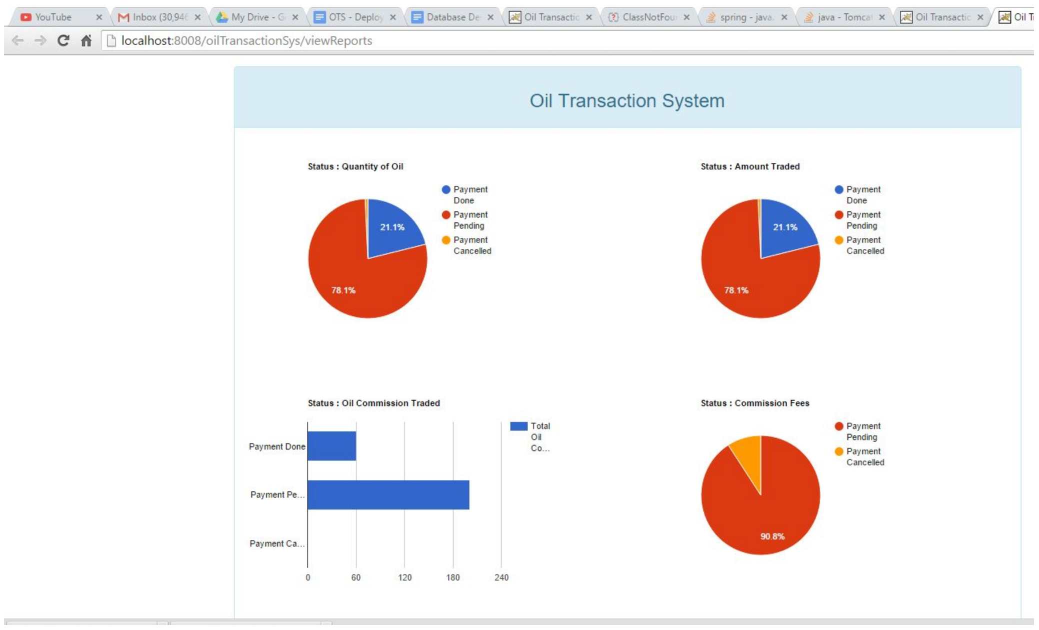
* Symbol on the form Indicates Mandatory input field

Transaction Summary

Check	#	Date	Type	Quantity	Amount($)	2% Commission($)	2% Commission(Barrel)
	0	2015-11-28	sell	100.0	5100.0	100.0	0.0
☐	1	2015-11-28	buy	980.0	50000.0	0.0	20.0

Account type: SILVER   Account Credit 0.0 $ Total Oil owned: 980.0 Barrel

15. Admin logs in and clicks on view reports button. He can see various reports of the data.



Triggers/Constraints defined

1. ON insert trigger either commission fees or oil adjusted quantity should be specified.

```
DELIMITER $$
CREATE TRIGGER order_commission AFTER INSERT ON OTS.ORDERS
FOR EACH ROW
BEGIN
  IF( ots.orders.commission_fees IS NULL && ots.orders.oil_adjusted_quantity)
        THEN
   INSERT INTO log (error_message) VALUES("commission fees not inserted");
  END IF;
```

```
END;
$$
DELIMITER ;
```

2. A Trigger that executes on insert of Order and if client has more than 30 records in a month, it updates the level field in client table to “GOLD”

```
DELIMITER $$
CREATE TRIGGER update_level AFTER INSERT ON OTS.PLACES
FOR EACH ROW
BEGIN
   UPDATE client SET level="GOLD" WHERE client_id=row.clint_id and (SELECT
COUNT(ID)  FROM ORDERS o JOIN PLACES p on p.order_id=o.id where
MONTH(o.date_placed) = MONTH(CURRENT_DATE)
AND YEAR(DATE_PLACED) = YEAR(CURRENT_DATE)  and
p.client_id=ROW.client_id)>=30;
 END;
$$
DELIMITER ;
```