A Web Based Oil Transaction System

Class Project for CS6360.006

Jiashuai Lu (Jxl173630)
Department of Computer Science, University of Texas at Dallas

December 6, 2017

1. Conceptual Design

Being an web based oil transaction system, according to the requirements, firstly, we at least need one entity set to store the users which can sign in this system. For this, we initiate our conceptual schema by adding an entity set *Users*.

Besides, this system also need we to separate users' rights between normal clients, traders, and managers, which could be a proper applicable scenario of *isa* relationship. After we use an *isa* relationship to capture the inheritance between users and its sub-classes, we attach all the common attributes to *Users*. We use *uid* as a user's primary key. By considering that use uid to login the system is inconvenient and will expose the uid, we also give every user an email attribute as a candidate key. This will be used as this user's login id for authenticating. For security, in the password attribute, we store not the plain but the encrypted text. At last, for offering different view to each kind of user, we also store the user type information in *Users*.

As a descendant of user, client is one of the most important entity of our system. Every transaction must be assigned a client as its owner. Also sometimes a transaction could be submitted by a trader of clients. In the requirements, this system should have three type base transactions. Two of them are buying and selling oils, and the other one is payment. After an elaborated analysis, we decide to give *Clients* entity set the following attributes: *first_name*, *last_name*, and other personal information of a client as well as *level*, *oil_balan*, and *text_balan* which are used to store a client's business information.

Moreover, in the ER diagram in Appendix A, we also have *Traders* and *Managers* as *Clients*' sibling entity sets to store corresponding information.

The second big part of the conceptual schema is the entity sets about transactions. Similar with the users part, we use an *isa* relationship as well. In our design, every transaction, no matter oil related transaction or payment transaction, needs a transaction id for primary key, a transaction date to store when it is occurred. Furthermore, also for differentiate them, we add a type attribute in the *Transactions* entity set.

Clients_trans and *Trader_trans* are used to capture the relation between *Clients* and *Transactions* as well as *Traders* and *Transactions* respectively. We also add some constraints on these relationships according to the reality. For example, every transaction must has exactly one owner (a many to one key constraint and a participant constraint) and every transaction has at most one trader (a many to one key constraint).

Oil_transactions is an entity set we used to express oil related transactions which have two kinds: buying and selling. For every oil transaction, a client could choose to pay for the commission with either her/his oil balance or with cash. We store commission by oil and commission by cash separately because this would make our development of the application easier. Besides, inspired by a lot of online bank system, we store the value of cash balance and oil balance after a transaction's effect in a transaction to let our customers be able to track their account information conveniently. Additionally, we store the current oil price, executed commission rate, and oil amount of this transaction as well to show the customers what is exactly happened during a transaction.

Being the only method of topping up in this system, a client can use a payment transaction anytime to pay for her/his cost of some previous oil transactions. We store only amount and the cash balance after a payment in a *Payment* entity.

After all, there are some entity sets which are auxiliary but not have explicit relationship with other entities. *Cancel_logs* are used to keep canceled transactions information for auditing. It is supposed to have every possible attribute of a transaction plus a log id as its primary key and a cancel date to track when this transaction is canceled.

2. Relational Schema

Given the ER diagram we get in the first part, we convert it to the following relational schema.

Users(*uid*: integer, *email*: string, *password*: string, *type*: integer)

Every children of *Users* also need to have the *uid* for referencing to its parent. And we do not explicitly have a sole entity set for managers, since managers do not have different attributes with users.

Clients(*uid*: integer, *first_name*: string, *last_name*: string, *phone_num*: string, *cellph_num*: string, *state*: string, *city*: string, *zip_code*: string, *level*: integer, *cash_balan*: float, *oil_balan*: float)
Traders(*uid*: integer, *trader_name*: string)

We combine the relationship *clients_trans* and *traders_trans* into the *transactions* relation, but use a not null foreign key to capture one transaction has exact one client and a foreign key to capture one transaction has at most one trader.

Clients_trans(*t_id*: integer, *t_date*: datetime, *c_id*: integer, *tr_id*: integer, *type*: integer)
Oil_transactions(*t_id*: integer, *comm_oil*: float, *comm_cash*: float, *oil_balan*: float, *cash_balan*: float, *comm_rate*: float, *price*: float, *amount*: integer)
Payments(*t_id*: integer, *cash_balan*: float, *amount*: integer)
Comm_rates(*rate_date*: datetime, *rate_silver*: float, *rate_gold*: float)
Oilprices(*price_date*: datetime, *price*: float)
Cancel_logs(*l_id*: integer, *c_date*: datetime, *t_id*: integer, *comm_oil*: float, *comm_cash*: float, *oil_balan*: float, *cash_balan*: float, *comm_rate*: float, *price*: float, *amount*: integer)

3. Software Architecture

Our system is a typical web application using MySQL 5.7 as its DBMS. We develop our backend *Brynhildr* based on a very light python webapp framework, Flask, and develop our frontend *Sigrdrifa* by using JavaScript mainly based on jQuery and Bootstrap which are also light framework. Figure 1 shows our system's architecture.

For easing the development and increasing the efficiency of transferring content between frontend and backend as well as improving our system's security, our backend offering web services in a RESTful way (Representational state transfer). This gives our frontend the ability to require and use web services in a consistent and predefined set of stateless operations. Moreover, this architecture also ensures the privacy because the backend will check every request's authority to see if this request has enough right to go to next step.

By using python3.7 as our backend development program language, we also use some extension of Flask to make our work easier. For example, we use Flask-Restful to develop our RESTful api, Flask-HttpAuth to handle authority control, Flask-MySQLdb to deal with all database operations in backend, Flask-Cors to control Cross-Origin Resource Sharing.

In the frontend, besides jQuery and Bootstrap, we use blueimp's JavaScript-MD5 module for encrypting the user's password in login step. Similarly, for the manager's view for looking up aggregation information of transactions, we use Xdan's jQuery-datetimepicker for the datetime picker addon (although HTML5 mode has its own datetime type input element).

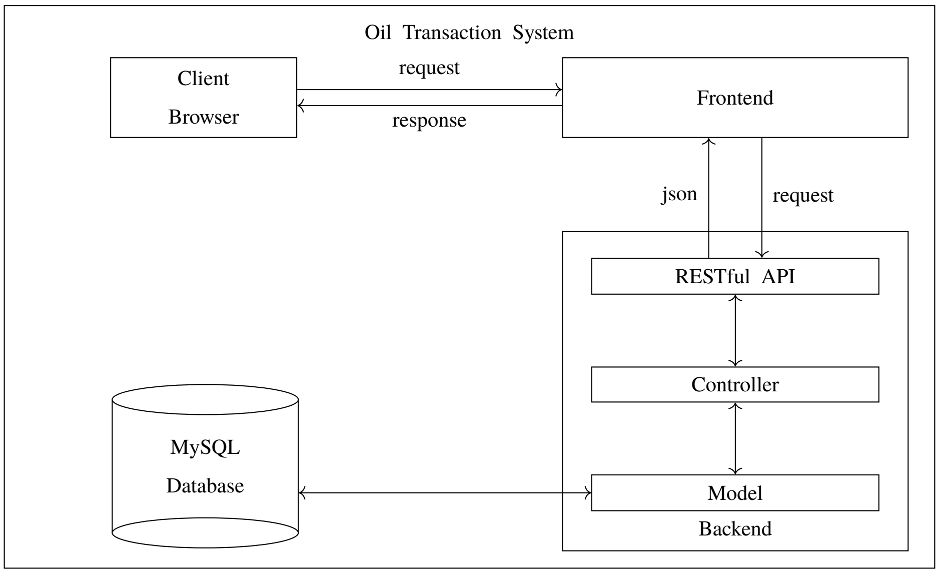


Figure 1. Software Architecture of Our System

4. Overview of Code

Being a restful web service backend, we only need to concentrate to implement necessary business logics and which part of data should be shown to specified type's user in the backend.

With the help of several extension package we refer to in the last part, our backend code are mainly separated to 6 parts. Table 1 demonstrates their functions briefly.

File name	Main Function
auth.py	token authentication
models.py	mapping to our entity sets
query.py	prepare statements
resources.py	RESTful api
utils.py	auxiliary methods (e.g. parse_int)
brynhildr.py	initial backend process

Table 1. Backend code function

For ensuring the atomicity and durability of our transactions, we use stored procedures for performing all business logical transactions. Table 2 shows some main procedures' function of our system. By the way, we also use a stored procedure to implement monthly rating level of clients. After we create a rating procedure, we use mysql's event to create a monthly schedule which will call our rating procedure on 00:00 AM of the first day every month.

Procedure	Main Function
buyoilproc	purchase oil on specified client account
selloilproc	sell oil
paymentproc	topping up cash on a client account
rating_monthly	rating every client according to last month's transactions
aggproc	gather aggregation information for manager's request
cancelproc	deal with a cancel transaction request

Table 2. Stored procedures function

Move on the frontend, since our application is based on RESTful web services, we organize our frontend in only one main html. The system will load corresponding content html modules for each type of users when a user has been passed the authentication module of backend and get her/his token.

If their is no valid token, the frontend will load the html module of sign in part and prompt the user to login. We use Table 3 to introduce main part of code in frondend.

File name	**Main Function**
index.html	main interface
sigrdrifa.js	main process of frontend
core_client.(html/js)	client module's interface and function
core_trader.(html/js)	trader module's interface and function
core_manager.(html/js)	manager module's interface and function
login.(html/js)	sign in method and log out
pagination.js	paginate transactions

Table 3. Frontend code function

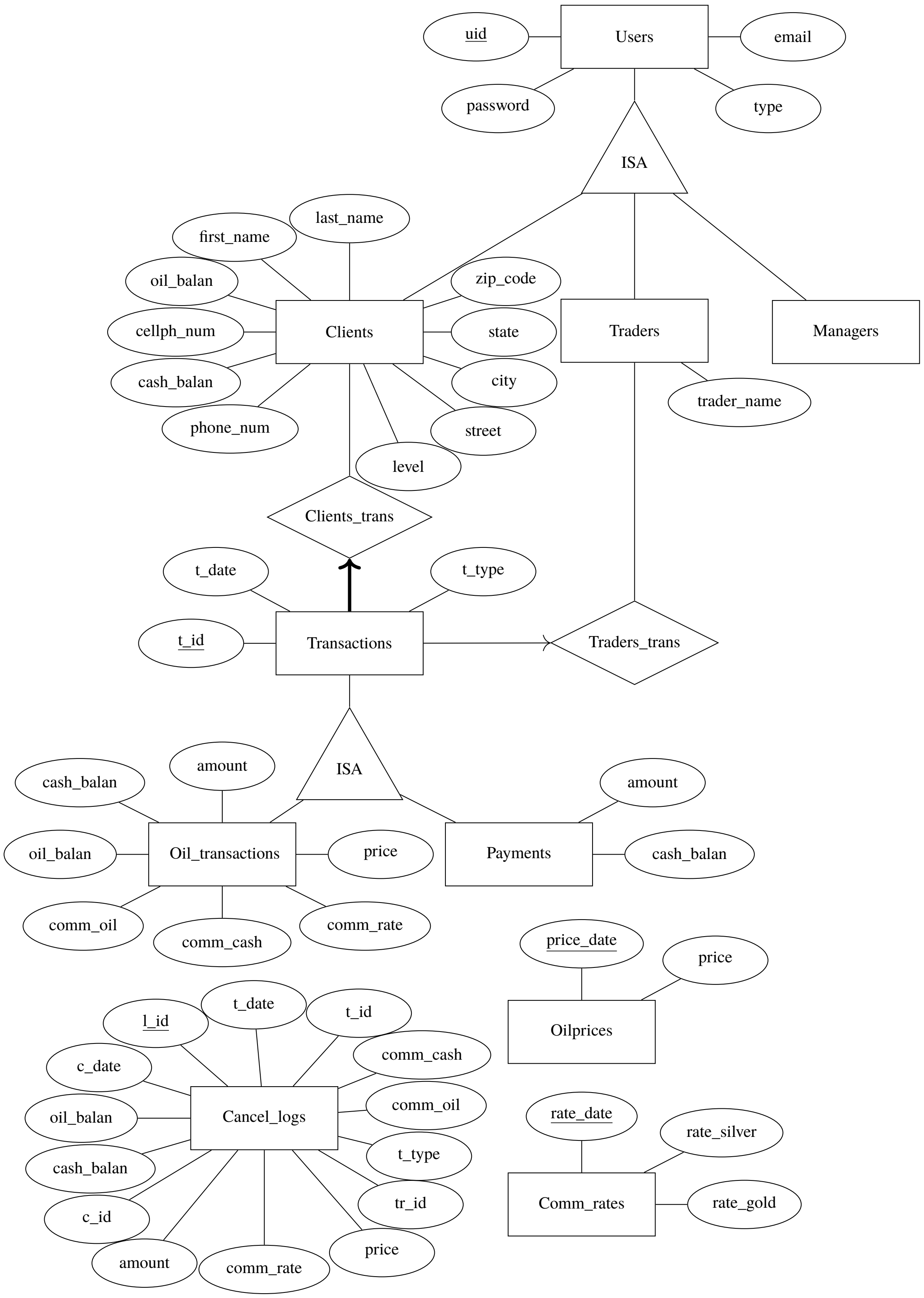
We also use an open source package JavaScript-MD5 for implementing encrypting the user's password in the frontend and then include the credential information into signin request.

MySQL 5.7 is the DBMS we choose for our system. In the database, we create 9 tables according to our relational schema, and the create table sentences are in the sqls.txt file. Besides, all the stored procedures, triggers, events we use in our system all also included in the source code files.

That is all about our web based oil transaction system. Thank you !

(Appendix A is in the last page)

Appendix A: ER Diagram



Users
uid
email
password
type
ISA
Clients
Traders
Managers
first_name
last_name
oil_balan
cellph_num
cash_balan
phone_num
zip_code
state
city
street
level
trader_name
Clients_trans
Traders_trans
Transactions
t_date
t_type
t_id
ISA
Oil_transactions
Payments
cash_balan
amount
oil_balan
price
comm_oil
comm_cash
comm_rate
amount
cash_balan
Oilprices
price_date
price
Cancel_logs
l_id
t_date
t_id
c_date
comm_cash
oil_balan
comm_oil
cash_balan
t_type
c_id
tr_id
amount
comm_rate
price
Comm_rates
rate_date
rate_silver
rate_gold