University of British Columbia, Department of Computer Science

CPSC 304

Summer 2018

Project Final Report

Project Formal Specification

Group Members:

Name	Student Number	Unix ID	Tutorial Section	Email Address
Wongelawit Teka Zewde	32493141	a8d0b	T1A	gospel.teka@gmail.com
Stephanie Wu	60030137	a5I0b	T1A	stephwu2000@gmail.com
Daniela Shklover	18491143	u5s0b	T1A	danielashklover@gmail.com
Ben Walker	34334169	m3c1b	T1E	bnwlkr@icloud.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above.

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia.

PROJECT DESCRIPTION:

We built a stock and cryptocurrency exchange tool which allows traders to buy and sell commodities. Trader can buy and/or sell commodity on specific exchanges and specify the quantity they want to purchase. Traders also can search for commodities and can also find the best commodity for arbitrage, where one exchange's asking price is lower than another exchanges biding price. A user can create an account with our system, with a username and password, then a user through our system can create accounts with different exchanges and fund their accounts. Afterwards a user is allowed to buy and sell on exchanges they have an account with if they have the currency to buy (as described above).

In addition, there are many other features a user can use, with various searches, including seeing their commodities owned, what exchanges they own commodities on, how many commodities they have, the value of their commodities (etc.), essentially its a full featured trading app, and it does everything one would need from a trading app, you can search by exchanges or you can search your own history transactions by date. One can also sort and see commodities that cost the most and their own commodities between a min and a max price. A user can also delete their account which triggers a delete cascade that deletes User info, Account info but not Transaction info, we only keep records of their old user ID and trades its not tied to the person. We used our own server and PHP and MySQL to accomplish all of this.

Additionally, we implemented a live API connection to update the Bitcoin Price on QuadrigaCX (Bonus!).

SCHEMA CHANGES:

The schema of our final product is a little different from our original proposition and the following are the changes we made.
We dropped Trend entity because we didn't find it useful to implement this feature for learning purpose of this course, which also led to not including Price and Volume entities. We also do not have Coin and Stock because we combined them under Commodity entities because there is no need to separate them as they are all still commodities and are bought and sold the same way with the same infor present, especially without our trends.

We have also made modification in attributes of some of the entities because we discovered we needed either more information or we didn't need all that we had. For instance the Trade entity now went from having a timestamp, a price, a user_id and a boolean for bought? , to having a timestamp, a user_id, an exchange_name, a commodity_name, a price, a quantity, and

move(which represents buy or sell), the reasons for these changes are we needed more information for our trades. The relations changed along side, since now Trade is related to the foreign key commodity_name and the foreign key exchange_name.

There are other similar situations where we added and removed attributes for similar reasons and it will be redundant to get into all of their details. However, here is another example of how we changed our Schema:

For out exchange entity we added fees to make it seem more real, these fees were important for arbitrage, where even if a ask price on one exchange was low the fees would also have an affect.

The new Schema is shown below:

SQL Table Schema

Primary Key: **Bold**
Foreign Key: *Italic*

- User (**id** INT, name CHAR(20), email CHAR(20) UNIQUE, password BINARY(64))

```
CREATE TABLE User(
     id INT PRIMARY KEY AUTO_INCREMENT,
     name CHAR(20),
     email CHAR(20) UNIQUE,
     password BINARY(64)
);
```

- Exchange (**name** CHAR(20), website CHAR(20))

```
CREATE TABLE Exchange(
name CHAR(20)
PRIMARY KEY, website CHAR(20)
);
```

- Commodity (**name** CHAR(20))

```
CREATE TABLE Commodity(
name CHAR(20) PRIMARY KEY
);
```

- Traded_On(***commodity_name*** CHAR(20), ***exchange_name*** CHAR(20), fees DOUBLE)

```
CREATE TABLE Traded_On(
commodity_name CHAR(20),
exchange_name CHAR(20),
fees DOUBLE,
PRIMARY KEY (commodity_name, exchange_name),
FOREIGN KEY (commodity_name) REFERENCES
Commodity(name),
FOREIGN KEY (exchange_name) REFERENCES Exchange(name)
);
```

- Account (**user_id** INT, ***exchange_name*** CHAR(20), ***commodity_name*** CHAR(20), value DOUBLE)

```
CREATE TABLE Account(
user_id INT,
exchange_name CHAR(20),
commodity_name CHAR(20),
value DOUBLE,
PRIMARY KEY (user_id, exchange_name, commodity_name),
FOREIGN KEY (user_id) REFERENCES User(id)
ON DELETE CASCADE,
FOREIGN KEY (exchange_name) REFERENCES Exchange(name),
FOREIGN KEY (commodity_name) REFERENCES
Commodity(name)
CONSTRAINT balance_check CHECK (NOT value<0)
);
```

- Trade (**timestamp**, *user_id* INT, *exchange_name* CHAR(20), *from_commodity_name* CHAR(20), *to_commodity_name* CHAR(20), from_value DOUBLE, to_value DOUBLE)

```
CREATE TABLE Trade (
timestamp TIMESTAMP PRIMARY KEY ON UPDATE
CURRENT_TIMESTAMP,
user_id INT,
exchange_name CHAR(20),
from_commodity_name CHAR(20),
```

```
to_commodity_name CHAR(20),
from_value DOUBLE,
to_value DOUBLE,
FOREIGN KEY (user_id) REFERENCES User(id)
    ON DELETE CASCADE,
FOREIGN KEY (exchange_name) REFERENCES Exchange(name),
FOREIGN KEY (to_commodity_name) REFERENCES
Commodity(name),
FOREIGN KEY (from_commodity_name) REFERENCES
Commodity(name)
);
```

- Matric (**id** INT PRIMARY KEY, *commodity_name* CHAR(20), *exchange_name* CHAR(20), bid INT, ask INT, volume INT,)

```
CREATE TABLE Metric(
id INT PRIMARY KEY,
commodity_name CHAR(20),
exchange_name CHAR(20),
bid INT,
ask INT,
volume INT,
FOREIGN KEY (commodity_name, exchange_name) REFERENCES
Traded_On (commodity_name, exchange_name)
    ON DELETE CASCADE
);
```

PLATFORMS:

We used a different platform from what we proposed because there were some limitations on the UBC Oracle server. The following are the platforms used

- MySQL
- PHP
- Linux Server

SYSTEM FUNCTIONALITY (ACCOMPLISHMENT):

Trader.php

- Buy functionality: allows trader to buy commodities of specific exchange and quantity. (UPDATE)
- Sell functionality: allows trader to buy commodities of specific exchange and quantity. (UPDATE)
- It creates and update a record of the transaction.

Arbitrageur.php

- Searches for the best bid of a give commodity

Updater.php

Gatekeeper.php

- Create an account
- Create exchange account
- Create fund account
- Checks for credentials using email and password of the trader

SQL QUERIES:

User Creates Different kinds Accounts: A user can create an account with our system, with a username and password, then a user through our system can create accounts with different exchanges and fund their accounts.

Check if a user has an account with a specific exchange, or accounts with commodities.

```
SELECT count(*) FROM Account WHERE user_id=$user_id
AND exchange_name=$exchange_name AND
commodity_name=$commodity_name GROUP BY
commodity_name;
```

Create account:

```
SELECT User.id, User.name, User.email FROM User WHERE
SHA1('$password') = User.password and User.email =
'$email';
```

User create exchange account (UPDATE QUERY)

```
INSERt INTO Account (user_id, commodity_name,
exchange_name) SELECT $user_id, Commodity.name as
commodity_name, Traded_On.exchange_name FROM Commodity
JOIN Traded_On ON Traded_On.commodity_name =
Commodity.name where
Traded_On.exchange_name='$exchange_name';
```

User fund their exchange account (`UPDATE QUERY`)

```
UPDATE Account SET value = value + $amount  WHERE
user_id=$user_id AND commodity_name='$commodity_name'
AND exchange_name='$exchange_name';
```

Check credentials

```
SELECT User.id, User.name, User.email FROM User WHERE
SHA1('$password') = User.password AND User.email =
'$email';
```

Trader buying or selling commodity Functionality: Purchases a commodity in specified quantity on a specified exchange. Sells a commodity in specified quantity on a specified exchange.

Check if a user has exchange account

```
SELECT * FROM Account WHERE
exchange_name='$exchange_name' AND user_id=$user_id;
```

Get asking price for a potential transaction:

```
SELECT Metric.ask FROM Metric JOIN Traded_On ON
Metric.commodity_name = Traded_On.commodity_name and
Metric.exchange_name = Traded_On.exchange_name where
Traded_On.exchange_name='$exchange_name' and
Traded_On.commodity_name ='$commodity_name';
```

Get the bid price of a particular commodity on particular exchange:

```
SELECT Metric.bid FROM Metric join Traded_On ON
Metric.commodity_name = Traded_On.commodity_name AND
Metric.exchange_name = Traded_On.exchange_name WHERE
Traded_On.exchange_name='$exchange_name' and
Traded_On.commodity_name ='$commodity_name';
```

Get Balance: (Can get based on Exchange/Commodity variations):

```
SELECT Account.value FROM Account WHERE
commodity_name='$commodity_name' and user_id=$user_id
and exchange_name='$exchange_name';
```

Buy or Sells Stock and Creates Transaction Record by Trigger (BONUS!)

```
INSERT INTO Trade (user_id, exchange_name,
commodity_name, price, quantity, move) VALUES
($user_id, '$exchange_name', '$commodity_name',
$price, $quantity, '$move');
```

Search Functionality: Obtain the best deal for arbitrage trading (the deal with the biggest difference between prices on different exchanges i.e. with a relatively low ask price on one exchange and a relatively high bid price on another exchange).

Search for the best bid (BONUS!):

```
SELECT a.commodity_name, b.exchange_name AS
buy_excahange, a.exchange_name AS sell_exchange,
a.bid*(1.0-get_fees(a.exchange_name)) /
b.ask*(get_fees(b.exchange_name)+1.0) AS MARGIN FROM
ECOMM_METRICS a JOIN ECOMM_METRICS b ON
b.commodity_name = a.commodity_name AND NOT
a.exchange_name = b.exchange_name WHERE
a.bid*(1.0-get_fees(a.exchange_name))>b.ask*(get_fees(
b.exchange_name)+1.0;
```

User has summary of account information: For example shows user's trades since a certain amount of days ago or all of them.

Get total of each commodity: (Aggregation Query)

```
SELECT  Account.commodity_name, sum(Account.value)
FROM Account JOIN User On Account.user_id = User.id
WHERE User.id = $user_id group by
Account.commodity_name;
```

Get count of how many commodities account owns: (Aggregation Query)

```
SELECT count(Account.commodity_name) FROM Account JOIN
User ON Account.user_id = User.id where User.id =
$user_id;
```

Summary by exchange name and commodity name

```
SELECT Account.exchange_name, Account.commodity_name,
Account.value FROM Account JOIN User On
Account.user_id = User.id WHERE User.id = $user_id AND
exchange_name='$exchange_name' AND
commodity_name='$commodity_name';
```

Summary by exchange name

```
SELECT Account.exchange_name, Account.commodity_name,
Account.value FROM Account JOIN User ON
Account.user_id = User.id  WHERE User.id = $user_id
AND exchange_name='$exchange_name';
```

Summary by commodity name

```
SELECT Account.exchange_name, Account.commodity_name,
Account.value FROM Account left JOIN User ON
Account.user_id=User.id  WHERE
Account.commodity_name='$commodity_name';
```

Summary of account with no inputs

```
SELECT Account.exchange_name, Account.commodity_name,
Account.value FROM Account JOIN User ON
Account.user_id = User.id WHERE User.id = $user_id;
```

Division Query: Find the commodity(likely currencies) that is traded on all exchanges.

```
SELECT commodity_name FROM Traded_On GROUP BY
commodity_name HAVING count(*) = (select count(*) FROM
Exchange)")));
```

Commodity Information: gives summary of commodities and where they are traded including bid and ask price.

Get all Commodity Metric information based on exchange/commodity and variations of:

```
SELECT Metric.* from Metric JOIN (select
Commodity.name, Traded_On.exchange_name FROM Commodity
JOIN Traded_On ON Commodity.name =
```

```
Traded_On.commodity_name WHERE
Traded_On.exchange_name='$exchange_name') AS comms ON
Metric.commodity_name = comms.name and
comms.exchange_name = Metric.exchange_name;

SELECT Metric.* from Metric JOIN (select
Commodity.name, Traded_On.exchange_name FROM Commodity
JOIN Traded_On ON Commodity.name =
Traded_On.commodity_name) AS comms ON
Metric.commodity_name = comms.name and
comms.exchange_name = Metric.exchange_name where
Metric.commodity_name = '$commodity_name';

SELECT Metric.* FROM Metric JOIN (select
Commodity.name, Traded_On.exchange_name FROM Commodity
JOIN Traded_On ON Commodity.name =
Traded_On.commodity_name WHERE
Traded_On.exchange_name='$exchange_name') AS comms ON
Metric.commodity_name = comms.name and
comms.exchange_name = Metric.exchange_name WHERE
Metric.commodity_name='$commodity_name';
```

Get all Commodity Metric information between a min and max price: (BONUS!)

```
SELECT Metric.* FROM Metric JOIN Commodity ON
Metric.commodity_name = Commodity.name where
Metric.ask BETWEEN $min_price and $max_price;
```

Exchange Information: Get all info about Exchanges, can choose to get info based on specific exchange:

```
SELECT * FROM Exchange WHERE name = '$name'
SELET *  FROM Exchange
```

Aggregate Min and Max: Perform a nested aggregation query of selected type combination on ask prices of all commodities.

Minimum

```
SELECT commodity_name, min(ask) FROM(select
commodity_name, avg(ask) ask FROM Metric WHERE
```

```
commodity_name <> 'cad' GROUP BY commodity_name ORDER
BY ask asc) AS T;
```

Maximum

```
SELECT commodity_name, max(ask) FROM(select
commodity_name, avg(ask) ask FROM Metric where
commodity_name <> 'cad' GROUP BY commodity_name ORDER
BY ask desc) AS T;
```

Trade Information

Get info of most expensive trade done by user: (Aggregation)

```
SELECT timestamp, commodity_name, exchange_name,
max(price * quantity) AS spent FROM Trade WHERE
user_id=$user_id GROUP BY user_id;
```

Get info of how many trades a user has done: (Aggregation)

```
SELECT count(*) AS nO_trades FROM Trade WHERE
user_id=1 GROUP BY user_id;

SELECT * FROM Trade WHERE UNIX_TIMESTAMP(timestamp) >
$since AND user_id=$user_id;
SELECT * FROM Trade WHERE user_id = $user_id;
```

Delete Functionality: Delete a user, except for their trading record data: Causes a cascade.

```
DELETE FROM User WHERE user_id = '$user_id
```

FUNCTIONAL DEPENDENCIES

We don't have any functional dependencies in our schema. Therefore, it is trivially in BCNF.