Exchange Matching Engine Report

Yuqiao Liang yl543
Xiaodi Nie xn15

1. Overview

In this homework, we used python3 and postgreSQL database and implemented a simple stock exchange engine where multiple clients and the server used xml to interact. Client can send commands to server to create accounts, create symbols, post and query orders. Server can send back relevant information and error message in response to operations made by clients. Client and Server connect using sockets and every time a new connection is made(by client), a new thread will be created to take care of the following operations.

2. Functional test

For functional tests, we first started the server program and run several client program to send different commands for each function to the server.

2.1. Test creating user and symbol (run python3 client1.py)

Xml send by user:

```
<create><account id="666" balance="1000"/><account id="888"
balance="1000"/><symbol sym="AAA"><account id="666">100</account><account
id="888">100</account></symbol><symbol sym="BBB"><account
id="666">200</account><account id="888">200</account></symbol></create>
```

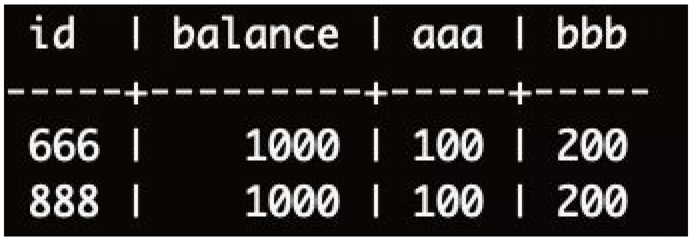
This will create a user with id “666” having initial balance of 1000 and a user with id “888” having initial balance of 1000 . Then it will create a symbol called “AAA” and assign 100 shares to user with id “666” and user with id “888”. Also will create a symbol called “BBB” and assign 200 shares to user with id “666” and user with id “888”.

Response is :

```
<?xml version="1.0" ?>
<results>
   <created id="666"/>
   <created id="888"/>
   <created id="666" sym="AAA"/>
   <created id="888" sym="AAA"/>
   <created id="666" sym="BBB"/>
   <created id="888" sym="BBB"/>
</results>
```

Response above means that we successfully create user “666” and user “888”. Also, we create a symbol called “AAA” and assign 100 shares to user with id “666” and user with id “888”. Also will create a symbol called “BBB” and assign 200 shares to user with id “666” and user with id “888”.

Below is the screenshot ACCOUNT table:



```
 id  | balance | aaa | bbb
-----+---------+-----+-----
 666 |    1000 | 100 | 200
 888 |    1000 | 100 | 200
```

2.2. Creating duplicate user

Xml send by user:

```
<create><account id="666" balance="1000"/></create>
```

This will create a user with id “666” having initial balance of 1000. But user id “666” already exists in database, so this will invoke error of duplicate user.

Response is :

```
<?xml version="1.0" ?>
<results>
  <error id="666">Account Already Exist!</error>
</results>
```

Response above means that we got error message of creating duplicate accounts.

2.3. Making transaction:

Xml send by user:

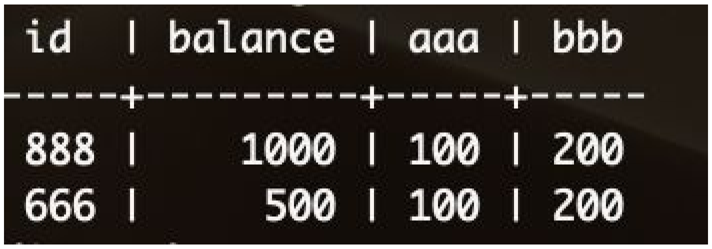
```
<transactions id="666"><order sym="AAA" amount="5" limit="100"/></transactions>
```

This transaction is saying that user “666” is trying to buy “AAA” for 5 shares and he is willing to pay 100 at most.

Response is :

```
<?xml version="1.0" ?>
<results>
  <order amount="5" id="2" limit="100" sym="AAA"/>
</results>
```

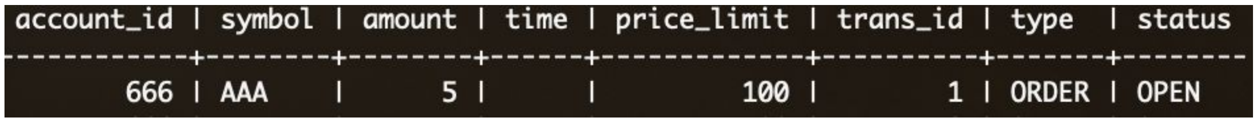
Below is the screenshot ACCOUNT table:



```
 id  | balance | aaa | bbb
-----+---------+-----+-----
 888 |    1000 | 100 | 200
 666 |     500 | 100 | 200
```

As we can see, balance of user “666” is deducted by 500 but the shares for AAA is not increased yet, because the order is still open, not executed.

Below is the screenshot TRANSACTION table:



```
account_id | symbol | amount | time | price_limit | trans_id | type  | status
-----------+--------+--------+------+------------+----------+-------+--------
       666 | AAA    |      5 |      |        100 |        1 | ORDER | OPEN
```

We can see that user “666” made an order to buy AAA, the maximum price is 100. The status is opened.

2.4. Making transaction with exceeded balance

Xml send by user:

<transactions id="666"><order sym="AAA" amount="5" limit="100"/></transactions>

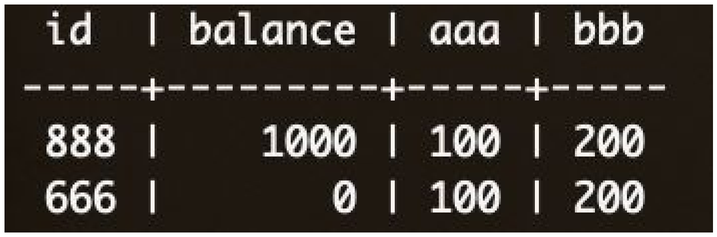
This transaction is saying that user “666” is trying to buy “AAA” for 5 shares and he is willing to pay 100 at most.We send it ***twice*** to test the insufficient balance.

Response is :

<?xml version="1.0" ?>
<results>
  <error amount="5" limit="100" sym="AAA">insufficient balance</error>
</results>

Because user “666” doesn’t have enough balance so server will response error.

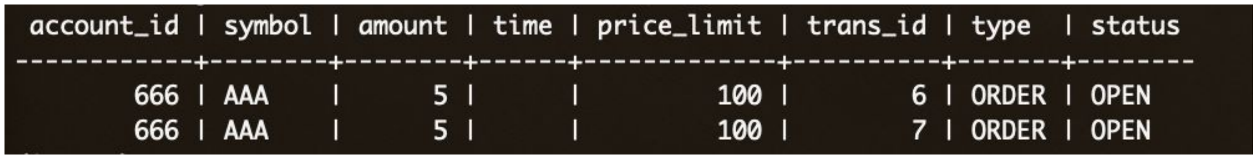
Below is the screenshot ACCOUNT table:



```
 id  | balance | aaa | bbb
-----+---------+-----+-----
 888 |    1000 | 100 | 200
 666 |       0 | 100 | 200
```

As we can see, balance of user “666” is deducted to 0.

Below is the screenshot TRANSACTION table:



```
 account_id | symbol | amount | time | price_limit | trans_id | type  | status
------------+--------+--------+------+-------------+----------+-------+--------
        666 | AAA    |      5 |      |         100 |        6 | ORDER | OPEN
        666 | AAA    |      5 |      |         100 |        7 | ORDER | OPEN
```

We can see that user “666” made two orders to buy AAA, the maximum price is 100. The status is opened. The third one is rejected so it’s not showing in the database.

2.5. Test matching order

Xml send by user:

<transactions id="888"><order sym="AAA" amount="-5" limit="80"/></transactions>

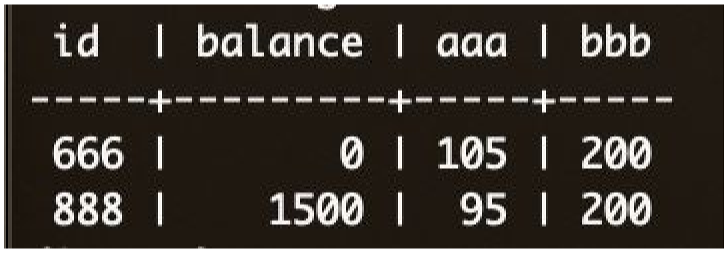
This transaction is saying that user “888” is trying to sell “AAA” for 5 shares and he is willing to accept 80 or above.

Response is :

```
<?xml version="1.0" ?>
<results>
   <order amount="-5" id="8" limit="80" sym="AAA"/>
</results>
```

This is correct because both user have enough money and the orders are both open.

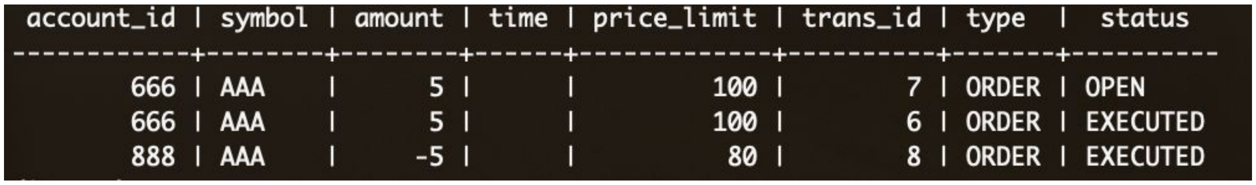
Below is the screenshot ACCOUNT table:



```
 id  | balance | aaa | bbb
-----+---------+-----+-----
 666 |       0 | 105 | 200
 888 |    1500 |  95 | 200
```

As we can see, balance of user “666” is deducted to 0 and user “666” have 105 shares of AAA. User “888” balance is increased by 500 and his share of AAA is deducted by 5 because he is selling AAA.

Below is the screenshot TRANSACTION table:



```
 account_id | symbol | amount | time | price_limit | trans_id | type  |  status
------------+--------+--------+------+-------------+----------+-------+----------
        666 | AAA    |      5 |      |         100 |        7 | ORDER | OPEN
        666 | AAA    |      5 |      |         100 |        6 | ORDER | EXECUTED
        888 | AAA    |     -5 |      |          80 |        8 | ORDER | EXECUTED
```

We can see that the first order made by user “666” has been matched with order made by user “888”. The final price is 100. Their status have been changed to EXECUTED.

2.6. 6. Test for non-existing account

Xml send by user:

```
<transactions id="555"><order sym="AAA" amount="5" limit="100"/></transactions>
```

User ‘555“ doesn’t exist so there should be an error.

Response is :

```
<?xml version="1.0" ?>
<results>
   <error amount="-5" limit="80" sym="AAA">ACCOUNT ID Doesn't Exist!</error>
```

</results>

2.7. Test for non-existing symbol

Xml send by user:
<transactions id="666"><order sym="ABC" amount="5" limit="100"/></transactions>
User 666“ does exist but there is no symbol named “ABC”.

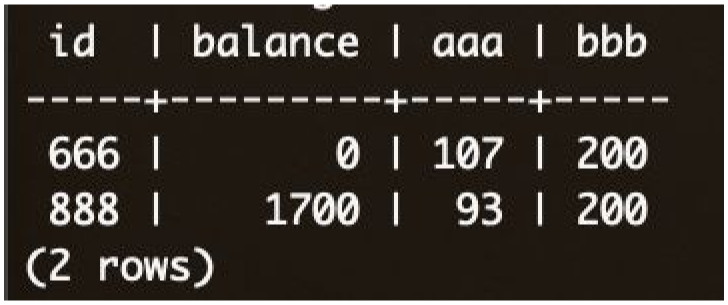
Response is :
```
<?xml version="1.0" ?>
<results>
   <error amount="-5" limit="80" sym="ABC">Symbol does not exist</error>
</results>
```

2.8. Test for partial order

Xml send by user:
<transactions id="666"><order sym="AAA" amount="5" limit="100"/></transactions>
User 666“ does exist but there is no symbol named “ABC”.

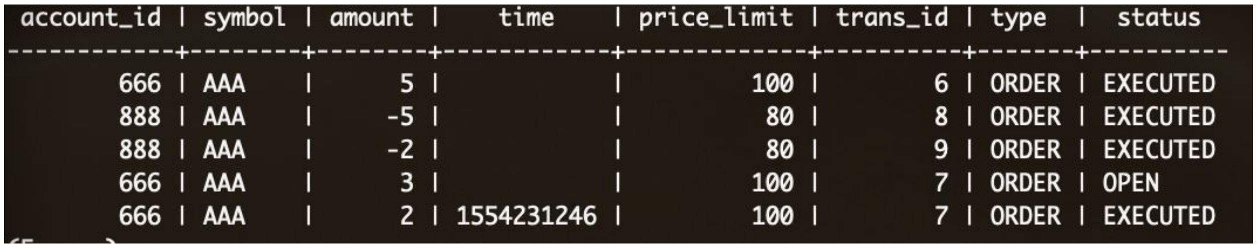
Response is :
```
<?xml version="1.0" ?>
<results>
   <error amount="-5" limit="80" sym="ABC">Symbol does not exist</error>
</results>
```

Below is the screenshot ACCOUNT table:


```
 id  | balance | aaa | bbb
-----+---------+-----+-----
 666 |       0 | 107 | 200
 888 |    1700 |  93 | 200
(2 rows)
```

As we can see, balance of user “666” is deducted to 0 and user “666” have 107 shares of AAA. User “888” balance is increased to 1700 and his share of AAA is deducted to 93 because he is selling AAA for 7 shares in total..

Below is the screenshot TRANSACTION table:



```
 account_id | symbol | amount |    time    | price_limit | trans_id | type  |  status
------------+--------+--------+------------+-------------+----------+-------+----------
        666 | AAA    |      5 |            |         100 |        6 | ORDER | EXECUTED
        888 | AAA    |     -5 |            |          80 |        8 | ORDER | EXECUTED
        888 | AAA    |     -2 |            |          80 |        9 | ORDER | EXECUTED
        666 | AAA    |      3 |            |         100 |        7 | ORDER | OPEN
        666 | AAA    |      2 | 1554231246 |         100 |        7 | ORDER | EXECUTED
```

We can see that the 3 shares of AAA is still open but 2 shares of AAA is executed. And they share the same trans_id.

2.9. Test for query

Xml send by user:

<transactions id="666"><query id="7"/></transactions>

We are now querying transaction id 7 under the user “666”.

Response is :

```
<?xml version="1.0" ?>
<results>
  <status id="7">
    <open shares="3"/>
    <executed price="100" shares="2" time="1554231246"/>
  </status>
</results>
```

We can see that transaction id 7 under the user “666” has two portions. One is already executed, the other is still open.

2.10. Test for cancel

Xml send by user:

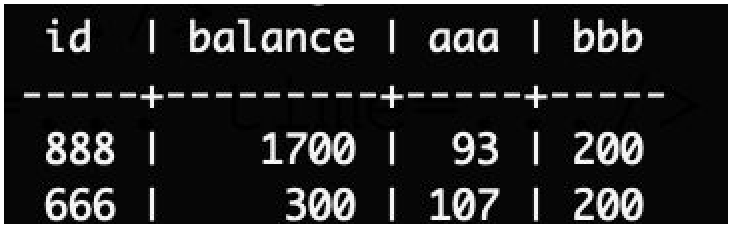
<transactions id="666"><cancel id="7"/></transactions>

This command will cancel transaction with id of 7 and under user “666”. Also it will withdr

Response is :

```
<?xml version="1.0" ?>
<results>
  <canceled id="7">
    <canceled shares="3" time="1554231246"/>
    <executed price="100" shares="2" time="1554231246"/>
  </canceled>
</results>
```

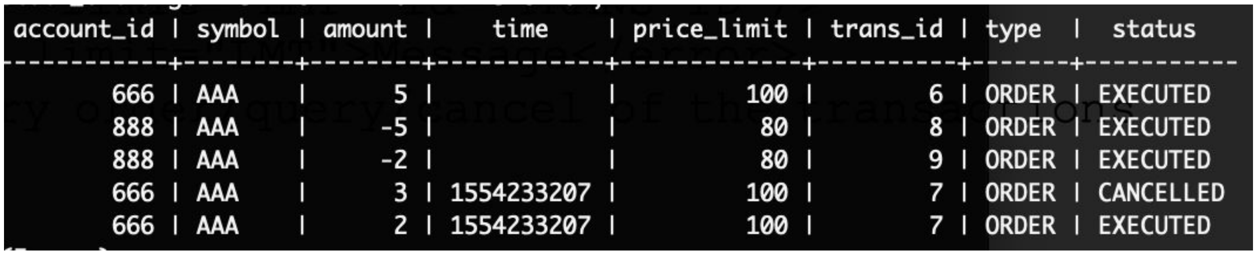
Below is the screenshot ACCOUNT table:



```
 id  | balance | aaa | bbb
-----+---------+-----+-----
 888 |    1700 |  93 | 200
 666 |     300 | 107 | 200
```

We can see that the balance of user “666” has restored.

Below is the screenshot TRANSACTION table:

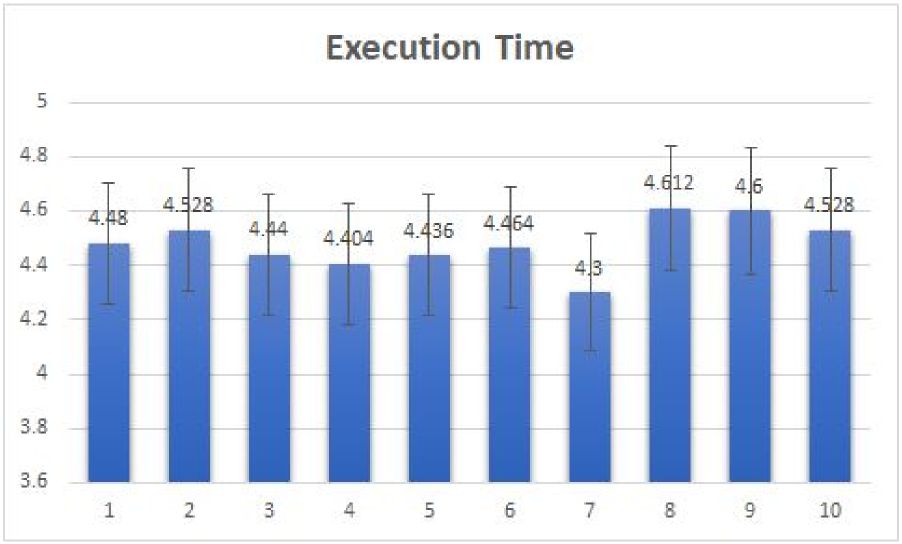


```
account_id | symbol | amount |    time    | price_limit | trans_id | type  |  status
-----------+--------+--------+------------+-------------+----------+-------+-----------
       666 | AAA    |      5 |            |         100 |        6 | ORDER | EXECUTED
       888 | AAA    |     -5 |            |          80 |        8 | ORDER | EXECUTED
       888 | AAA    |     -2 |            |          80 |        9 | ORDER | EXECUTED
       666 | AAA    |      3 | 1554233207 |         100 |        7 | ORDER | CANCELLED
       666 | AAA    |      2 | 1554233207 |         100 |        7 | ORDER | EXECUTED
```

We can see that the 3 shares of AAA has been canceled. The executed order is not affected.

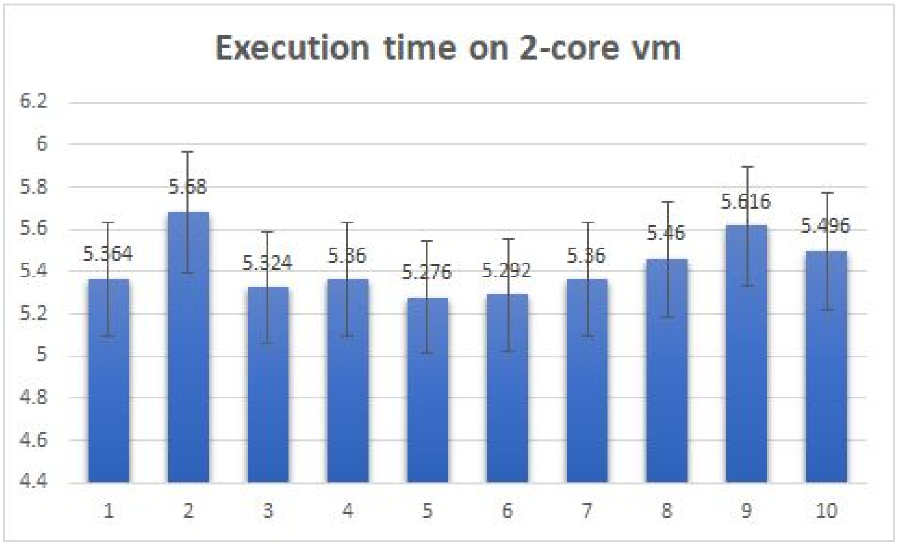
3. Scalability test

For scalability test, we first started the server program and run a script(start.sh) where several client programs are automatically run to imitate multi-client access to the system. In the script, 3 client programs sending out create and order command were executed 50 times respectively. The script was run 10 times and their execution time were recorded.



The above graph indicated the execution time of the 10 experiments. The error bar was for 95% confidence interval. It indicated that 95% of the interval has the true average execution time of the experiment in it. From this graph we can get the grasp of the average execution time of the concurrency experiment and it has relatively fast speed to handle concurrent access to the database and can respond in time when there are lots of client asking for information.

Note that the above test is carried out on the 4-core vm, and we also put the same experiment on the 2-core regular vm and tests its performance. The execution time was as follows:



Also the error bar here was for 95% confidence interval. We can see from this bar chart that the average execution time for the same experiment on a 2-core machine is relatively longer than that on a 4-core machine, which is expected since 2 cores cannot handle multi-thread as well as 4 cores. It can still finish its job in less than 2 times of the 4-core execution time,