CS5421: project5 report

Question 1

The question 1 ask us to calculate the sum of all the balance over the table. To achieve this, we add code into function sum_balance(sess) like the following:

```
return sess.execute("SELECT SUM(ACCOUNT.BALANCE) FROM ACCOUNT").scalar()
```

`.execute("SELECT SUM(ACCOUNT.BALANCE) FROM ACCOUNT")` is to execute the sum up query over the table. `.scalar()` is to get the query result.

Question 2

The question 2 ask us to swap the balance of two rows randomly. To achieve this, we define the exchange function like the following:

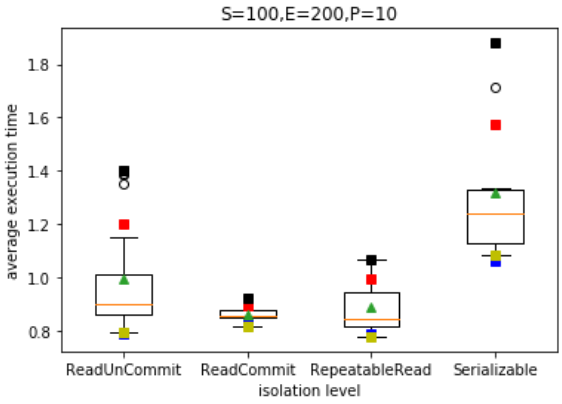
```
def exchange(sess):

    ## 1. Get the balance of two random ids

    A_1 = random.randint(1,100000)

    A_2 = random.randint(1,100000)

    V_1 = sess.execute("SELECT ACCOUNT.BALANCE FROM ACCOUNT WHERE ACCOUNT.ID
=:A_1",{"A_1":A_1}).scalar()

    V_2 = sess.execute("SELECT ACCOUNT.BALANCE FROM ACCOUNT WHERE ACCOUNT.ID
=:A_2",{"A_2":A_2}).scalar()

    ## 2. Swap the value

    sess.execute("UPDATE ACCOUNT SET BALANCE =:V_2 WHERE ACCOUNT.ID =:A_1",{"V_2":V_2,"A_1":A_1})

    sess.execute("UPDATE ACCOUNT SET BALANCE =:V_1 WHERE ACCOUNT.ID =:A_2",{"V_1":V_1,"A_2":A_2})

    sess.commit()

return 1
```

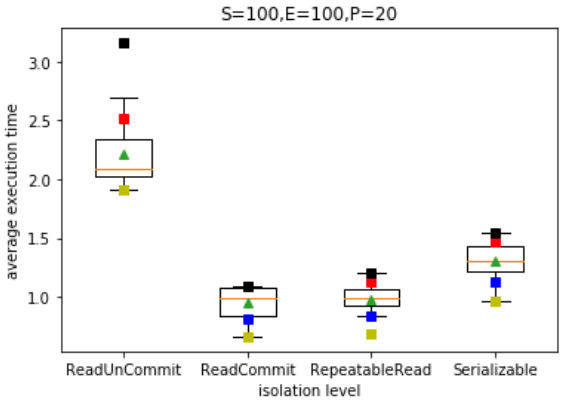
Question 3

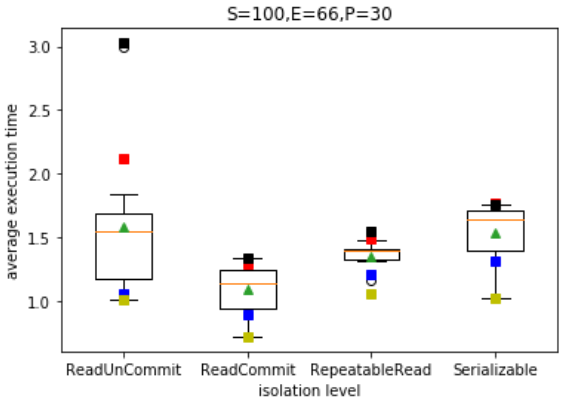
Simulation environment: Window 10, Postgre Database, CPU-i7 8550U

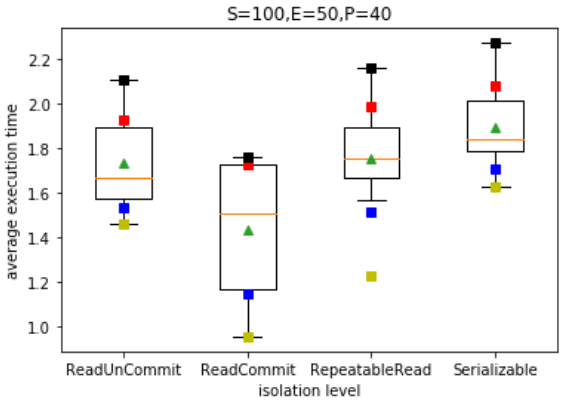
Question 3. (a)

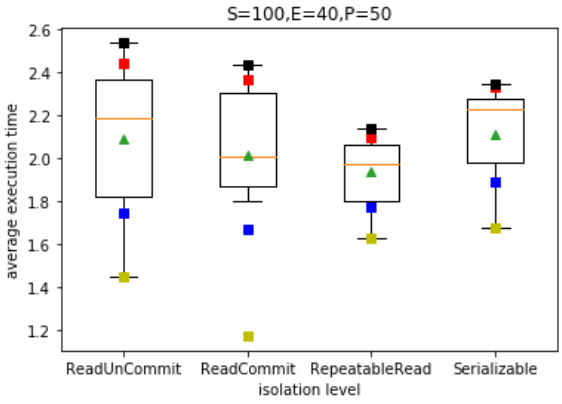
The box plot for average execution time as a function of the total number of subprocesses P for each of the 4 isolation levels is shown below.

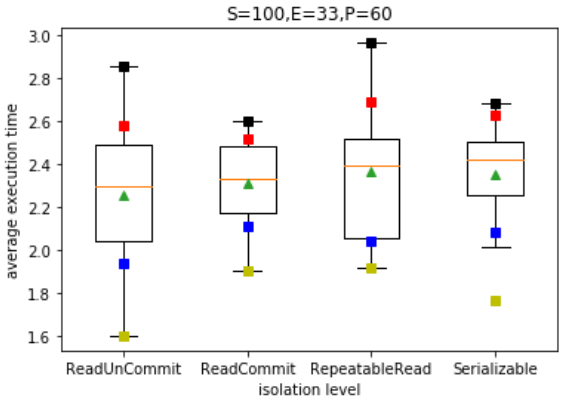

S=100,E=200,P=10
average execution time
1.8
1.6
1.4
1.2
1.0
0.8
ReadUnCommit
ReadCommit
RepeatableRead
Serializable
isolation level

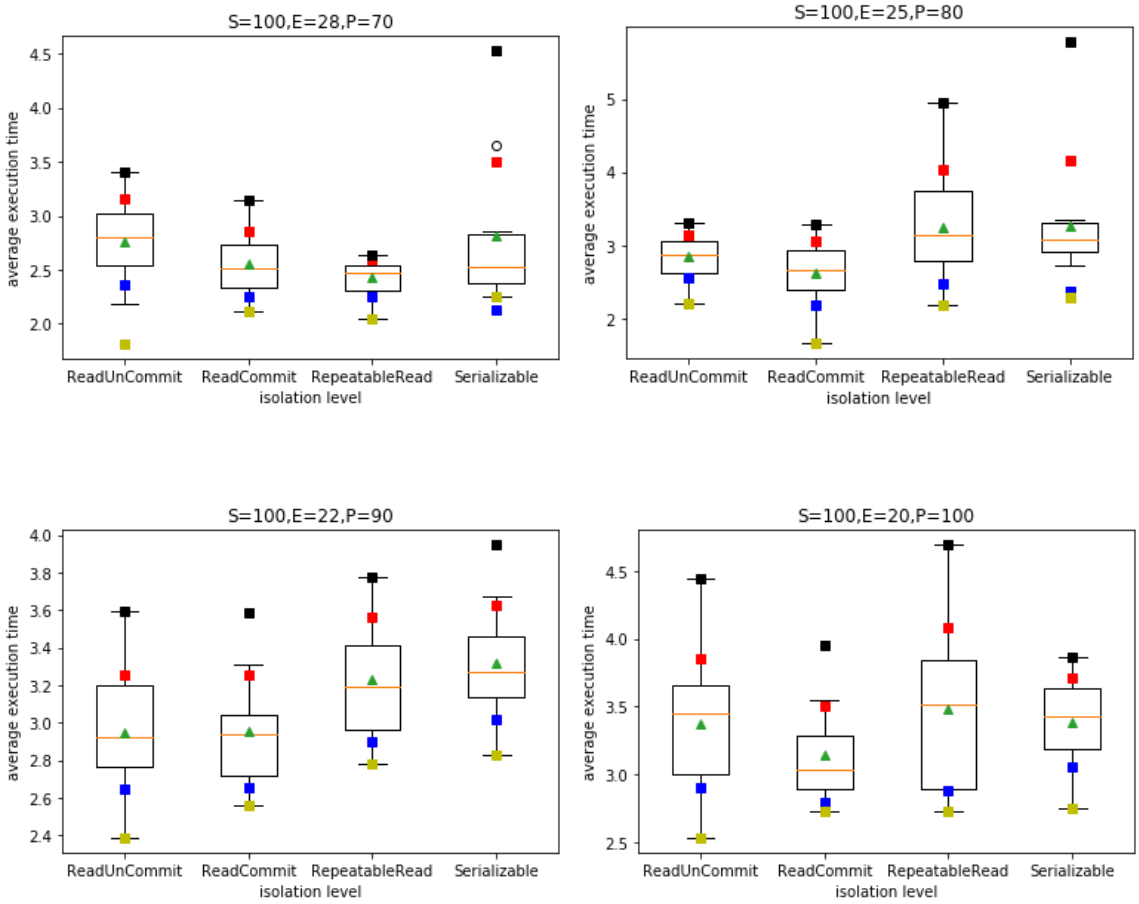

S=100,E=100,P=20
average execution time
3.0
2.5
2.0
1.5
1.0
ReadUnCommit
ReadCommit
RepeatableRead
Serializable
isolation level


S=100,E=66,P=30
average execution time
3.0
2.5
2.0
1.5
1.0
ReadUnCommit
ReadCommit
RepeatableRead
Serializable
isolation level


S=100,E=50,P=40
average execution time
2.2
2.0
1.8
1.6
1.4
1.2
1.0
ReadUnCommit
ReadCommit
RepeatableRead
Serializable
isolation level


S=100,E=40,P=50
average execution time
2.6
2.4
2.2
2.0
1.8
1.6
1.4
1.2
ReadUnCommit
ReadCommit
RepeatableRead
Serializable
isolation level


S=100,E=33,P=60
average execution time
3.0
2.8
2.6
2.4
2.2
2.0
1.8
1.6
ReadUnCommit
ReadCommit
RepeatableRead
Serializable
isolation level

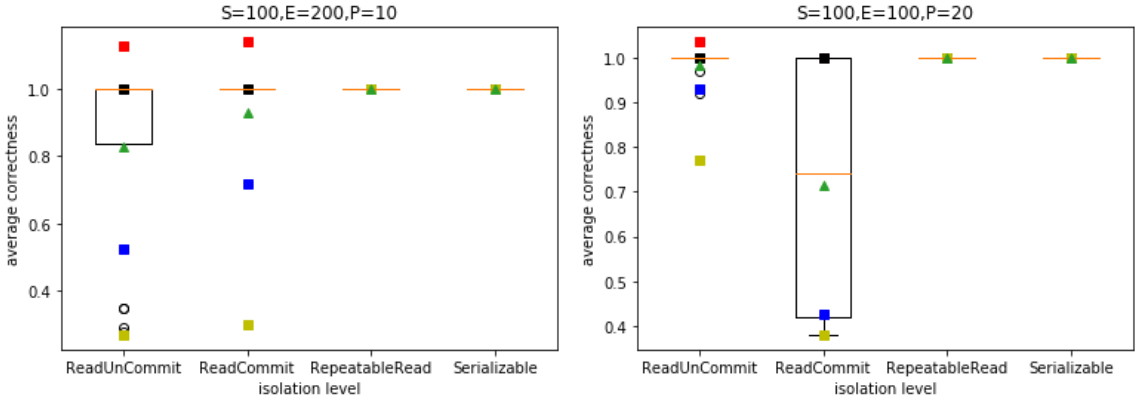


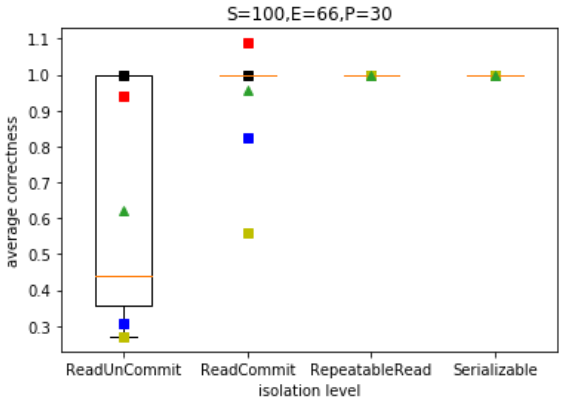
From above figures, we can see the following phenomenon:

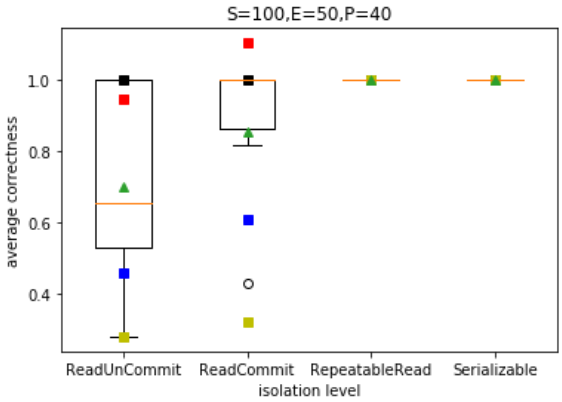
- The monotonicity between the execution time and the number of subprocesses is not obvious. When P=10, the execution time is the smallest. When P=70 and 80, the execution time seem slightly large than others. There may be several reasons for this phenomenon:
    - The available resources for execution are varying. Because there are other programs and process running in the OS, these tasks may affect the available resources for the query execution.
    - As the number of subprocesses increases, it will speed up the execution speed for the parallelization. However, on the other hand, as the number of subprocesses increases, the probability that the same cell in the database is updated by different subprocesses, which will trigger lock in the database and increase the execution time.
- The execution time of 'ReadCommit' is usually the smallest. The execution of 'ReadUnCommit' is quite similar to that of 'ReadCommit'. The execution time of 'RepeatableRead' and 'Serializable' is obviously large than these of the first two isolation level. Specifically, the execution time of 'Serializable' is usually the largest among them. There may be several reasons for this phenomenon:
    - The isolation level defines four levels of transaction isolation. The strict order is 'ReadUnCommit', 'ReadCommit', 'RepeatableRead', 'Serializable' [1]. Consequently, the execution time of 'Serializable' is the largest for the fact that it is the strictest.

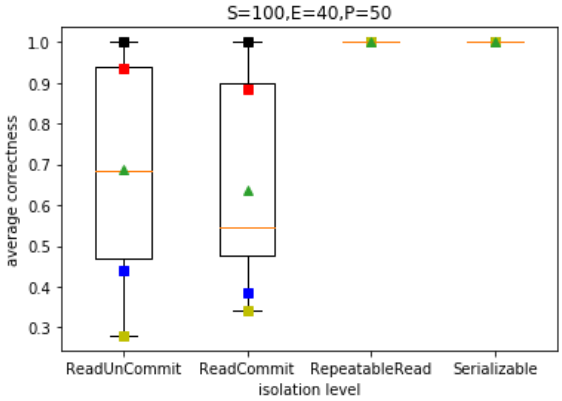
Question 3.(b)

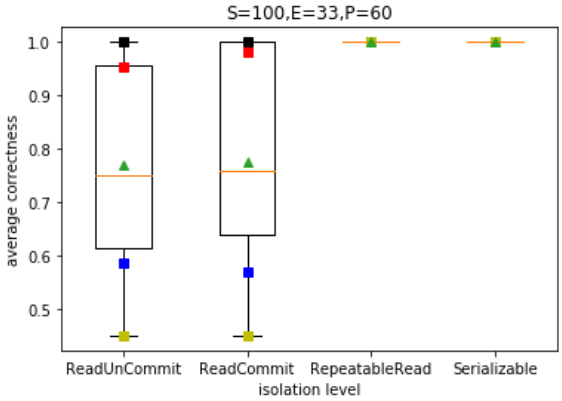
Box plot for average correctness as a function of the total number of subprocesses P for each of the 4 isolation levels is shown below.

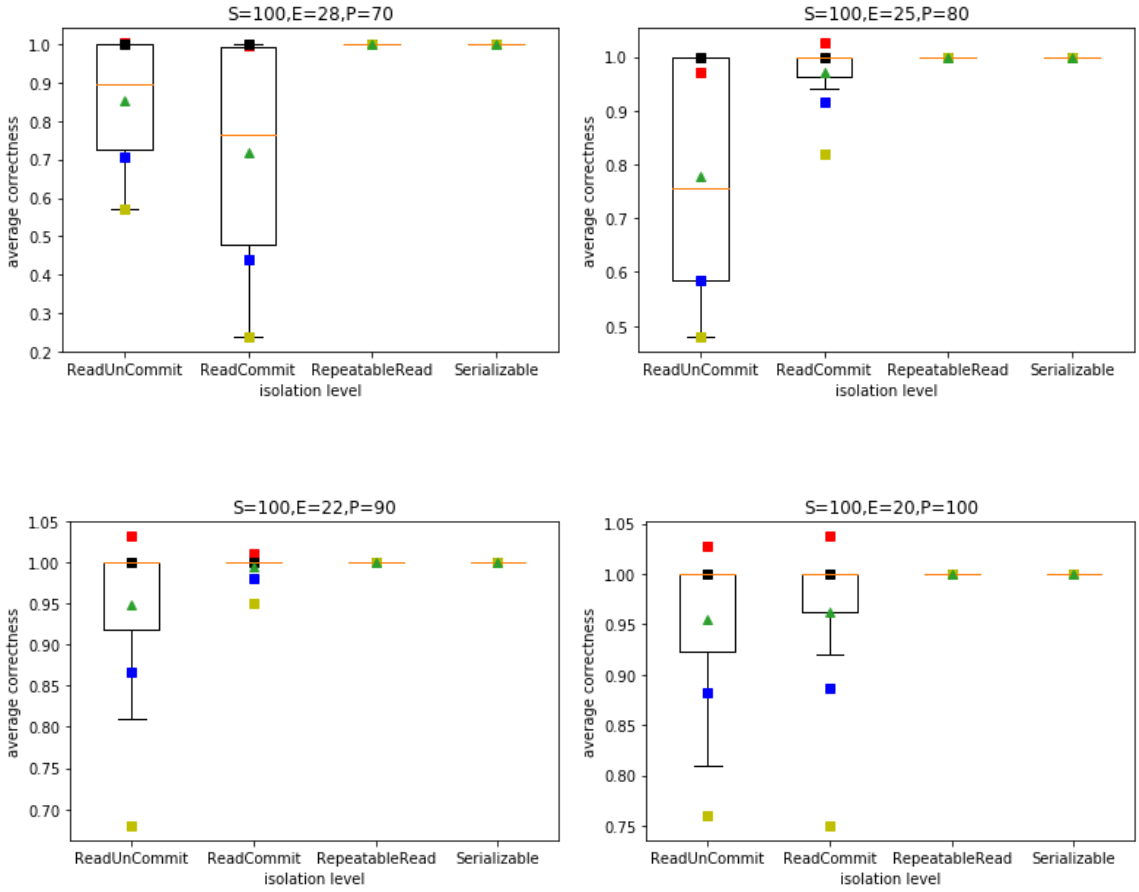












From above figures, we can see the following phenomenon:

- The correctness of ‘RepeatableRead’ and ‘Serializable’ is always 1. The correctness of ‘ReadUnCommit’ is smallest and the variance of it is largest. There may be several reasons for this phenomenon:
    - Because ‘Serializable’ is strictest among the four level. From [1], we can know that ‘Serializable’ can prevent Dirty Read, Nonrepeatable Read and Phantom Read. ‘RepeatableRead’ can prevent Dirty Read and Nonrepeatable Read. ‘ReadCommit’ can prevent Dirty Read. Thus, we can see that the error cases of ‘ReadUnCommit’ is largest and error cases of ‘Serializable’ is rare.

Reference

[1] https://www.postgresql.org/docs/9.1/transaction-iso.html