DBMS Project Deadline 6

Shobhit Pandey 2021287, Agamdeep Singh 2021306

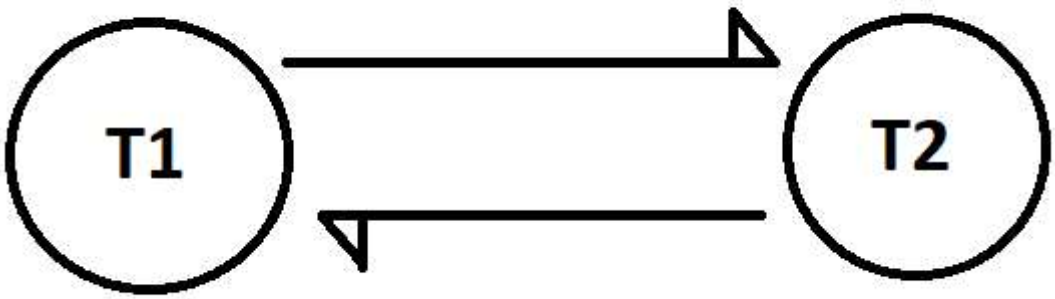
Cab Booking (Transaction 1)	**Wallet Updation (Transaction 2)**
Begin Transaction	
Give locations to choose the start of the trip.	
Select drivers who are in the chosen city.	
Subtract trip money from the wallet.	
If the wallet has sufficient funds the trips will start else cancel	
The trip ends and the customer reaches their destination	
Driver location is updated to the new city that they are in	
	Begin Transaction
	The customer adds money to their wallet.
	Check Balance

In the above schedule, if during the cab booking, the customer doesn't have sufficient funds in their wallet, the trips doesn't go through even though they added money into their wallet

Precedence graph for Schedule 1

An alternative to this schedule is that before changing the wallet for cab booking, the schedule checks for any incoming requests where money is being added to the wallet. If yes, then the booking transaction continues after the money is added.

Cab Booking (Transaction 1)	**Wallet Updation (Transaction 2)**
Begin transaction	Begin transaction
Give locations to choose the start of the trip	
Select drivers who are in the chosen city.	
	The customer adds money to their wallet.
	Check Balance
Subtract trip money from the wallet	
If the wallet has sufficient funds, the trips will Start the trip else cancel	
The trip ends, and the customer reaches their destination.	
Driver location is updated to the new city that they are in	



Precedence Graph for schedule 2

In the first schedule of transactions :

1. Read wallet
2. If sufficient funds then the trip continues else it is cancelled
3. Add money to the wallet

If we make the precedence graph for this schedule, then this schedule is conflict serialisable due to the absence of cycles.

For the second transactions

1. Read wallet
2. Update wallet value
3. If sufficient funds are available, then the trip continues, or else if cancelled.

In the precedence graph that we make for Transaction 1 and Transaction 2, there will be a cycle which means that this schedule is non-conflict serialisable.

In the illustration of transaction 1 below, the queries were working in DBMS, they were later incorporated into the backend code.

In the illustration below for transaction 2, we have used **locks** which let us update the wallet with priority before any further operations can be done on it by other transactions. This will help prevent any form of WR or RW concurrency errors and prevent data access issues in the database.

Transaction 1

```
set autocommit = 0;
start transaction;

select * from cities;

update customer
set Wallet = Wallet - 500
where C_Phone_number = 9811443772;

create table temp
select *
from drivers
where drivers.Car_number in (select number from car where category = 5 order by con desc)
and drivers.Cities_name = 'sextubercular'
order by rating desc
limit 1;

update drivers
set drivers.Revenue = drivers.Revenue + 500
where drivers.D_Phone_number in (select D_phone_number from temp);

insert into trips values(123456789,500,'start time','1:11:2',(select temp.Cities_name from temp),'destination',(select temp.D_Phone_number from temp));

update drivers
set drivers.Cities_name = 'destination'
where drivers.D_phone_number in (select D_phone_number from temp);

drop table temp;

commit;
```

Transaction 2

```
start transaction;
lock tables customer write;
update customer
set customer.wallet = customer.wallet + 500
where C_phone_number = 9811443772;
unlock table;
commit;
```

Non Conflicting Transactions

-
```
SET autocommit = 0; #switching autocommit off so that changes are reflected only after our updating
start transaction;
insert into cities value('noida');  #new location just dropped
commit;
```

-
```
SET autocommit = 0;
start transaction; #new driver just registered
insert into car values('UP 16 DD 6012',5,1);
insert into drivers values(9811443772,'Shobhit','shobhit21287@iiitd.ac.in',5,0,'noida','UP 16 DD 6012',0);
commit;
```

-
```
SET autocommit = 0;
start transaction;        #new customer just registered
insert into customer values('9811443772','shobhit21287@iiitd.ac.in','shobhit',4200,5,null,null,null,null);
insert into locations values('my home address','noida sec 41',0,'-1',9811443772,'shobhit21287@iiitd.ac.in'); #customer enters address as the first frequently travelled location
commit;
```

-
```
SET autocommit = 0;
start transaction;      #new admin and is being given a batch of drivers to manag
insert into admin values(11,'shobhit','shobhit21287@iiitc.ac.in');
update drivers set Admin_employee_ID = 11 where rating = 4;
commit;
```

The above transactions are all pairwise non-conflicting transactions. Since they all make changes on different tables or columns, they do not affect each other's functioning. [essentially six pairs of non-conflicting transactions]