Task 1:

Create table “account”

```
create table account (
        userid serial primary key,
        username char(256),
        credit int,
        currency char(256)
);

insert into account (username, credit, currency)
values ('Bill', 1000, 'RUB'), ('Ann', 1000, 'RUB'), ('Alan', 1000, 'RUB');
```

Commit the transactions

```
begin;
update account
set credit = credit - 500
where userid = 1;
update account
set credit = credit + 500
where userid = 3;
commit;

begin;
update account
set credit = credit - 700
where userid = 2;
update account
set credit = credit + 700
where userid = 1;
commit;

begin;
update account
set credit = credit - 100
where userid = 2;
update account
set credit = credit + 100
where userid = 3;
commit;
```

	userid [PK] integer	username character (256)	credit integer	currency character (256)
1	1	Bill	1200	RUB
2	2	Ann	200	RUB
3	3	Alan	1600	RUB

Return initial values

```
begin;
update account
set credit =  1000
where userid = 1;
update account
set credit =  1000
where userid = 2;
update account
set credit =  1000
where userid = 3;
commit;
```

	userid [PK] integer	username character (256)	credit integer	currency character (256)
1	1	Bill	1000	RUB
2	2	Ann	1000	RUB
3	3	Alan	1000	RUB

Rollback the transactions

```
begin;
update account
set credit = credit - 500
where userid = 1;
update account
set credit = credit + 500
where userid = 3;
rollback;

begin;
update account
set credit = credit - 700
where userid = 2;
update account
set credit = credit + 700
where userid = 1;
rollback;
```

```
begin;
update account
set credit = credit - 100
where userid = 2;
update account
set credit = credit + 100
where userid = 3;
rollback;
```

	userid [PK] integer	username character (256)	credit integer	currency character (256)
1	1	Bill	1000	RUB
2	2	Ann	1000	RUB
3	3	Alan	1000	RUB

Add new column: “bankname”

```
alter table account
add column bankname char(256);

update account
set bankname = 'sberbank'
where userid = 1 or userid = 3;

update account
set bankname = 'tinkoff'
where userid = 2;
```

Add new record: “fees” (external transactions' fees will be sent to this account)

```
insert into account(username, credit, currency)
values (fees, 0, 'RUB');
```

	userid [PK] integer	username character (256)	credit integer	currency character (256)	bankname character (256)
1	2	Ann	1000	RUB	tinkoff
2	1	Bill	1000	RUB	sberbank
3	3	Alan	1000	RUB	sberbank
4	4	fees	0	RUB	[null]

Do the same transactions, but with fees for external transactions

```
# python 3.8
import psycopg2
```

```
conn = psycopg2.connect(database='postgres', user='postgres',
password='1308249756', host='localhost', port='5432')
cur = conn.cursor()


def transaction(id1, id2, money):
   cur.execute('select bankname from account where userid = ' + str(id1) + ' or
userid = ' + str(id2) + ';')
   banks = cur.fetchall()

   if banks[0] != banks[1]:
       cur.execute('begin; update account set credit = credit + 30 where userid =
4; commit;')
       cur.execute('begin; update account set credit = credit - 30 where userid =
' + str(id1) + '; commit;')
   cur.execute('begin;')
   cur.execute('update account set credit = credit - ' + str(money) + ' where
userid = ' + str(id1) + ';')
   cur.execute('update account set credit = credit + ' + str(money) + ' where
userid = ' + str(id2) + ';')
   cur.execute('commit;')


transaction(1, 3, 500)
transaction(2, 1, 700)
transaction(2, 3, 100)

conn.close()
```

	userid [PK] integer	username character (256)	credit integer	currency character (256)	bankname character (256)
1	2	Ann	140	RUB	tinkoff
2	1	Bill	1200	RUB	sberbank
3	4	fees	60	RUB	[null]
4	3	Alan	1600	RUB	sberbank

Create table “ledger”

```
create table ledger (
        ledgerid serial,
        fromid int,
        toid int,
        fee int,
        amount int,
        date_time timestamp
)
```

Return initial values

```
begin;
update account
set credit =  1000
where userid = 1;
update account
set credit =  1000
where userid = 2;
update account
set credit =  1000
where userid = 3;
update account
set credit =  0
where userid = 4;
commit;
```

	userid [PK] integer	username character (256)	credit integer	currency character (256)	bankname character (256)
1	1	Bill	1000	RUB	sberbank
2	2	Ann	1000	RUB	tinkoff
3	3	Alan	1000	RUB	sberbank
4	4	fees	0	RUB	[null]

Do the same transactions, but with records in the new table

```
# python 3.8
import psycopg2
import time
import datetime

conn = psycopg2.connect(database='postgres', user='postgres',
password='1308249756', host='localhost', port='5432')
cur = conn.cursor()


def transaction(id1, id2, money):
    fee = 0
    cur.execute('select bankname from account where userid = ' + str(id1) + ' or
userid = ' + str(id2) + ';')
    banks = cur.fetchall()

    if banks[0] != banks[1]:
        cur.execute('begin; update account set credit = credit + 30 where userid =
4; commit;')
        cur.execute('begin; update account set credit = credit - 30 where userid =
' + str(id1) + '; commit;')
```

```
        fee = 30

    cur.execute('begin;')
    cur.execute('update account set credit = credit - ' + str(money) + ' where
userid = ' + str(id1) + ';')
    cur.execute('update account set credit = credit + ' + str(money) + ' where
userid = ' + str(id2) + ';')
    cur.execute('commit;')

    dt = datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d
%H:%M:%S')
    cur.execute('insert into ledger (fromid, toid, fee, amount, date_time) values
(' +
                str(id1) + ',' + str(id2) + ',' + str(fee) + ',' + str(money) +
",'" + str(dt) + "');")


transaction(1, 3, 500)
transaction(2, 1, 700)
transaction(2, 3, 100)

conn.close()
```

	ledgerid integer	fromid integer	toid integer	fee integer	amount integer	date_time timestamp without time zone
1	1	1	3	0	500	2022-04-22 21:11:47
2	2	2	1	30	700	2022-04-22 21:11:47
3	3	2	3	30	100	2022-04-22 21:11:47

	userid [PK] integer	username character (256)	credit integer	currency character (256)	bankname character (256)
1	2	Ann	140	RUB	tinkoff
2	1	Bill	1200	RUB	sberbank
3	4	fees	60	RUB	[null]
4	3	Alan	1600	RUB	sberbank

Task 2:

Create table “account”

```
create table account (
        username text,
        fullname text,
        balance int,
        Group_id int
);
```

insert into account (username, fullname, balance, Group_id)
values ('jones', 'Alice Jones', 82, 1), ('bitdiddl', 'Ben Bitdiddle', 65, 1), ('mike', 'Michael Dole', 73, 2), ('alyssa', 'Alyssa P. Hacker', 79, 3), ('bbrown', 'Bob Brown', 100, 3);

	username text	fullname text	balance integer	group_id integer
1	jones	Alice Jones	82	1
2	bitdiddl	Ben Bitdiddle	65	1
3	mike	Michael Dole	73	2
4	alyssa	Alyssa P. Hacker	79	3
5	bbrown	Bob Brown	100	3

Read committed:

T1:
```
begin;
set transaction isolation level read committed;
select * from account;
```

	username	fullname	balance	group_id
1	jones	Alice Jones	82	1
2	bitdiddl	Ben Bitdiddle	65	1
3	mike	Michael Dole	73	2
4	alyssa	Alyssa P. Hacker	79	3
5	bbrown	Bob Brown	100	3

T2:
```
begin;
set transaction isolation level read committed;
update account
set username = 'ajones'
where fullname = 'Alice Jones';
```

T1:
```
select * from account;
```

	username	fullname	balance	group_id
1	jones	Alice Jones	82	1
2	bitdiddl	Ben Bitdiddle	65	1
3	mike	Michael Dole	73	2
4	alyssa	Alyssa P. Hacker	79	3
5	bbrown	Bob Brown	100	3

T2:
```
select * from account;
```

	username	fullname	balance	group_id
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	ajones	Alice Jones	82	1

Result: terminals show different information information, because terminals cannot see not commited changes that were made by other transactions.

T2:
```
commit;
select * from account;
```

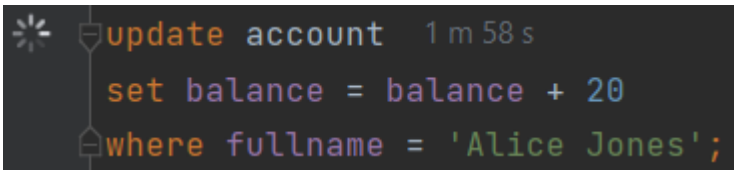
	username	fullname	balance	group_id
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	ajones	Alice Jones	82	1

T1:
```
select * from account;
```

	username	fullname	balance	group_id
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	ajones	Alice Jones	82	1

Result: both tables show the same data.

T2:
```
begin;
set transaction isolation level read committed;
```
T1:
```
update account
set balance = balance + 10
where fullname = 'Alice Jones';
```
T2:
```
update account
set balance = balance + 20
where fullname = 'Alice Jones';
```



Result: the second terminal cannot do the query until the first transaction is finished. Therefore, its data will not be changed.

	username	fullname	balance	group_id
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	ajones	Alice Jones	82	1

T1:
```
commit;
```
T2:
```
rollback;
```
Result: after finishing the transactions the table looks like that:

	username text	fullname text	balance integer	group_id integer
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	ajones	Alice Jones	92	1

Change Alice's username to 'jones' back:

	username	fullname	balance	group_id
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	jones	Alice Jones	92	1

Repeatable read:

T1:
```
begin;
set transaction isolation level repeatable read;
select * from account;
```

	username	fullname	balance	group_id
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	jones	Alice Jones	92	1

T2:
```
begin;
set transaction isolation level repeatable read;
update account
set username = 'ajones'
where fullname = 'Alice Jones';
```

T1:
```
select * from account;
```

	username	fullname	balance	group_id
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	jones	Alice Jones	92	1

T2:
```
select * from account;
```

	username	fullname	balance	group_id
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	ajones	Alice Jones	92	1

Result: terminals show different information information, because terminals cannot see not commited changes that were made by other transactions.

T2:
```
commit;
select * from account;
```

	username	fullname	balance	group_id
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	ajones	Alice Jones	82	1

T1:
```
select * from account;
```

	username	fullname	balance	group_id
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	jones	Alice Jones	92	1

Result: terminals show different data again, because using repeatable read a transaction can only see the data that was committed before it starts.

T2:
```
begin;
set transaction isolation level repeatable read;
```
T1:
```
update account
set balance = balance + 10
where fullname = 'Alice Jones';
```
Error: could not serialize access due to read/write dependencies among transactions.
Explanation (from the PostgreSQL documentation): if the first updater commits (and actually updated or deleted the row, not just locked it) then the repeatable read transaction will be rolled back with the message
```
ERROR:  could not serialize access due to concurrent update
```
because a repeatable read transaction cannot modify or lock rows changed by other transactions after the repeatable read transaction began.

T1:
```
select * from account;
```

	username	fullname	balance	group_id
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	jones	Alice Jones	92	1

T2:
```
update account
set balance = balance + 20
where fullname = 'Alice Jones';
select * from account;
```

	username	fullname	balance	group_id
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	ajones	Alice Jones	112	1

Result: terminals show different data, because the data could not be changed during the first transaction.

	username	fullname	balance	group_id
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	ajones	Alice Jones	82	1

T1:
```
commit;
```
T2:
```
rollback;
```
Result: after finishing the transactions the table looks like that:

	username text	fullname text	balance integer	group_id integer
1	bitdiddl	Ben Bitdiddle	65	1
2	mike	Michael Dole	73	2
3	alyssa	Alyssa P. Hacker	79	3
4	bbrown	Bob Brown	100	3
5	ajones	Alice Jones	92	1

Task 2 - part 2:
Read committed:
T1:
```
begin;
set transaction isolation level read committed;
```
T2:
```
begin;
set transaction isolation level read committed;
```
T1:
```
select * from account where group_id = 2;
```

	username	fullname	balance	group_id
1	mike	Michael Dole	73	2

T2:
```
update account
set group_id = 2
where fullname = 'Bob Brown';
```
T1:
```
select * from account where group_id = 2;
```

	username	fullname	balance	group_id
1	mike	Michael Dole	73	2

Explanation: Bob is not in the group 2, because terminals cannot see not commited changes that were made by other transactions.
T1:
```
update account
set balance = balance + 15
where group_id = 2;
```
T1:
```
commit;
```
T2:
```
commit;
```
Result: after finishing the transactions the table looks like that (eventually, Bob is in the group 2, but he stays with the same balance):

	username text	fullname text	balance integer	group_id integer
1	bitdiddl	Ben Bitdiddle	65	1
2	alyssa	Alyssa P. Hacker	79	3
3	ajones	Alice Jones	92	1
4	bbrown	Bob Brown	100	2
5	mike	Michael Dole	88	2

Return Bob to the group 3:

	username text	fullname text	balance integer	group_id integer
1	bitdiddl	Ben Bitdiddle	65	1
2	alyssa	Alyssa P. Hacker	79	3
3	ajones	Alice Jones	92	1
4	mike	Michael Dole	88	2
5	bbrown	Bob Brown	100	3

Repeatable read:

T1:
```
begin;
set transaction isolation level read committed;
```
T2:
```
begin;
set transaction isolation level read committed;
```
T1:
```
select * from account where group_id = 2;
```

	username	fullname	balance	group_id
1	mike	Michael Dole	88	2

T2:
```
update account
set group_id = 2
where fullname = 'Bob Brown';
```
T1:
```
select * from account where group_id = 2;
```

	username	fullname	balance	group_id
1	mike	Michael Dole	88	2

Result: Bob is not in the group 2, because using repeatable read a transaction can only see the data that was committed before it starts.

T1:
```
update account
set balance = balance + 15
where group_id = 2;
```
T1:
```
commit;
```
T2:
```
commit;
```
Result: after finishing the transactions the table looks like that (eventually, Bob is in the group 2, but he stays with the same balance):

	username text	fullname text	balance integer	group_id integer
1	bitdiddl	Ben Bitdiddle	65	1
2	alyssa	Alyssa P. Hacker	79	3
3	ajones	Alice Jones	92	1
4	bbrown	Bob Brown	100	2
5	mike	Michael Dole	103	2