Screencast 6-3

Using database transactions in python programs

Transactions

A transaction in a database is a sequence of operations that need to be executed together but not separately. Such sequence is treated as an indivisible action.

Changes which are done in a transaction should be confirmed or cancelled at the end of it. There are two keywords to do it – COMMIT to save changes and ROLLBACK to undo the corrections.

Psycopg2 library can work in two modes – with and without auto commit. If auto commit is on, then every single manipulation is saved permanently after execution. Library starts an implicit transaction in other cases and rolls back if there was no commit.

A connection object has two methods – commit() and rollback().

The library also allows to work with multiple transactions. In this example only one transaction is used.

Task

Write a python function *new_reservation(car_id, res_start_date, res_end_date, driver_id, pick_at)* to create a new reservation and return its identifier.

This function should solve the following tasks:

1) check if there are no overlapping reservations.
2) book a car (write a record into res table)
3) increase car's reservation counter

If something breaks inside this sequence, then all changes must be cancelled.

Database: cars2.
Identity option of reservation identifier should be activated. The following code adds this setting:
alter table res alter column bid add generated always as identity (start 100)

Let us consider a function that checks overlapping reservations and writes a new record:

```
def new_reservation(car, start:date, finish:date, driver, pickAt)->int:
    queryResNr = """select count(*) as overlapping from res
        where finish between %s and %s
              or start between %s and %s
              or (start <= %s and finish >= %s)"""
    queryNewRes = """insert into res(cid, start, finish, pickat, days, did)
        values (%s, %s, %s, %s, %s, %s)
        returning bid;"""
    queryNumInc = """update car
        set res_number = res_number + 1
```

```
        where cid = %s;"""
    retValue = 0
    con = psycopg2.connect(database="cars2", user="postgres", password="sql",
host="localhost")
    cur = con.cursor()
    try:
        cur.execute(queryResNr, (start, finish, start, finish, start, finish))
        oln = cur.fetchone()[0]
        if oln > 0:
            retValue = -1
        else:
            cur.execute(queryNewRes, (car, start, finish, pickAt, (finish-
start).days, driver))
            resId = cur.fetchone()[0]
            cur.execute(queryNumInc, (car,))
            cur.close()
            retValue = resId
    except (Exception) as err:
        retValue = -1
    con.close()
    return retValue
```

Call this function:

```
res = new_reservation(6, date(2018, 3, 26), date(2018, 3, 28), 2, 'airport')
print(res)
```

This call will not change anything, because auto commit is off by default. To make this function work we need to call commit() function of the connection object.

The following code contains correct fragment of new_reservation:

```
    try:
        cur.execute(queryResNr, (start, finish, start, finish, start, finish))
        oln = cur.fetchone()[0]
        if oln > 0:
            retValue = -1
        else:
            cur.execute(queryNewRes, (car, start, finish, pickAt, (finish-
start).days, driver))
            resId = cur.fetchone()[0]
            cur.execute(queryNumInc, (car,))
            con.commit()
            cur.close()
            retValue = resId
    except (Exception) as err:
        con.rollback()
        retValue = -1
    con.close()
    return retValue
```

Try to call new_reservation with non-existent identifiers to check if transaction is used.

Additional information on transactions

You can write transactions without creating applications like we did in this example. An equivalent in Postgres written in PL/PGSQL programming language is shown below. Though procedures are not included in this course, you can notice that this script and python function have much in common:

```
do $$
declare
v_carid int := 6;
v_start date := '2018-03-26';
v_finish date := '2018-03-28';
v_pickat varchar(15) := 'airport';
v_driver int:=2;

v_has_ol int:=0;
v_retval int:= -1;
begin
	select count(*) as overlapping into v_has_ol from res
		where finish between v_start and v_finish
			or start between v_start and v_finish
			or (start <= v_start and finish >= v_finish);

	begin
		if v_has_ol <> 0 then
			v_retval:=-1;
		else
			insert into res(cid, start, finish, pickat, days, did)
				values (v_carid, v_start, v_finish, v_pickat, v_finish-v_start,
v_driver)
				returning bid into v_retval;

			update car
				set res_number = res_number + 1
				where cid = v_carid;
		end if;
	exception when others then
		raise notice 'errors occured';
		v_retval:= -1;
	end;
	if v_retval = -1 then rollback;
		else commit;
		end if;
```

```
        raise notice 'result: %', v_retval;
end $$ language plpgsql;
```

Databases provide instruments to manage transactions and to control concurrent access to the same data. Important setting is transaction isolation level which defines what data should be accessible to concurrent transactions.