ADBMS Experiment 1

Ojas Patil

2019130048

TE COMPS

Batch: B

Aim: To demonstrate the concept of deadlock in distributed system.

Procedure:

Step 1:

First dropped the tables prereq, time_slot, advisor, takes, teaches, section, instructor, course, classroom, student, department if there are any. For this, I used the commands given in ddl_drop.docx which are as follows.

```
drop table prereq;
drop table time_slot;
drop table advisor;
drop table takes;
drop table teaches;
drop table section;
drop table instructor;
drop table course;
drop table classroom;
drop table student;
drop table department;
```

Step 2:

After dropping the tables, I created new tables with same names using commands given in ddl_sql.docx which are as follows:

```
create table classroom
        (building              varchar(15),
         room_number           varchar(7),
         capacity              numeric(4,0),
         primary key (building, room_number)
        );

create table department
        (dept_name             varchar(20),
         building              varchar(15),
         budget                 numeric(12,2) check (budget > 0),
         primary key (dept_name)
        );

create table course
        (course_id             varchar(8),
         title                 varchar(50),
         dept_name             varchar(20),
         credits               numeric(2,0) check (credits > 0),
         primary key (course_id),
         foreign key (dept_name) references department
                on delete set null
        );

create table instructor
        (ID                     varchar(5),
         name                   varchar(20) not null,
         dept_name              varchar(20),
         salary                 numeric(8,2) check (salary > 29000),
         primary key (ID),
         foreign key (dept_name) references department
                on delete set null
        );

create table section
        (course_id              varchar(8),
         sec_id                 varchar(8),
         semester               varchar(6)
                check (semester in ('Fall', 'Winter', 'Spring', 'Summer')),
         year                   numeric(4,0) check (year > 1701 and year <
2100),
         building               varchar(15),
         room_number            varchar(7),
         time_slot_id           varchar(4),
         primary key (course_id, sec_id, semester, year),
         foreign key (course_id) references course
                on delete cascade,
         foreign key (building, room_number) references classroom
                on delete set null
        );

create table teaches
        (ID                     varchar(5),
         course_id              varchar(8),
         sec_id                 varchar(8),
         semester               varchar(6),
         year                   numeric(4,0),
         primary key (ID, course_id, sec_id, semester, year),
         foreign key (course_id,sec_id, semester, year) references section
                on delete cascade,
         foreign key (ID) references instructor
                on delete cascade
        );

create table student
        (ID                     varchar(5),
         name                   varchar(20) not null,
         dept_name              varchar(20),
         tot_cred               numeric(3,0) check (tot_cred >= 0),
         primary key (ID),
         foreign key (dept_name) references department
                on delete set null
        );

create table takes
        (ID                     varchar(5),
         course_id              varchar(8),
         sec_id                 varchar(8),
         semester               varchar(6),
         year                   numeric(4,0),
         grade                  varchar(2),
         primary key (ID, course_id, sec_id, semester, year),
         foreign key (course_id,sec_id, semester, year) references section
                on delete cascade,
         foreign key (ID) references student
                on delete cascade
        );

create table advisor
        (s_ID                  varchar(5),
         i_ID                  varchar(5),
         primary key (s_ID),
         foreign key (i_ID) references instructor (ID)
                on delete set null,
         foreign key (s_ID) references student (ID)
                on delete cascade
        );

create table time_slot
        (time_slot_id          varchar(4),
         day                   varchar(1),
         start_hr              numeric(2) check (start_hr >= 0 and start_hr
< 24),
         start_min             numeric(2) check (start_min >= 0 and
start_min < 60),
         end_hr                numeric(2) check (end_hr >= 0 and end_hr <
24),
         end_min               numeric(2) check (end_min >= 0 and end_min <
60),
         primary key (time_slot_id, day, start_hr, start_min)
        );

create table prereq
        (course_id             varchar(8),
         prereq_id             varchar(8),
         primary key (course_id, prereq_id),
         foreign key (course_id) references course
                on delete cascade,
         foreign key (prereq_id) references course
        );
```

<u>Step 3</u>:

Inserting data into tables created from insert.docx

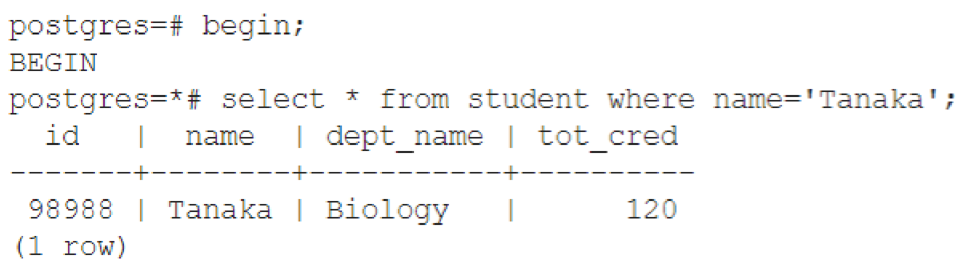
```
delete from prereq;
delete from time_slot;
delete from advisor;
delete from takes;
delete from student;
delete from teaches;
delete from section;
delete from instructor;
delete from course;
delete from department;
delete from classroom;
insert into classroom values ('Packard', '101', '500');
insert into classroom values ('Painter', '514', '10');
insert into classroom values ('Taylor', '3128', '70');
insert into classroom values ('Watson', '100', '30');
insert into classroom values ('Watson', '120', '50');
insert into department values ('Biology', 'Watson', '90000');
insert into department values ('Comp. Sci.', 'Taylor', '100000');
insert into department values ('Elec. Eng.', 'Taylor', '85000');
insert into department values ('Finance', 'Painter', '120000');
insert into department values ('History', 'Painter', '50000');
insert into department values ('Music', 'Packard', '80000');
insert into department values ('Physics', 'Watson', '70000');
insert into course values ('BIO-101', 'Intro. to Biology', 'Biology', '4');
insert into course values ('BIO-301', 'Genetics', 'Biology', '4');
insert into course values ('BIO-399', 'Computational Biology', 'Biology',
'3');
insert into course values ('CS-101', 'Intro. to Computer Science', 'Comp.
Sci.', '4');
insert into course values ('CS-190', 'Game Design', 'Comp. Sci.', '4');
insert into course values ('CS-315', 'Robotics', 'Comp. Sci.', '3');
insert into course values ('CS-319', 'Image Processing', 'Comp. Sci.',
'3');
insert into course values ('CS-347', 'Database System Concepts', 'Comp.
Sci.', '3');
insert into course values ('EE-181', 'Intro. to Digital Systems', 'Elec.
Eng.', '3');
insert into course values ('FIN-201', 'Investment Banking', 'Finance',
'3');
insert into course values ('HIS-351', 'World History', 'History', '3');
insert into course values ('MU-199', 'Music Video Production', 'Music',
'3');
insert into course values ('PHY-101', 'Physical Principles', 'Physics',
'4');
insert into instructor values ('10101', 'Srinivasan', 'Comp. Sci.',
'65000');
insert into instructor values ('12121', 'Wu', 'Finance', '90000');
insert into instructor values ('15151', 'Mozart', 'Music', '40000');
insert into instructor values ('22222', 'Einstein', 'Physics', '95000');
insert into instructor values ('32343', 'El Said', 'History', '60000');
insert into instructor values ('33456', 'Gold', 'Physics', '87000');
insert into instructor values ('45565', 'Katz', 'Comp. Sci.', '75000');
insert into instructor values ('58583', 'Califieri', 'History', '62000');
insert into instructor values ('76543', 'Singh', 'Finance', '80000');
insert into instructor values ('76766', 'Crick', 'Biology', '72000');
insert into instructor values ('83821', 'Brandt', 'Comp. Sci.', '92000');
insert into instructor values ('98345', 'Kim', 'Elec. Eng.', '80000');
insert into section values ('BIO-101', '1', 'Summer', '2009', 'Painter',
'514', 'B');
insert into section values ('BIO-301', '1', 'Summer', '2010', 'Painter',
'514', 'A');
insert into section values ('CS-101', '1', 'Fall', '2009', 'Packard',
'101', 'H');
insert into section values ('CS-101', '1', 'Spring', '2010', 'Packard',
'101', 'F');
insert into section values ('CS-190', '1', 'Spring', '2009', 'Taylor',
'3128', 'E');
insert into section values ('CS-190', '2', 'Spring', '2009', 'Taylor',
'3128', 'A');
insert into section values ('CS-315', '1', 'Spring', '2010', 'Watson',
'120', 'D');
insert into section values ('CS-319', '1', 'Spring', '2010', 'Watson',
'100', 'B');
insert into section values ('CS-319', '2', 'Spring', '2010', 'Taylor',
'3128', 'C');
insert into section values ('CS-347', '1', 'Fall', '2009', 'Taylor',
'3128', 'A');
```

```
insert into section values ('EE-181', '1', 'Spring', '2009', 'Taylor',
'3128', 'C');
insert into section values ('FIN-201', '1', 'Spring', '2010', 'Packard',
'101', 'B');
insert into section values ('HIS-351', '1', 'Spring', '2010', 'Painter',
'514', 'C');
insert into section values ('MU-199', '1', 'Spring', '2010', 'Packard',
'101', 'D');
insert into section values ('PHY-101', '1', 'Fall', '2009', 'Watson',
'100', 'A');
insert into teaches values ('10101', 'CS-101', '1', 'Fall', '2009');
insert into teaches values ('10101', 'CS-315', '1', 'Spring', '2010');
insert into teaches values ('10101', 'CS-347', '1', 'Fall', '2009');
insert into teaches values ('12121', 'FIN-201', '1', 'Spring', '2010');
insert into teaches values ('15151', 'MU-199', '1', 'Spring', '2010');
insert into teaches values ('22222', 'PHY-101', '1', 'Fall', '2009');
insert into teaches values ('32343', 'HIS-351', '1', 'Spring', '2010');
insert into teaches values ('45565', 'CS-101', '1', 'Spring', '2010');
insert into teaches values ('45565', 'CS-319', '1', 'Spring', '2010');
insert into teaches values ('76766', 'BIO-101', '1', 'Summer', '2009');
insert into teaches values ('76766', 'BIO-301', '1', 'Summer', '2010');
insert into teaches values ('83821', 'CS-190', '1', 'Spring', '2009');
insert into teaches values ('83821', 'CS-190', '2', 'Spring', '2009');
insert into teaches values ('83821', 'CS-319', '2', 'Spring', '2010');
insert into teaches values ('98345', 'EE-181', '1', 'Spring', '2009');
insert into student values ('00128', 'Zhang', 'Comp. Sci.', '102');
insert into student values ('12345', 'Shankar', 'Comp. Sci.', '32');
insert into student values ('19991', 'Brandt', 'History', '80');
insert into student values ('23121', 'Chavez', 'Finance', '110');
insert into student values ('44553', 'Peltier', 'Physics', '56');
insert into student values ('45678', 'Levy', 'Physics', '46');
insert into student values ('54321', 'Williams', 'Comp. Sci.', '54');
insert into student values ('55739', 'Sanchez', 'Music', '38');
insert into student values ('70557', 'Snow', 'Physics', '0');
insert into student values ('76543', 'Brown', 'Comp. Sci.', '58');
insert into student values ('76653', 'Aoi', 'Elec. Eng.', '60');
insert into student values ('98765', 'Bourikas', 'Elec. Eng.', '98');
insert into student values ('98988', 'Tanaka', 'Biology', '120');
insert into takes values ('00128', 'CS-101', '1', 'Fall', '2009', 'A');
insert into takes values ('00128', 'CS-347', '1', 'Fall', '2009', 'A-');
insert into takes values ('12345', 'CS-101', '1', 'Fall', '2009', 'C');
insert into takes values ('12345', 'CS-190', '2', 'Spring', '2009', 'A');
insert into takes values ('12345', 'CS-315', '1', 'Spring', '2010', 'A');
insert into takes values ('12345', 'CS-347', '1', 'Fall', '2009', 'A');
insert into takes values ('19991', 'HIS-351', '1', 'Spring', '2010', 'B');
insert into takes values ('23121', 'FIN-201', '1', 'Spring', '2010', 'C+');
insert into takes values ('44553', 'PHY-101', '1', 'Fall', '2009', 'B-');
insert into takes values ('45678', 'CS-101', '1', 'Fall', '2009', 'F');
insert into takes values ('45678', 'CS-101', '1', 'Spring', '2010', 'B+');
insert into takes values ('45678', 'CS-319', '1', 'Spring', '2010', 'B');
insert into takes values ('54321', 'CS-101', '1', 'Fall', '2009', 'A-');
insert into takes values ('54321', 'CS-190', '2', 'Spring', '2009', 'B+');
insert into takes values ('55739', 'MU-199', '1', 'Spring', '2010', 'A-');
insert into takes values ('76543', 'CS-101', '1', 'Fall', '2009', 'A');
insert into takes values ('76543', 'CS-319', '2', 'Spring', '2010', 'A');
insert into takes values ('76653', 'EE-181', '1', 'Spring', '2009', 'C');
insert into takes values ('98765', 'CS-101', '1', 'Fall', '2009', 'C-');
insert into takes values ('98765', 'CS-315', '1', 'Spring', '2010', 'B');
insert into takes values ('98988', 'BIO-101', '1', 'Summer', '2009', 'A');
insert into takes values ('98988', 'BIO-301', '1', 'Summer', '2010', null);
insert into advisor values ('00128', '45565');
insert into advisor values ('12345', '10101');
insert into advisor values ('23121', '76543');
insert into advisor values ('44553', '22222');
insert into advisor values ('45678', '22222');
insert into advisor values ('76543', '45565');
insert into advisor values ('76653', '98345');
insert into advisor values ('98765', '98345');
insert into advisor values ('98988', '76766');
insert into time_slot values ('A', 'M', '8', '0', '8', '50');
insert into time_slot values ('A', 'W', '8', '0', '8', '50');
insert into time_slot values ('A', 'F', '8', '0', '8', '50');
insert into time_slot values ('B', 'M', '9', '0', '9', '50');
insert into time_slot values ('B', 'W', '9', '0', '9', '50');
insert into time_slot values ('B', 'F', '9', '0', '9', '50');
insert into time_slot values ('C', 'M', '11', '0', '11', '50');
insert into time_slot values ('C', 'W', '11', '0', '11', '50');
insert into time_slot values ('C', 'F', '11', '0', '11', '50');
insert into time_slot values ('D', 'M', '13', '0', '13', '50');
insert into time_slot values ('D', 'W', '13', '0', '13', '50');
insert into time_slot values ('D', 'F', '13', '0', '13', '50');
insert into time_slot values ('E', 'T', '10', '30', '11', '45 ');
insert into time_slot values ('E', 'R', '10', '30', '11', '45 ');
insert into time_slot values ('F', 'T', '14', '30', '15', '45 ');
insert into time_slot values ('F', 'R', '14', '30', '15', '45 ');
insert into time_slot values ('G', 'M', '16', '0', '16', '50');
insert into time_slot values ('G', 'W', '16', '0', '16', '50');
insert into time_slot values ('G', 'F', '16', '0', '16', '50');
insert into time_slot values ('H', 'W', '10', '0', '12', '30');
insert into prereq values ('BIO-301', 'BIO-101');
insert into prereq values ('BIO-399', 'BIO-101');
insert into prereq values ('CS-190', 'CS-101');
insert into prereq values ('CS-315', 'CS-101');
insert into prereq values ('CS-319', 'CS-101');
insert into prereq values ('CS-347', 'CS-101');
insert into prereq values ('EE-181', 'PHY-101');
```

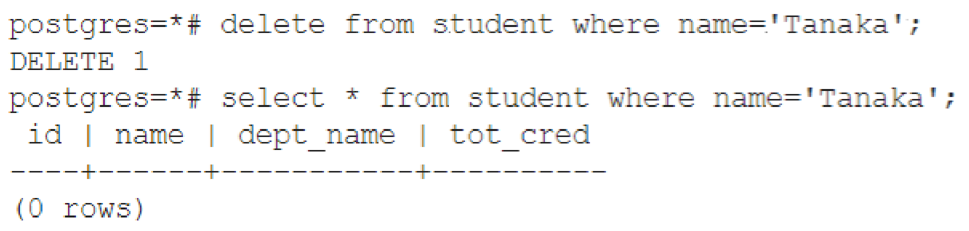
Step 4:

Transactions

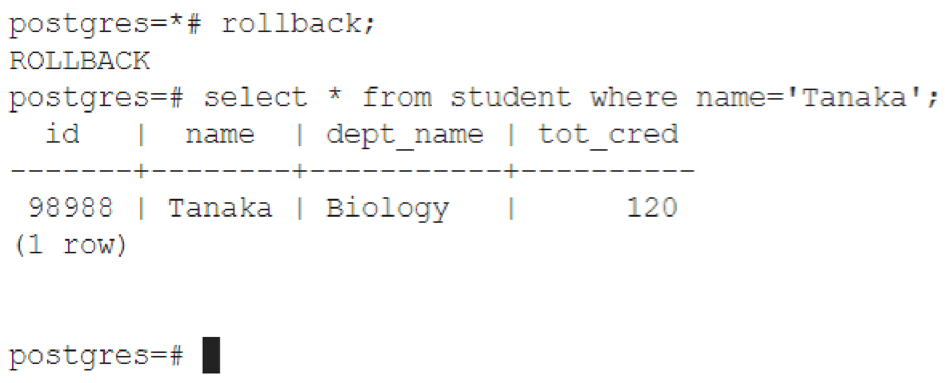
1. In this exercise, you will see how to rollback or commit transactions. By default PostgreSQL commits each SQL statement as soon as it is submitted. To prevent the transaction from committing immediately, you have to issue a command **begin;** to tell PostgreSQL to not commit immediately. You can issue any number of SQL statements after this, and then either **commit;** to commit the transaction, or **rollback;** to rollback the transaction. To see the effect, execute the following commands *one at a time*
   - begin ;
   - select * from student where name = 'Tanaka';
   - delete from student where name = 'Tanaka';
   - select * from student where name = 'Tanaka';
   - rollback;
   - select * from student where name = 'Tanaka';



```
postgres=# begin;
BEGIN
postgres=*# select * from student where name='Tanaka';
  id   |  name  | dept_name | tot_cred
-------+--------+-----------+----------
 98988 | Tanaka | Biology   |      120
(1 row)
```



```
postgres=*# delete from student where name='Tanaka';
DELETE 1
postgres=*# select * from student where name='Tanaka';
 id | name | dept_name | tot_cred
----+------+-----------+----------
(0 rows)
```



```
postgres=*# rollback;
ROLLBACK
postgres=# select * from student where name='Tanaka';
  id   |  name  | dept_name | tot_cred
-------+--------+-----------+----------
 98988 | Tanaka | Biology   |      120
(1 row)


postgres=# █
```

Observation: I observed that begin command sets the auto_commit flag false. On execution of first select command, it showed some data belonging to Tanaka. Then I deleted the entry of Tanaka. On execution of 2nd select query, no data was showed. Then I executed rollback query. Now, on execution of 3rd select query, data belonging to Tanaka was showed again.

Conclusion: From above observation, I concluded that, after execution of begin query, auto commit was set off. So, whatever changes made in to data either insertion, updation or deletion, those changes were not saved into permanent memory. Hence until we execute commit statement, they can be rollbacked using rollback query. On execution of rollback query, database is reset to last commit.

Concurrency

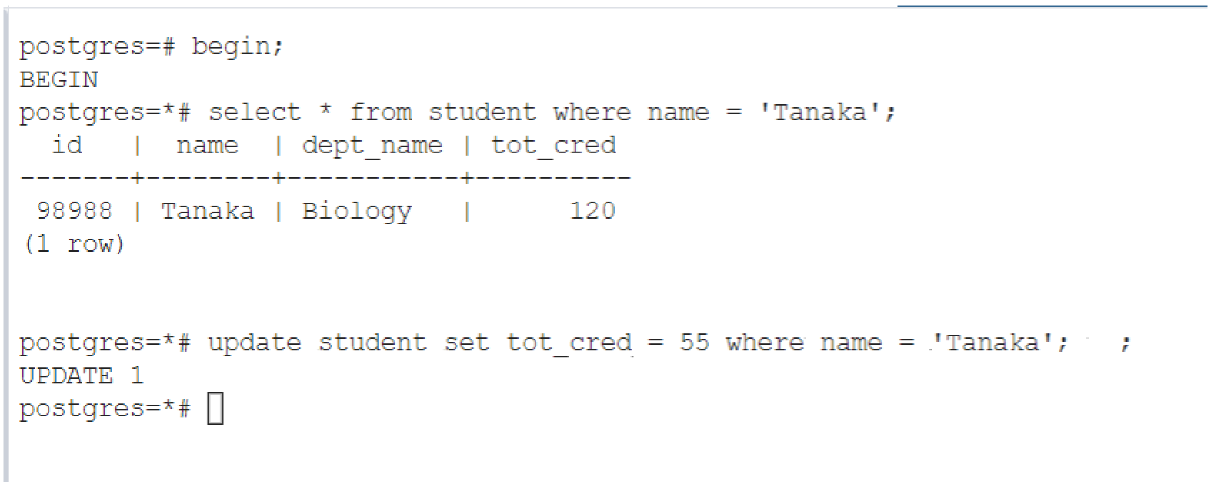
1] In this exercise you will run transactions concurrently from two different pgAdmin3 windows, to see how updates by one transaction affect another.

- Open two pgAdmin3 connections to the same database. Execute the following commands in sequence in the first window

1. begin ;
2. update student set tot_cred = 55 where name = 'Tanaka';

- Now in the second window execute
   1. begin;
   2. select * from student where name = 'Tanaka';
      - Look at the value of tot_cred. Can you figure out why you got the result that you saw? What does this tell you about concurrency control in PostgreSQL?

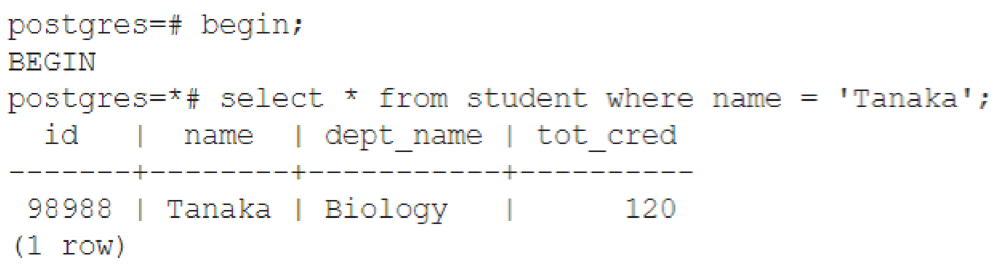
Shell 1



```
postgres=# begin;
BEGIN
postgres=*# select * from student where name = 'Tanaka';
  id   |  name  | dept_name | tot_cred
-------+--------+-----------+----------
 98988 | Tanaka | Biology   |      120
(1 row)


postgres=*# update student set tot_cred = 55 where name = 'Tanaka';   ;
UPDATE 1
postgres=*#
```

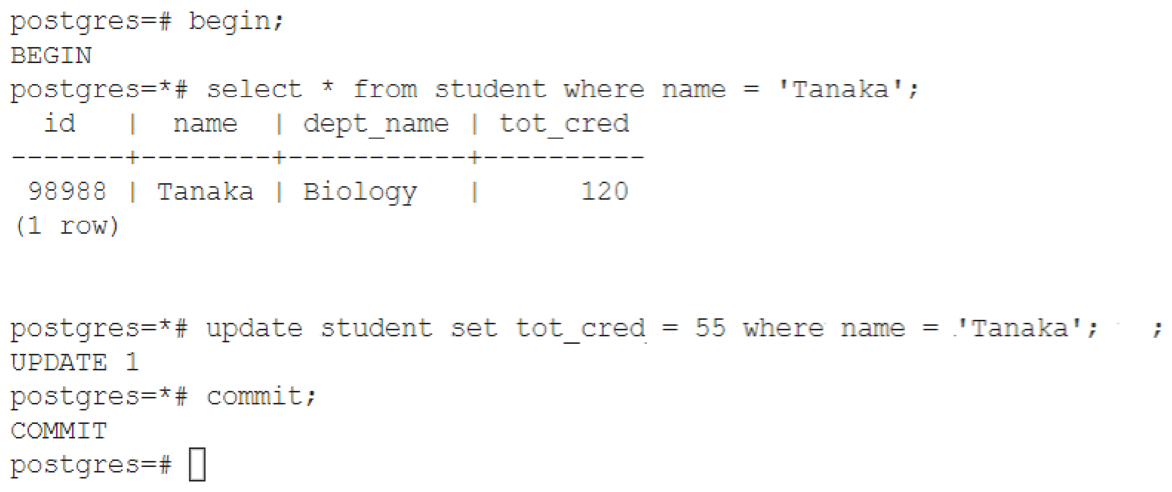
Shell 2



```
postgres=# begin;
BEGIN
postgres=*# select * from student where name = 'Tanaka';
  id   |  name  | dept_name | tot_cred
-------+--------+-----------+----------
 98988 | Tanaka | Biology   |      120
(1 row)
```

From execution of above queries, I came to know that PostgreSQL uses read committed isolation level as data hasn't been updated until I commit it. That's why we are seeing old values in other shell.

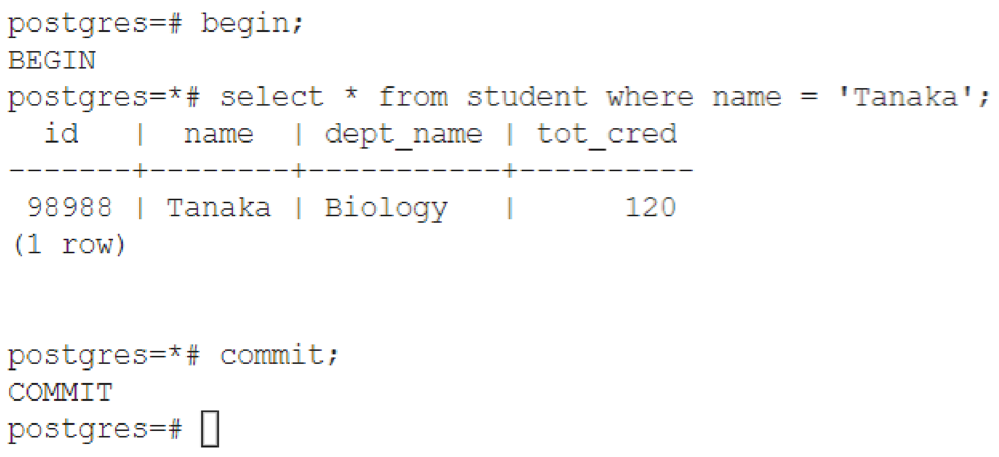
Shell 1



```
postgres=# begin;
BEGIN
postgres=*# select * from student where name = 'Tanaka';
  id   |  name  | dept_name | tot_cred
-------+--------+-----------+----------
 98988 | Tanaka | Biology   |      120
(1 row)


postgres=*# update student set tot_cred = 55 where name = 'Tanaka';    ;
UPDATE 1
postgres=*# commit;
COMMIT
postgres=#
```

Shell 2

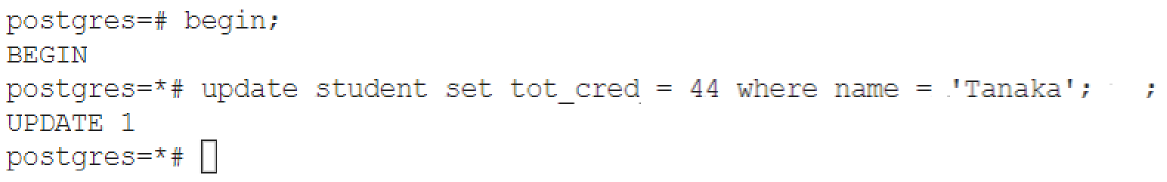


```
postgres=# begin;
BEGIN
postgres=*# select * from student where name = 'Tanaka';
  id   |  name  | dept_name | tot_cred
-------+--------+-----------+----------
 98988 | Tanaka | Biology   |      120
(1 row)


postgres=*# commit;
COMMIT
postgres=#
```

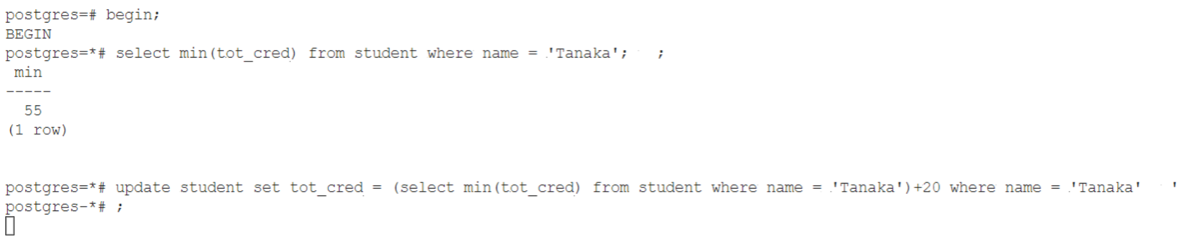
Both the shells committed successfully. In the first shell, changes got committed where as there were no changes in shell 2 it also got committed to new commit.

Shell 1



```
postgres=# begin;
BEGIN
postgres=*# update student set tot_cred = 44 where name = 'Tanaka';    ;
UPDATE 1
postgres=*#
```

Shell 2

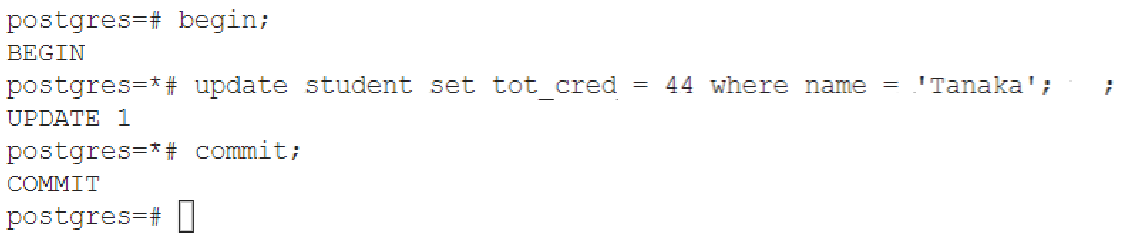


```
postgres=# begin;
BEGIN
postgres=*# select min(tot_cred) from student where name = 'Tanaka';   ;
 min
-----
  55
(1 row)


postgres=*# update student set tot_cred = (select min(tot_cred) from student where name = 'Tanaka')+20 where name = 'Tanaka'   '
postgres-*# ;
```

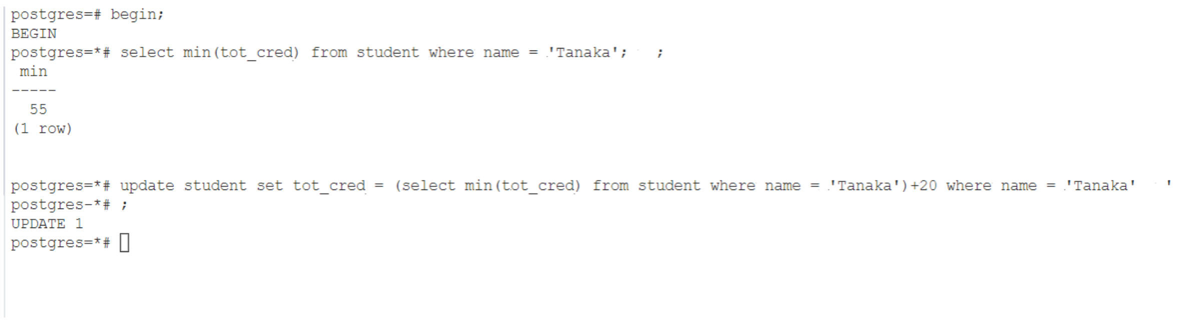
Here, query in shell to keeps on hanging as query in shell 1 is not yet committed. The lock acquired by shell 1 is not allowing shell 2 to make changes i.e. write in DB but it allows it to read from DB.

Shell 1



```
postgres=# begin;
BEGIN
postgres=*# update student set tot_cred = 44 where name = 'Tanaka';   ;
UPDATE 1
postgres=*# commit;
COMMIT
postgres=#
```

Shell 2

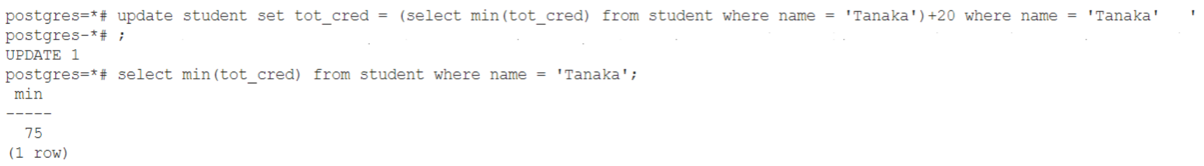


```
postgres=# begin;
BEGIN
postgres=*# select min(tot_cred) from student where name = 'Tanaka';   ;
 min
-----
  55
(1 row)


postgres=*# update student set tot_cred = (select min(tot_cred) from student where name = 'Tanaka')+20 where name = 'Tanaka'   '
postgres-*# ;
UPDATE 1
postgres=*#
```

On executing commit in shell 1, it releases lock and update query in shell 2 which was kept hanging got executed after acquiring write lock. (Serialized)

Shell 2



```
postgres=*# update student set tot_cred = (select min(tot_cred) from student where name = 'Tanaka')+20 where name = 'Tanaka'   '
postgres-*# ;
UPDATE 1
postgres=*# select min(tot_cred) from student where name = 'Tanaka';
 min
-----
  75
(1 row)
```

Select query showed 75 because command asked it to read value of tot_cred which was 55 at the time of reading as shell 1 query of updating it to 44 was not committed. Read commit isolation level doesn't restrict other accessors from reading data but restricts them from writing hence shell 2 read 55 even if it was updated by shell 1 but kept uncommitted.

(a) Next, in both windows execute the following commands:
begin;
set transaction isolation level serializable;
and reexecute the first query (one txn updates, the other reads), and see what happens. Describe what you observed, and explain why it happened.

In shell 1,

"ERROR: current transaction is aborted, commands ignored until end of transaction block"

Above message had appeared.

In shell 2,

It showed the data belonging to Tanaka with total_cred value of 75

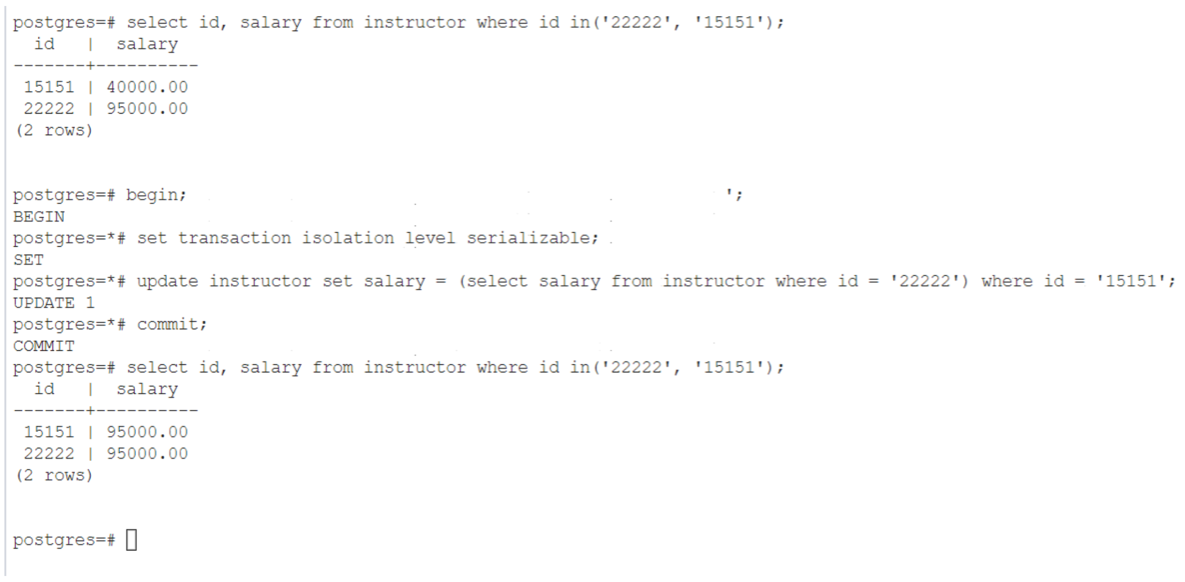
It is bcause serializable isolation level didn't allow shell 2 to make write changes nor any reads but allowed shell 2 to read as it was the first to initiate a transaction.

(b) Next do the same for the second query (both txns update).

After performing the queries in reverse order, I concluded that as shell 2 was the first to initiate the transaction, it was the only shell to write or read from db whereas shell 1 was not allowed to read as well.

Step 4:

Shell 1

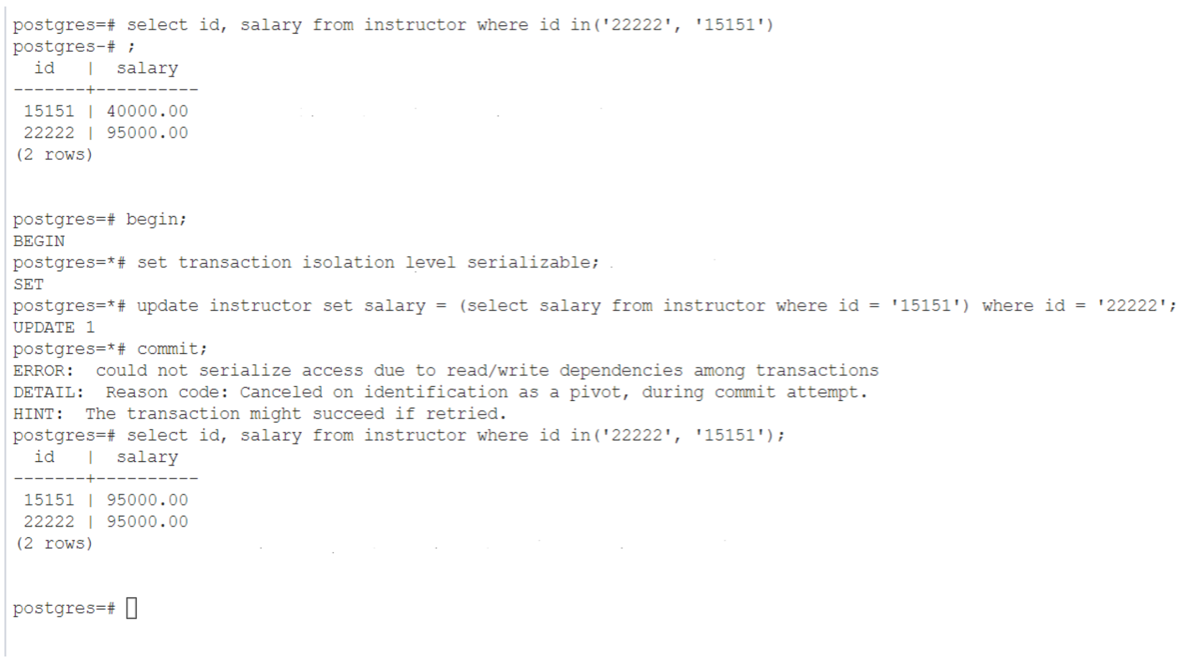


```
postgres=# select id, salary from instructor where id in('22222', '15151');
  id   |  salary
-------+----------
 15151 | 40000.00
 22222 | 95000.00
(2 rows)


postgres=# begin;                                                          ';
BEGIN
postgres=*# set transaction isolation level serializable;
SET
postgres=*# update instructor set salary = (select salary from instructor where id = '22222') where id = '15151';
UPDATE 1
postgres=*# commit;
COMMIT
postgres=# select id, salary from instructor where id in('22222', '15151');
  id   |  salary
-------+----------
 15151 | 95000.00
 22222 | 95000.00
(2 rows)


postgres=#
```

Shell 2



```
postgres=# select id, salary from instructor where id in('22222', '15151')
postgres-# ;
  id   |  salary
-------+----------
 15151 | 40000.00
 22222 | 95000.00
(2 rows)


postgres=# begin;
BEGIN
postgres=*# set transaction isolation level serializable;
SET
postgres=*# update instructor set salary = (select salary from instructor where id = '15151') where id = '22222';
UPDATE 1
postgres=*# commit;
ERROR:  could not serialize access due to read/write dependencies among transactions
DETAIL:  Reason code: Canceled on identification as a pivot, during commit attempt.
HINT:  The transaction might succeed if retried.
postgres=# select id, salary from instructor where id in('22222', '15151');
  id   |  salary
-------+----------
 15151 | 95000.00
 22222 | 95000.00
(2 rows)


postgres=#
```

As both the shells were working on same set of rows with serializable isolation level, shell 1 which first initiated the transaction, and updated the values. At the same time, shell 2 also tried to update values in same rows, so it was denied the access to them. In this way queries were serialized even if executed at the same time. After committing shell 1, shell 2 was able to read the data and write into it. Hence it considered new value only unlike read committed isolation level which allowed it to read older value.

Conclusion:

From above experiment, I understood difference between read committed isolation level and serializable isolation level. Read committed allowed to read but didn't allowed to write whereas serializable isolation didn’t even allowed to read once transaction is initiated concerning the same data.