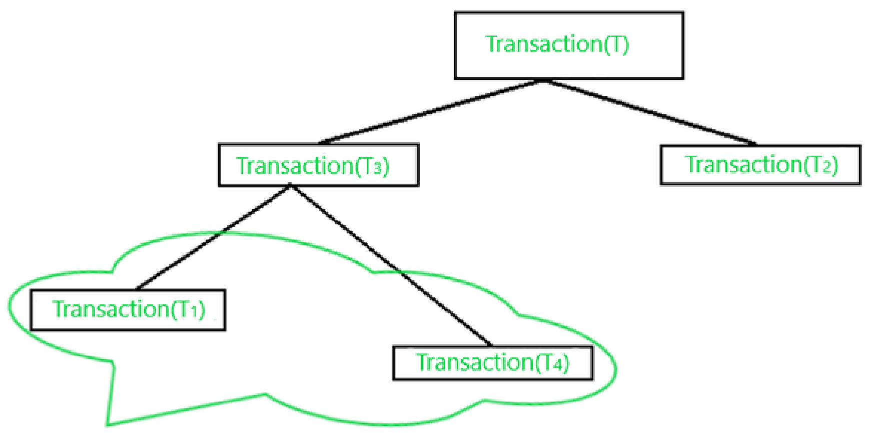
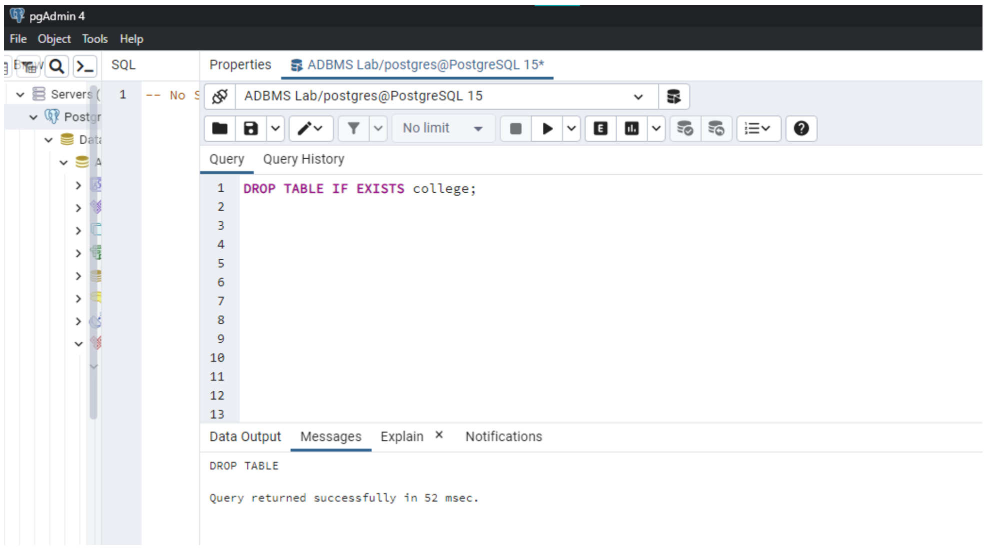
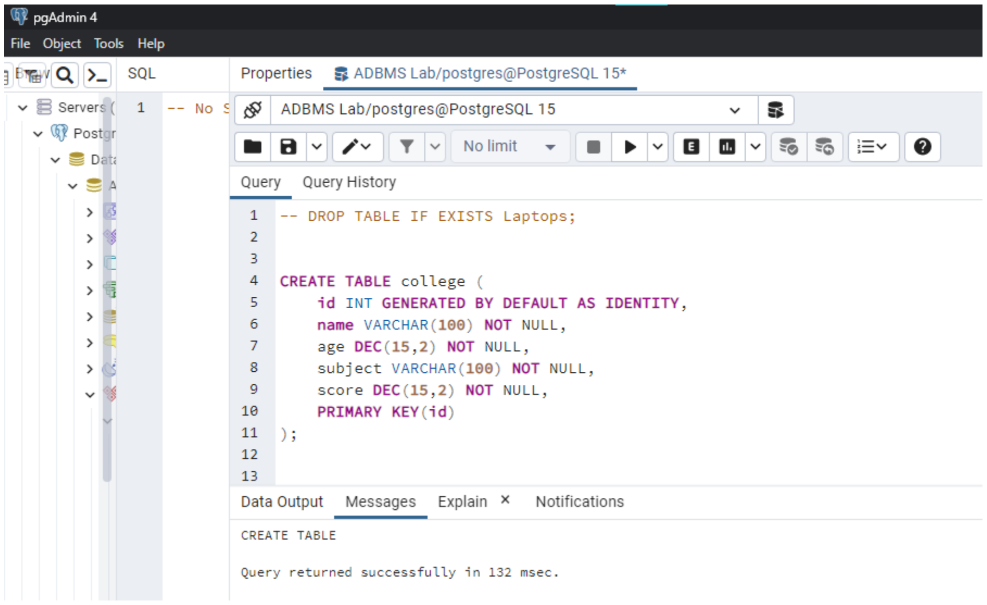
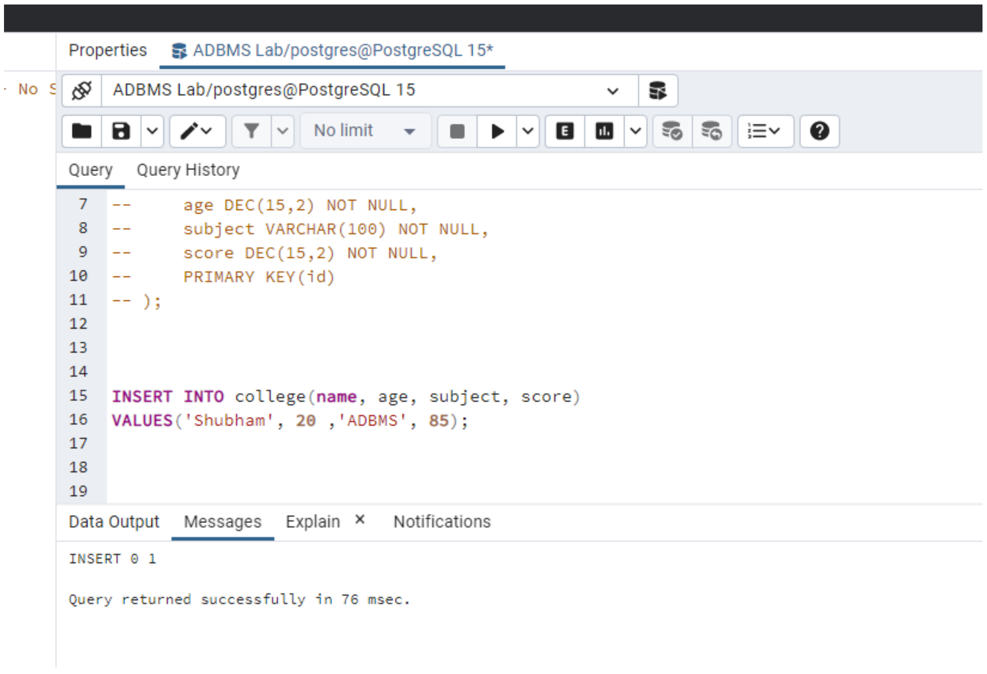
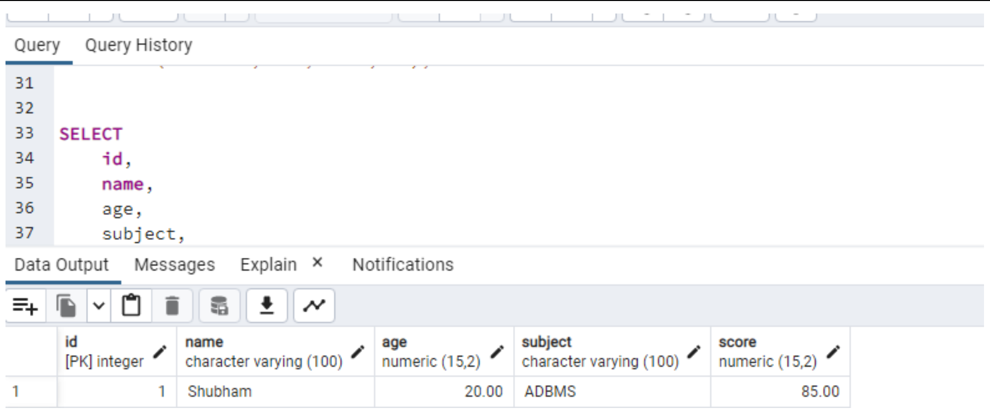
Name	Shubham Golwal
UID no.	2020300015
Experiment No.	01

AIM:	Concurrency control
Program 1	
Problem Statement :	Perform concurrency control in PostgreSQL Database with college database.
Theory :	**What are transactions?** A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity. **What is concurrency control?** Concurrency control concept comes under the Transaction in database management system (DBMS). It is a procedure in DBMS which helps us for the management of two simultaneous processes to execute without conflicts between each other, these conflicts occur in multi user systems. Concurrency can simply be said to be executing multiple transactions at a time. It is required to increase time efficiency. If many transactions try to access the same data, then inconsistency arises. Concurrency control required to maintain consistency data. For example, if we take ATM machines and do not use concurrency, multiple persons cannot draw money at a time in different places. This is where we need concurrency. **The advantages of concurrency control are as follows −** • Waiting time will be decreased. • Response time will decrease. • Resource utilization will increase. • System performance & Efficiency is increased.

	 Transaction(T)
	Transaction(T3)
	Transaction(T2)
	Transaction(T1)
	Transaction(T4)
	All the descendents release the held locks and retained locks
	**credits to gfg*

Output	Drop:  `DROP TABLE IF EXISTS college;` DROP TABLE Query returned successfully in 52 msec.

	Create:
	
	 pgAdmin 4
	File Object Tools Help
	SQL
	Properties
	ADBMS Lab/postgres@PostgreSQL 15*
	ADBMS Lab/postgres@PostgreSQL 15
	No limit
	Query Query History
	Data Output Messages Explain Notifications
	`-- DROP TABLE IF EXISTS Laptops;`
	
	`CREATE TABLE college (`
	`    id INT GENERATED BY DEFAULT AS IDENTITY,`
	`    name VARCHAR(100) NOT NULL,`
	`    age DEC(15,2) NOT NULL,`
	`    subject VARCHAR(100) NOT NULL,`
	`    score DEC(15,2) NOT NULL,`
	`    PRIMARY KEY(id)`
	`);`
	
	`CREATE TABLE`
	
	`Query returned successfully in 132 msec.`
	
	Insert:
	
	 Properties
	ADBMS Lab/postgres@PostgreSQL 15*
	ADBMS Lab/postgres@PostgreSQL 15
	No limit
	Query Query History
	Data Output Messages Explain Notifications
	`--     age DEC(15,2) NOT NULL,`
	`--     subject VARCHAR(100) NOT NULL,`
	`--     score DEC(15,2) NOT NULL,`
	`--     PRIMARY KEY(id)`
	`-- );`
	
	`INSERT INTO college(name, age, subject, score)`
	`VALUES('Shubham', 20 ,'ADBMS', 85);`
	
	`INSERT 0 1`
	
	`Query returned successfully in 76 msec.`

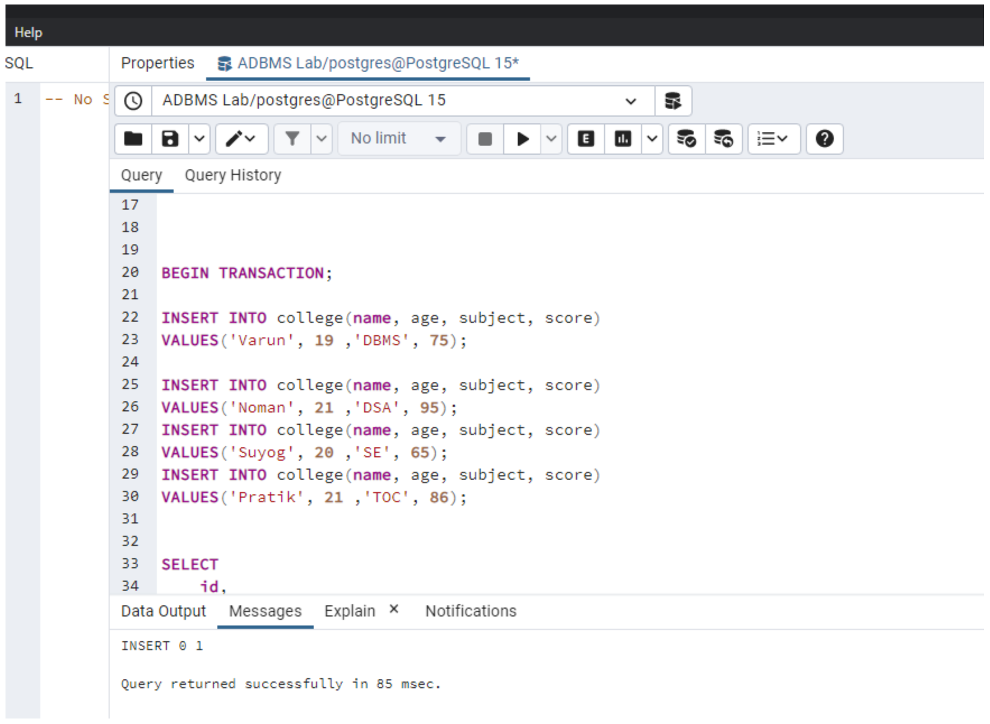


```
SELECT
    id,
    name,
    age,
    subject,
```

	id [PK] integer	name character varying (100)	age numeric (15,2)	subject character varying (100)	score numeric (15,2)
1	1	Shubham	20.00	ADBMS	85.00

Initial insert into database

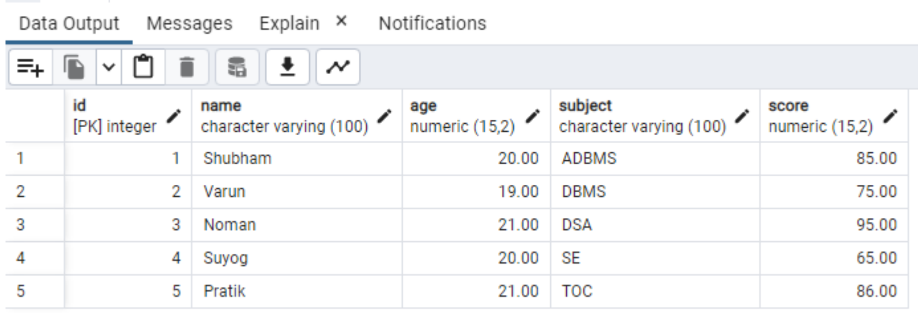
Begin Transaction:


Help
SQL
Properties
ADBMS Lab/postgres@PostgreSQL 15*
ADBMS Lab/postgres@PostgreSQL 15
No limit
Query
Query History

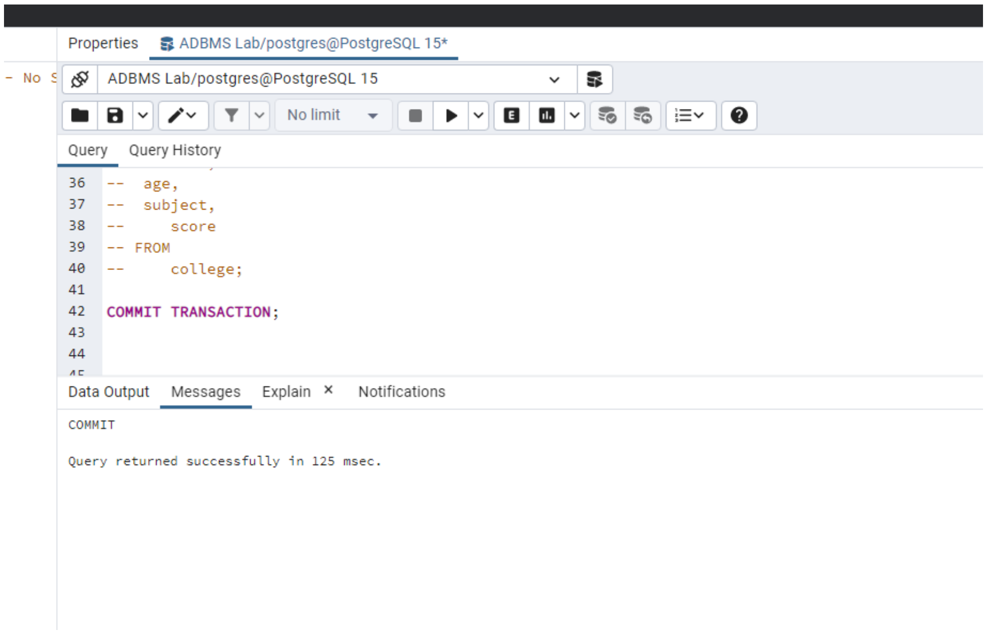
```
-- No S
```

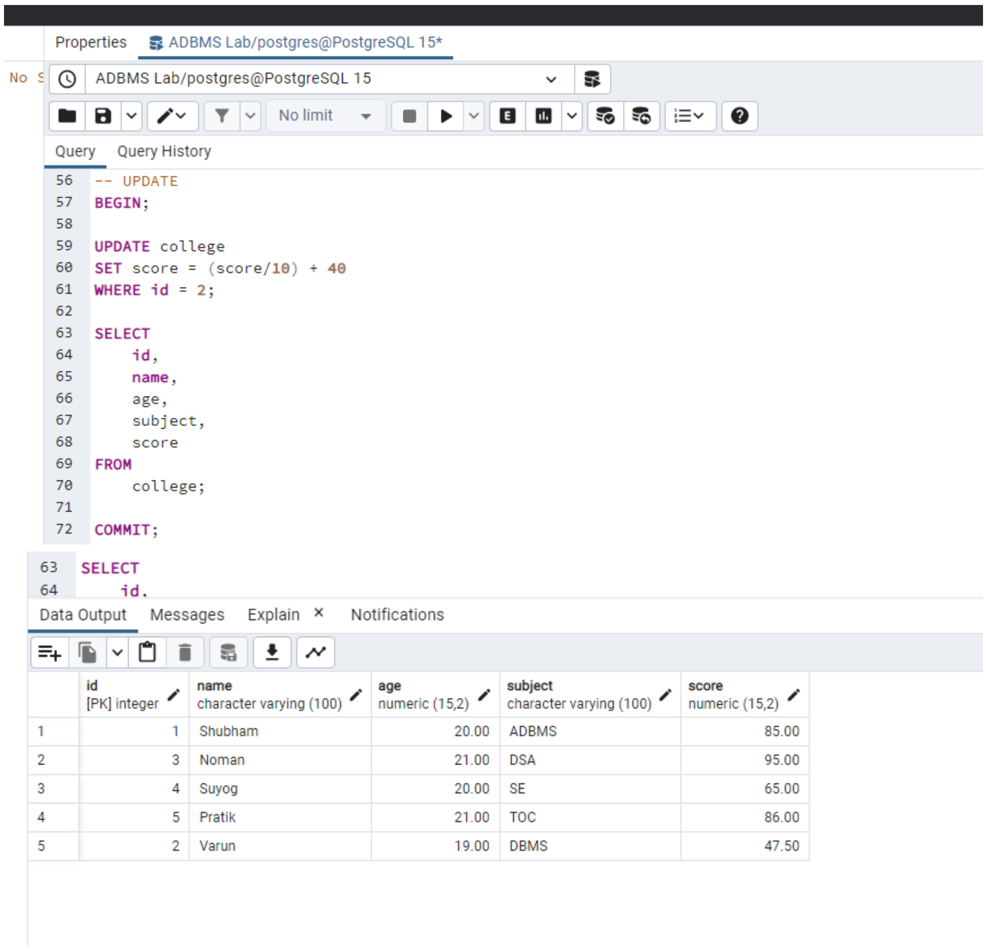
```
BEGIN TRANSACTION;

INSERT INTO college(name, age, subject, score)
VALUES('Varun', 19 ,'DBMS', 75);

INSERT INTO college(name, age, subject, score)
VALUES('Noman', 21 ,'DSA', 95);
INSERT INTO college(name, age, subject, score)
VALUES('Suyog', 20 ,'SE', 65);
INSERT INTO college(name, age, subject, score)
VALUES('Pratik', 21 ,'TOC', 86);


SELECT
    id,
```

```
INSERT 0 1

Query returned successfully in 85 msec.
```



	id [PK] integer	name character varying (100)	age numeric (15,2)	subject character varying (100)	score numeric (15,2)
1	1	Shubham	20.00	ADBMS	85.00
2	2	Varun	19.00	DBMS	75.00
3	3	Noman	21.00	DSA	95.00
4	4	Suyog	20.00	SE	65.00
5	5	Pratik	21.00	TOC	86.00

	Started a new transaction and insert a new student into the college table:
	
	Commit:
	
	Properties ADBMS Lab/postgres@PostgreSQL 15*
	ADBMS Lab/postgres@PostgreSQL 15
	No limit
	Query Query History
	`-- age,`
	`-- subject,`
	`--     score`
	`-- FROM`
	`--     college;`
	
	`COMMIT TRANSACTION;`
	Data Output Messages Explain Notifications
	`COMMIT`
	
	`Query returned successfully in 125 msec.`
	
	Made the changes become visible to other sessions (or users), committed the transaction by using the COMMIT TRANSACTION statement
	
	Alternative: COMMIT WORK, COMMIT;
	
	After executing the COMMIT statement, PostgreSQL also guarantees that the change will be durable if a crash happens.
	
	------x--------------------
	
	Update:
	
	Currently score at id = 2 is 75.



Properties | ADBMS Lab/postgres@PostgreSQL 15*

ADBMS Lab/postgres@PostgreSQL 15

No limit

Query | Query History

```
-- UPDATE
BEGIN;

UPDATE college
SET score = (score/10) + 40
WHERE id = 2;

SELECT
    id,
    name,
    age,
    subject,
    score
FROM
    college;

COMMIT;
```

```
SELECT
    id,
```

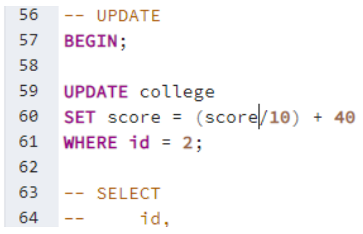
Data Output | Messages | Explain | Notifications

	id [PK] integer	name character varying (100)	age numeric (15,2)	subject character varying (100)	score numeric (15,2)
1	1	Shubham	20.00	ADBMS	85.00
2	3	Noman	21.00	DSA	95.00
3	4	Suyog	20.00	SE	65.00
4	5	Pratik	21.00	TOC	86.00
5	2	Varun	19.00	DBMS	47.50

Changes is made to id =2 [Varun] and score is updated to 47.5
After committing changes are made visible.

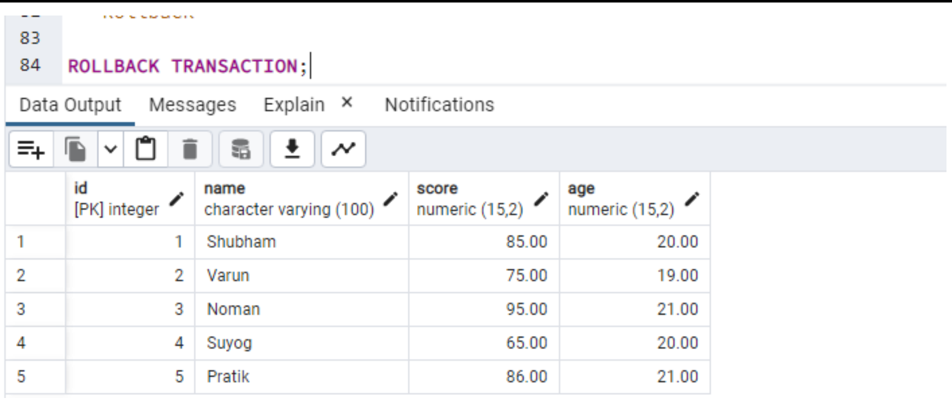
Rollback:

Updated but not committed:



```
-- UPDATE
BEGIN;

UPDATE college
SET score = (score/10) + 40
WHERE id = 2;

-- SELECT
--      id,
```

Score at id = 2 is 47.5.

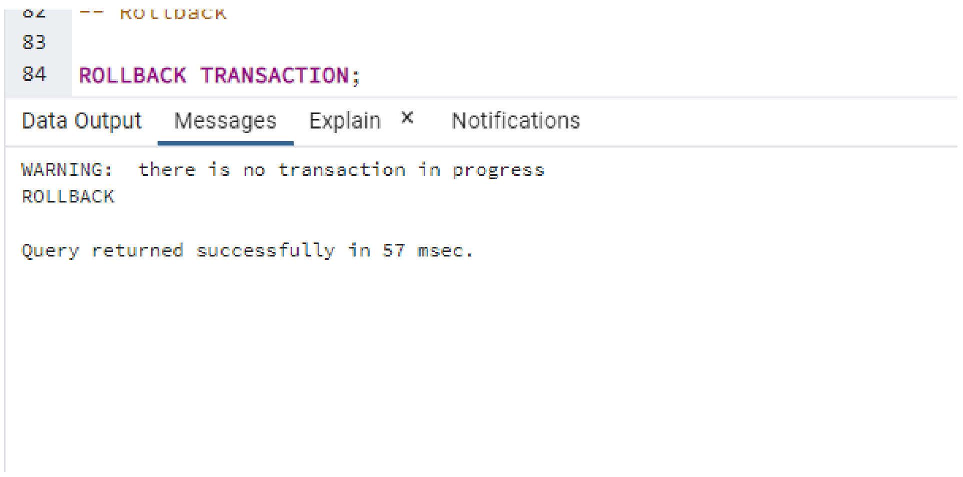
```
ROLLBACK TRANSACTION;
```

Data Output | Messages | Explain | Notifications

	id [PK] integer	name character varying (100)	score numeric (15,2)	age numeric (15,2)
1	1	Shubham	85.00	20.00
2	2	Varun	75.00	19.00
3	3	Noman	95.00	21.00
4	4	Suyog	65.00	20.00
5	5	Pratik	86.00	21.00

After executing rollback score at id = 2 is reverted to 75

But if transaction is committed then we can’t revert back.



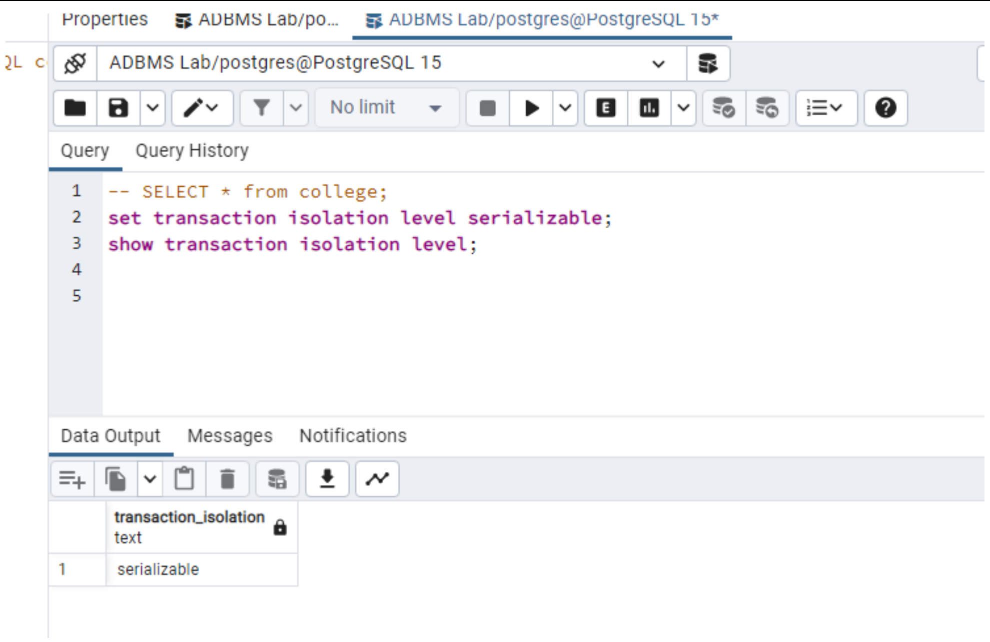
```
-- Rollback

ROLLBACK TRANSACTION;
```

Data Output | Messages | Explain | Notifications

```
WARNING:  there is no transaction in progress
ROLLBACK

Query returned successfully in 57 msec.
```

Serializability:

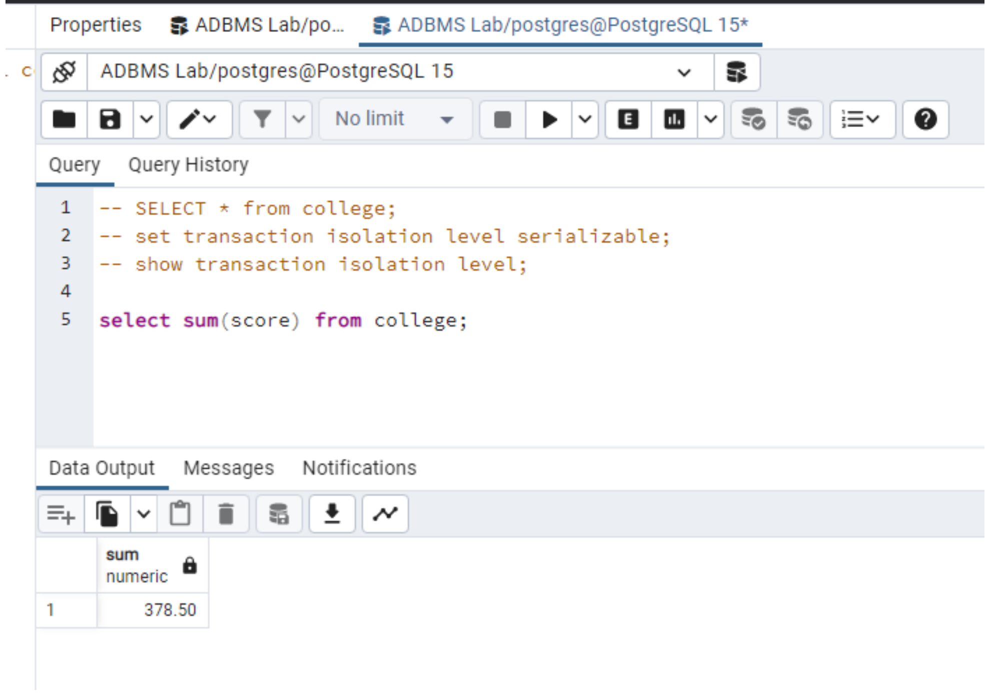
I’m gonna start 2 new transactions, then set their isolation level to serializable.



```
-- SELECT * from college;
set transaction isolation level serializable;
show transaction isolation level;
```

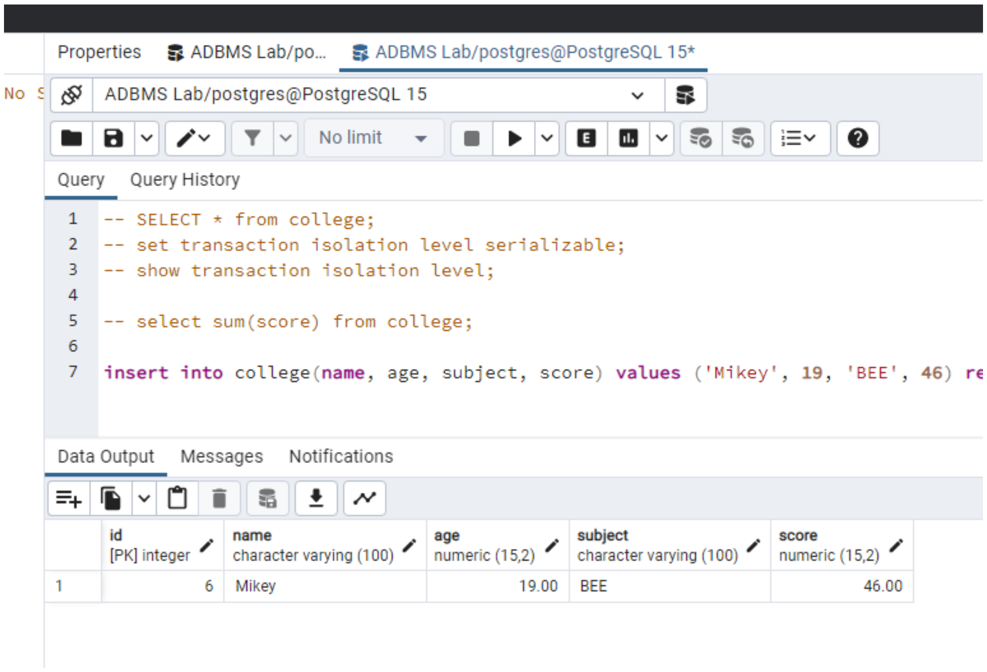
```
   transaction_isolation
   text
1  serializable
```

Get sum of all the scores of students



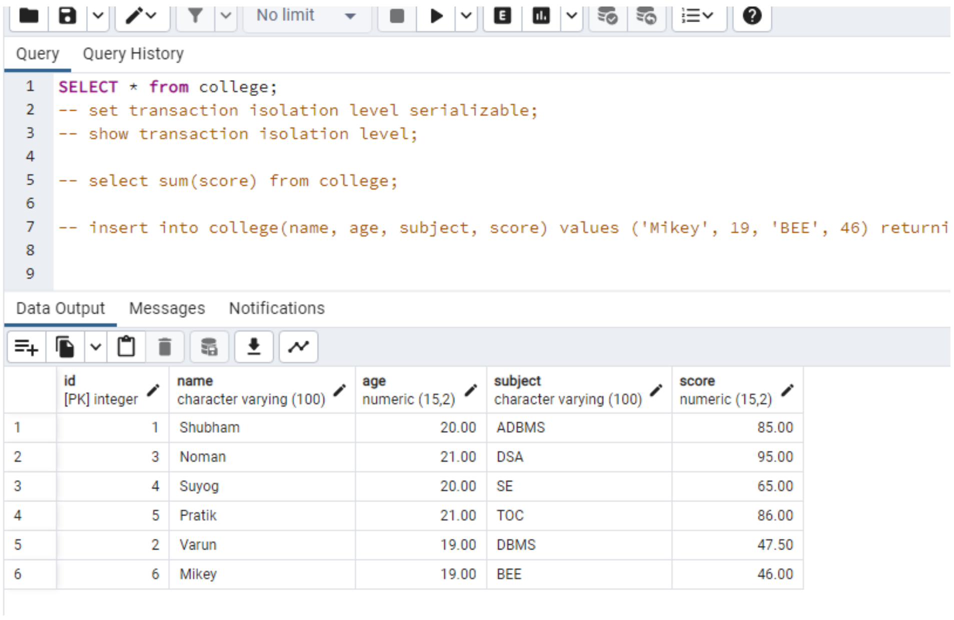
```
-- SELECT * from college;
-- set transaction isolation level serializable;
-- show transaction isolation level;

select sum(score) from college;
```

```
   sum
   numeric
1  378.50
```



```
-- SELECT * from college;
-- set transaction isolation level serializable;
-- show transaction isolation level;

-- select sum(score) from college;

insert into college(name, age, subject, score) values ('Mikey', 19, 'BEE', 46) re
```

	id [PK] integer	name character varying (100)	age numeric (15,2)	subject character varying (100)	score numeric (15,2)
1	6	Mikey	19.00	BEE	46.00

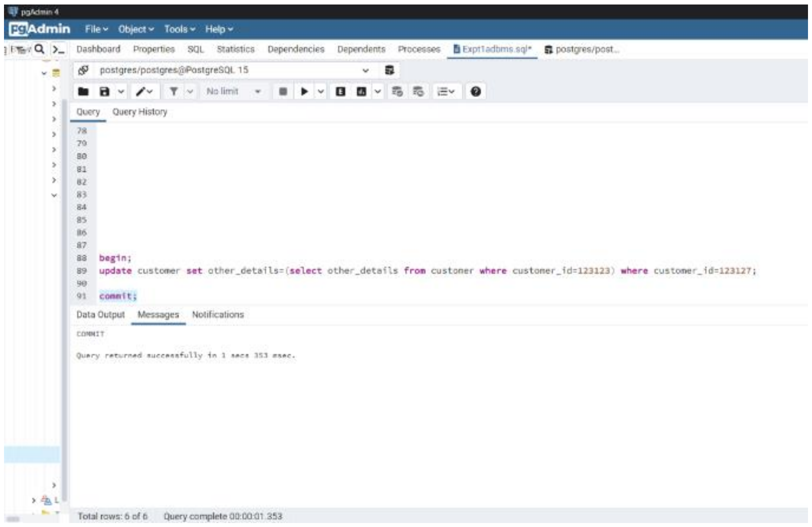


```
SELECT * from college;
-- set transaction isolation level serializable;
-- show transaction isolation level;

-- select sum(score) from college;

-- insert into college(name, age, subject, score) values ('Mikey', 19, 'BEE', 46) returni


```

	id [PK] integer	name character varying (100)	age numeric (15,2)	subject character varying (100)	score numeric (15,2)
1	1	Shubham	20.00	ADBMS	85.00
2	3	Noman	21.00	DSA	95.00
3	4	Suyog	20.00	SE	65.00
4	5	Pratik	21.00	TOC	86.00
5	2	Varun	19.00	DBMS	47.50
6	6	Mikey	19.00	BEE	46.00

Serialized updates:

Window 1:

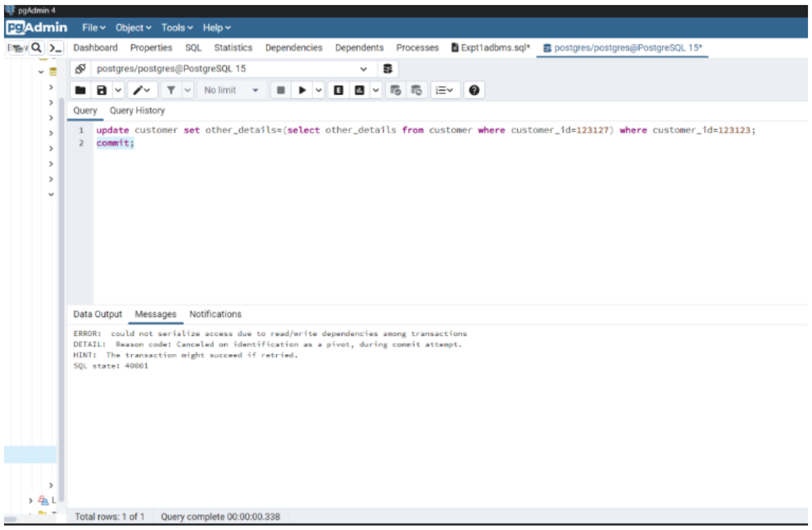
	

```
begin;
update customer set other_details=(select other_details from customer where customer_id=123123) where customer_id=123127;

commit;
```

```
COMMIT

Query returned successfully in 1 secs 353 msec.
```

Total rows: 6 of 6   Query complete 00:00:01.353

Window 2:



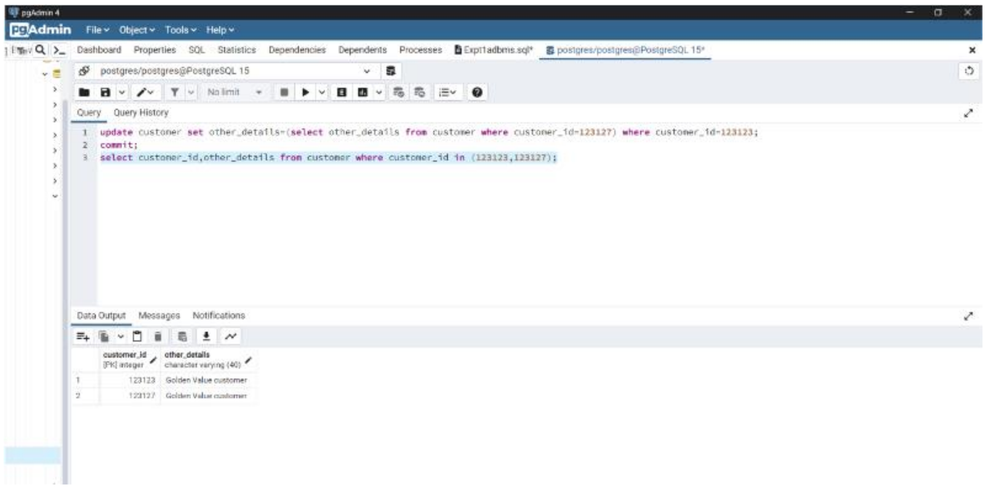
```
update customer set other_details=(select other_details from customer where customer_id=123127) where customer_id=123123;
commit;
```

```
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.
SQL state: 40001
```

Total rows: 1 of 1   Query complete 00:00:00.338

Error: could not serialize access due to read/write dependencies among transactions DETAIL:
Reason code: Cancelled on identification as a pivot, during commit attempt.
HINT: The transaction might succeed if retried. SQL state: 40001

Why has it happened?
When we committed first update operation it worked fine. But when we committed second one on same tuple then it throws serializable error.



```
update customer set other_details=(select other_details from customer where customer_id=123127) where customer_id=123123;
commit;
select customer_id,other_details from customer where customer_id in (123123,123127);
```

```
   customer_id      other_details
   [PK] integer     character varying (40)
1       123123      Golden Value customer
2       123127      Golden Value customer
```

Checking serializable:
Window 2:

Is this equivalent to any serializable schedule?
Yes, this schedule is conflict serializable. Because we are performing write operation from both transactions at the same time.

Conclusion:

- I learnt meaning of serializability and concurrency in a transaction.
- I also learnt that various Transaction language (TCL) commands are used for managing and controlling the transactions.
- Executed serializability and under the conflict in serializability.