**Міністерство освіти і науки України**

**Національний технічний університет України**

**“Київський політехнічний інститут ім. Ігоря Сікорського”**

**Факультет прикладної математики**

**Кафедра спеціалізованих комп’ютерних систем**

**Лабораторна робота № 3**

з дисципліни «Бази даних і засоби управління»

«Ознайомлення з базовими операціями СУБД PostgreSQL»

Виконала:

студентка групи КП-73

Берещенко Анастасія

Перевірив:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Завдання

Завдання роботи полягає у наступному:

1.Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).

2.Створити та проаналізувати різні типи індексів у PostgreSQL.

3.Розробити тригер бази даних PostgreSQL.

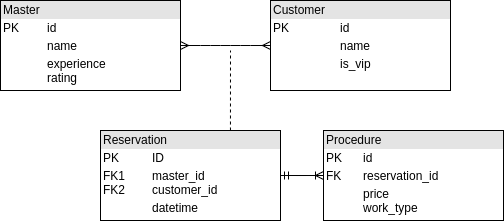
4.Навести приклади та проаналізувати рівні ізоляції транзакцій уPostgreSQL.

Порядок виконання роботи

В ході роботи розроблено:

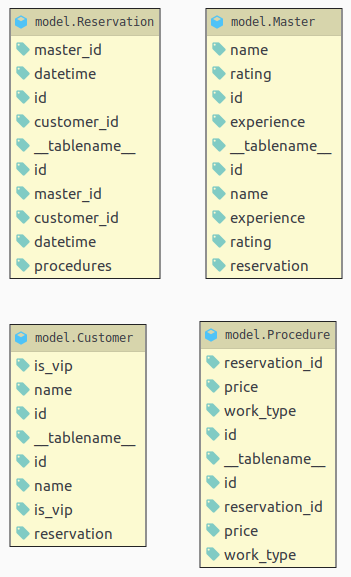
1. Логічну модель БД та Діаграму класів;
2. Функціонал програмного додатку;
3. ОО програмний додаток роботи з БД " … ". Для взаємодії з БД використано ORM модуль SQLAlchemy.

Логічна модель бази даних наведена на Рис 1.



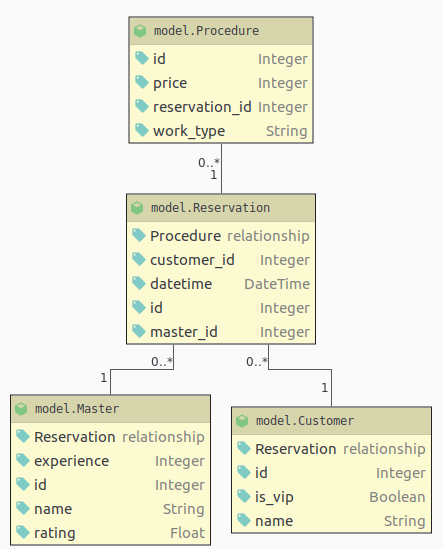
*Рис 1. Логічна модель бази даних*

Сутнісні класи програми наведені на Рис 2.



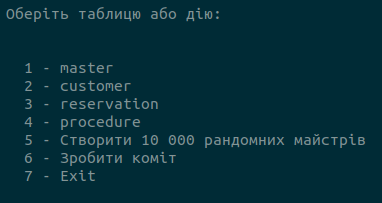
*Рис 2. Фрагмент UML діаграми сутнісних класів*

Зв’язки між сутнісними класами, сгенеровані за допомогою SqlAlchemy, наведені на Рис 3.

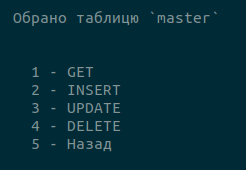


*Рис 3. Зв’язки між сутнісними класами*

Меню програми наведене на схемі нижче.



Пункт 1-4:



Пункт 5:



Пункт 6:



Пункт 7 (вихід з програми):



Лістинг програми

*import* controller  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 controller.Controller().show\_start\_menu()

controller.py

*from* consolemenu *import* SelectionMenu  
  
*import* model  
*import* view  
*import* scanner  
  
  
*class* Controller:  
 *def \_\_init\_\_*(*self*):  
 *self*.model = model.Model()  
 *self*.view = view.View()  
  
 *self*.tables = *list*(model.TABLES.keys())  
  
 *def* get\_table\_name(*self*, index):  
 *try*:  
 *return self*.tables[index]  
 *except IndexError*:  
 *return None*  
  
 *def* show\_start\_menu(*self*, subtitle='', \*\*kwargs):  
 menu\_options = *self*.tables + ['Створити 10 000 рандомних майстрів',  
 'Зробити коміт']  
 next\_steps = [*self*.show\_table\_menu] \* *len*(*self*.tables) + [  
 *self*.create\_random\_masters,  
 *self*.commit  
 ]  
 menu = SelectionMenu(menu\_options, subtitle=subtitle,  
 title="Оберіть таблицю або дію:")  
 menu.show()  
  
 index = menu.selected\_option  
 *if* index < *len*(menu\_options):  
 table\_name = *self*.get\_table\_name(index)  
 next\_step = next\_steps[index]  
 *try*:  
 next\_step(table\_name=table\_name)  
 *except Exception as* err:  
 *self*.show\_start\_menu(subtitle=*str*(err))  
 *else*:  
 *print*('Пака!!!!')  
  
 *def* show\_table\_menu(*self*, table\_name, subtitle=''):  
 next\_steps = [*self*.get, *self*.insert, *self*.update, *self*.delete, *self*.show\_start\_menu]  
 menu = SelectionMenu(  
 ['GET', 'INSERT', 'UPDATE', 'DELETE'], subtitle=subtitle,  
 title=f'Обрано таблицю `{table\_name}`', exit\_option\_text='Назад', )  
 menu.show()  
  
 next\_step = next\_steps[menu.selected\_option]  
 next\_step(table\_name=table\_name)  
  
 *def* get(*self*, table\_name):  
 filter\_by = scanner.input\_dict(table\_name, 'За чим фільтрувати запит? Залиште пустим щоб отримати всі рядки:')  
 data = *self*.model.get(table\_name, \*\*filter\_by)  
 *self*.view.print\_entities(table\_name, data)  
 scanner.press\_enter()  
 *self*.show\_table\_menu(table\_name)  
  
 *def* insert(*self*, table\_name):  
 new\_values = scanner.input\_dict(table\_name, 'Введіть нові значення:')  
 *self*.model.insert(table\_name, \*\*new\_values)  
 *self*.show\_table\_menu(table\_name, 'Вставка відбулася успішно')  
  
 *def* update(*self*, table\_name):  
 filter\_by = scanner.input\_dict(table\_name, 'Який рядок треба змінити?:', limit=1)  
 new\_values = scanner.input\_dict(table\_name, 'Введіть нові значення:')  
 *self*.model.update(table\_name, *list*(filter\_by.items())[0], \*\*new\_values)  
 *self*.show\_table\_menu(table\_name, 'Оновлення відбулося успішно')  
  
 *def* delete(*self*, table\_name):  
 filter\_by = scanner.input\_dict(table\_name, 'Які рядки треба видалити?')  
 *self*.model.delete(table\_name, \*\*filter\_by)  
 *self*.show\_table\_menu(table\_name, 'Видалення відбулося успішно')  
  
 *def* create\_random\_masters(*self*, \*\*kwargs):  
 *self*.model.create\_random\_masters()  
 *self*.show\_start\_menu('10 000 випадкових майстрів додано')  
  
 *def* commit(*self*, \*\*kwargs):  
 *self*.model.commit()  
 *self*.show\_start\_menu(subtitle='Зміни успішно збережені')

model.py

*from* sqlalchemy *import* Column, Integer, String, DateTime, \  
 Boolean, ForeignKey, Float, create\_engine  
*from* sqlalchemy.ext.declarative *import* declarative\_base  
*from* sqlalchemy.orm *import* relationship, sessionmaker  
  
db\_str = 'postgres://admin:admin@localhost:5432/kpi'  
db = create\_engine(db\_str)  
Base = declarative\_base()  
  
  
*class* ReprMixin:  
 *def \_\_repr\_\_*(*self*):  
 clean\_dict = *self*.*\_\_dict\_\_*.copy()  
 clean\_dict.pop('\_sa\_instance\_state')  
 *return* f'<{*self*.\_\_class\_\_.*\_\_name\_\_*}>{clean\_dict})'  
  
  
*class* Master(Base, ReprMixin):  
 \_\_tablename\_\_ = 'master'  
  
 id = Column(Integer, primary\_key=*True*)  
 name = Column(String)  
 experience = Column(Integer)  
 rating = Column(Float)  
  
 reservation = relationship('Reservation')  
  
 *def \_\_init\_\_*(*self*, id=*None*, name=*None*, experience=*None*, rating=*None*):  
 *self*.id = id  
 *self*.name = name  
 *self*.experience = experience  
 *self*.rating = rating  
  
  
*class* Customer(Base, ReprMixin):  
 \_\_tablename\_\_ = 'customer'  
  
 id = Column(Integer, primary\_key=*True*)  
 name = Column(String)  
 is\_vip = Column(Boolean)  
  
 reservation = relationship('Reservation')  
  
 *def \_\_init\_\_*(*self*, id=*None*, name=*None*, is\_vip=*None*):  
 *self*.id = id  
 *self*.name = name  
 *self*.is\_vip = is\_vip  
  
  
*class* Reservation(Base, ReprMixin):  
 \_\_tablename\_\_ = 'reservation'  
  
 id = Column(Integer, primary\_key=*True*)  
 master\_id = Column(Integer, ForeignKey('master.id'))  
 customer\_id = Column(Integer, ForeignKey('customer.id'))  
 datetime = Column(DateTime)  
  
 procedures = relationship('Procedure')  
  
 *def \_\_init\_\_*(*self*, id=*None*, master\_id=*None*, customer\_id=*None*, datetime=*None*):  
 *self*.id = id  
 *self*.master\_id = master\_id  
 *self*.customer\_id = customer\_id  
 *self*.datetime = datetime  
  
  
*class* Procedure(Base, ReprMixin):  
 \_\_tablename\_\_ = 'procedure'  
  
 id = Column(Integer, primary\_key=*True*)  
 reservation\_id = Column(Integer, ForeignKey('reservation.id'))  
 price = Column(Integer)  
 work\_type = Column(String)  
  
 *def \_\_init\_\_*(*self*, id=*None*, reservation\_id=*None*, price=*None*, work\_type=*None*):  
 *self*.id = id  
 *self*.reservation\_id = reservation\_id  
 *self*.price = price  
 *self*.work\_type = work\_type  
  
  
Base.metadata.create\_all(db)  
  
TABLES = {  
 'master': ('id', 'name', 'experience', 'rating'),  
 'customer': ('id', 'name', 'is\_vip'),  
 'reservation': ('id', 'master\_id', 'customer\_id', 'datetime'),  
 'procedure': ('id', 'reservation\_id', 'price', 'work\_type')  
}  
  
CLASSES = {  
 'procedure': Procedure, 'customer': Customer,  
 'reservation': Reservation, 'master': Master  
}  
  
  
*class* Model:  
 *def \_\_init\_\_*(*self*):  
 *self*.session = sessionmaker(db)()  
  
 *def* create\_tables(*self*):  
 *with open*('scripts/create.sql') *as* file:  
 command = file.read()  
 *self*.session.execute(command)  
 *self*.session.commit()  
  
 *def* get(*self*, table\_name, \*\*filter\_by):  
 objects\_class = CLASSES[table\_name]  
 objects = *self*.session.query(objects\_class)  
 *for* key, item *in* filter\_by.items():  
 objects = objects.filter(*getattr*(objects\_class, key) == item)  
  
 *return list*(objects)  
  
 *def* insert(*self*, table\_name, \*\*new\_values):  
 object\_class = CLASSES[table\_name]  
 obj = object\_class(\*\*new\_values)  
 *self*.session.add(obj)  
  
 *def* update(*self*, table\_name, condition, \*\*new\_values):  
 *if not* new\_values:  
 *raise Exception*('Не вказані поля, які треба оновити')  
  
 column, value = condition  
 object\_class = CLASSES[table\_name]  
 filter\_attr = *getattr*(object\_class, column)  
 objects = *self*.session.query(object\_class).filter(filter\_attr == value)  
  
 *for* obj *in* objects:  
 *for* key, item *in* new\_values.items():  
 *setattr*(obj, key, item)  
  
 *def* delete(*self*, table\_name, \*\*filter\_by):  
 *if not* filter\_by:  
 *raise Exception*('Не вказані умови для рядків, які треба видалити')  
  
 objects\_class = CLASSES[table\_name]  
 objects = *self*.session.query(objects\_class)  
 *for* key, item *in* filter\_by.items():  
 objects = objects.filter(*getattr*(objects\_class, key) == item)  
  
 objects.delete()  
  
 *def* commit(*self*):  
 *self*.session.commit()  
  
 *def* create\_random\_masters(*self*):  
 *with open*('scripts/random.sql') *as* file:  
 sql = file.read()  
 *self*.session.execute(sql)

view.py

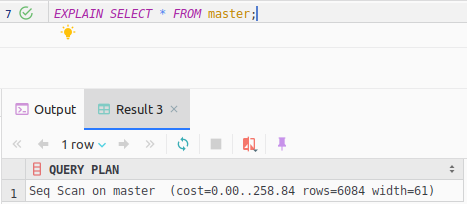
*class* View:  
 *def \_\_init\_\_*(*self*):  
 *self*.SEPARATOR\_WIDTH = 30  
  
 *def* print\_entities(*self*, table\_name, data):  
 separator\_line = '-' \* *self*.SEPARATOR\_WIDTH  
  
 *print*(f'Результат для таблиці `{table\_name}`', end='\n\n')  
 *for* entity *in* data:  
 *print*(entity)

**Індекси**

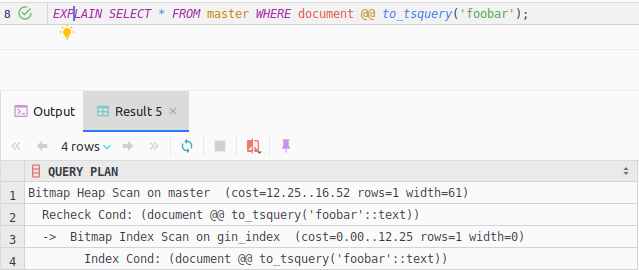
**Gin індекс:**

*ALTER TABLE* master *ADD COLUMN document tsvector*;  
*UPDATE* master *SET document* = *to\_tsvector*(*name*) *WHERE true*;  
*CREATE INDEX* gin\_index *ON* master *using* gin(*document*);

Порядок звертання до таблиці без використання фільтру по колонці, на яку додано індекс (створений індекс не використовується):



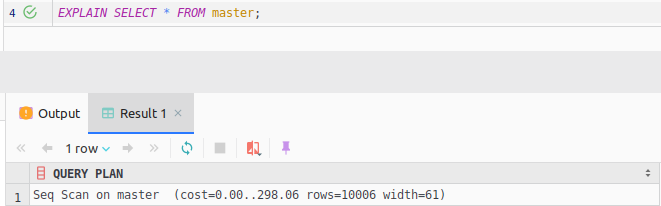
Порядок звертання до таблиці з використанням фільтру по колонці, на яку додано індекс (пошук відбувається за допомогою створеного індексу):



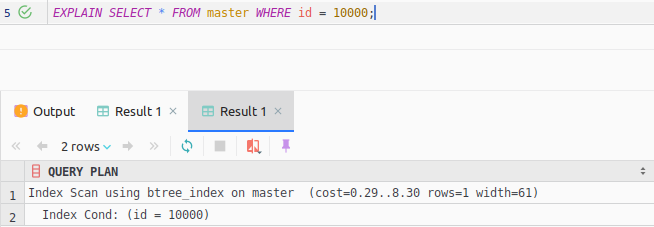
**Btree індекс:**

*CREATE INDEX* btree\_index *ON* master *using* btree(id);

Порядок звертання до таблиці без використання фільтру по колонці, на яку додано індекс (створений індекс не використовується):



Порядок звертання до таблиці з використанням фільтру по колонці, на яку додано індекс (пошук відбувається за допомогою створеного індексу):



**Тригер**

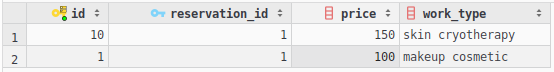
**AFTER INSERT**

Якщо для до броні додається нова процедура, то всі процедури в цій броні отримують знижку 10%.

Код:

*CREATE OR REPLACE FUNCTION after\_insert\_procedure*()  
*RETURNS TRIGGER LANGUAGE* PLPGSQL *AS* $$  
*DECLARE*  
other\_procedures *CURSOR IS SELECT \* FROM procedure WHERE* reservation\_id = *NEW*.reservation\_id;  
*BEGIN*  
 *FOR* p *IN* other\_procedures *LOOP*  
 *UPDATE procedure SET* price = price \* 0.9 *WHERE* id = p.id;  
 *END LOOP*;  
 *RETURN NEW*;  
*END*;  
$$;

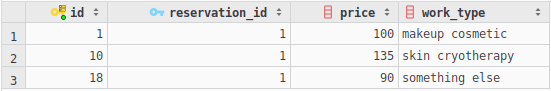
Приклади результатів:

Процедури для броні з id=1;

Додамо нову процедуру:

*INSERT INTO procedure*(reservation\_id, price, work\_type)  
*VALUES* (1, 100, 'something else');

Процедури після цього:



**AFTER UPDATE**

Якщо оновлюється бронь і їй у відповідність встановлюється ВІП-клієнт, то майстер, до якого записаний цей клієнт отримує +10% до рейтингу.

Код:

*CREATE OR REPLACE FUNCTION after\_update\_reservation*()  
*RETURNS TRIGGER LANGUAGE* PLPGSQL *AS* $$  
*DECLARE*  
*BEGIN*  
 *IF true IN* (*SELECT* is\_vip *FROM* customer *WHERE* id = *NEW*.customer\_id) *THEN*  
 *UPDATE* master *SET* rating = rating \* 1.1 *WHERE* id = *NEW*.master\_id;  
 *END IF*;  
 *RETURN NEW*;  
*END*;  
$$;

Приклади результатів:

*INSERT INTO* master(id, *name*, experience, rating) *VALUES* (1337, 'Sample Master', 15, 4);  
*INSERT INTO* customer(id, *name*, is\_vip) *VALUES* (42, 'Simple Customer', *false*);  
*INSERT INTO* customer(id, *name*, is\_vip) *VALUES* (1337, 'Vip Customer', *true*);  
*INSERT INTO* reservation(master\_id, customer\_id) *VALUES* (1337, 42);  
*UPDATE* reservation *SET* customer\_id = 1337 *WHERE* master\_id = 1337;  
*SELECT \* FROM* master *WHERE name* = 'Sample Master';



Початковий рейтинг майстра був 4. В його броні змінився клієнт (новий клієнт ВІП). Майстер отримав збільшення рейтингу до 4.4.

**Дослідження рівнів ізоляції**

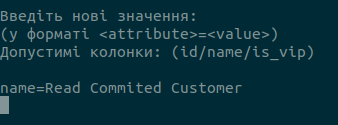
Для перевірки аномалій буде використовуватися розроблений програмний додаток, запущений у двох екземплярах паралельно.

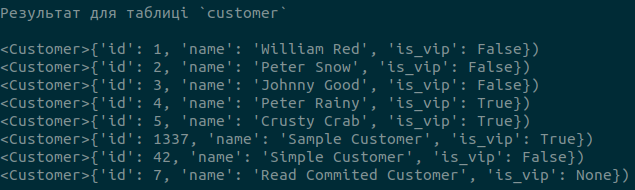
1. **READ COMMITTED**

Перевіримо наявність аномалії “dirty read”, коли транзакція читає дані, які ще не були закомічені паралельною транзакцією.

Транзакція №1

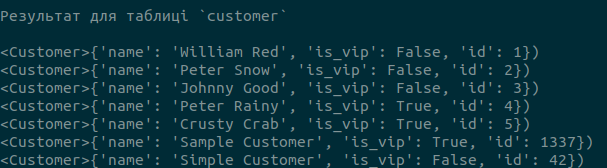
Створює нового Клієнта і перевіряє його наявність для свого запиту SELECT





Транзакція №2

Отримує список всіх Клієнтів.



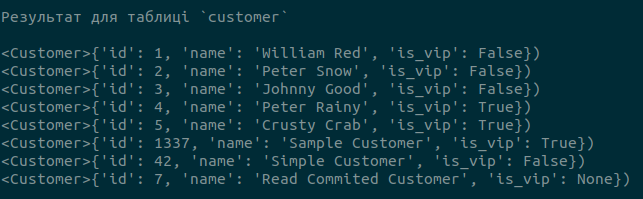
Транзакція №1

Робить коміт.



Транзакція №2

Отримує список всіх Клієнтів.



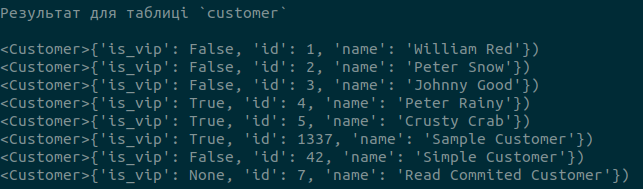
Як бачимо Транзація №2 не бачила зміни, внесені до таблиці Транзакцією №1, до ти пір поки остання не закомітила свої зміни. Отже, було доведено, що рівень ізоляції READ COMMITTED захищає від аномалії “брудного читання”.

2. **REPEATABLE READ**

Перевіримо наявність аномалії “nonrepeatable read”, коли транзакція повторно зчитує дані і вони виявляються модифіковані комітом паралельної транзакції.

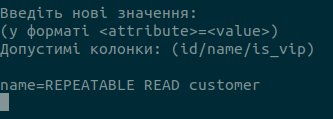
Транзакція №1

Отримує список усіх Клієнтів.

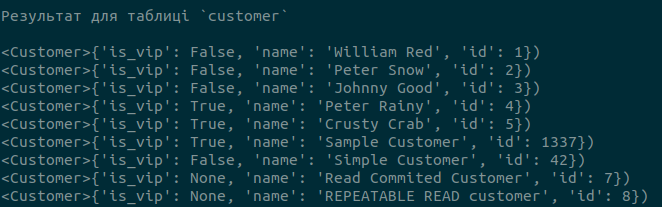


Транзакція №2

Створює нового Клієнта, комітить зміни і перевіряє його наявність в списку усіх клієнтів.

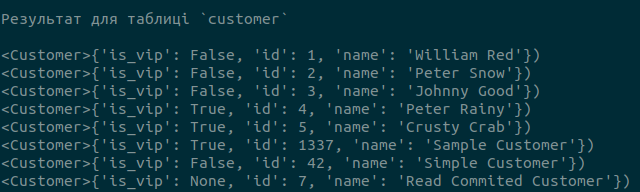






Транзакція №1

Заново отримує список усіх клієнтів.



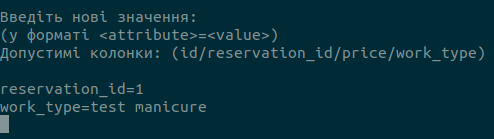
Як бачимо, Транзакція №1 так і не побачила закомічені зміни Транзакції №2, так як перша з них зчитувала дані з таблиці ще до коміту. Отже, було доведено, що рівень ізоляції REPEATBLE READ захищає від аномалії “nonrepeatable read”.

3. **SERIALIZABLE**

Перевіримо наявність аномалії “serialization anomaly”, коли дві паралельні транзакції хочуть закомітити свої результати і при цьому є різниця, в якому порядку виконувати команди, виконані кожною з транзакцій.

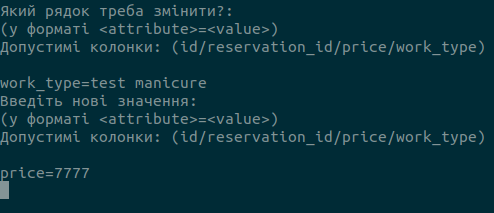
Транзакція №1

Створює нову Процедуру “test manicure”.



Транзакція №2

Змінює ціну для всіх процедур “test manicure”.



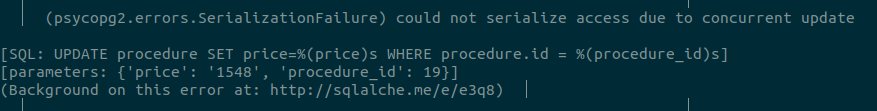
Транзакція №1

Намагається закомітити зміни



Транзакція №2

Намагається закомітити зміни



Як бачимо Транзація №2 не змогла закомітити зміни бо порядок виконання команд з обох транзакцій змінює вихідний результат. Отже, було доведено, що рівень ізоляції SERIALIZABLE захищає від “serialization anomaly”.