НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування та спеціалізованих комп'ютерних систем

Лабораторна робота №3

з дисципліни

«Бази даних та засоби управління»

Тема «Засоби оптимізації роботи СУБД PostgreSQL»

Виконав: студент III курсу
ФПМ групи КВ-04
Томчик О.

Київ – 2023

Завдання

Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

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 у PostgreSQL.

Структура бази даних з лабораторної роботи №1

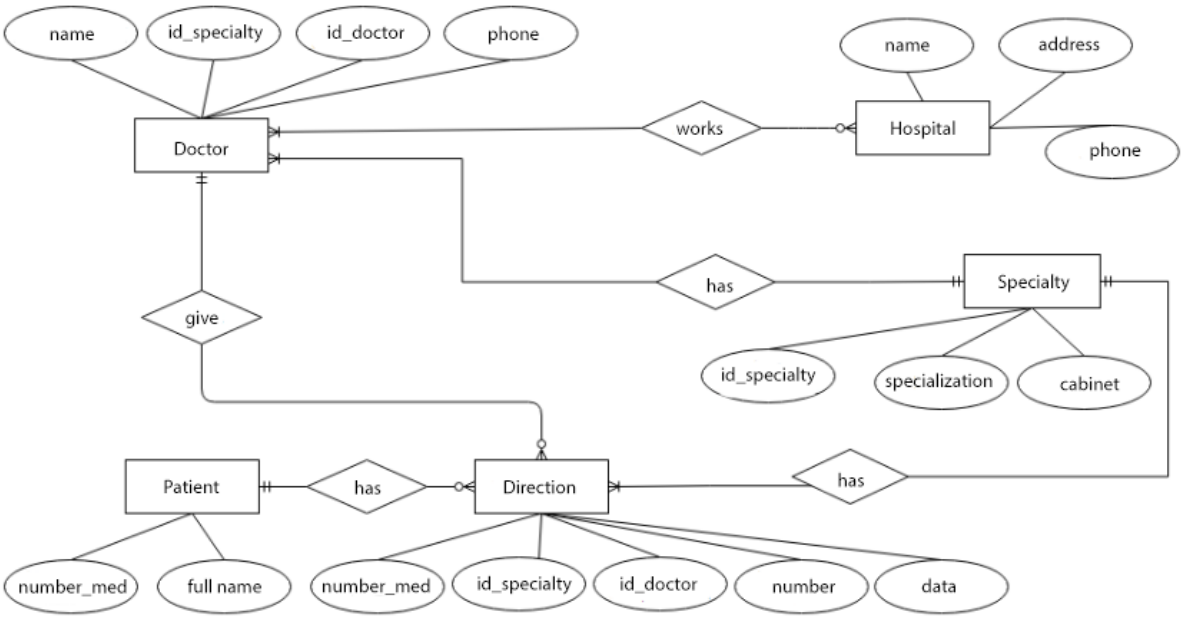


Рис.1. ER-діаграма, побудована за нотацією "Пташиної лапки"

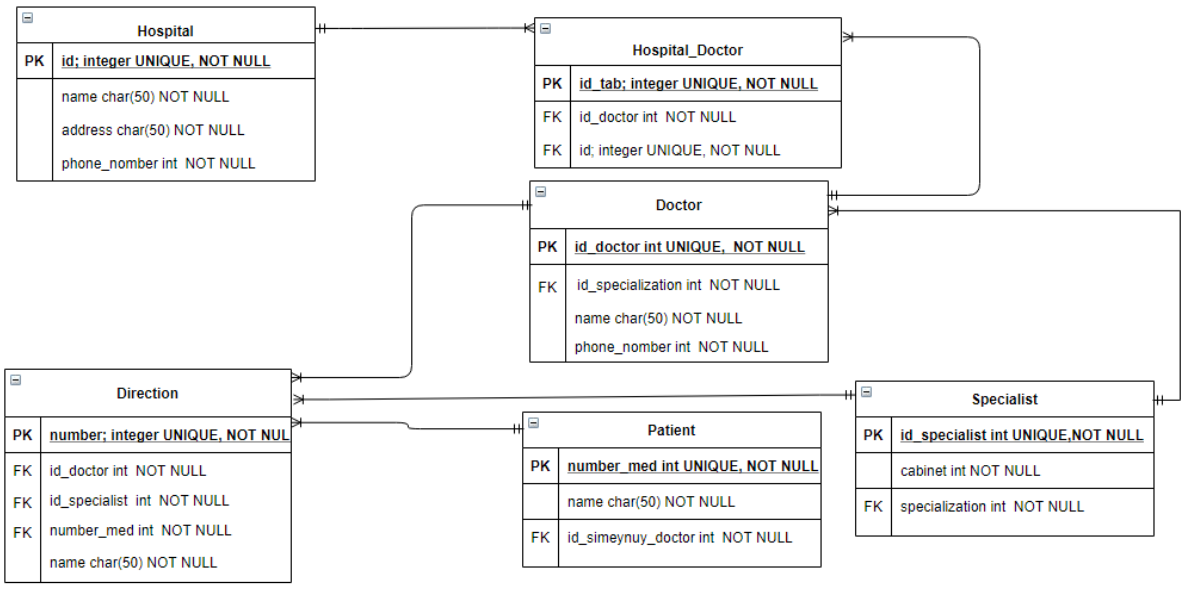


Рис.2. Схема бази даних у графічному вигляді

Обрана предметна галузь передбачає отримання і обробку потрібної інформації щодо лікарні, лікарів, що в ній працюють, пацієнтів, які в ній обслуговуються, спеціальностей лікарів та направлень, які отримують пацієнти.

Виконання
Завдання 1

Була використана бібліотека *sqlAlchemy,* яка перетворює функції, що реалізують запити до об'єктної бази даних.

Класи-сутності ORM в модулі *model.py:*

```
class Direction(base):
    __tablename__ = 'Direction'
    number = Column(Integer, primary_key=True, nullable=False)
    number_med = Column(Integer, ForeignKey('Patient.number_med'),
nullable=False)
    id_doctor = Column(Integer, ForeignKey('Doctor.id_doctor'),
nullable=False)
    id_specialist = Column(Integer,
ForeignKey('Specialist.id_specialist'), nullable=False)
    data = Column(String(50), nullable=False)
    Doctor = relationship('Doctor')
    Specialist = relationship('Specialist')
    Patient = relationship('Patient')

    def __init__(self, number_med, id_doctor, id_specialist, data,
number=-1):
        self.number_med = number_med
        self.id_doctor = id_doctor
        self.id_specialist = id_specialist
        self.data = data
        if id != -1:
            self.number = number

    format_str = '{:^8}{:^12}{:^12}{:^12}{:^20}'

    def __repr__(self):
        return self.format_str.format(self.number, self.number_med,
self.id_doctor, self.id_specialist, self.data)

    def __attributes_print__(self):
        return self.format_str.format('number', 'number_med',
'id_doctor', 'id_specialist', 'data')


class Doctor(base):
    __tablename__ = 'Doctor'
```

```
    id_doctor = Column(Integer, primary_key=True, nullable=False)
    id_specialist = Column(Integer,
ForeignKey('Specialist.id_specialist'), nullable=False)
    name_doc = Column(String(50), nullable=False)
    phone_num = Column(Integer, nullable=False)
    Specialist = relationship('Specialist')

    def __init__(self, id_specialist, name_doc, phone_num,
id_doctor=-1):
        self.phone_num = phone_num
        self.id_specialist = id_specialist
        self.name_doc = name_doc
        if id_doctor != -1:
            self.id_doctor = id_doctor

    format_str = '{:^10}{:^12}{:^12}{:^30}'

    def __repr__(self):
        return self.format_str.format(self.id_doctor,
self.id_specialist, self.name_doc, self.phone_num)

    def __attributes_print__(self):
        return self.format_str.format('id_doctor', 'id_specialist',
'name_doc', 'phone_num')


class Hospital(base):
    __tablename__ = 'Hospital'
    id = Column(Integer, primary_key=True, nullable=False)
    name = Column(String(50), nullable=False)
    address = Column(String(50), nullable=False)
    phone = Column(Integer, nullable=False)

    def __init__(self, name, address, phone, id=-1):
        self.phone = phone
        self.name = name
        self.address = address
        if id != -1:
            self.id = id

    format_str = '{:^8}{:^30}{:^40}{:^20}'

    def __repr__(self):
        return self.format_str.format(self.id, self.name,
self.address, self.phone)

    def __attributes_print__(self):
        return self.format_str.format('id', 'name', 'address',
'phone')
```

```
class Hospital_Doctor(base):
    __tablename__ = 'Hospital_Doctor'
    id_tab = Column(Integer, primary_key=True, nullable=False)
    id = Column(Integer, ForeignKey('Hospital.id'), nullable=False)
    id_doctor = Column(Integer, ForeignKey('Doctor.id_doctor'),
nullable=False)
    Hospital = relationship('Hospital')
    Doctor = relationship('Doctor')

    def __init__(self, id, id_doctor, id_tab=-1):
        self.id_doctor = id_doctor
        self.id = id
        if id_tab != -1:
            self.id_tab = id_tab

    format_str = '{:^8}{:^12}{:^12}'

    def __repr__(self):
        return self.format_str.format(self.id_tab, self.id,
self.id_doctor)

    def __attributes_print__(self):
        return self.format_str.format('id_tab', 'id', 'id_doctor')


class Patient(base):
    __tablename__ = 'Patient'
    number_med = Column(Integer, primary_key=True, nullable=False)
    name = Column(String(50), nullable=False)

    def __init__(self, name, number_med=-1):
        self.name = name
        if number_med != -1:
            self.number_med = number_med

    format_str = '{:^10}{:^30}'

    def __repr__(self):
        return self.format_str.format(self.number_med, self.name)

    def __attributes_print__(self):
        return self.format_str.format('number_med', 'name')


class Specialist(base):
    __tablename__ = 'Specialist'
    id_specialist = Column(Integer, primary_key=True,
nullable=False)
    cabinet = Column(Integer, nullable=False)
    specialization = Column(String(50), nullable=False)
```

```
    def __init__(self, cabinet, specialization, phone,
id_specialist=-1):
        self.phone = phone
        self.name = name
        self.specialization = specialization
        if id_specialist != -1:
            self.id_specialist = id_specialist

    format_str = '{:^12}{:^12}{:^40}'

    def __repr__(self):
        return self.format_str.format(self.id_specialist,
self.cabinet, self.specialization)

    def __attributes_print__(self):
        return self.format_str.format('id_specialist', 'cabinet',
'specialization')
```

Запити засобами SQLAlchemy по роботі з об’єктами

Початковий стан:

```
1. Insert data in table
2. Edit data in table
3. Delete data from table
4. Print rows
5. Generate random data
6. Search from tables
0. Exit
    Select option 0-6: 4
Select table, from 0-6
1. Direction
2. Doctor
3. Hospital
4. Hospital_Doctor
5. Patient
6. Specialist
0. Return to main menu
2
Input quantity of rows to print: 100
id_doctor id_specialist  name_doc           phone_num
    1          2        sGY7C4vn             92300
    2          6        ribbAify             48103
    3          5        acN36FvU             50875
    4          2        6xjkJDhi             23943
    5          5        T9nY5Set             99226
    6          5        KBfnqc9L             95563
    7          4        2obGhMBk             42495
    8          6        21CErKTn             52323
    9          2        EmOaj6Pf             23429
    10         6        7a78uO7u             60591
```

Вставка екземплярів класів-сутностей:

```
1. Insert data in table
2. Edit data in table
3. Delete data from table
4. Print rows
5. Generate random data
6. Search from tables
0. Exit
    Select option 0-6: 1
Select the table to insert data, from 0-6
1. Direction
2. Doctor
3. Hospital
4. Hospital_Doctor
5. Patient
6. Specialist
0. Return to main menu
2
Insert value seperated by comma
Input: id_specialist->integer, name_doc->char[50], phone_num->integer
4, Ivan,1234123
Inserted successfully
```

id_doctor	id_specialist	name_doc	phone_num
1	2	sGY7C4vn	92300
2	6	ribbAify	48103
3	5	acN36FvU	50875
4	2	6xjkJDhi	23943
5	5	T9nY5Set	99226
6	5	KBfnqc9L	95563
7	4	2obGhMBk	42495
8	6	21CErKTn	52323
10	6	7a78uO7u	60591
11	4	Ivan	1234123

Видалення екземплярів класів-сутностей:

```
1. Insert data in table
2. Edit data in table
3. Delete data from table
4. Print rows
5. Generate random data
6. Search from tables
0. Exit
    Select option 0-6: 3
Select table, from 0-6
1. Direction
2. Doctor
3. Hospital
4. Hospital_Doctor
5. Patient
6. Specialist
0. Return to main menu
2
Enter id of row you want to delete
'p' -> print rows
'r' -> return to menu
9
Deleted successfully
```

id_doctor	id_specialist	name_doc	phone_num
1	2	sGY7C4vn	92300
2	6	ribbAify	48103
3	5	acN36FvU	50875
4	2	6xjkJDhi	23943
5	5	T9nY5Set	99226
6	5	KBfnqc9L	95563
7	4	2obGhMBk	42495
8	6	21CErKTn	52323
10	6	7a78uO7u	60591

Редагування екземплярів класів-сутностей:

```
1. Insert data in table
2. Edit data in table
3. Delete data from table
4. Print rows
5. Generate random data
6. Search from tables
0. Exit
    Select option 0-6: 2
Select table, from 0-6
1. Direction
2. Doctor
3. Hospital
4. Hospital_Doctor
5. Patient
6. Specialist
0. Return to main menu
2
Enter id of row you want to change
'p' -> print rows
'r' -> return to menu
11
id_doctor id_specialist  name_doc            phone_num
    11         4          Ivan                1234123
If you don't want to change column -> write as it was
Insert value seperated by comma
Input: id_specialist->integer, name_doc->char[50], phone_num->integer
5, Ivan Petrovich, 432134
Updated successfully
```

id_doctor	id_specialist	name_doc	phone_num
1	2	sGY7C4vn	92300
2	6	ribbAify	48103
3	5	acN36FvU	50875
4	2	6xjkJDhi	23943
5	5	T9nY5Set	99226
6	5	KBfnqc9L	95563
7	4	2obGhMBk	42495
8	6	21CErKTn	52323
10	6	7a78uO7u	60591
11	5	Ivan Petrovich	432134

Частина 4

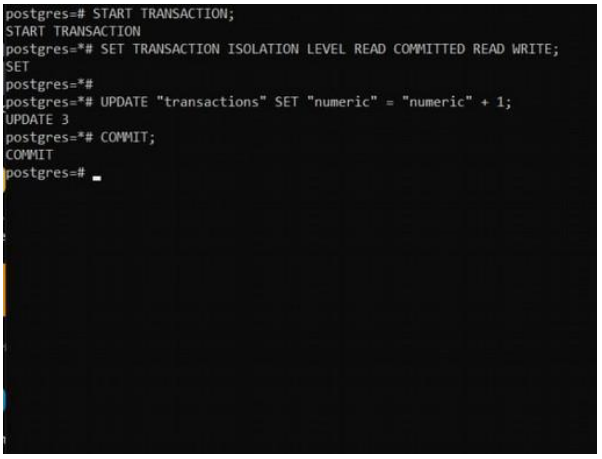
Щоб проаналізувати на прикладах використання рівнів ізоляції транзакцій READ COMMITTED, REPEATABLE READ та SERIALIZABLE та продемонструвати феномени, які виникають, і способи їх уникнення завдяки встановленню відповідного рівня ізоляції транзакцій, була створена окрема таблиця:

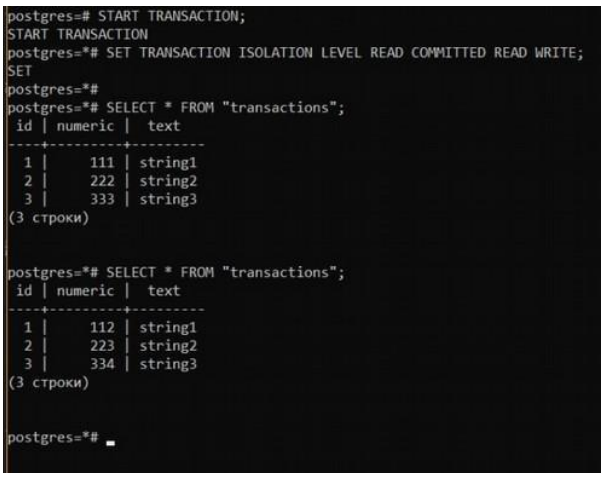
```
DROP TABLE IF EXISTS "transactions";
CREATE TABLE "transactions"
(
"id" bigserial PRIMARY KEY,
"numeric" bigint,
"text" text
);

INSERT INTO "transactions"("numeric", "text") VALUES (111,
'string1'), (222, 'string2'), (333, 'string3');
```

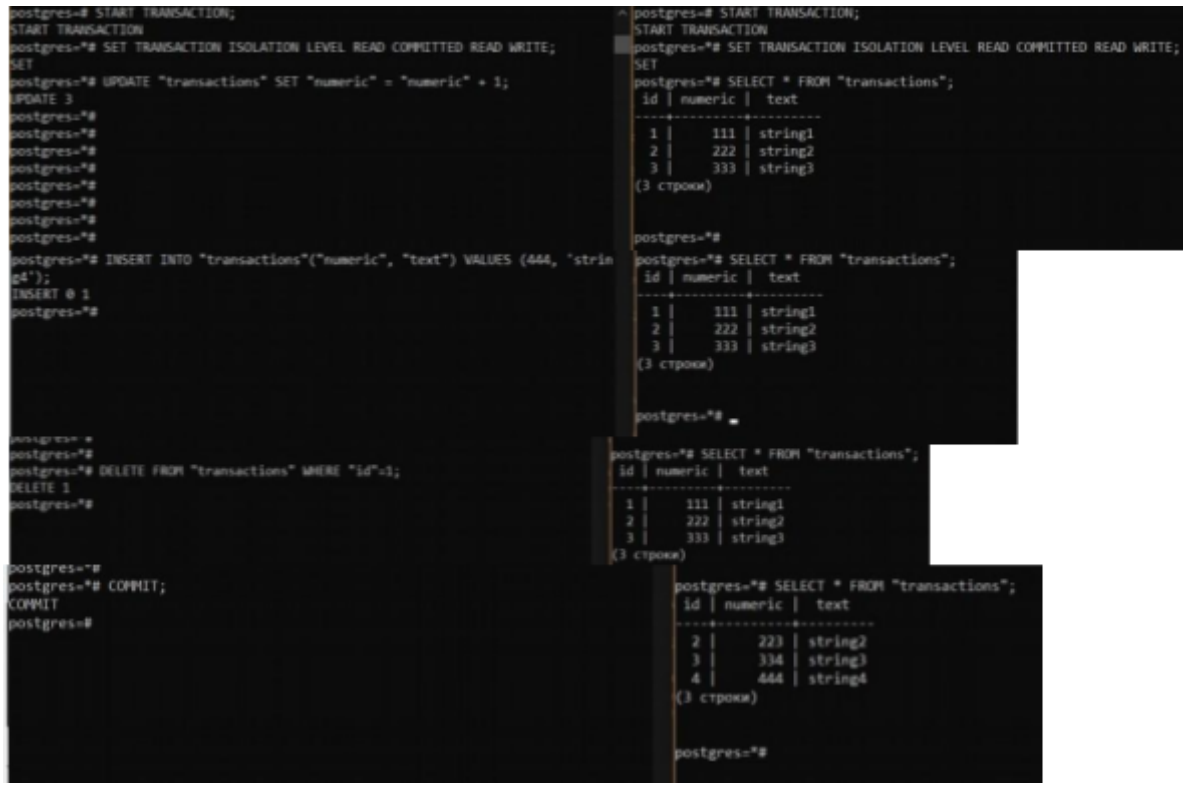
READ COMMITTED:

На цьому рівні ізоляції одна транзакція не бачить змін у базі даних, викликаних іншою доки та не завершить своє виконання (командою COMMIT або ROLLBACK).

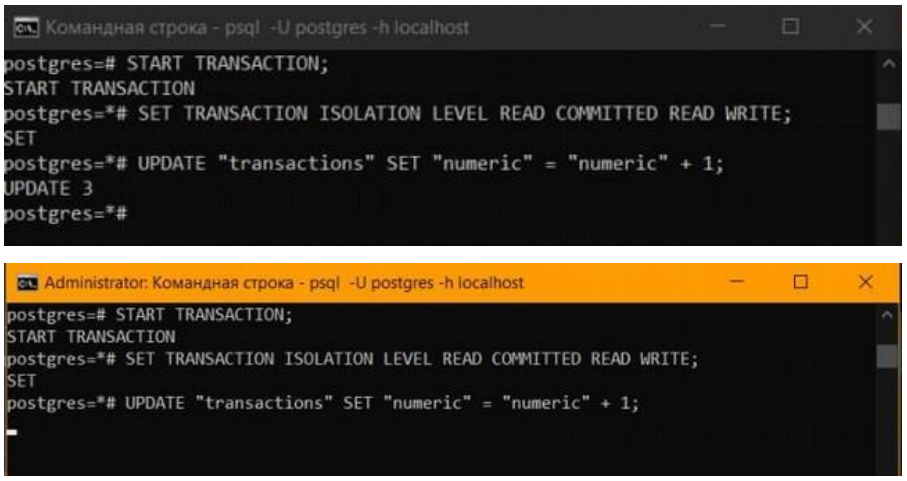




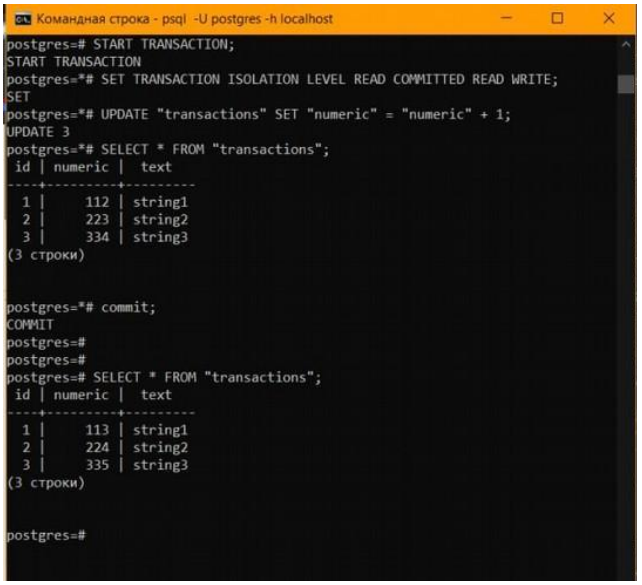
Дані після вставки та видалення так само будуть видні другій тільки після завершення першої.

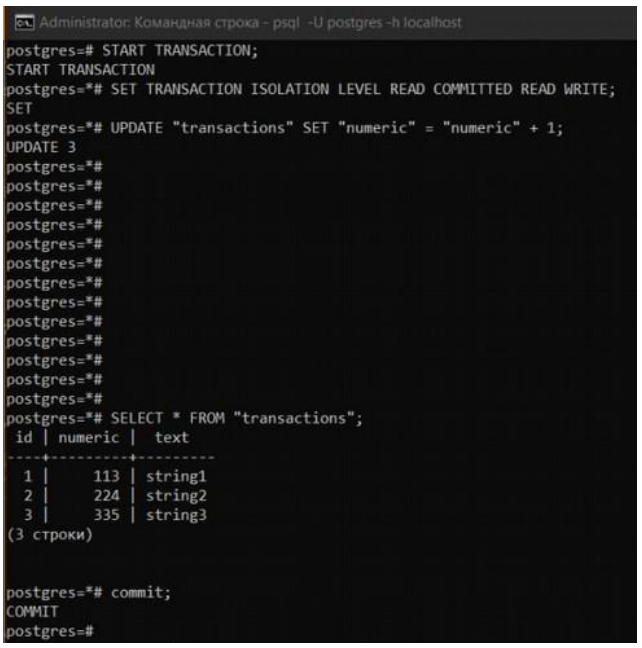


На цьому знімку також бачимо, що друга транзакція не може внести дані у базу, доки не завершилась попередня.



А тут бачимо, що після завершення першої, друга транзакція виконала запит, змінивши вже ті дані, що були закомічені першою транзакцією.





Коли T2 бачить дані T1 запитів UPDATE, DELETE виникає феномен повторного читаня, а коли бачить дані запиту INSERT – читання фантомів. Цей рівень ізоляції забезпечує захист від явища брудного читання.

REPEATABLE READ:

На цьому рівні ізоляції T2 не бачитиме змінені дані транзакцією T1, але також не зможе отримати доступ до тих самих даних.

Тут видно, що друга не бачить змін з першої:

```
postgres=# START TRANSACTION;SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
 READ WRITE;
START TRANSACTION
SET
postgres=*# UPDATE "transactions" SET "numeric" = "numeric" + 1;INSERT INTO
"transactions"("numeric", "text") VALUES (444, 'string4');DELETE FROM "trans
actions" WHERE "id"=1;
UPDATE 3
INSERT 0 1
DELETE 1
postgres=*#
```

```
postgres=# START TRANSACTION;SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
 READ WRITE;
START TRANSACTION
SET
postgres=*# SELECT * FROM "transactions";
 id | numeric |  text
----+---------+---------
  1 |     111 | string1
  2 |     222 | string2
  3 |     333 | string3
(3 строки)

postgres=*#
```

А тут, що отримуємо помилку при спробі доступу до тих самих даних:

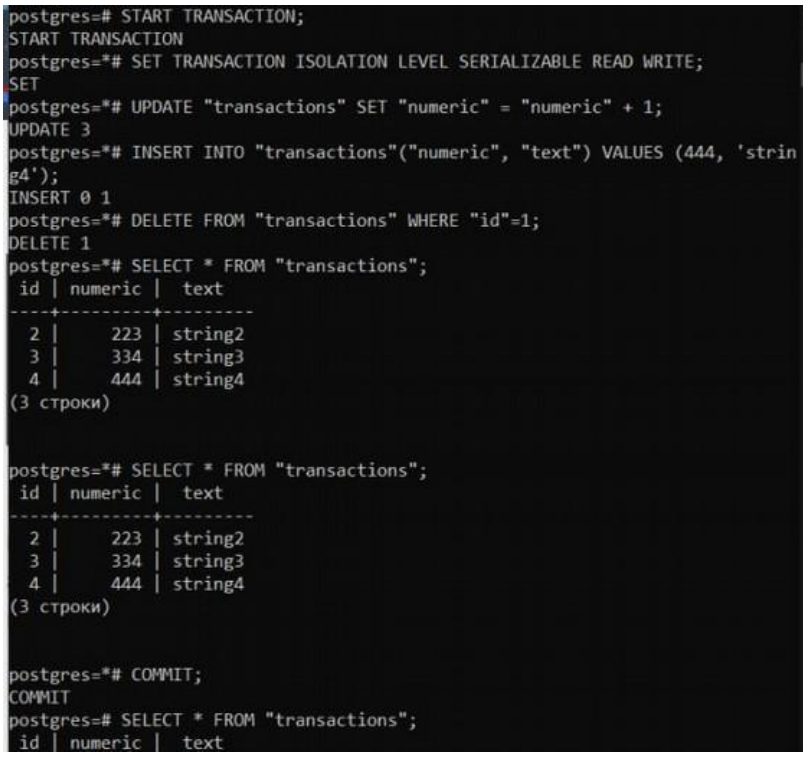
```
postgres=# START TRANSACTION;
START TRANSACTION
postgres=*# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ READ WRITE;
SET
postgres=*# UPDATE "transactions" SET "numeric" = "numeric" + 1;
UPDATE 3
postgres=*# COMMIT;
COMMIT
postgres=#
```

```
postgres=# START TRANSACTION;
START TRANSACTION
postgres=*# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ READ WRITE;
SET
postgres=*# UPDATE "transactions" SET "numeric" = "numeric" + 1;
ПОМИЛКА:  не вдалося серіалізувати доступ через паралельне оновлення
postgres=!# SELECT * FROM "transactions";
ПОМИЛКА:  поточна транзакція перервана, команди до кінця блока транзакції пропускаються
postgres=!# COMMIT;
ROLLBACK
postgres=# SELECT * FROM "transactions";
 id | numeric |  text
----+---------+---------
  1 |     112 | string1
  2 |     223 | string2
  3 |     334 | string3
(3 строки)
```

Бачимо, що не виникає читання фантомів та повторного читання, а також заборонено одночасний доступ до незбережених даних. Хоча класично цей рівень ізоляції призначений для попередження повторного читання.

SERIALIZABLE:

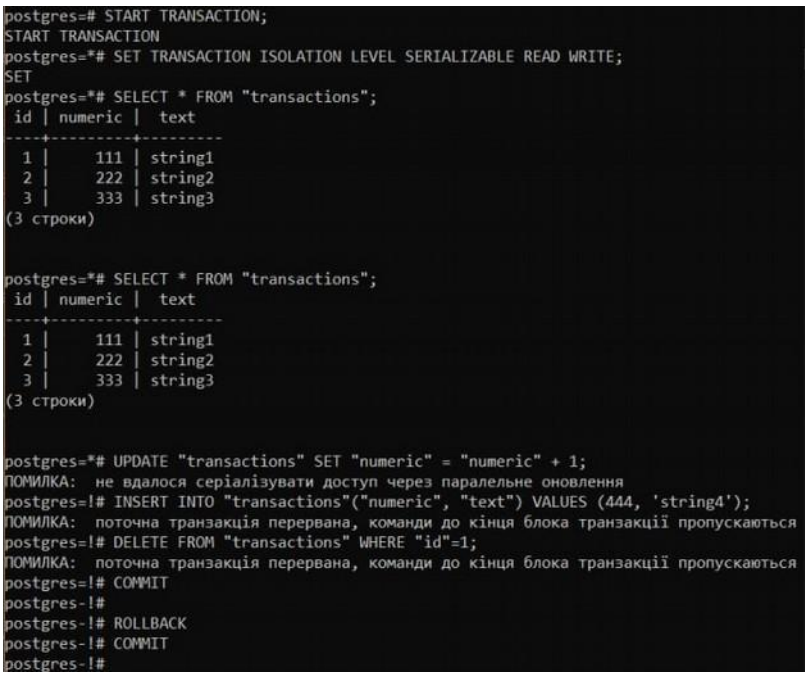
На цьому рівні транзакіції поводять себе так, ніби вони не знають одна про одну. Вони не можуть вплинути одна на одну і одночасний доступ строго заборонений.



```
postgres=# START TRANSACTION;
START TRANSACTION
postgres=*# SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ WRITE;
SET
postgres=*# UPDATE "transactions" SET "numeric" = "numeric" + 1;
UPDATE 3
postgres=*# INSERT INTO "transactions"("numeric", "text") VALUES (444, 'strin
g4');
INSERT 0 1
postgres=*# DELETE FROM "transactions" WHERE "id"=1;
DELETE 1
postgres=*# SELECT * FROM "transactions";
 id | numeric |  text
----+---------+--------
  2 |     223 | string2
  3 |     334 | string3
  4 |     444 | string4
(3 строки)


postgres=*# SELECT * FROM "transactions";
 id | numeric |  text
----+---------+--------
  2 |     223 | string2
  3 |     334 | string3
  4 |     444 | string4
(3 строки)


postgres=*# COMMIT;
COMMIT
postgres=# SELECT * FROM "transactions";
 id | numeric |  text
```



```
postgres=# START TRANSACTION;
START TRANSACTION
postgres=*# SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ WRITE;
SET
postgres=*# SELECT * FROM "transactions";
 id | numeric |  text
----+---------+--------
  1 |     111 | string1
  2 |     222 | string2
  3 |     333 | string3
(3 строки)


postgres=*# SELECT * FROM "transactions";
 id | numeric |  text
----+---------+--------
  1 |     111 | string1
  2 |     222 | string2
  3 |     333 | string3
(3 строки)


postgres=*# UPDATE "transactions" SET "numeric" = "numeric" + 1;
ПОМИЛКА:  не вдалося серіалізувати доступ через паралельне оновлення
postgres=!# INSERT INTO "transactions"("numeric", "text") VALUES (444, 'string4');
ПОМИЛКА:  поточна транзакція перервана, команди до кінця блока транзакції пропускаються
postgres=!# DELETE FROM "transactions" WHERE "id"=1;
ПОМИЛКА:  поточна транзакція перервана, команди до кінця блока транзакції пропускаються
postgres=!# COMMIT
postgres-!#
postgres-!# ROLLBACK
postgres-!# COMMIT
postgres-!#
```

У попередньому випадку вдалось “відкатити” другу тразакцію і це не вплинуло на подальшу можливість роботи в терміналі. На цьому ж рівні навіть після завершення першої не вдалося зробити ні COMMIT ні ROLLBACK для другої транзакції. Взагалі, в класичному представленні цей рівень призначений для недопущення явища читання фантомів. На цьому рівні ізоляції ми отримуємо максимальну узгодженість даних і можемо бути впевнені, що зайві дані не будуть зафіксовані.