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 і засоби управління**

*на тему: " Засоби оптимізації роботи СУБД PostgreSQL"*

Виконав:
студент 3 курсу
групи КВ-94
Поляруш Є. М.
Перевірив:
Петрашенко А. В.

Київ – 2021

# Постановка задачі

*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.

| *№ варіанта* | *Види індексів* | *Умови для тригера* |
|---|---|---|
| *20* | *GIN, BRIN* | *after insert, update* |

Посилання на репозиторій у GitHub з вихідним кодом програми та звітом: https://github.com/EugenePoliarush/Lab3

# Завдання №1

Дана предметна галузь передбачає роботу компанії певного житлового комплексу і її зв'язок з мешканцями

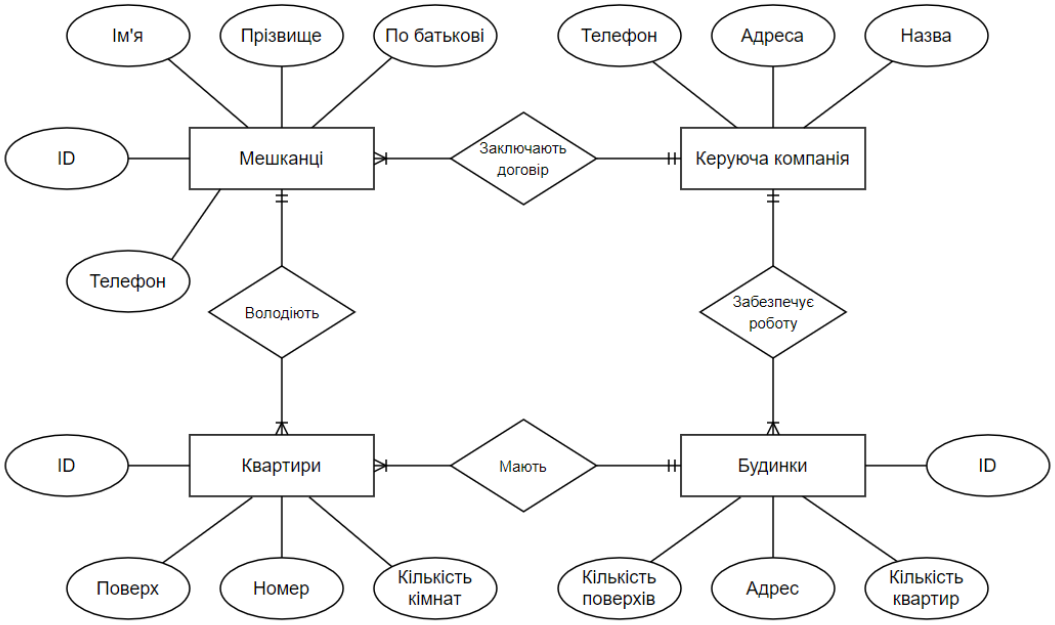



Рисунок 1. ER-діаграма, побудована за нотацією Чена

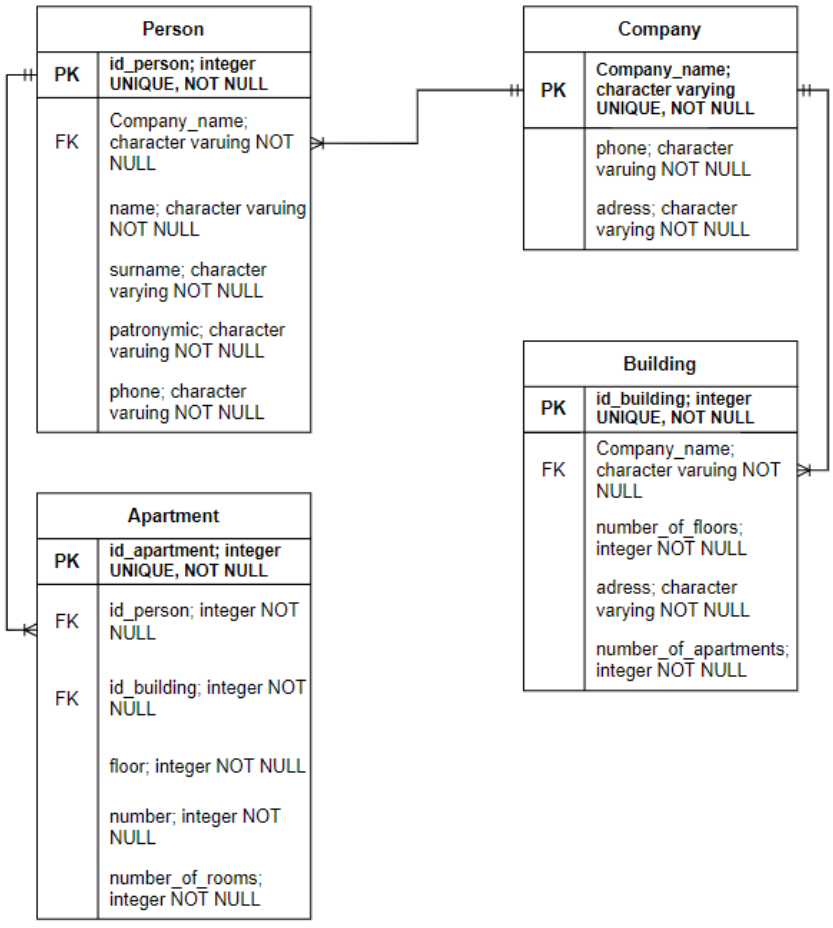



Рисунок 2. Схема бази даних

# Класи ORM у модулі Model

```
 class Apartment(Base):
    __tablename__ = 'Apartment'
    id_apartment = Column(Integer, primary_key=True)
    floor = Column(Integer)
    number = Column(Integer)
    number_of_rooms = Column(Integer)
    id_person = Column(Integer, ForeignKey('Person.id_person'))
    id_building = Column(Integer, ForeignKey('Building.id_building'))

    def __init__(self, id_apartment, floor, number, number_of_rooms, id_person, id_building):
        self.id_apartment = id_apartment
        self.floor = floor
        self.number = number
        self.number_of_rooms = number_of_rooms
        self.id_person = id_person
        self.id_building = id_building

    def __repr__(self):
        return "{} {} {} {} {} {}".format(self.id_apartment, self.floor, self.number,
self.number_of_rooms, self.id_person, self.id_building)

class Building(Base):
    __tablename__ = 'Building'
    id_building = Column(Integer, primary_key=True)
    adress = Column(String)
    number_of_apartments = Column(Integer)
    number_of_floors = Column(Integer)
    Company_name = Column(String, ForeignKey('Company.Company_name'))

    def __init__(self, id_building, adress, number_of_apartments, number_of_floors,
Company_name):
        self.id_building = id_building
        self.adress = adress
        self.number_of_apartments = number_of_apartments
        self.number_of_floors = number_of_floors
        self.Company_name = Company_name

    def __repr__(self):
        return "{} {} {} {} {}".format(self.id_building, self.adress, self.number_of_apartments,
self.number_of_floors, self.Company_name)

class Company(Base):
    __tablename__ = 'Company'
    Company_name = Column(String, primary_key=True)
    adress = Column(String)
    phone = Column(String)
```

```
    def __init__(self, Company_name, adress, phone):
        self.Company_name = Company_name
        self.adress = adress
        self.phone = phone

    def __repr__(self):
        return "{} {} {}".format(self.Company_name, self.adress, self.phone)

class Person(Base):
    __tablename__ = 'Person'
    id_person = Column(Integer, primary_key=True)
    name = Column(String)
    surname = Column(String)
    patronymic = Column(String)
    phone = Column(String)
    Company_name = Column(String, ForeignKey('Company.Company_name'))

    def __init__(self, id_person, name, surname, patronymic, phone, Company_name):
        self.id_person = id_person
        self.name = name
        self.surname = surname
        self.patronymic = patronymic
        self.phone = phone
        self.Company_name = Company_name

    def __repr__(self):
        return "{} {} {} {} {} {}".format(self.id_person, self.name, self.surname, self.patronymic,
self.phone, self.Company_name)
```

## Результати роботи

### Початкова таблиця

Apartment table:

| id_apartment | floor | number | number_of_rooms | id_person | id_building |
|---|---|---|---|---|---|
| 1 | 3 | 1 | 2 | 3 | 1 |
| 2 | 2 | 4 | 1 | 2 | 2 |
| 3 | 2 | 5 | 3 | 2 | 2 |
| 4 | 5 | 1 | 3 | 3 | 1 |
| 5 | 8 | 4 | 2 | 2 | 1 |
| 6 | 2 | 7 | 1 | 3 | 2 |
| 7 | 6 | 3 | 1 | 1 | 1 |
| 8 | 4 | 7 | 1 | 3 | 1 |
| 9 | 1 | 2 | 2 | 3 | 1 |
| 10 | 6 | 8 | 1 | 2 | 3 |
| 11 | 3 | 8 | 1 | 2 | 1 |
| 12 | 5 | 3 | 1 | 3 | 3 |
| 13 | 2 | 2 | 2 | 4 | 4 |
| 14 | 3 | 3 | 3 | 2 | 2 |

Видалення запису

```
Choose number of table -> 1
id_apartment of row for deletion = 13

                Menu
        1 -> Show one table
        2 -> Show all table
        3 -> Insert data
        4 -> Delete data
        5 -> Update data
        6 -> Exit

Select from 1 to 6 -> 1

        1 -> Apartment
        2 -> Building
        3 -> Company
        4 -> Person

Choose number of table -> 1
Apartment table:
+--------------+-------+--------+-----------------+-----------+-------------+
| id_apartment | floor | number | number_of_rooms | id_person | id_building |
+--------------+-------+--------+-----------------+-----------+-------------+
|      1       |   3   |   1    |        2        |     3     |      1      |
|      2       |   2   |   4    |        1        |     2     |      2      |
|      3       |   2   |   5    |        3        |     2     |      2      |
|      4       |   5   |   1    |        3        |     3     |      1      |
|      5       |   8   |   4    |        2        |     2     |      1      |
|      6       |   2   |   7    |        1        |     3     |      2      |
|      7       |   6   |   3    |        1        |     1     |      1      |
|      8       |   4   |   7    |        1        |     3     |      1      |
|      9       |   1   |   2    |        2        |     3     |      1      |
|      10      |   6   |   8    |        1        |     2     |      3      |
|      11      |   3   |   8    |        1        |     2     |      1      |
|      12      |   5   |   3    |        1        |     3     |      3      |
|      14      |   3   |   3    |        3        |     2     |      2      |
+--------------+-------+--------+-----------------+-----------+-------------+
```

Вставка запису

```
"id_apartment"=13
"floor"=4
"number"=4
"number_of_rooms"=4
"id_person"=4
"id_building"=4

                Menu
        1 -> Show one table
        2 -> Show all table
        3 -> Insert data
        4 -> Delete data
        5 -> Update data
        6 -> Exit

Select from 1 to 6 -> 1

        1 -> Apartment
        2 -> Building
        3 -> Company
        4 -> Person

Choose number of table -> 1
Apartment table:
+--------------+-------+--------+-----------------+-----------+-------------+
| id_apartment | floor | number | number_of_rooms | id_person | id_building |
+--------------+-------+--------+-----------------+-----------+-------------+
|      1       |   3   |   1    |        2        |     3     |      1      |
|      2       |   2   |   4    |        1        |     2     |      2      |
|      3       |   2   |   5    |        3        |     2     |      2      |
|      4       |   5   |   1    |        3        |     3     |      1      |
|      5       |   8   |   4    |        2        |     2     |      1      |
|      6       |   2   |   7    |        1        |     3     |      2      |
|      7       |   6   |   3    |        1        |     1     |      1      |
|      8       |   4   |   7    |        1        |     3     |      1      |
|      9       |   1   |   2    |        2        |     3     |      1      |
|      10      |   6   |   8    |        1        |     2     |      3      |
|      11      |   3   |   8    |        1        |     2     |      1      |
|      12      |   5   |   3    |        1        |     3     |      3      |
|      13      |   4   |   4    |        4        |     4     |      4      |
|      14      |   3   |   3    |        3        |     2     |      2      |
+--------------+-------+--------+-----------------+-----------+-------------+
```

Редагування запису

```
"id_apartment"=12
"floor"=2
"number"=2
"number_of_rooms"=2
"id_person"=2
"id_building"=2

                Menu
        1 -> Show one table
        2 -> Show all table
        3 -> Insert data
        4 -> Delete data
        5 -> Update data
        6 -> Exit

Select from 1 to 6 -> 1

        1 -> Apartment
        2 -> Building
        3 -> Company
        4 -> Person

Choose number of table -> 1
Apartment table:
+--------------+-------+--------+-----------------+-----------+-------------+
| id_apartment | floor | number | number_of_rooms | id_person | id_building |
+--------------+-------+--------+-----------------+-----------+-------------+
|      1       |   3   |   1    |        2        |     3     |      1      |
|      2       |   2   |   4    |        1        |     2     |      2      |
|      3       |   2   |   5    |        3        |     2     |      2      |
|      4       |   5   |   1    |        3        |     3     |      1      |
|      5       |   8   |   4    |        2        |     2     |      1      |
|      6       |   2   |   7    |        1        |     3     |      2      |
|      7       |   6   |   3    |        1        |     1     |      1      |
|      8       |   4   |   7    |        1        |     3     |      1      |
|      9       |   1   |   2    |        2        |     3     |      1      |
|      10      |   6   |   8    |        1        |     2     |      3      |
|      11      |   3   |   8    |        1        |     2     |      1      |
|      12      |   2   |   2    |        2        |     2     |      2      |
|      13      |   4   |   4    |        4        |     4     |      4      |
|      14      |   3   |   3    |        3        |     2     |      2      |
+--------------+-------+--------+-----------------+-----------+-------------+
```

# Завдання №2

Для тестування індексів було створено окремі таблиці у базі даних з 1000000записів.

GIN

GIN призначений для обробки випадків, коли елементи, що підлягають індексації, є складеними значеннями (наприклад - реченнями), а запити, які обробляються індексом, мають шукати значення елементів, які з'являються в складених елементах (повторювані частини слів або речень). Індекс GIN зберігає набір пар (ключ, список появи ключа), де список появи — це набір ідентифікаторіврядків, у яких міститься ключ. Один і той самий ідентифікатор рядка може знаходитись у кількох списках, оскільки елемент може містити більше одного ключа. Кожне значення ключа зберігається лише один раз, тому індекс GIN дуже швидкий для випадків, коли один і той же ключ з’являється багато разів. Цей індекс може взаємодіяти тільки з полем типу tsvector.

Створення таблиці БД:

```
CREATE TABLE "test_table"("id" bigserial PRIMARY KEY, "some_text" text, "vector" tsvector);

INSERT INTO "test_table"("some_text") SELECT substr(characters, (random() * length(characters) + 1)::integer, 10) FROM (VALUES('qwertyuiopasdfghjklzxcvbnm')) as symbols(characters), generate_series(1, 1000000) as q;

UPDATE "test_table" set "vector" = to_tsvector("some_text");
```

Запити для тестування:

```
SELECT SUM("id") FROM "test_table" WHERE ("vector" @@ to_tsquery('asd')) OR ("vector" @@ to_tsquery('nm'));
SELECT MIN("id"), MAX("id") FROM "test_table" WHERE ("vector" @@ to_tsquery('nm')) GROUP BY "id" % 2;
SELECT COUNT(*) FROM "test_table" WHERE "id" % 2 = 0;
SELECT COUNT(*) FROM "test_table" WHERE ("vector" @@ to_tsquery('nm'));
```

Створення індексу:

```
CREATE INDEX "gin_index" ON "test_table" USING gin("vector");
```

Результати без індексування

```
Database1=# \timing on
Таймер ув?мкнено.
Database1=# SELECT SUM("id") FROM "test_table" WHERE ("vector" @@ to_tsquery('asd')) OR ("vector" @@ to_tsquery('nm'));
     sum
-------------
 19226053355
(1 Ё фюъ)


Час: 2043,513 мс (00:02,044)
Database1=# SELECT MIN("id"), MAX("id") FROM "test_table" WHERE ("vector" @@ to_tsquery('nm')) GROUP BY "id" % 2;
 min |  max
-----+--------
  14 | 999992
  25 | 999957
(2 Ё фъш)


Час: 1181,572 мс (00:01,182)
Database1=# SELECT COUNT(*) FROM "test_table" WHERE "id" % 2 = 0;
 count
--------
 500000
(1 Ё фюъ)


Час: 596,714 мс
Database1=# SELECT COUNT(*) FROM "test_table" WHERE ("vector" @@ to_tsquery('nm'));
 count
-------
 38665
(1 Ё фюъ)


Час: 1161,344 мс (00:01,161)
Database1=#
```

Результати з індексуванням

```
Database1=# SELECT SUM("id") FROM "test_table" WHERE ("vector" @@ to_tsquery('asd')) OR ("vector" @@ to_tsquery('nm'));
     sum
-------------
 19185504312
(1 Ё фюъ)


Час: 565,239 мс
Database1=# SELECT MIN("id"), MAX("id") FROM "test_table" WHERE ("vector" @@ to_tsquery('nm')) GROUP BY "id" % 2;
 min |  max
-----+--------
  26 | 999966
 173 | 999985
(2 Ё фъш)


Час: 586,826 мс
Database1=# SELECT COUNT(*) FROM "test_table" WHERE "id" % 2 = 0;
 count
--------
 500000
(1 Ё фюъ)


Час: 598,353 мс
Database1=# SELECT COUNT(*) FROM "test_table" WHERE ("vector" @@ to_tsquery('nm'));
 count
-------
 38332
(1 Ё фюъ)


Час: 647,675 мс
Database1=#
```

Порівнявши результати, можна побачити, що пошук з індексацією значно швидший, за пошук без індексації.  Крім 3 запита, оскільки на нього вона не впливає. Пошук проходить швидше через головну особливість індексування GIN: кожне значення шуканого ключа зберігається одинраз і запит іде не по всій таблиці, а лише по тим даним, що містяться у списку появи цього ключа.

BRIN

Ідея BRIN не в тому, щоб швидко знайти потрібні строки, а в тому, щоб уникнути перегляду відомостей не потрібних. Це завжди неточний індекс: він взагалі не містить ID- табличних рядків.

Спрощено кажучи, BRIN добре працює для тих стовпців, значення в яких співвідносяться з їх фізичним розташуванням в таблиці. Іншими словами, якщо запит без пропозицій ORDER BY видає значення стовпця практично в порядку зростання або зменшення.

Створення таблиці БД:

```
CREATE TABLE "test_table"("id" bigserial PRIMARY KEY, "some_text" text, "vector" tsvector);

INSERT INTO "test_table"("some_text") SELECT substr(characters, (random() * length(characters) + 1)::integer, 10) FROM (VALUES('qwertyuiopasdfghjklzxcvbnm')) as symbols(characters), generate_series(1, 1000000) as q;

UPDATE "test_table" set "vector" = to_tsvector("some_text");
```

Запити для тестування:

```
SELECT SUM("id") FROM "test_table" WHERE ("some_text" @@ to_tsquery('asd')) OR ("vector" @@ to_tsquery('nm'));
SELECT MIN("id"), MAX("id") FROM "test_table" WHERE ("some_text" @@ to_tsquery('nm')) GROUP BY "id" % 2;
SELECT COUNT(*) FROM "test_table" WHERE "id" % 2 = 0;
SELECT COUNT(*) FROM "test_table" WHERE ("some_text" @@ to_tsquery('nm'));
```

Створення індексу:

```
DROP TABLE IF EXISTS "test_table";
CREATE INDEX "brin_index" ON "test_table" USING brin("some_text");
```

Результати без індексування

```
Database1=# SELECT SUM("id") FROM "test_table" WHERE ("some_text" @@ to_tsquery('asd')) OR ("vector" @@ to_tsquery('nm'));
     sum
-------------
 19197312622
(1 Ё фюъ)


Час: 2509,294 мс (00:02,509)
Database1=# SELECT MIN("id"), MAX("id") FROM "test_table" WHERE ("some_text" @@ to_tsquery('nm')) GROUP BY "id" % 2;
 min |  max
-----+--------
  12 | 999986
 207 | 999995
(2 Ё фъш)


Час: 1725,349 мс (00:01,725)
Database1=# SELECT COUNT(*) FROM "test_table" WHERE "id" % 2 = 0;
 count
--------
 500000
(1 Ё фюъ)


Час: 519,452 мс
Database1=# SELECT COUNT(*) FROM "test_table" WHERE ("some_text" @@ to_tsquery('nm'));
 count
-------
 38443
(1 Ё фюъ)


Час: 1693,522 мс (00:01,694)
```

Результати з індексуванням

```
Database1=# SELECT SUM("id") FROM "test_table" WHERE ("some_text" @@ to_tsquery('asd')) OR ("vector" @@ to_tsquery('nm'));
     sum
-------------
 19197312622
(1 Ё фюъ)


Час: 2387,963 мс (00:02,388)
Database1=# SELECT MIN("id"), MAX("id") FROM "test_table" WHERE ("some_text" @@ to_tsquery('nm')) GROUP BY "id" % 2;
 min |  max
-----+--------
  12 | 999986
 207 | 999995
(2 Ё фъш)


Час: 1697,097 мс (00:01,697)
Database1=# SELECT COUNT(*) FROM "test_table" WHERE "id" % 2 = 0;
 count
--------
 500000
(1 Ё фюъ)


Час: 626,164 мс
Database1=# SELECT COUNT(*) FROM "test_table" WHERE ("some_text" @@ to_tsquery('nm'));
 count
-------
 38443
(1 Ё фюъ)


Час: 1678,810 мс (00:01,679)
Database1=#
```

Оскільки для даного типу індексування більш важливим є запобігання перегляду непотрібних даних, а не швидко знайти потрібні рядки, то і швидкість з індексуванням не є значно швидшою ніж без індексування. Але в усіх випадках крім 3 показані кращі результати.

# Завдання №3

Для тестування тригера було створено дві таблиці:

```
DROP TABLE IF EXISTS "test_table_1";
CREATE TABLE "test_table_1"("id" bigserial PRIMARY KEY, "Text" text);

DROP TABLE IF EXISTS "test_table_2";
CREATE TABLE "test_table_2"("id" bigserial PRIMARY KEY, "ID" bigint, "Text" text);
```

Початкові дані у таблицях:

```
INSERT INTO "test_table_1"("Text")
VALUES ('Text1'), ('Text2'), ('Text3'), ('Text4'), ('Text5'), ('Text6'), ('Text7'),
('Text8'), ('Text9'), ('Text10');
```

Команди, що ініціюють виконання тригера:

```
CREATE TRIGGER "after_insert_update_trigger" AFTER INSERT OR UPDATE ON
"test_table_1" FOR EACH ROW
EXECUTE procedure after_insert_update_func();
```

Текст тригера:

```
CREATE FUNCTION after_insert_update_func() RETURNS TRIGGER as $trigger$ DECLARE
CURSOR_LOG CURSOR FOR SELECT * FROM "test_table_2"; row_ "test_table_2"%ROWTYPE;

BEGIN
IF new."id" % 2 = 0 THEN
      RAISE NOTICE 'id is multiple on 2'; FOR row_ IN CURSOR_LOG LOOP
      UPDATE "test_table_2" SET "ID" = row_."id" WHERE "id" = row_."id";
      END LOOP; RETURN OLD;
ELSE
      RAISE NOTICE 'id is not multiple on 2';
      Insert into "test_table_2" ("ID","Text") values (new."id", 'new_Text');
      RETURN NEW;
END IF; END;
```

Початковий стан

| | id [PK] bigint | Text text |
|---|---|---|
| 1 | 1 | Text1 |
| 2 | 2 | Text2 |
| 3 | 3 | Text3 |
| 4 | 4 | Text4 |
| 5 | 5 | Text5 |
| 6 | 6 | Text6 |
| 7 | 7 | Text7 |
| 8 | 8 | Text8 |
| 9 | 9 | Text9 |
| 10 | 10 | Text10 |
| | | |

| | id [PK] | ID bigir | Text text |
|---|---|---|---|
| | | | |

Після вставки

INSERT INTO "test_table_1"("Text") VALUES ('New text');

| | id [PK] bigint | Text text |
|---|---|---|
| 1 | 1 | Text1 |
| 2 | 2 | Text2 |
| 3 | 3 | Text3 |
| 4 | 4 | Text4 |
| 5 | 5 | Text5 |
| 6 | 6 | Text6 |
| 7 | 7 | Text7 |
| 8 | 8 | Text8 |
| 9 | 9 | Text9 |
| 10 | 10 | Text10 |
| 11 | 11 | New text |
| | | |

| | id [PK] bigint | ID bigint | Text text |
|---|---|---|---|
| 1 | 1 | 11 | new_Text |
| | | | |

Повторна вставка(парне id)

INSERT INTO "test_table_1"("Text") VALUES ('New text');

| | id [PK] b | Text text |
|---|---|---|
| 1 | 1 | Text1 |
| 2 | 2 | Text2 |
| 3 | 3 | Text3 |
| 4 | 4 | Text4 |
| 5 | 5 | Text5 |
| 6 | 6 | Text6 |
| 7 | 7 | Text7 |
| 8 | 8 | Text8 |
| 9 | 9 | Text9 |
| 10 | 10 | Text10 |
| 11 | 11 | New text |
| 12 | 12 | New text |
| | | |

| | id [PK] bigint | ID bigint | Text text |
|---|---|---|---|
| 1 | 1 | 1 | new_Text |
| | | | |

За таблицями можна побачити як виконалась інша гілка коду у випадку коли id ділиться націло на 2.

Після редагування

update "test_table_1" set "Text" = 'NEW_TEXT' where "id"= 11;

| | id [PK] b | Text text |
|---|---|---|
| 1 | 1 | Text1 |
| 2 | 2 | Text2 |
| 3 | 3 | Text3 |
| 4 | 4 | Text4 |
| 5 | 5 | Text5 |
| 6 | 6 | Text6 |
| 7 | 7 | Text7 |
| 8 | 8 | Text8 |
| 9 | 9 | Text9 |
| 10 | 10 | Text10 |
| 11 | 11 | NEW_TEXT |
| 12 | 12 | New text |
| | | |

| | id [PK] bigint | ID bigint | Text text |
|---|---|---|---|
| 1 | 1 | 1 | new_Text |
| 2 | 3 | 11 | new_Text |
| | | | |

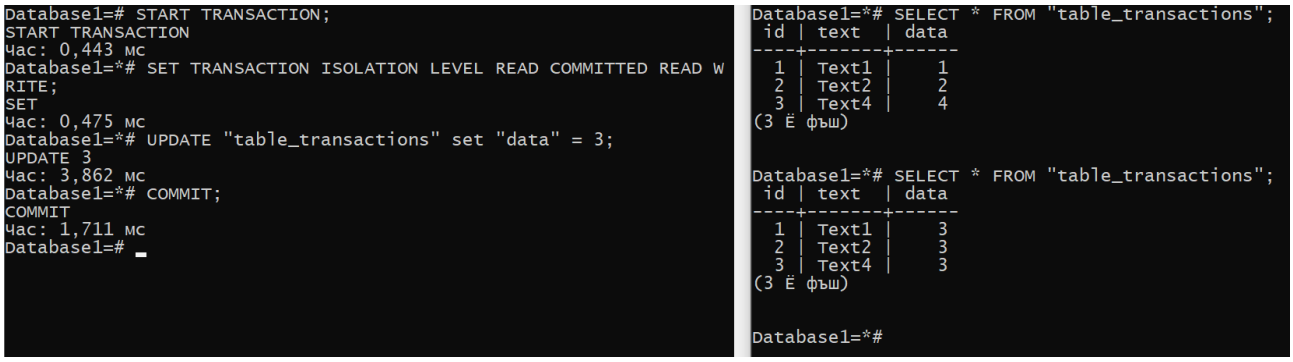
# Завдання №4

Створимо таблицю з початковими даними.

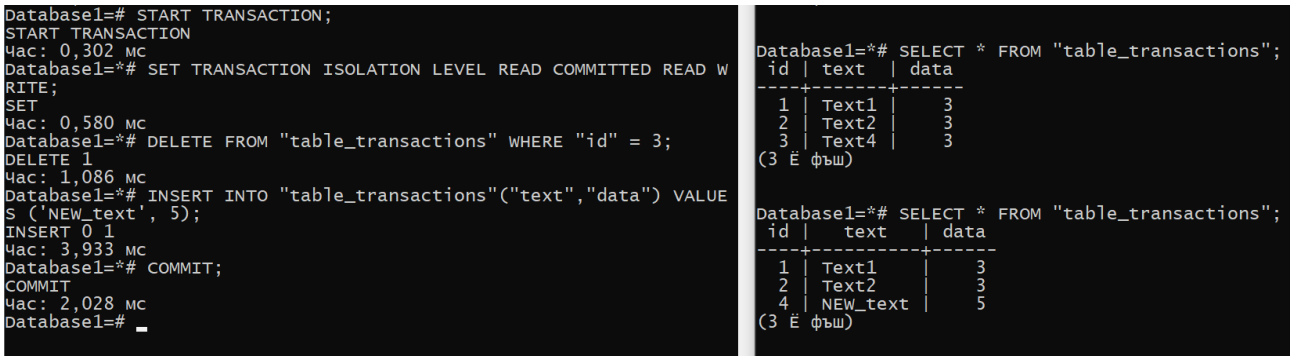
```
DROP TABLE IF EXISTS "table_transactions";
CREATE TABLE "table_transactions"("id" bigserial PRIMARY KEY, "text" text,
"data" bigint);

INSERT INTO "table_transactions"("text", "data") VALUES ('Text1', 1), ('Text2',
2), ('Text4', 4);
```

## 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).

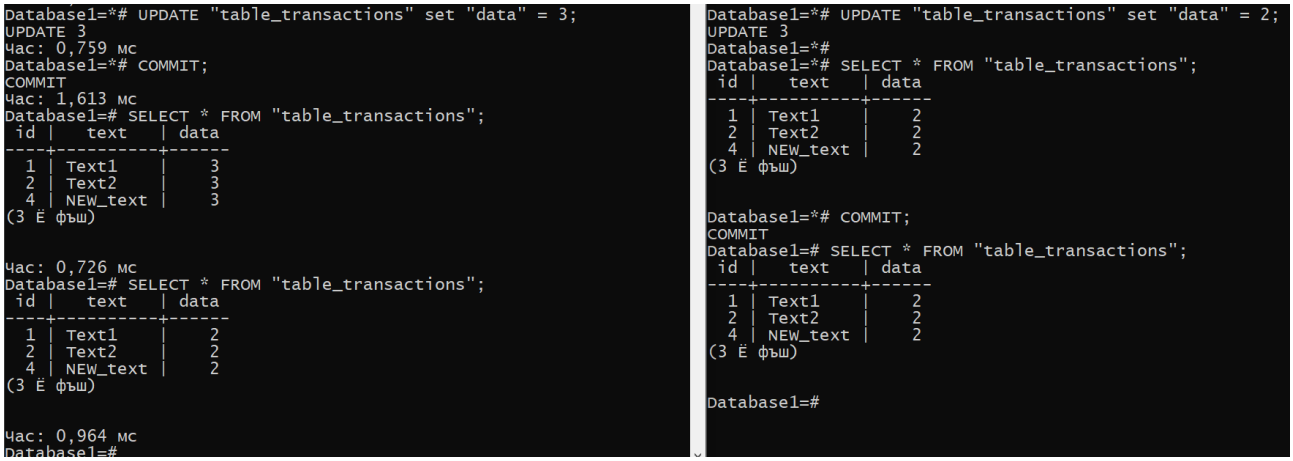



Аналогічна ситуація для вставки, видалення даних, другій транзакції дані будуть видні тільки після завершення першої.




Ось приклад, коли одна транзакція не може внести дані, поки не завершилась попередня.


І вже після закінчення лівої транзакції, запит виконала друга, змінивши дані, знову. При чому дані в першій транзакції після коміта зміняться перший раз і потім ще раз після коміта другої транзакції, що і зображено на скріншоті.

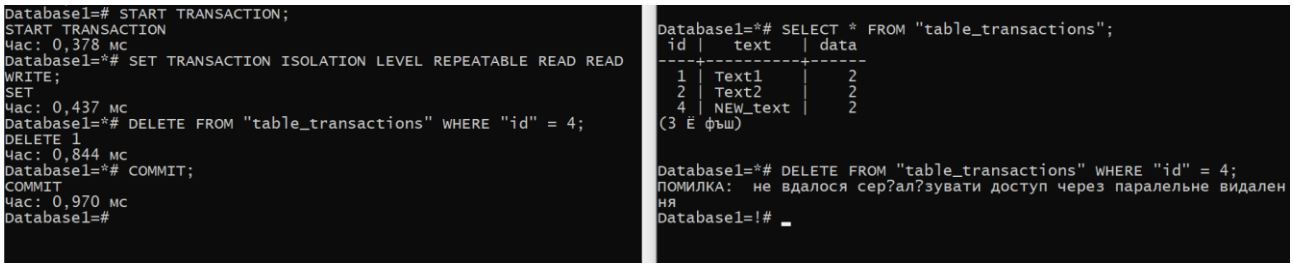



Отже, коли транзакція 2 бачить дані транзакції 1, запитів UPDATE, DELETE виникає феномен повторного читаня, а коли бачить дані запиту INSERT – читання фантомів. Цей рівень ізоляціїзабезпечує захист від явища брудного читання.

## REPEATABLE READ

На цьому рівні ізоляції транзакція 2 не бачитиме змінені дані транзакцією 1, але також незможе отримати доступ до тих самих даних.

На даному скріншоті можна переконатися в тому, що друга транзакція не бачить змін в першій транзакції.

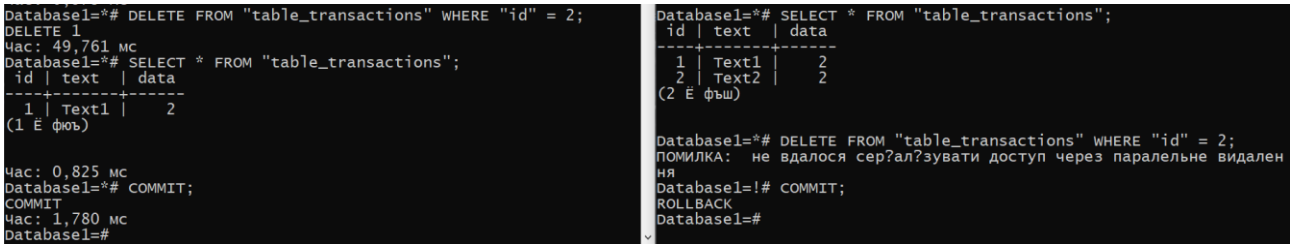



А при спробі доступу будемо отримувати помилку, що також зображено на скріншоті.

Як можна побачити, 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.




На даному рівні ізоляції ми можемо отримати максимальну узгодженість даних і можемо бути впевнені, що зайві дані не будуть зафіксовані.