



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ  
імені ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики  
Кафедра системного програмування і спеціалізованих комп’ютерних систем

**Лабораторна робота № 3**  
з дисципліни «Бази даних і засоби управління»  
«Засоби оптимізації роботи СУБД PostgreSQL»

Виконав: Чекмезов Г.В.  
Студент групи КВ-94  
Перевірів: Петрашенко А.В.

**Київ 2021**

### Лабораторна робота № 3.

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.

Варіант 25

25	<i>BTree, GIN</i>	<i>after delete, insert</i>
----	-------------------	-----------------------------

**GitHub:** <https://github.com/glebbovski/DataBase/tree/main/Lab3>

# Завдання 1

## “Сутність-зв’язок”

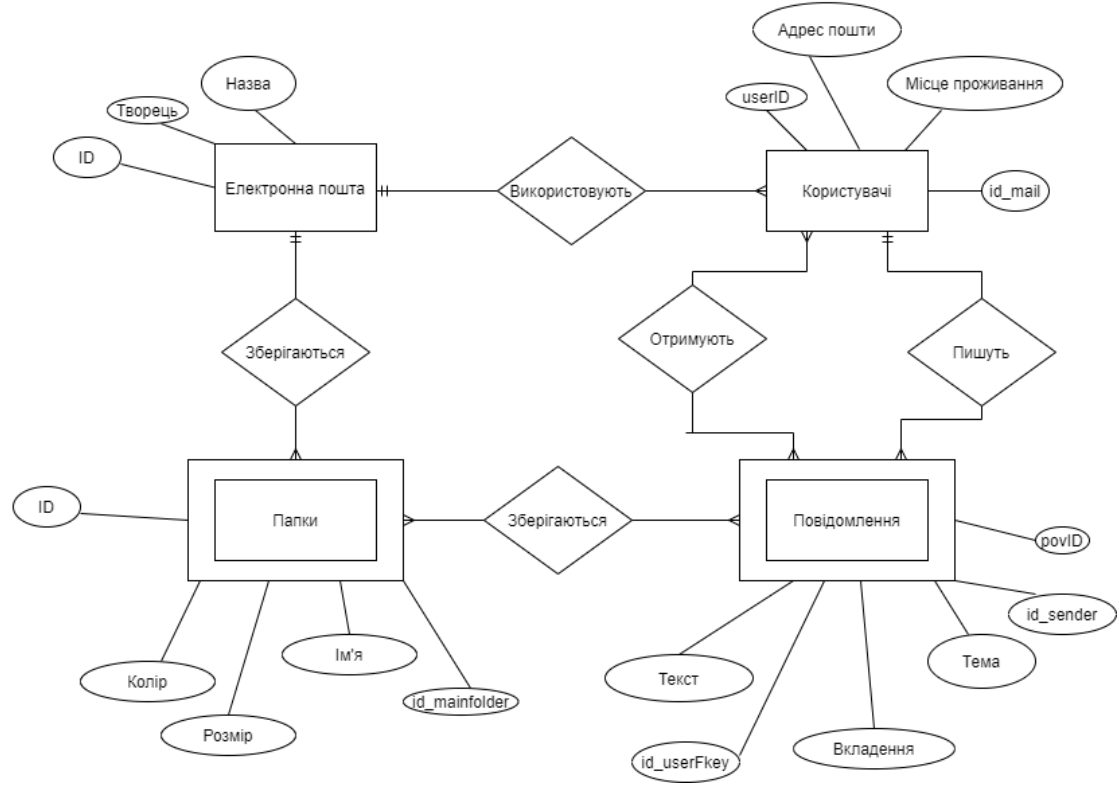
### Сутності

Електронна пошта

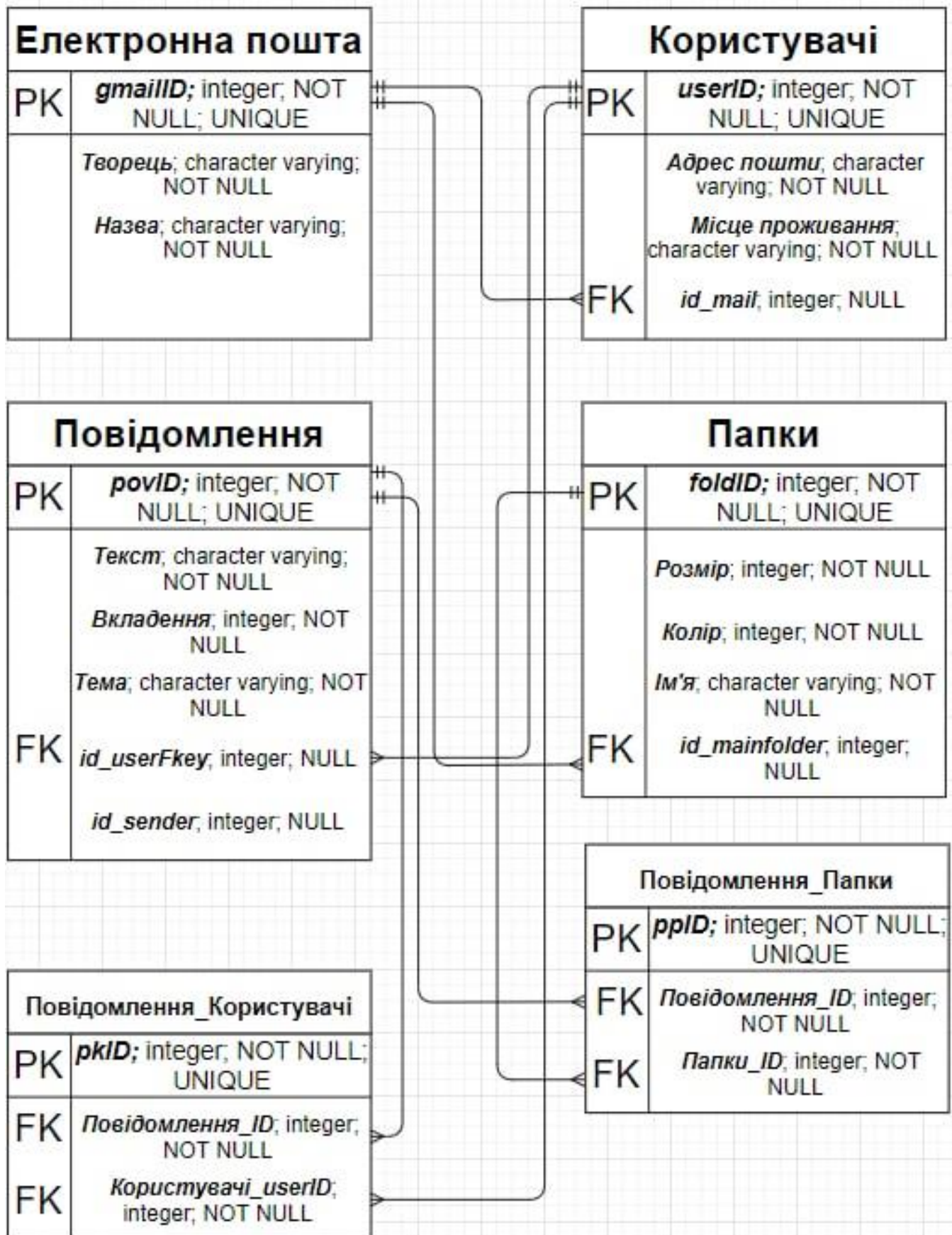
Користувачі

Повідомлення

Папки



### Схема бази даних у графічному вигляді:



## *Таблиці бази даних у pgAdmin 4*

-- Table: public.email

-- DROP TABLE public.email;

CREATE TABLE IF NOT EXISTS public.email

```
(
    "gmailID" integer NOT NULL DEFAULT nextval('"email_gmailID_seq"'::regclass),
    creator character varying COLLATE pg_catalog."default" NOT NULL,
    name character varying COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT email_pkey PRIMARY KEY ("gmailID")
)
```

TABLESPACE pg\_default;

ALTER TABLE public.email

OWNER to postgres;

-----  
-- Table: public.folders

-- DROP TABLE public.folders;

CREATE TABLE IF NOT EXISTS public.folders

```
(
    "foldID" integer NOT NULL DEFAULT nextval('"folders_foldID_seq"'::regclass),
    size integer NOT NULL,
    colour integer NOT NULL,
    nameof character varying COLLATE pg_catalog."default" NOT NULL,
    id_mainfolder integer NOT NULL,
    CONSTRAINT folders_pkey PRIMARY KEY ("foldID"),
    CONSTRAINT folders_id_mainfolder_fkey FOREIGN KEY (id_mainfolder)
        REFERENCES public.email ("gmailID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

TABLESPACE pg\_default;

ALTER TABLE public.folders

OWNER to postgres;

-----  
-- Table: public.folders\_notifications

-- DROP TABLE public.folders\_notifications;

```

CREATE TABLE IF NOT EXISTS public.folders_notifications
(
    "ppID" integer NOT NULL,
    "pov_ID" integer NOT NULL,
    "fold_ID" integer NOT NULL,
    CONSTRAINT folders_notifications_pkey PRIMARY KEY ("ppID", "pov_ID", "fold_ID"),
    CONSTRAINT "folders_notifications_fold_ID_fkey" FOREIGN KEY ("fold_ID")
        REFERENCES public.folders ("foldID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE,
    CONSTRAINT "folders_notifications_pov_ID_fkey" FOREIGN KEY ("pov_ID")
        REFERENCES public.notifications ("povID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

```

```

TABLESPACE pg_default;

```

```

ALTER TABLE public.folders_notifications
    OWNER to postgres;

```

---

```

-- Table: public.notifications

```

```

-- DROP TABLE public.notifications;

```

```

CREATE TABLE IF NOT EXISTS public.notifications
(
    "povID" integer NOT NULL DEFAULT nextval('"notifications_povID_seq"'::regclass),
    text character varying COLLATE pg_catalog."default" NOT NULL,
    addfiles integer NOT NULL,
    title character varying COLLATE pg_catalog."default" NOT NULL,
    "id_userFkey" integer,
    id_sender integer,
    CONSTRAINT notifications_pkey PRIMARY KEY ("povID"),
    CONSTRAINT "notifications_id_userFkey_fkey" FOREIGN KEY ("id_userFkey")
        REFERENCES public.users ("userID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

```

```

TABLESPACE pg_default;

```

```

ALTER TABLE public.notifications
    OWNER to postgres;

```

-----  
-- Table: public.users

-- DROP TABLE public.users;

CREATE TABLE IF NOT EXISTS public.users

(  
    "userID" integer NOT NULL DEFAULT nextval('users\_userID\_seq'::regclass),  
    adress character varying COLLATE pg\_catalog."default" NOT NULL,  
    place character varying COLLATE pg\_catalog."default" NOT NULL,  
    id\_mail integer NOT NULL,  
    CONSTRAINT users\_pkey PRIMARY KEY ("userID"),  
    CONSTRAINT users\_id\_mail\_fkey FOREIGN KEY (id\_mail)  
        REFERENCES public.email ("gmailID") MATCH SIMPLE  
        ON UPDATE NO ACTION  
        ON DELETE NO ACTION  
)

TABLESPACE pg\_default;

ALTER TABLE public.users

    OWNER to postgres;

-----  
-- Table: public.users\_notifications

-- DROP TABLE public.users\_notifications;

CREATE TABLE IF NOT EXISTS public.users\_notifications

(  
    "pkID" integer NOT NULL DEFAULT nextval('users\_notifications\_pkID\_seq'::regclass),  
    "pov\_ID" integer NOT NULL,  
    "users\_ID" integer NOT NULL,  
    CONSTRAINT users\_notifications\_pkey PRIMARY KEY ("pkID"),  
    CONSTRAINT "users\_notifications\_pov\_ID\_fkey" FOREIGN KEY ("pov\_ID")  
        REFERENCES public.notifications ("povID") MATCH SIMPLE  
        ON UPDATE NO ACTION  
        ON DELETE CASCADE,  
    CONSTRAINT "users\_notifications\_users\_ID\_fkey" FOREIGN KEY ("users\_ID")  
        REFERENCES public.users ("userID") MATCH SIMPLE  
        ON UPDATE NO ACTION  
        ON DELETE CASCADE  
)

TABLESPACE pg\_default;

```
ALTER TABLE public.users_notifications
    OWNER to postgres;
```

---

## Реалізовані класи ORM

```
class email(Base):
    __tablename__ = 'email'
    gmailID = Column(Integer, primary_key=True, nullable=False, unique=True)
    creator = Column(String, nullable=False)
    name = Column(String, nullable=False)
    users = relationship("users")
    folders = relationship("folders")

    def __init__(self, gmailID, creator, name):
        self.gmailID = gmailID
        self.creator = creator
        self.name = name

    def __repr__(self):
        return "<email(gmailID={}, creator='{}', name='{}')>"\
            .format(self.gmailID, self.creator, self.name)
```

---

```
class folders(Base):
    __tablename__ = 'folders'
    foldID = Column(Integer, primary_key=True, nullable=False, unique=True)
    size = Column(Integer, nullable=False)
    colour = Column(Integer, nullable=False)
    nameof = Column(String, nullable=False)
    id_mainfolder = Column(Integer, ForeignKey('email.gmailID'),
        nullable=False)

    fol_not = relationship("notifications",
        secondary=folders_notifications_association, back_populates="not_fol",
        passive_deletes='all', lazy='dynamic')

    def __init__(self, foldID, size, colour, nameof, id_mainfolder):
        self.foldID = foldID
        self.size = size
        self.colour = colour
        self.nameof = nameof
        self.id_mainfolder = id_mainfolder

    def __repr__(self):
        return "<folders(foldID={} ,size={}, colour={}, nameof='{}',
            id_mainfolder={})>"\
            .format(self.foldID, self.size, self.colour, self.nameof,
                self.id_mainfolder)
```

---

```
class users(Base):
    __tablename__ = 'users'
    userID = Column(Integer, primary_key=True, nullable=False, unique=True)
    adress = Column(String, nullable=False)
    place = Column(String, nullable=False)
    id_mail = Column(Integer, ForeignKey('email.gmailID'), nullable=False)

    us_not = relationship("notifications",
        secondary=users_notifications_association, back_populates="not_us",
        passive_deletes='all', lazy='dynamic')

    counter = relationship('notifications', cascade="all,
        delete", back_populates='proper', passive_deletes=True)
```



```

def __init__(self, userID, address, place, id_mail):
    self.userID = userID
    self.address = address
    self.place = place
    self.id_mail = id_mail

def __repr__(self):
    return "<users(userID={}, address='{}', place={}, id_mail={})>"\
        .format(self.userID, self.address, self.place, self.id_mail)

```

---

```

class notifications(Base):
    __tablename__ = 'notifications'
    povID = Column(Integer, primary_key=True, nullable=False, unique=True)
    text = Column(String, nullable=False)
    addfiles = Column(Integer, nullable=False)
    title = Column(String, nullable=False)
    id_userFkey = Column(Integer, ForeignKey('users.userID',
ondelete="CASCADE"), nullable=True)
    id_sender = Column(Integer, nullable=True)

    not_fol = relationship("folders",
secondary=folders_notifications_association, back_populates="fol_not",
passive_deletes='all', lazy='dynamic')

    not_us = relationship("users", secondary=users_notifications_association,
back_populates="us_not",
passive_deletes='all', lazy='dynamic')

    propper = relationship('users', back_populates="counter")

    def __init__(self, povID, text, addfiles, title, id_userFkey, id_sender):
        self.povID = povID
        self.text = text
        self.addfiles = addfiles
        self.title = title
        self.id_userFkey = id_userFkey
        self.id_sender = id_sender

    def __repr__(self):
        return "<notifications(povID={}, text='{}', addfiles={}, title='{}',
id_userFkey={}, id_sender={})>"\
            .format(self.povID, self.text, self.addfiles, self.title,
self.id_userFkey, self.id_sender)

```

---

```

folders_notifications_association = Table(
    'folders_notifications', Base.metadata,
    Column('ppID', Integer, primary_key=True, nullable=False),
    Column('pov_ID', Integer, ForeignKey('notifications.povID',
ondelete="CASCADE"), primary_key=True, nullable=False),
    Column('fold_ID', Integer, ForeignKey('folders.foldID',
ondelete="cascade"), primary_key=True, nullable=False)
)

```

---

```

users_notifications_association = Table(
    'users_notifications', Base.metadata,
    Column('pkID', Integer, primary_key=True, nullable=False),
    Column('pov_ID', Integer, ForeignKey('notifications.povID',
ondelete="CASCADE"), nullable=False),
    Column('users_ID', Integer, ForeignKey('users.userID',

```

```
ondelete="cascade"), nullable=False)  
)
```

---

## Запити у вигляді ORM

### Insert

*На прикладі таблиці email.*

Стан таблиці до вставки:

```
<email(gmailID=1, creator='HGKJ', name='UOYI')>  
<email(gmailID=2, creator='POPSTAR', name='STRAPOP')>
```

Після вставки двох рядків:

```
<email(gmailID=1, creator='HGKJ', name='UOYI')>  
<email(gmailID=2, creator='POPSTAR', name='STRAPOP')>  
<email(gmailID=3, creator='Professor', name='kaboba')>  
<email(gmailID=4, creator='Yeppi', name='Peppi')>
```

### Delete

Стан таблиці до видалення:

```
<email(gmailID=1, creator='HGKJ', name='UOYI')>  
<email(gmailID=2, creator='POPSTAR', name='STRAPOP')>  
<email(gmailID=3, creator='Professor', name='kaboba')>  
<email(gmailID=4, creator='Yeppi', name='Peppi')>
```

Стан таблиці після видалення:

```
<email(gmailID=1, creator='HGKJ', name='UOYI')>  
<email(gmailID=3, creator='Professor', name='kaboba')>  
<email(gmailID=4, creator='Yeppi', name='Peppi')>
```

### Update

Стан таблиці до редагування рядка:

```
<email(gmailID=1, creator='HGKJ', name='UOYI')>  
<email(gmailID=3, creator='Professor', name='kaboba')>  
<email(gmailID=4, creator='Yeppi', name='Peppi')>  
<email(gmailID=6, creator='рпловаі', name='аврпол')>
```

Стан таблиці після редагування рядка:

```
<email(gmailID=1, creator='HGKJ', name='UOYI')>  
<email(gmailID=3, creator='Professor', name='kaboba')>  
<email(gmailID=4, creator='Yeppi', name='Peppi')>  
<email(gmailID=6, creator='glebBOSSik', name='glebthebest')>
```

## *Завдання 2*

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

### *BTree*

Індекс *BTree* призначений для даних, які можна відсортувати. Іншими словами, для типу даних мають бути визначені оператори «більше», «більше або дорівнює», «менше», «менше або дорівнює» та «дорівнює». Пошук починається з кореня вузла, і потрібно визначити, по якому з дочірніх вузлів спускатися. Знаючи ключи в корені, можна зрозуміти діапазони значень в дочірніх вузлах. Процедура повторюється до тих пір, поки не буде знайдено вузол, з якого можна отримати необхідні дані.

### **SQL запити**

#### *Створення таблиці БД:*

```
DROP TABLE IF EXISTS "btree_test";  
  
CREATE TABLE "btree_test" ("id" bigserial PRIMARY KEY, "count" integer);  
  
insert into "btree_test"("count") select random()*999999 from generate_series(1,  
1000000) as q;
```

#### *Запити для тестування:*

Усього було проведено 4 запити.

1 – виведення рядків, ідентифікатор яких кратний 3.

2 – виведення рядків, значення “count” яких більше або дорівнює 30000.

3 – виведення середнього значення ідентифікаторів записів, у яких значення “count” знаходиться у проміжку 49789 та 987001.

4 – виведення максимального ідентифікатора, у якого значення “count” знаходиться між 55555 та 555555, та сортується за кратними 3 ідентифікаторами.

```
SELECT COUNT(*) FROM "btree_test" where "id" % 3 = 0;  
SELECT COUNT(*) FROM "btree_test" where "count" >= 30000;  
SELECT AVG("id") from "btree_test" where "count" >= 49789 and "count" <=  
987001;  
SELECT MAX("id") FROM "btree_test" where "count" between 55555 and 555555  
group by "id" % 3 = 0;
```

### Створення індексу:

```
DROP INDEX IF EXISTS "btree_time_index";  
CREATE INDEX "btree_time_index" ON "btree_test" USING btree("id");
```

### Результати та час виконання

#### Без індексування:

```
SQL Shell (psql)  
test=# SELECT COUNT(*) FROM "btree_test" where "id" % 3 = 0;  
count  
-----  
333333  
(1 řĖĖĭŭŗ)  
  
Время: 126,885 мс  
test=# SELECT COUNT(*) FROM "btree_test" where "count" >= 30000;  
count  
-----  
969853  
(1 řĖĖĭŭŗ)  
  
Время: 123,091 мс  
test=# SELECT AVG("id") from "btree_test" where "count" >= 49789 and "count" <= 987001;  
avg  
-----  
499950.827190046659  
(1 řĖĖĭŭŗ)  
  
Время: 139,630 мс  
test=# SELECT MAX("id") FROM "btree_test" where "count" between 55555 and 555555 group by "id" % 3 = 0;  
max  
-----  
999995  
999996  
(2 řĖĖĭŭŗ)  
  
Время: 132,911 мс  
test=#
```

#### З індексуванням:

```
SQL Shell (psql)  
test=# CREATE INDEX "btree_time_index" ON "btree_test" USING btree("id");  
CREATE INDEX  
Время: 2415,847 мс (00:02,416)  
test=# SELECT COUNT(*) FROM "btree_test" where "id" % 3 = 0;  
count  
-----  
333333  
(1 řĖĖĭŭŗ)  
  
Время: 121,826 мс  
test=# SELECT COUNT(*) FROM "btree_test" where "count" >= 30000;  
count  
-----  
969853  
(1 řĖĖĭŭŗ)  
  
Время: 122,674 мс  
test=# SELECT AVG("id") from "btree_test" where "count" >= 49789 and "count" <= 987001;  
avg  
-----  
499950.827190046659  
(1 řĖĖĭŭŗ)  
  
Время: 139,276 мс  
test=# SELECT MAX("id") FROM "btree_test" where "count" between 55555 and 555555 group by "id" % 3 = 0;  
max  
-----  
999995  
999996  
(2 řĖĖĭŭŗ)  
  
Время: 128,995 мс  
test=#
```

З результатів запитів, індексування за допомогою Btree майже не відрізняється за швидкістю, а у деяких випадках взагалі працює довше за запити без індексування. Це можна пояснити тим, що даний метод індексування взагалі не має ефективного обходу по дереву (ефективних засобів вибірки даних), які упорядковані за властивістю, відмінною від обраного ключа. А у нашому випадку вибірка саме така. Даний метод є ефективним для даних, які можливо відсортувати, тобто для таких, для яких можливо використати оператори: більше, більше або дорівнює, менше, менше або дорівнює, дорівнює.

### *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 ідентифікаторів рядків, 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.

### **SQL запити**

#### *Створення таблиці БД:*

```
DROP TABLE IF EXISTS "gin_test";
```

```
create table "gin_test" ("id" bigserial PRIMARY KEY, "doc" text, "doc_tsv"  
tsvector);
```

```
insert into "gin_test"("doc") select substr(characters,  
(random()*length(characters)+1)::int, 15) from  
(values('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as  
symbols(characters), generate_series(1, 1000000) as q;
```

```
update "gin_test" set "doc_tsv" = to_tsvector("doc");
```

#### *Запити для тестування:*

Усього було проведено 4 запити.

```

SELECT COUNT(*) FROM "gin_test" where "id" % 3 = 0;
SELECT COUNT(*) FROM "gin_test" WHERE ("doc_tsv" @@ to_tsquery('ZXCVBNM'));
SELECT avg("id") from "gin_test" where ("doc_tsv" @@
to_tsquery('WERTYUIOPASDFGH')) or ("doc_tsv" @@ to_tsquery('ZXCVBNM'));
select min("id"), max("id") from "gin_test" where ("doc_tsv" @@
to_tsquery('ZXCVBNM')) group by "id" % 3 = 0;

```

*Створення індексу:*

```

DROP INDEX IF EXISTS "gin_time_index";
CREATE INDEX "gin_time_index" ON "gin_test" USING gin("doc_tsv");

```

## Результати та час виконання

Без індексування:

```

SQL Shell (psql)
test=# \timing on
Секундомер включён.
test=# \timing on
Секундомер включён.
test=# SELECT COUNT(*) FROM "gin_test" where "id" % 3 = 0;
count
-----
333333
(1 řĖĖĖĖĖĖ)

Время: 132,534 мс
test=# SELECT COUNT(*) FROM "gin_test" WHERE ("doc_tsv" @@ to_tsquery('ZXCVBNM'));
count
-----
19234
(1 řĖĖĖĖĖĖ)

Время: 789,129 мс
test=# SELECT avg("id") from "gin_test" where ("doc_tsv" @@ to_tsquery('WERTYUIOPASDFGH')) or ("doc_tsv" @@ to_tsquery('
ZXCVBNM'));
avg
-----
500132.585212946012
(1 řĖĖĖĖĖĖ)

Время: 1925,613 мс (00:01,926)
test=# select min("id"), max("id") from "gin_test" where ("doc_tsv" @@ to_tsquery('ZXCVBNM')) group by "id" % 3 = 0;
min | max
-----+-----
302 | 999857
45 | 999837
(2 řĖĖĖĖĖĖ)

Время: 797,659 мс
test=#

```

З індексуванням:

```
SQL Shell (psql)

Время: 797,659 мс
test=# create index "gin_time_index" ON "gin_test" using gin("doc_tsv");
CREATE INDEX
Время: 486,271 мс
test=# SELECT COUNT(*) FROM "gin_test" where "id" % 3 = 0;
 count
-----
 333333
(1 ėĖĖĖĖĖ)

Время: 130,144 мс
test=# SELECT COUNT(*) FROM "gin_test" WHERE ("doc_tsv" @@ to_tsquery('ZXCVBNM'));
 count
-----
 19234
(1 ėĖĖĖĖĖ)

Время: 137,803 мс
test=# SELECT avg("id") from "gin_test" where ("doc_tsv" @@ to_tsquery('WERTYUIOPASDFGH')) or ("doc_tsv" @@ to_tsquery('ZXCVBNM'));
      avg
-----
500132.585212946012
(1 ėĖĖĖĖĖ)

Время: 155,976 мс
test=# select min("id"), max("id") from "gin_test" where ("doc_tsv" @@ to_tsquery('ZXCVBNM')) group by "id" % 3 = 0;
 min | max
-----+-----
 302 | 999857
  45 | 999837
(2 ėĖĖĖĖĖ)

Время: 120,472 мс
test=#
```

Отримавши результати, бачимо, що в усіх викликах, окрім першого(тут індексація на час не впливає), пошук відбувається набагато швидше з індексацією, ніж без неї. Такий результат є очікуваним, тому що це і є основна привілеція індексування GIN – кожне значення шуканого ключа зберігається лише один раз, і запит йде не по всій таблиці, а лише по даним, які містяться у списку появи цього ключа. Таким чином ми можемо зберегти велику кількість часу, не витрачаючи його на непотрібні пошуки.

### Завдання 3

Для тестування тригера було створено дві таблиці:

```
drop table if exists "trigger_test";
create table "trigger_test"("trigger_test_id" bigserial primary key,
"trigger_test_text" text);
```

```

drop table if exists "trigger_test_log";
create table "trigger_test_log" ("id" bigserial primary key,
"trigger_test_log_id" bigint, "trigger_test_log_text" text);

```

### Задання початкових даних у таблицях:

```

insert into "trigger_test" ("trigger_test_text") select chr(trunc(65 +
random()*25)::int)||chr(trunc(65 + random()*25)::int)||chr(trunc(65 +
random()*25)::int)||chr(trunc(65 + random()*25)::int)
from generate_series(1, 100);

```

### Команди ініціювання виконання тригера:

```

create trigger "after_delete_insert_trigger"
AFTER DELETE OR INSERT ON "trigger_test"
for each row
execute procedure after_delete_insert_changes();

```

### Реалізація тригера:

```

create or replace function after_delete_insert_changes()
returns trigger
as $$
DECLARE
    CURSOR_LOG CURSOR FOR SELECT * FROM "trigger_test";
    row_ "trigger_test"%ROWTYPE;
begin
    if old."trigger_test_id" is not null then
        if old."trigger_test_id" % 3 = 0 then
            raise notice 'OLD';
            FOR row_ IN CURSOR_LOG LOOP
                DELETE FROM "trigger_test_log" where
"trigger_test_log_id" = row_."trigger_test_id";
            end loop;
            insert into
"trigger_test_log"("trigger_test_log_id","trigger_test_log_text") values
(old."trigger_test_id", old."trigger_test_text");
            return new;
        else
            raise notice 'NEW';
            FOR row_ IN CURSOR_LOG LOOP
                DELETE FROM "trigger_test_log" where
"trigger_test_log_id" = row_."trigger_test_id";
            end loop;
            insert into
"trigger_test_log"("trigger_test_log_id","trigger_test_log_text") values (old."trigg
er_test_id", 'unspecial');
            return new;
        end if;
    end if;
end if;

```



```

else
    raise notice 'old value is null';
    return old;
end if;
end;
$$ language plpgsql;

```

Результати роботи:

Початковий стан таблиць:

	 trigger_test_id [PK] bigint 	trigger_test_text 
1	1	PHPK
2	2	PYWK
3	3	RKVF
4	4	DPUO
5	5	KPOW
6	6	EPFA
7	7	NIAP
8	8	VQCK
9	9	DDSQ
10	10	WQPF
11	11	UFTD
12	12	TRCL
13	13	MWWQ
14	14	OEUM
15	15	CKPG
16	16	OODW

Query Editor

Query History

Data Output

Messages

Successfully run. Total query runtime  
100 rows affected.



	trigger_test_id [PK] bigint	trigger_test_text text
3	3	UNCH
4	4	PWIIH
5	5	OLDP
6	6	EDRO
7	7	KIGO
8	8	WDHW
9	9	CQAX
10	91	KDDN
11	92	BHMM
12	93	MCWC
13	94	KOLK
14	95	GNCB
15	96	EGKA
16	97	WQME
17	98	HPWF
18	99	SECW
19	100	WSMN

Query Editor   Query History   **Data Output**   Ex

### Messages

Successfully run. Total query runtime: 1  
19 rows affected.

	id [PK] bigint	trigger_test_log_id bigint	trigger_test_log_text text
65	65	74	unspecial
66	66	75	JKUG
67	67	76	unspecial
68	68	77	unspecial
69	69	78	SNKR
70	70	79	unspecial
71	71	80	unspecial
72	72	81	JCGM
73	73	82	unspecial
74	74	83	unspecial
75	75	84	PFHF
76	76	85	unspecial
77	77	86	unspecial
78	78	87	OBTD
79	79	88	unspecial
80	80	89	unspecial
81	81	90	BYIB

Можна побачити, що після виконання запиту видалення, у таблиці trigger\_test видаляються усі рядки з id від 10 до 90, після чого аналізуючи видалені дані, заповнюється таблиця trigger\_test\_log.

Таблиці після окремого виконання запиту на вставку:

```
INSERT INTO "trigger_test"("trigger_test_id",
"trigger_test_text") values (444, 'special');
```

88	88	OQQB
89	89	MQUV
90	90	IFWH
91	91	YEQI
92	92	DVOY
93	93	FOLE
94	94	RTSM
95	95	VVQL
96	96	SYXH
97	97	HAKJ
98	98	ALYF
99	99	EPTQ
100	100	KBYB
101	444	special

Query Editor
Query History
Data Output
E

Messages

Successfully run. Total query runtime:  
101 rows affected.





	id [PK] bigint	trigger_test_log_id bigint	trigger_test_log_text text

З результатів видно, що після виконання запиту на вставку, в таблиці trigger\_test з'явився новий рядок, а таблиця trigger\_test\_log пуста. Це пояснюється тим, що в реалізації нашого триггеру приймають участь “старі” індекси (old id). Тобто, зрозуміло, що при виконанні операції видалення, є так звані “старі” індекси, і їх можна якось використати, а ось у вставки “старих”

індексів бути не може, з'являються тільки “нові”, які у нашому тригері не використовуються. Саме тому така поведінка таблиць є очікуваною.

Таблиці після виконання запитів підряд:

4	4	PQFJ
5	5	NIGU
6	6	VUTG
7	7	XTTR
8	8	QSQY
9	9	LDLF
10	91	CGBP
11	92	YKNK
12	93	QVUU
13	94	POAJ
14	95	BUWW
15	96	EUFA
16	97	AAWF
17	98	XKFL
18	99	PDTG
19	100	NPWC
20	444	special
Query Editor   Query History   Data Output		
Messages		
Successfully run. Total query runti 20 rows affected.		

	 id [PK] bigint 	trigger_test_log_id bigint 	trigger_test_log_text text 	
65	65	74	unspecial	
66	66	75	RNYF	
67	67	76	unspecial	
68	68	77	unspecial	
69	69	78	KCRK	
70	70	79	unspecial	
71	71	80	unspecial	
72	72	81	LEQP	
73	73	82	unspecial	
74	74	83	unspecial	
75	75	84	TWOQ	
76	76	85	unspecial	
77	77	86	unspecial	
78	78	87	VWJP	
79	79	88	unspecial	
80	80	89	unspecial	
81	81	90	ARJN	

[Query Editor](#)
[Query History](#)
[Data Output](#)
[Explain](#)
[Notifications](#)

**Messages**

Successfully run. Total query runtime: 384 msec.  
 81 rows affected.

Зі скріншотів бачимо також очікуваний результат спрацьовування, а саме – у таблиці trigger\_test видалилися рядки з id від 10 до 90, та додався рядок з id 444, та заповнилася таблиця trigger\_test\_log тими значеннями, на які спрямований тригер.

## Завдання 4

Створення таблиці:

```
DROP TABLE IF EXISTS "phenomen";  
  
CREATE TABLE "phenomen" ("id" bigserial PRIMARY KEY, "numeric"  
bigint, "text" text);  
  
INSERT INTO "phenomen" ("numeric", "text") VALUES (111,  
'text1'), (222,  
'text2'), (333, 'text3'), (444, 'text4');
```

### READ COMMITTED

Read Committed — рівень ізоляції транзакції, який вибирається в Postgres Pro за замовчуванням. У транзакції, що працює на цьому рівні, запит SELECT бачить ті дані, які були зафіксовані до початку запиту; він ніколи не побачить незафіксованих даних або змін, внесених у процесі виконання запиту паралельними транзакціями. По суті, запит SELECT бачить знімок бази даних у момент початку виконання запиту. Тобто, доки паралельні транзакції не завершать своє виконання (commit), якихось змін у даних транзакції видно не буде.

Дані після вставки, видалення та редагування у одній транзакції та іншій:



SQL Shell (psql)

Server [localhost]: localhost  
Database [postgres]: test  
Port [5432]: 5432  
Username [postgres]: postgres  
Пароль пользователя postgres:  
psql (13.4)  
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной  
страницы Windows (1251).  
8-битовые (русские) символы могут отображаться некорректно.  
Подробнее об этом смотрите документацию psql, раздел  
"Notes for Windows users".  
Введите "help", чтобы получить справку.  
  
test=# start transaction;  
START TRANSACTION  
test=# set transaction isolation level read committed read write;  
SET  
test=# select \* from "phenomen";  
id | numeric | text  
-----  
1 | 112 | lubli katletki  
2 | 223 | lubli katletki  
3 | 334 | lubli katletki  
6 | 556 | lubli katletki  
(4 стр.)  
  
test=# select \* from "phenomen";  
id | numeric | text  
-----  
1 | 112 | lubli katletki  
2 | 223 | lubli katletki  
3 | 334 | lubli katletki  
6 | 556 | lubli katletki  
(4 стр.)  
  
test=# select \* from "phenomen";  
id | numeric | text  
-----  
1 | 112 | lubli katletki  
2 | 223 | lubli katletki  
3 | 334 | lubli katletki  
6 | 556 | lubli katletki  
(4 стр.)  
  
test=# select \* from "phenomen";  
id | numeric | text  
-----  
1 | 112 | kataper  
2 | 223 | kataper  
3 | 334 | kataper  
7 | 777 | kataper  
(4 стр.)

SQL Shell (psql)

Server [localhost]: localhost  
Database [postgres]: test  
Port [5432]: 5432  
Username [postgres]: postgres  
Пароль пользователя postgres:  
psql (13.4)  
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной  
страницы Windows (1251).  
8-битовые (русские) символы могут отображаться некорректно.  
Подробнее об этом смотрите документацию psql, раздел  
"Notes for Windows users".  
Введите "help", чтобы получить справку.  
  
test=# start transaction;  
START TRANSACTION  
test=# set transaction isolation level read committed read write;  
SET  
test=# delete from "phenomen" where "numeric" = 556;  
DELETE 1  
test=# insert into "phenomen"("numeric", "text") values (777, 'katapa');  
INSERT 0 1  
test=# update "phenomen" set "text" = 'kataper';  
UPDATE 4  
test=# commit;  
COMMIT  
test=#

Друга транзакція не може вносити змін доки не завершиться перша транзакція:

SQL Shell (psql)

Server [localhost]: localhost  
Database [postgres]: test  
Port [5432]: 5432  
Username [postgres]: postgres  
Пароль пользователя postgres:  
psql (13.4)  
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной  
страницы Windows (1251).  
8-битовые (русские) символы могут отображаться некорректно.  
Подробнее об этом смотрите документацию psql, раздел  
"Notes for Windows users".  
Введите "help", чтобы получить справку.  
  
test=# start transaction;  
START TRANSACTION  
test=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;  
SET  
test=# UPDATE "phenomen" set "text" = 'lubli katletki';  
UPDATE 4  
test=#

SQL Shell (psql)

Server [localhost]: localhost  
Database [postgres]: test  
Port [5432]: 5432  
Username [postgres]: postgres  
Пароль пользователя postgres:  
psql (13.4)  
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной  
страницы Windows (1251).  
8-битовые (русские) символы могут отображаться некорректно.  
Подробнее об этом смотрите документацию psql, раздел  
"Notes for Windows users".  
Введите "help", чтобы получить справку.  
  
test=# start transaction;  
START TRANSACTION  
test=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;  
SET  
test=# SELECT \* FROM "phenomen";  
id | numeric | text  
-----  
1 | 112 | text1  
2 | 223 | text2  
3 | 334 | text3  
6 | 556 | text666  
(4 стр.)  
  
test=# SELECT \* FROM "phenomen";  
id | numeric | text  
-----  
1 | 112 | text1  
2 | 223 | text2  
3 | 334 | text3  
6 | 556 | text666  
(4 стр.)  
  
test=# UPDATE "phenomen" set "text" = 'lubli katletki';  
-

Після того, як запити у першій транзакції були “закомічені”, друга транзакція має змогу викликати запити та виконувати їх:

```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (13.4)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
страницы Windows (1251).
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документацию psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.

test=# start transaction;
START TRANSACTION
test=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
SET
test=# UPDATE "phenomen" set "text" = 'lubli katletki';
UPDATE 4
test=# commit;
COMMIT
test=#
```

```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (13.4)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
страницы Windows (1251).
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документацию psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.

test=# start transaction;
START TRANSACTION
test=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
SET
test=# SELECT * FROM "phenomen";
 id | numeric | text
-----+-----+-----
  1 |      112 | text1
  2 |      223 | text2
  3 |      334 | text3
  6 |      556 | text666
(4 строк)

test=# SELECT * FROM "phenomen";
 id | numeric | text
-----+-----+-----
  1 |      112 | text1
  2 |      223 | text2
  3 |      334 | text3
  6 |      556 | text666
(4 строк)

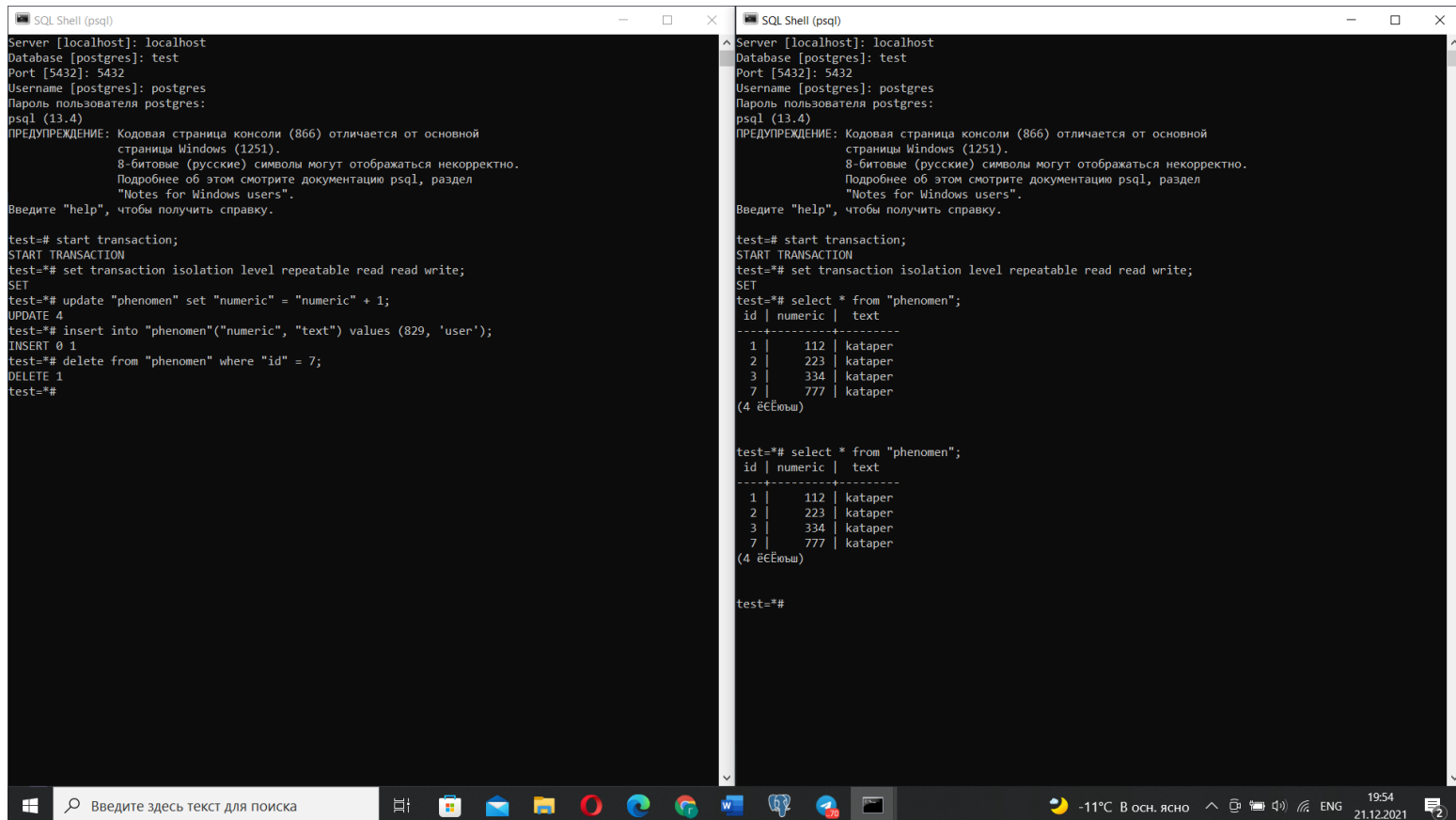
test=# UPDATE "phenomen" set "text" = 'lubli katletki';
UPDATE 4
test=#
```

Тобто, коли друга транзакція бачить зміни у першій транзакції за допомогою запитів UPDATE та DELETE, то виникає феномен повторного читання (транзакція, що читає, «не бачить» зміни даних, які були нею раніше прочитані), а при INSERT виникає читання фантомів (ситуація, коли при повторному читанні в рамках однієї транзакції одна і та ж вибірка дає різні множини рядків). Тобто, на цьому рівні забезпечується захист від чорнового, «брудного» читання, проте, в процесі роботи однієї транзакції інша може бути успішно завершена та зроблені нею зміни зафіксовані.

## REPEATABLE READ

У режимі Repeatable Read видно лише ті дані, які були зафіксовані до початку транзакції, але не видно незафіксовані дані та зміни, здійснені іншими транзакціями в процесі виконання цієї транзакції (однак запит бачитиме ефекти попередніх змін у своїй транзакції, незважаючи на те, що вони не зафіксовані).

Друга транзакція не бачить змін першої:



```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (13.4)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
страницы Windows (1251).
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документацию psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.

test=# start transaction;
START TRANSACTION
test=# set transaction isolation level repeatable read read write;
SET
test=# update "phenomen" set "numeric" = "numeric" + 1;
UPDATE 4
test=# insert into "phenomen"("numeric", "text") values (829, 'user');
INSERT 0 1
test=# delete from "phenomen" where "id" = 7;
DELETE 1
test=#

SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (13.4)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
страницы Windows (1251).
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документацию psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.

test=# start transaction;
START TRANSACTION
test=# set transaction isolation level repeatable read read write;
SET
test=# select * from "phenomen";
 id | numeric | text
-----
  1 |    112 | kataper
  2 |    223 | kataper
  3 |    334 | kataper
  7 |    777 | kataper
(4 строк)

test=# select * from "phenomen";
 id | numeric | text
-----
  1 |    112 | kataper
  2 |    223 | kataper
  3 |    334 | kataper
  7 |    777 | kataper
(4 строк)

test=#
```

Спроба втручання до тих самих даних після commit другою транзакцією:

```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (13.4)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
страницы Windows (1251).
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документацию psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.

test=# start transaction;
START TRANSACTION
test=# set transaction isolation level repeatable read read write;
SET
test=# update "phenomen" set "numeric" = "numeric" + 1;
UPDATE 4
test=# insert into "phenomen"("numeric", "text") values (829, 'user');
INSERT 0 1
test=# delete from "phenomen" where "id" = 7;
DELETE 1
test=# commit;
COMMIT
test=#
```

```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (13.4)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
страницы Windows (1251).
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документацию psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.

test=# start transaction;
START TRANSACTION
test=# set transaction isolation level repeatable read read write;
SET
test=# select * from "phenomen";
 id | numeric | text
-----+-----+-----
  1 |    112 | kataper
  2 |    223 | kataper
  3 |    334 | kataper
  7 |    777 | kataper
(4 строк)

test=# select * from "phenomen";
 id | numeric | text
-----+-----+-----
  1 |    112 | kataper
  2 |    223 | kataper
  3 |    334 | kataper
  7 |    777 | kataper
(4 строк)

test=# update "phenomen" set "numeric" = "numeric" + 1;
ОШИБКА: не удалось сериализовать доступ из-за параллельного изменения
test=#
test=#
```

Видно, що читання фантомів не виникає, хоча взагалі цей рівень ізоляції призначений для попередження повторного читання. Користуватися даним та вищими рівнями транзакцій без необхідності зазвичай не рекомендується.

## SERIALIZABLE

Найвищий рівень ізолюваності; транзакції повністю ізолюються одна від одної, кожна виконується так, ніби паралельних транзакцій не існує. Тільки на цьому рівні паралельні транзакції не схильні до ефекту “фантомного читання”.

Дії у першій та другій транзакції:

```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (13.4)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
страницы Windows (1251).
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документацию psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.

test=# start transaction;
START TRANSACTION
test=# set transaction isolation level serializable read write;
SET
test=# update "phenomen" set "numeric" = "numeric" + 1;
UPDATE 4
test=# insert into "phenomen"("numeric", "text") values (555, 'kartova');
INSERT 0 1
test=# delete from "phenomen" where "id" = 2;
DELETE 1
test=# commit;
COMMIT
test=#
```

```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (13.4)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
страницы Windows (1251).
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документацию psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.

test=#
test=# start transaction;
START TRANSACTION
test=# set transaction isolation level serializable read write;
SET
test=# select * from "phenomen";
 id | numeric | text
-----+-----+-----
  2 |    225 | kataper
  3 |    336 | kataper
  8 |    830 | user
  9 |    555 | kartoxa
(4 строк)

test=# select * from "phenomen";
 id | numeric | text
-----+-----+-----
  2 |    225 | kataper
  3 |    336 | kataper
  8 |    830 | user
  9 |    555 | kartoxa
(4 строк)

test=# update "phenomen" set "numeric" = "numeric" + 1;
ОШИБКА: не удалось сериализовать доступ из-за параллельного изменения
test=# commit;
ROLLBACK
test=#
```

Даний рівень призначений для недопущення читання фантомів. На цьому рівні ізоляції гарантується максимальна узгодженість даних.