



**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 Базы даних і засоби управління
на тему: “Засоби оптимізації роботи СУБД PostgreSQL”**

Виконала:
студентка III курсу
групи KB-94
Романенко М. В.
Перевірів:
Петрашенко А. В.

Київ – 2021

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.

21	<i>Btree, Hash</i>	<i>before delete, update</i>
----	--------------------	------------------------------

Посилання на репозиторій у GitHub з вихідним кодом програми та звітом:

<https://github.com/marromanenko/data-base-course/tree/main/LAB3>

Завдання №1

Обрана предметна галузь передбачає отримання інформації про користувачів деякої соціальної мережі, про їхні пости та коментарі під ними.

Entities

Post

User

Comments

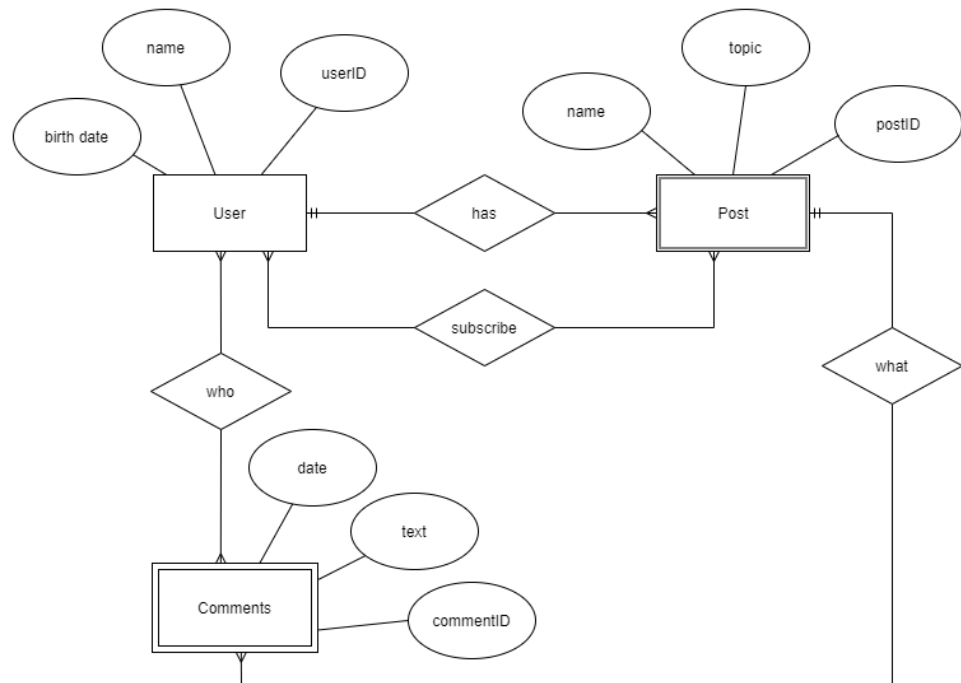


Рис. 1. ER-діаграма, побудована за нотацією "Пташиної лапки"

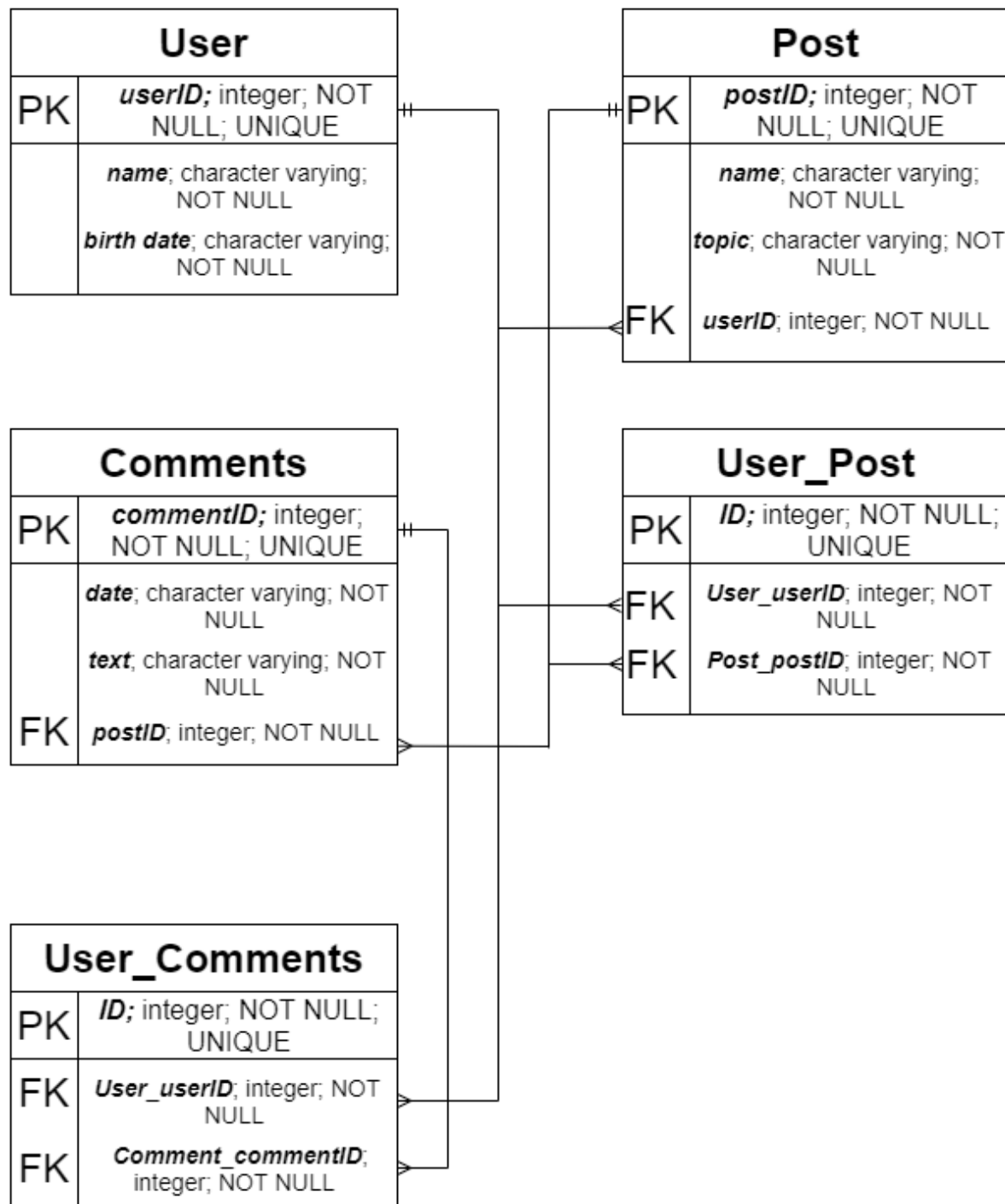


Рис. 2. Схема бази даних у графічному вигляді

Таблиці бази даних у середовищі PgAdmin4

```

-- Table: public.User

-- DROP TABLE public."User";

CREATE TABLE IF NOT EXISTS public."User"
(
    "userID" integer NOT NULL DEFAULT nextval('"User_userID_seq"'::regclass),
    name character varying COLLATE pg_catalog."default",
    birth_date date,
    CONSTRAINT "User_pkey" PRIMARY KEY ("userID")
)

TABLESPACE pg_default;

ALTER TABLE public."User"
    OWNER to postgres;

-- Table: public.Post

-- DROP TABLE public."Post";

CREATE TABLE IF NOT EXISTS public."Post"
(
    "postID" integer NOT NULL DEFAULT nextval('"Post_postID_seq"'::regclass),
    namepost character varying COLLATE pg_catalog."default",
    topic character varying COLLATE pg_catalog."default",
    "userID" integer,
    CONSTRAINT "Post_pkey" PRIMARY KEY ("postID"),
    CONSTRAINT "Post_userID_fkey" FOREIGN KEY ("userID")
        REFERENCES public."User" ("userID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE public."Post"
    OWNER to postgres;

```

```

|-- Table: public.Comments

-- DROP TABLE public."Comments";

CREATE TABLE IF NOT EXISTS public."Comments"
(
    "commentID" integer NOT NULL DEFAULT nextval('"Comments_commentID_seq"'::regclass),
    date date,
    text character varying COLLATE pg_catalog."default",
    "postID" integer,
    CONSTRAINT "Comments_pkey" PRIMARY KEY ("commentID"),
    CONSTRAINT "Comments_postID_fkey" FOREIGN KEY ("postID")
        REFERENCES public."Post" ("postID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE public."Comments"
    OWNER to postgres;

|-- Table: public.User_Comments

-- DROP TABLE public."User_Comments";

CREATE TABLE IF NOT EXISTS public."User_Comments"
(
    "User_userID" integer NOT NULL,
    "Comments_commentID" integer NOT NULL,
    CONSTRAINT "User_Comments_pkey" PRIMARY KEY ("User_userID", "Comments_commentID"),
    CONSTRAINT "User_Comments_Comments_commentID_fkey" FOREIGN KEY ("Comments_commentID")
        REFERENCES public."Comments" ("commentID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT "User_Comments_User_userID_fkey" FOREIGN KEY ("User_userID")
        REFERENCES public."User" ("userID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE public."User_Comments"
    OWNER to postgres;

```

```

|-- Table: public.User_Post

-- DROP TABLE public."User_Post";

CREATE TABLE IF NOT EXISTS public."User_Post"
(
    "User_userID" integer NOT NULL,
    "Post_postID" integer NOT NULL,
    CONSTRAINT "User_Post_pkey" PRIMARY KEY ("User_userID", "Post_postID"),
    CONSTRAINT "User_Post_Post_postID_fkey" FOREIGN KEY ("Post_postID")
        REFERENCES public."Post" ("postID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT "User_Post_User_userID_fkey" FOREIGN KEY ("User_userID")
        REFERENCES public."User" ("userID") MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE public."User_Post"
    OWNER to postgres;

```

Класи ORM у реалізованому модулі Model

```

user_post_association = Table('User_Post', Base.metadata, Column('User_userID',
Integer, ForeignKey('User.userID'), primary_key=True),
                             Column('Post_postID', Integer,
ForeignKey('Post.postID'), primary_key=True))

user_comments_association = Table('User_Comments', Base.metadata,
Column('User_userID', Integer, ForeignKey('User.userID'), primary_key=True),
      Column('Comments_commentID', Integer,
ForeignKey('Comments.commentID'), primary_key=True))

class User(Base):
    __tablename__ = 'User'
    userID = Column(Integer, primary_key=True)
    name = Column(String)
    birth_date = Column(Date)

    posts = relationship('Post')
    comments = relationship("Comments", secondary=user_comments_association)
    posts1 = relationship("Post", secondary=user_post_association)

    def __init__(self, userID, name, birth_date):
        self.userID = userID
        self.name = name

```

```

        self.birth_date = birth_date

    def __repr__(self):
        return "<User(name='{ }', birth_date={})>".format(self.name,
self.birth_date)

class Post(Base):
    __tablename__ = 'Post'
    postID = Column(Integer, primary_key=True)
    namepost = Column(String)
    topic = Column(String)
    userID = Column(Integer, ForeignKey('User.userID'))
    comments = relationship("Comments")

    def __init__(self, postID, namepost, topic, userID):
        self.postID = postID
        self.namepost = namepost
        self.topic = topic
        self.userID = userID

    def __repr__(self):
        return "<Post(namepost='{ }', topic={})>".format(self.namepost,
self.topic)

class Comments(Base):
    __tablename__ = 'Comments'
    commentID = Column(Integer, primary_key=True)
    date = Column(Date)
    text = Column(String)
    postID = Column(Integer, ForeignKey('Post.postID'))

    def __init__(self, commentID, date, text, postID):
        self.commentID = commentID
        self.date = date
        self.text = text
        self.postID = postID

    def __repr__(self):
        return "<Comment(date={ }, text='{ }')>".format(self.date, self.text)

```

Запити у вигляді ORM

Продемонструємо вставку, виучення, редагування даних на прикладі таблиці Product

Початковий стан:


```

User
SQL query =>  select * from public."User"

-----
ID = 1
Name =  Mariia
Birth date =  2002-04-14
-----
ID = 3
Name =  Sofiia
Birth date =  2001-10-03
-----
ID = 4
Name =  Yarina
Birth date =  2010-09-12
-----
ID = 2
Name =  Gleb
Birth date =  2002-07-26
-----

```

Видалення запису:

```

Main menu
0 => Show one table
1 => Show all table
2 => Insert data
3 => Delete data
4 => Update data
5 => Select data
6 => Randomize data
7 => Exit
Make your choice => 3

1 => User
2 => Post
3 => Comments

Choose table number => 1
Attribute to delete User ID = 4
Continue deletion? 1 - Yes; 2 - No =>2

```

```

Make your choice => 0

1 => User
2 => Post
3 => Comments
4 => User_Comments
5 => User_Post

Choose table number => 1
User
SQL query => select * from public."User"

-----
ID = 1
Name = Mariia
Birth date = 2002-04-14
-----
ID = 3
Name = Sofiia
Birth date = 2001-10-03
-----
ID = 2
Name = Gleb
Birth date = 2002-07-26
-----

```

Вставка запису:

```

Make your choice => 2

1 => User
2 => Post
3 => Comments

Choose table number => 1
Name = Sasha
Birth date = 2002-08-17
User ID = 4
Continue insertion? 1 - Yes; 2 - No =>|

```

```

User
SQL query =>  select * from public."User"

-----
ID = 1
Name =  Mariia
Birth date =  2002-04-14
-----
ID = 3
Name =  Sofiaia
Birth date =  2001-10-03
-----
ID = 2
Name =  Gleb
Birth date =  2002-07-26
-----
ID = 4
Name =  Sasha
Birth date =  2002-08-17
-----

```

Редагування запису:

```

Make your choice => 4

1 => User
2 => Post
3 => Comments

Choose table number => 1
Attribute to update(where) User ID = 4
New name = Yaroslava
New birth date = 2000-01-01
Continue updation? 1 - Yes; 2 - No =>

```

```

User
SQL query => select * from public."User"

-----
ID = 1
Name = Mariia
Birth date = 2002-04-14
-----
ID = 3
Name = Sofia
Birth date = 2001-10-03
-----
ID = 2
Name = Gleb
Birth date = 2002-07-26
-----
ID = 4
Name = Yaroslava
Birth date = 2000-01-01
-----

```

Завдання №2

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 ключі 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, "time" timestamp);

INSERT INTO "btree_test"("time") SELECT (timestamp '2021-01-01' + random()*(timestamp
'2020-01-01'-timestamp '2022-01-01')) FROM
(VALUES('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
generate_series(1, 1000000) as q;

```

Запити для тестування:

Було протестовано 4 запити: 1 – виведення записів, ідентифікатор яких кратний 2; 2 - виведення записів, у яких час більше або дорівнює 2019-10-01; 3 -

виведення середнього значення ідентифікаторів записів, у яких час знаходиться в проміжку між 2019-10-01 та 2021-12-7; 4 - виведення суми ідентифікаторів, а також максимального ідентифікатора записів, у яких час знаходиться в проміжку між 2020-05-05 та 2021-05-05, сортування за кратними 2 ідентифікаторами.

```
SELECT COUNT(*) FROM "btree_test" WHERE "id" % 2 = 0;
```

```
SELECT COUNT(*) FROM "btree_test" WHERE "time" >= '20191001';
```

```
SELECT AVG("id") FROM "btree_test" WHERE "time" >= '20191001' AND "time" <= '20211207';
```

```
SELECT SUM("id"), MAX("id") FROM "btree_test" WHERE "time" >= '20200505' AND "time" <= '20210505' GROUP BY "id"%2;
```

Створення індексу:

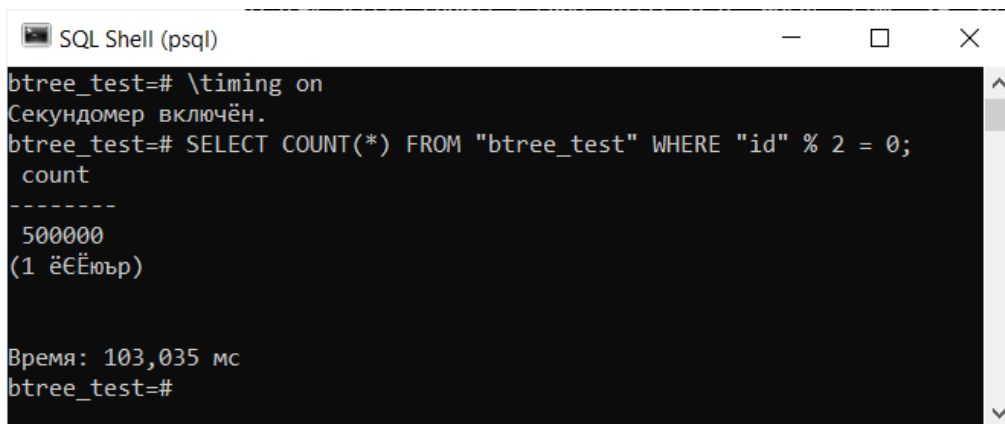
```
DROP INDEX IF EXISTS "btree_index";
```

```
CREATE INDEX "btree_time_index" ON "btree_test" ("id");
```

Результати виконання запитів

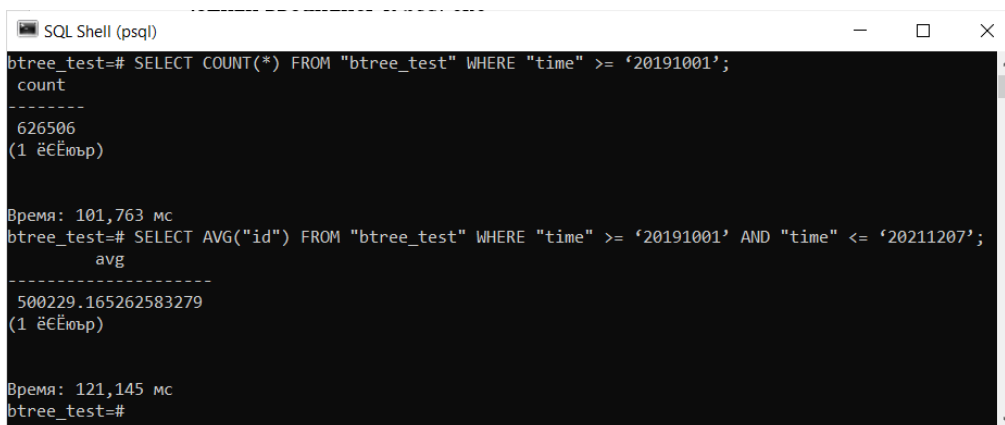
Запити вводились у psql.exe.

Запити без індексування:



```
SQL Shell (psql)
btree_test=# \timing on
Секундометр включён.
btree_test=# SELECT COUNT(*) FROM "btree_test" WHERE "id" % 2 = 0;
 count
-----
 500000
(1 řĖĖĥ)

Время: 103,035 мс
btree_test=#
```



```
SQL Shell (psql)
btree_test=# SELECT COUNT(*) FROM "btree_test" WHERE "time" >= '20191001';
 count
-----
 626506
(1 řĖĖĥ)

Время: 101,763 мс
btree_test=# SELECT AVG("id") FROM "btree_test" WHERE "time" >= '20191001' AND "time" <= '20211207';
      avg
-----
500229.165262583279
(1 řĖĖĥ)

Время: 121,145 мс
btree_test=#
```

```
SQL Shell (psql)
btree_test=# SELECT SUM("id"), MAX("id") FROM "btree_test" WHERE "time" >= '20200505' AND "time" <= '20210505' GROUP BY "id"%2;
      sum      | max
-----+-----
 82273546660 | 999998
 82648086755 | 999991
(2 rows)

Время: 210,172 мс
btree_test=#
```

Час виконання запитів

Операція 1	Операція 2	Операція 3	Операція 4
103,035 мс	101,763 мс	121,145	210,172 мс

Запити з індексуванням:

```
SQL Shell (psql)
btree_test=# CREATE INDEX "btree_time_index" ON "btree_test" ("id");
CREATE INDEX
Время: 901,514 мс
```

```
Выбрать SQL Shell (psql)
btree_test=# SELECT COUNT(*) FROM "btree_test" WHERE "id" % 2 = 0;
 count
-----
 500000
(1 row)

Время: 95,780 мс
btree_test=# SELECT COUNT(*) FROM "btree_test" WHERE "time" >= '20191001';
 count
-----
 626506
(1 row)

Время: 94,006 мс
```

```
SQL Shell (psql)
btree_test=# SELECT AVG("id") FROM "btree_test" WHERE "time" >= '20191001' AND "time" <= '20211207';
      avg
-----
500229.165262583279
(1 row)

Время: 121,022 мс
btree_test=#
```

```
SQL Shell (psql)
btree_test=# SELECT SUM("id"), MAX("id") FROM "btree_test" WHERE "time" >= '20200505' AND "time" <= '20210505' GROUP BY "id"%2;
      sum      | max
-----+-----
 82273546660 | 999998
 82648086755 | 999991
(2 rows)

Время: 208,870 мс
btree_test=#
```

Час виконання запитів

Операція 1	Операція 2	Операція 3	Операція 4
95,780 мс	94,006 мс	121,022	208,870 мс

Як бачимо з результатів, за допомогою використання індексу BTree виконання операцій майже не змінилося. Це пов'язано з тим, що основний недолік В-дерев полягає у відсутності для них ефективних засобів вибірки даних (тобто методу обходу дерева), упорядкованих за якістю, відмінною від обраного ключа. Цей індекс рекомендовано використовувати саме для операцій пошуку с порівнянням (нерівностями), що і було продемонстровано в запитах.

Hash

Хеш-індекси в PostgreSQL використовують форму структури даних хеш-таблиці (використовують хеш-функцію). Хеш-коди поділені на обмежену кількість комірок. Коли до індексу додається нове значення, PostgreSQL застосовує хеш-функцію до значення і поміщає хеш-код і вказівник на кортеж у відповідну комірку. Коли відбувається запит за допомогою індексу хешування, PostgreSQL бере значення індексу і застосовує хеш-функцію, щоб визначити, яка комірка може містити потрібні дані.

SQL запити

Створення таблиці БД:

```
DROP TABLE IF EXISTS "hash_test";

CREATE TABLE "hash_test"("id" bigserial PRIMARY KEY, "time" timestamp);

INSERT INTO "hash_test"("time") SELECT (timestamp '2021-01-01' + random() * (timestamp '2020-01-01' - timestamp '2022-01-01')) FROM
(VALUE('qwertyuiopasdfghjklzxcvbnmQWERTYUIOPASDFGHJKLZXCVBNM')) as symbols(characters),
generate_series(1, 1000000) as q;
```

Запити для тестування:

```
SELECT COUNT(*) FROM "hash_test" WHERE "id" % 2 = 0;

SELECT COUNT(*) FROM "hash_test" WHERE "time" >= '20191001';

SELECT AVG("id") FROM "hash_test" WHERE "time" >= '20191001' AND "time" <= '20211207';

SELECT SUM("id"), MAX("id") FROM "hash_test" WHERE "time" >= '20200505' AND "time" <= '20210505' GROUP BY "id" % 2;
```

Створення індексу:

```
DROP TABLE IF EXISTS "hash_test";

CREATE INDEX "time_hash_index" ON "hash_test" USING hash("id");
```

Результати виконання запитів

Запити без індексування:

```
SQL Shell (psql)
btree_test=# \timing on
Секундомер включён.
btree_test=# SELECT COUNT(*) FROM "btree_test" WHERE "id" % 2 = 0;
count
-----
500000
(1 řĖĖĭŕ)

Время: 103,035 мс
btree_test=#
```

```
SQL Shell (psql)
btree_test=# SELECT COUNT(*) FROM "btree_test" WHERE "time" >= '20191001';
count
-----
626506
(1 řĖĖĭŕ)

Время: 101,763 мс
btree_test=# SELECT AVG("id") FROM "btree_test" WHERE "time" >= '20191001' AND "time" <= '20211207';
avg
-----
500229.165262583279
(1 řĖĖĭŕ)

Время: 121,145 мс
btree_test=#
```

```
SQL Shell (psql)
btree_test=# SELECT SUM("id"), MAX("id") FROM "btree_test" WHERE "time" >= '20200505' AND "time" <= '20210505' GROUP BY "id"%2;
sum      | max
-----+-----
8227354660 | 999998
82648086755 | 999991
(2 řĖĖĭŕŕ)

Время: 210,172 мс
btree_test=#
```

Час виконання запитів

Операція 1	Операція 2	Операція 3	Операція 4
103,035 мс	101,763 мс	121,145	210,172 мс

Запити з індексуванням:


```
SQL Shell (psql)
btree_test=# SELECT COUNT(*) FROM "hash_test" WHERE "id" % 2 = 0;
count
-----
500000
(1 66171 мс)

Время: 66,171 мс
btree_test=# SELECT COUNT(*) FROM "hash_test" WHERE "time" >= '20191001';
count
-----
626127
(1 140,615 мс)

Время: 140,615 мс
btree_test=# SELECT AVG("id") FROM "hash_test" WHERE "time" >= '20191001' AND "time" <= '20211207';
avg
-----
499992.804285712004
(1 71,534 мс)

Время: 71,534 мс
btree_test=# SELECT SUM("id"), MAX("id") FROM "hash_test" WHERE "time" >= '20200505' AND "time" <= '20210505' GROUP BY "id" % 2;
sum | max
-----+-----
82472381958 | 999998
82293169080 | 999999
(2 122,508 мс)

Время: 122,508 мс
btree_test=#
```

Час виконання запитів

Операція 1	Операція 2	Операція 3	Операція 4
66,171 мс	140,615 мс	71,534	122,508 мс

Як можна побачити індексування за допомогою hash не значно пришвидшує пошук даних у таблиці, а іноді навіть показує гірші результати, ніж запити без індексування. Так виходить тому що це один із найпримітивніших методів індексування і для пошуку потрібних даних алгоритм все одно проходить через усі записи у таблиці. Він ефективний при застосуванні до поля числового типу.

Висновок

Індекс HASH підходить для операцій порівняння на рівність. Його не можна використовувати для прискорення порядку операцій, а також неможливо визначити кількість рядків між двома значеннями, що вплине на ефективність виконання деяких запитів. І лише ключове слово цілком може використовуватись для пошуку рядка.

Індекс BTREE можна використовувати при використанні операторів більше, менше, BETWEEN, не рівне і LIKE. Якщо ви вибираєте діапазон у полі індексу через індекс, можна отримати доступ лише до індексу BTREE. Індекс HASH фактично є повне сканування таблиці.

Завдання №3

Для тестування тригера було створено дві таблиці:

```
DROP TABLE IF EXISTS "trigger_test";
```

```
CREATE TABLE "trigger_test"( "trigger_testID" bigserial PRIMARY KEY, "trigger_testName" 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_name" text );
```

Початкові дані у таблицях:

```
INSERT INTO "trigger_test"("trigger_testName") VALUES ('trigger_test1'),  
('trigger_test2'), ('trigger_test3'), ('trigger_test4'), ('trigger_test5'),  
('trigger_test6'), ('trigger_test7'), ('trigger_test8'), ('trigger_test9'),  
('trigger_test10');
```

Команди, що ініціюють виконання тригера:

```
CREATE TRIGGER "before_delete_update_trigger"  
BEFORE DELETE OR UPDATE ON "trigger_test"  
FOR EACH ROW  
EXECUTE procedure before_delete_update_func();
```

Текст тригера:

```
CREATE OR REPLACE FUNCTION before_delete_update_func() RETURNS TRIGGER as $trigger$  
DECLARE  
CURSOR_LOG CURSOR FOR SELECT * FROM "trigger_test_log";  
row_ "trigger_test_log"%ROWTYPE;  
  
BEGIN  
IF old."trigger_testID" % 2 = 0 THEN  
IF old."trigger_testID" % 3 = 0 THEN  
RAISE NOTICE 'trigger_testID is multiple of 2 and 3';  
FOR row_ IN CURSOR_LOG LOOP  
UPDATE "trigger_test_log" SET "trigger_test_log_name" = '_' ||  
row_."trigger_test_log_name" || '_log' WHERE "id" = row_."id";  
END LOOP;  
RETURN OLD;  
ELSE  
RAISE NOTICE 'trigger_testID is even';  
INSERT INTO "trigger_test_log"("trigger_test_log_ID", "trigger_test_log_name")  
VALUES (old."trigger_testID", old."trigger_testName");  
UPDATE "trigger_test_log" SET "trigger_test_log_name" = trim(BOTH '_log' FROM  
"trigger_test_log_name");  
RETURN NEW;
```

```

END IF;
ELSE
RAISE NOTICE 'trigger_testID is odd';
FOR row_ IN CURSOR_LOG LOOP
UPDATE "trigger_test_log" SET "trigger_test_log_name" = '_' ||
row_."trigger_test_log_name" || '_log' WHERE "id" = row_."id";
END LOOP;
RETURN OLD;
END IF;
END;
$trigger$ LANGUAGE plpgsql;

```

Скріншоти зі змінами у таблицях бази даних

Початковий стан

```
SELECT * FROM "trigger_test";
```

```
SELECT * FROM "trigger_test_log";
```

	trigger_testID [PK] bigint	trigger_testName text
1	1	trigger_test1
2	2	trigger_test2
3	3	trigger_test3
4	4	trigger_test4
5	5	trigger_test5
6	6	trigger_test6
7	7	trigger_test7
8	8	trigger_test8
9	9	trigger_test9
10	10	trigger_test10

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text

Після виконання запиту на видалення

```
DELETE FROM "trigger_test" WHERE "trigger_testID" % 3 = 0;
```

	trigger_testID [PK] bigint	trigger_testName text
1	1	trigger_test1
2	2	trigger_test2
3	4	trigger_test4
4	5	trigger_test5
5	7	trigger_test7
6	8	trigger_test8
7	10	trigger_test10

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text

Після виконання запиту на видалення бачимо видалення кратних трьом рядків у таблиці trigger_test та пусту таблицю trigger_test_log (тому що в ній відбуваються зміни для існуючих записів, а не запис нових)

Після виконання запиту на оновлення

	trigger_testID [PK] bigint	trigger_testName text
1	1	trigger_test1
2	2	trigger_test2_log
3	3	trigger_test3
4	4	trigger_test4_log
5	5	trigger_test5
6	6	trigger_test6
7	7	trigger_test7
8	8	trigger_test8_log
9	9	trigger_test9
10	10	trigger_test10_log

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text
1	1	2	trigger_test2
2	2	4	trigger_test4
3	3	8	trigger_test8
4	4	10	trigger_test10

Після виконання запиту на оновлення бачимо виконання алгоритму тригера за парними рядками (в тому числі для 6, але з поверненням стану old).

Після виконання запиту видалення і оновлення

	trigger_testID [PK] bigint	trigger_testName text
1	1	trigger_test1
2	2	trigger_test2_log
3	4	trigger_test4_log
4	5	trigger_test5
5	7	trigger_test7
6	8	trigger_test8_log
7	10	trigger_test10_log

	id [PK] bigint	trigger_test_log_ID bigint	trigger_test_log_name text
1	1	2	trigger_test2
2	2	4	trigger_test4
3	3	8	trigger_test8
4	4	10	trigger_test10

Після виконання запиту на видалення та оновлення бачимо видалення кратних трьом запитів, а до парних рядків додавання певного тексту, як і очікувалося.

Завдання №4

Для цього завдання також створювалась окрема таблиця з деякими початковими даними:

```

DROP TABLE IF EXISTS "transactions";

CREATE TABLE "transactions"(
    "id" bigserial PRIMARY KEY,
    "numeric" bigint,
    "text" text
);

INSERT INTO "transactions"("numeric", "text") VALUES (1, 'str1'), (2, 'str2'), (3, 'str3');

```

READ COMMITTED

На цьому рівні ізоляції одна транзакція не бачить змін у базі даних, викликаних іншою доки та не завершить своє виконання.

```

SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: btree_test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (14.0)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
страницы Windows (1251).
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документации psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.

btree_test=# START TRANSACTION;
START TRANSACTION
btree_test=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
SET
btree_test=# UPDATE "transactions" SET "numeric" = "numeric" + 1;
UPDATE 4
btree_test=# INSERT INTO "transactions"("numeric", "text") VALUES (5, 'str5');
INSERT 0 1
btree_test=# DELETE FROM "transactions" WHERE "id"=1;
DELETE 1
btree_test=# COMMIT;
COMMIT
btree_test=#

SQL Shell (psql)
btree_test=# START TRANSACTION;
START TRANSACTION
btree_test=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
SET
btree_test=# SELECT * FROM "transactions";
 id | numeric | text
-----
 1 |      2 | str1
 2 |      3 | str2
 3 |      4 | str3
 4 |      4 | str4
(4 строк)

btree_test=# SELECT * FROM "transactions";
 id | numeric | text
-----
 1 |      2 | str1
 2 |      3 | str2
 3 |      4 | str3
 4 |      4 | str4
(4 строк)

btree_test=# SELECT * FROM "transactions";
 id | numeric | text
-----
 1 |      2 | str1
 2 |      3 | str2
 3 |      4 | str3
 4 |      4 | str4
(4 строк)

btree_test=# SELECT * FROM "transactions";
 id | numeric | text
-----
 2 |      4 | str2
 3 |      5 | str3
 4 |      5 | str4
 5 |      5 | str5
(4 строк)

btree_test=#

```

Тобто можемо зробити висновок, що операції вставки, оновлення та видалення будуть видні другій тільки після завершення першої транзакції, і зі скріншотів бачимо що після завершення першої, друга транзакція виконала запит, змінивши вже ті дані, що були закомічені першою транзакцією.

SQL Shell (psql)

Server [localhost]: localhost
Database [postgres]: btree_test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (14.0)
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.

btree_test=# START TRANSACTION;
START TRANSACTION
btree_test=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
SET
btree_test=# UPDATE "transactions" SET "numeric" = "numeric" + 1;
UPDATE 4
btree_test=#

SQL Shell (psql)

Server [localhost]: localhost
Database [postgres]: btree_test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (14.0)
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.

btree_test=# START TRANSACTION;
START TRANSACTION
btree_test=# SET TRANSACTION ISOLATION LEVEL READ COMMITTED READ WRITE;
SET
btree_test=# UPDATE "transactions" SET "numeric" = "numeric" + 1;

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

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:

SQL Shell (psql)

Database [postgres]: btree_test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (14.0)
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.

btree_test=# START TRANSACTION;
START TRANSACTION
btree_test=# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ READ WRITE;
SET
btree_test=# UPDATE "transactions" SET "numeric" = "numeric" + 1;
UPDATE 4
btree_test=# INSERT INTO "transactions"("numeric", "text") VALUES (4, 'str4');
INSERT 0 1
btree_test=# DELETE FROM "transactions" WHERE "id"=1;
DELETE 0
btree_test=#

SQL Shell (psql)

Database [postgres]: btree_test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (14.0)
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.

btree_test=# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ READ WRITE;
SET
btree_test=# SELECT * FROM "transactions";
id | numeric | text

2 | 4 | str2
3 | 5 | str3
4 | 5 | str4
5 | 5 | str5
(4 RECORDS)

btree_test=# SELECT * FROM "transactions";
id | numeric | text

2 | 4 | str2
3 | 5 | str3
4 | 5 | str4
5 | 5 | str5
(4 RECORDS)

btree_test=#

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

SQL Shell (psql)

Server [localhost]: localhost
Database [postgres]: btree_test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (14.0)
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.

btree_test=# START TRANSACTION;
START TRANSACTION
btree_test=# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ READ WRITE;
SET
btree_test=# UPDATE "transactions" SET "numeric" = "numeric" + 1;
UPDATE 4
btree_test=#

SQL Shell (psql)

Server [localhost]: localhost
Database [postgres]: btree_test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (14.0)
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.

btree_test=# START TRANSACTION;
START TRANSACTION
btree_test=# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ READ WRITE;
SET
btree_test=# UPDATE "transactions" SET "numeric" = "numeric" + 1;

```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: btree_test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (14.0)
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.

btree_test=# START TRANSACTION;
START TRANSACTION
btree_test=# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ READ WRITE;
SET
btree_test=# UPDATE "transactions" SET "numeric" = "numeric" + 1;
UPDATE 4
btree_test=# COMMIT;
COMMIT
btree_test=#
```

```
SQL Shell (psql)
Server [localhost]: localhost
Database [postgres]: btree_test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (14.0)
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.

btree_test=# START TRANSACTION;
START TRANSACTION
btree_test=# SET TRANSACTION ISOLATION LEVEL REPEATABLE READ READ WRITE;
SET
btree_test=# UPDATE "transactions" SET "numeric" = "numeric" + 1;
ОШИБКА: не удалось сериализовать доступ из-за параллельного изменения
btree_test=#
```

Цей рівень ізоляції призначений для попередження повторного читання.

SERIALIZABLE

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

```
SQL Shell (psql)
Database [postgres]: btree_test
Port [5432]: 5432
Username [postgres]: postgres
Пароль пользователя postgres:
psql (14.0)
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.

btree_test=# START TRANSACTION;
START TRANSACTION
btree_test=# SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ WRITE;
SET
btree_test=# UPDATE "transactions" SET "numeric" = "numeric" + 1;
UPDATE 4
btree_test=# INSERT INTO "transactions"("numeric", "text") VALUES (8, 'str8');
INSERT 0 1
btree_test=# DELETE FROM "transactions" WHERE "id"=2;
DELETE 1
btree_test=#
```

```
SQL Shell (psql)
btree_test=# SET
btree_test=# SELECT * FROM "transactions";
 id | numeric | text
-----
 2 |      5 | str2
 3 |      6 | str3
 4 |      6 | str4
 5 |      6 | str5
(4 строк)

btree_test=# SELECT * FROM "transactions";
 id | numeric | text
-----
 2 |      5 | str2
 3 |      6 | str3
 4 |      6 | str4
 5 |      6 | str5
(4 строк)

btree_test=# UPDATE "transactions" SET "numeric" = "numeric" + 1;
```

```
SQL Shell (psql)
8-битовые (русские) символы могут отображаться некорректно.
Подробнее об этом смотрите документацию psql, раздел
"Notes for Windows users".
Введите "help", чтобы получить справку.

btree_test=# START TRANSACTION;
START TRANSACTION
btree_test=# SET TRANSACTION ISOLATION LEVEL SERIALIZABLE READ WRITE;
SET
btree_test=# UPDATE "transactions" SET "numeric" = "numeric" + 1;
UPDATE 4
btree_test=# INSERT INTO "transactions"("numeric", "text") VALUES (8, 'str8');
INSERT 0 1
btree_test=# DELETE FROM "transactions" WHERE "id"=2;
DELETE 1
btree_test=# COMMIT
btree_test=# COMMIT;
ОШИБКА: ошибка синтаксиса (примерное положение: "COMMIT")
СТРОКА 2: COMMIT;
      ^
btree_test=# COMMIT;
ROLLBACK
btree_test=#
```

```
SQL Shell (psql)
btree_test=# SET
btree_test=# SELECT * FROM "transactions";
 id | numeric | text
-----
 3 |      6 | str3
 4 |      6 | str4
 5 |      6 | str5
(4 строк)

btree_test=# SELECT * FROM "transactions";
 id | numeric | text
-----
 2 |      5 | str2
 3 |      6 | str3
 4 |      6 | str4
 5 |      6 | str5
(4 строк)

btree_test=# UPDATE "transactions" SET "numeric" = "numeric" + 1;
UPDATE 4
btree_test=#
```

В класичному представленні цей рівень призначений для недопущення явища читання фантомів.