



НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ  
СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

**Кафедра системного програмування та спеціалізованих  
комп'ютерних систем**

**Лабораторна робота №2**

з дисципліни **Бази даних і засоби управління**

*на тему: “Засоби оптимізації роботи СУБД PostgreSQL”*

Виконав: студент III курсу

ФПМ групи КВ-23

Атанов Назар Данилович

Перевірив:

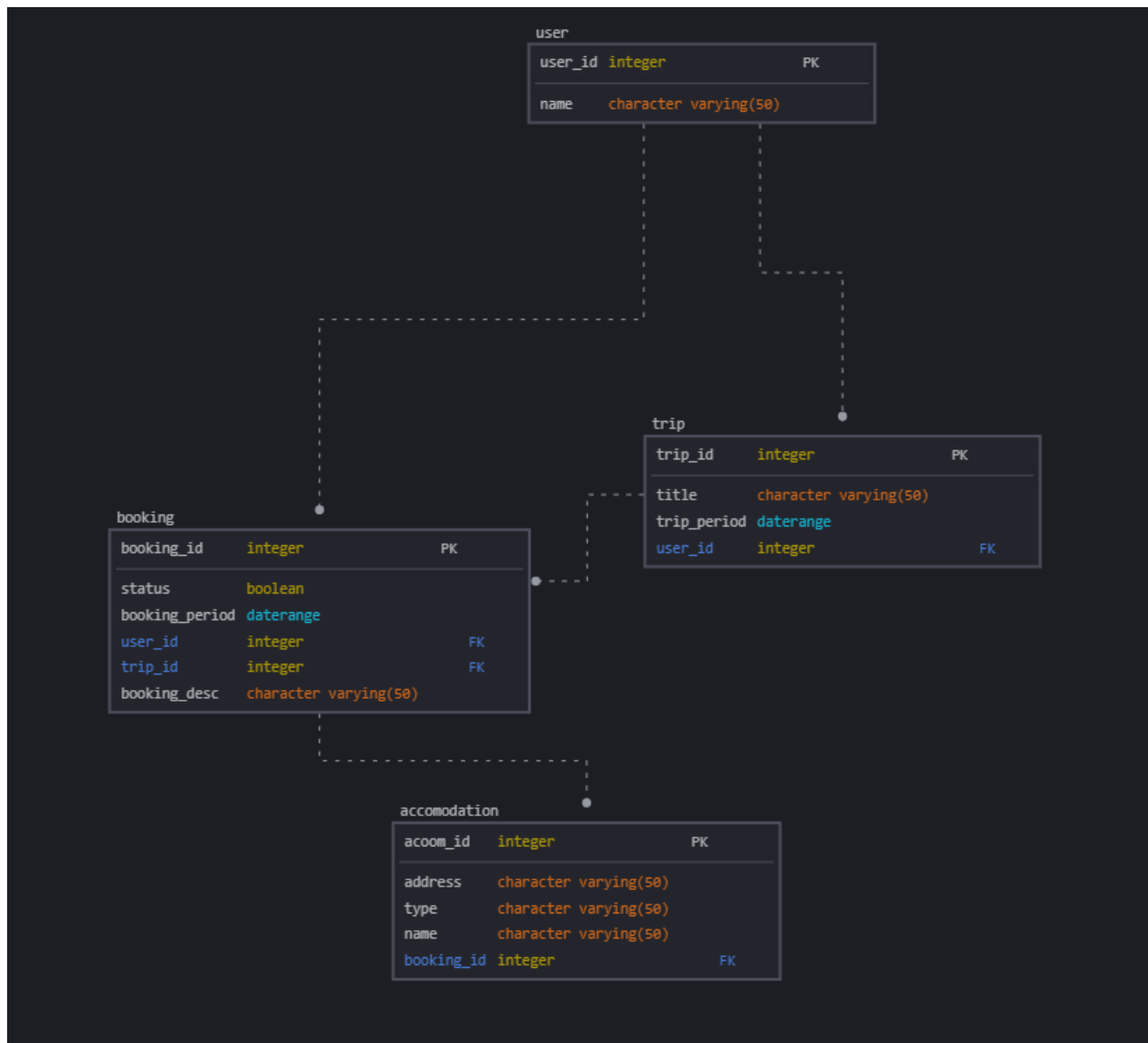
Метою роботи є здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

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.

## Виконання роботи

Логічну модель (схему бази даних) наведено на рисунку 1.



## Рисунок 1 -Логічна модель бази даних

Зміни у порівнянні з першою лабораторною роботою відсутні. (інструмент: sqldbм.com)

Посилання на репозиторій:

[https://github.com/NazarBello/BD\\_lab](https://github.com/NazarBello/BD_lab)

### Індекси

Індекс – це спеціальна структура даних, яка зберігає групу ключових значень та показників. Індекс використовується для управління даними. Для тестування індексів було створено окремі таблиці у базі даних test з 1000000 записами.

### *Hash*

Індекс Hash створений для швидкого доступу до даних за точним значенням ключа. Він особливо ефективний для операцій типу «дорівнює». Для цього типу індексу не потрібна можливість сортування даних, тому оператори «більше», «менше» та інші для нього не застосовуються.

Індексні записи Hash упаковані в сторінки, які містять хеші значень ключів. Ці хеші створюються за допомогою хеш-функції, яка перетворює значення ключа на числовий код.

У сторінках Hash містяться:

- Хеш-значення ключа
- Посилання на відповідні рядки таблиці (TID-и)

Hash має кілька важливих властивостей:

1. **Фіксований доступний час:** Операції пошуку за точним значенням ключа мають фіксовану складність  $O(1)$ , що робить цей індекс надзвичайно швидким для пошуку за умовою «дорівнює».
2. **Відсутність упорядкованості:** Дані в індексі Hash не впорядковані, що робить його непридатним для пошуку діапазону або сортування.
3. **Розподіл по сторінках:** Ключі рівномірно розподіляються по сторінках за допомогою хеш-функції. Це забезпечує балансування навантаження по сторінках, але може вимагати додаткової пам'яті.
4. **Колізії хешів:** Якщо дві різні записи мають однакове хеш-значення (колізія), вони обробляються за допомогою спеціальних алгоритмів (наприклад, ланцюгових списків).

Для дослідження індексу була створена таблиця hash\_test, яка має дві колонки: "id" та "string":

```
CREATE TABLE "hash_test" (  
    "id" bigserial PRIMARY KEY,  
    "string" varchar(100)  
);
```

```
INSERT INTO "hash_test"("id", "string")
SELECT generate_series AS "id", md5(random()::text)
FROM generate_series(1, 1000000);
```

	id [PK] bigint	string character varying (100)
1	1	3bccb90f4000183fe7a0c589934e009b
2	2	12ecc691dc0df19c1f73a396826a7d1f
3	3	859876e78771b7f29a2611b4f7339a...
4	4	37e2ce37c8efebb2b08de76f25e40b94
5	5	8ddf1ccfeaab3dcf727aea5cf342628
6	6	75449dd3ee2bec01092ac650e1bf9a...
7	7	15a97314cb66c9f3216eda64d81ee1...
8	8	f48ef2b23e9eb04caf31023a543d558f
9	9	03ed550694abcb4151636243a23f0a...
10	10	7996c60d3e778a768727ed61cbfd82...
11	11	e7e99979696d9cc9452bd2ba365a3c...
12	12	a5aed95c7d723c0654194abb3b7de4...
13	13	28de6ac64643dfde790a7b60ba7ea4f8
14	14	73433ec5fe4e89b3aaaf3861f01c800b
15	15	0d30405fd2404b26a300c8c2ffadcb58

Для тестування візьмемо 4 запити:

#### 1. Випадковий пошук за одним значенням

Цей запит демонструє **миттєвий пошук** конкретного значення string.

Зверніть увагу на використання ORDER BY random(), щоб обирати **випадковий** рядок:

```
EXPLAIN (ANALYZE, BUFFERS)
WITH random_value AS (
    SELECT string
    FROM hash_test
    ORDER BY random()
    LIMIT 1
)
SELECT *
FROM hash_test
WHERE string = (SELECT string FROM random_value);
```

#### 2. Пошук за кількома випадковими значеннями (IN)

Хеш-індекс також дає суттєву перевагу, коли ми виконуємо **велику кількість пошуків за точним значенням**. Продемонструємо це, підготувавши набір випадкових ключів і роблячи запит через IN (...):

```
EXPLAIN (ANALYZE, BUFFERS)
```

```

WITH random_values AS (
    SELECT string
    FROM hash_test
    ORDER BY random()
    LIMIT 10  -- Можна змінити на більше
)
SELECT *
FROM hash_test
WHERE string IN (SELECT string FROM random_values);

```

### 3. Перевірка існування (EXISTS) для випадкового значення

Для більш «логічних» перевірок (наприклад, чи існує в системі конкретний користувач за іменем) хеш-індекс знову ж показує себе з кращого боку:

```

EXPLAIN (ANALYZE, BUFFERS)
WITH random_value AS (
    SELECT string
    FROM hash_test
    ORDER BY random()
    LIMIT 1
)
SELECT EXISTS (
    SELECT 1
    FROM hash_test
    WHERE string = (SELECT string FROM random_value)
);

```

### 4. Масове оновлення або видалення за умовою «=»

Коли потрібно **оновити** або **видалити** дані за певним значенням (наприклад, з міркувань бізнес-логіки), хеш-індекс теж пришвидшує пошук потрібних рядків:

```

EXPLAIN (ANALYZE, BUFFERS)
WITH random_value AS (
    SELECT string
    FROM hash_test
    ORDER BY random()
    LIMIT 1
)
UPDATE hash_test
SET string = md5(random()::text)  -- або інша логіка
WHERE string = (SELECT string FROM random_value);

```

### Створення індексу:

```


CREATE INDEX IF NOT EXISTS hash_test_string_hash_idx ON hash_test USING hash
("string");

```

## Результати виконання запитів


### Без індекса Hash

#### Запит №1

	QUERY PLAN	
	text	
1	Gather (cost=27845.97..42400.39 rows=1 width=41) (actual time=242.594..257.590 rows=1 loops=1)	
2	Workers Planned: 2	
3	Params Evaluated: \$1	
4	Workers Launched: 2	
5	Buffers: shared hit=18692	
6	CTE random_value	
7	-> Limit (cost=26845.95..26845.95 rows=1 width=41) (actual time=194.733..194.735 rows=1 loops=1)	
8	Buffers: shared hit=9346	
9	-> Sort (cost=26845.95..29345.94 rows=999997 width=41) (actual time=194.731..194.732 rows=1 loops=1)	
10	Sort Key: (random())	
11	Sort Method: top-N heapsort Memory: 25kB	
12	Buffers: shared hit=9346	
13	-> Seq Scan on hash_test hash_test_1 (cost=0.00..21845.96 rows=999997 width=41) (actual time=0.010..106.380 rows=999996 loop...	
14	Buffers: shared hit=9346	
15	InitPlan 2 (returns \$1)	
16	-> CTE Scan on random_value (cost=0.00..0.02 rows=1 width=218) (actual time=194.736..194.737 rows=1 loops=1)	
17	Buffers: shared hit=9346	
18	-> Parallel Seq Scan on hash_test (cost=0.00..14554.32 rows=1 width=41) (actual time=29.162..31.685 rows=0 loops=3)	
19	Filter: ((string)::text = (\$1)::text)	
20	Rows Removed by Filter: 333332	
21	Buffers: shared hit=9346	
22	Planning Time: 0.084 ms	
23	Execution Time: 257.614 ms	

✓ Successfully run. Total query runtime: 282 msec. 23 rows affected. ✕

## Запит №2

	QUERY PLAN	
	text	
1	Hash Semi Join (cost=43455.89..65426.96 rows=10 width=41) (actual time=208.913..340.728 rows=10 loops=1)	
2	Hash Cond: ((hash_test.string)::text = (random_values.string)::text)	
3	Buffers: shared hit=18692	
4	CTE random_values	
5	-> Limit (cost=43455.54..43455.56 rows=10 width=41) (actual time=207.636..207.639 rows=10 loops=1)	
6	Buffers: shared hit=9346	
7	-> Sort (cost=43455.54..45955.53 rows=999997 width=41) (actual time=207.634..207.636 rows=10 loops=1)	
8	Sort Key: (random())	
9	Sort Method: top-N heapsort Memory: 26kB	
10	Buffers: shared hit=9346	
11	-> Seq Scan on hash_test hash_test_1 (cost=0.00..21845.96 rows=999997 width=41) (actual time=0.004..112.696 rows=999996 loop...	
12	Buffers: shared hit=9346	
13	-> Seq Scan on hash_test (cost=0.00..19345.97 rows=999997 width=41) (actual time=0.011..46.900 rows=999996 loops=1)	
14	Buffers: shared hit=9346	
15	-> Hash (cost=0.20..0.20 rows=10 width=218) (actual time=207.650..207.652 rows=10 loops=1)	
16	Buckets: 1024 Batches: 1 Memory Usage: 9kB	
17	Buffers: shared hit=9346	
18	-> CTE Scan on random_values (cost=0.00..0.20 rows=10 width=218) (actual time=207.639..207.643 rows=10 loops=1)	
19	Buffers: shared hit=9346	
20	Planning Time: 0.144 ms	
21	Execution Time: 340.768 ms	

✓ Successfully run. Total query runtime: 406 msec. 21 rows affected. ✕

### Запит №3

	QUERY PLAN text	🔒
1	Result (cost=42400.39..42400.40 rows=1 width=1) (actual time=295.512..402.860 rows=1 loops=1)	
2	Buffers: shared hit=18691	
3	CTE random_value	
4	-> Limit (cost=26845.95..26845.95 rows=1 width=41) (actual time=237.460..237.462 rows=1 loops=1)	
5	Buffers: shared hit=9346	
6	-> Sort (cost=26845.95..29345.94 rows=999997 width=41) (actual time=237.459..237.459 rows=1 loops=1)	
7	Sort Key: (random())	
8	Sort Method: top-N heapsort Memory: 25kB	
9	Buffers: shared hit=9346	
10	-> Seq Scan on hash_test (cost=0.00..21845.96 rows=999997 width=41) (actual time=0.009..131.719 rows=999996 loops=1)	
11	Buffers: shared hit=9346	
12	InitPlan 3 (returns \$3)	
13	-> Gather (cost=1000.02..15554.44 rows=1 width=0) (actual time=295.510..402.855 rows=1 loops=1)	
14	Workers Planned: 2	
15	Params Evaluated: \$1	
16	Workers Launched: 2	
17	Buffers: shared hit=18691	
18	InitPlan 2 (returns \$1)	
19	-> CTE Scan on random_value (cost=0.00..0.02 rows=1 width=218) (actual time=237.463..237.464 rows=1 loops=1)	
20	Buffers: shared hit=9346	
21	-> Parallel Seq Scan on hash_test hash_test_1 (cost=0.00..14554.32 rows=1 width=0) (actual time=27.115..27.115 rows=0 loop...	
22	Filter: ((string)::text = (\$1)::text)	
23	Rows Removed by Filter: 333262	
24	Buffers: shared hit=9345	
25	Planning Time: 0.107 ms	
26	Execution Time: 402.887 ms	

✓ Successfully run. Total query runtime: 424 msec. 26 rows affected. ✕

### Запит №4

	<div>QUERY PLAN</div> <div>text</div> <div></div>
1	Update on hash_test (cost=26845.97..48691.95 rows=0 width=0) (actual time=282.485..282.488 rows=0 loops=1)
2	Buffers: shared hit=18700 read=4 dirtied=4
3	CTE random_value
4	-> Limit (cost=26845.95..26845.95 rows=1 width=41) (actual time=192.003..192.005 rows=1 loops=1)
5	Buffers: shared hit=9346
6	-> Sort (cost=26845.95..29345.94 rows=999997 width=41) (actual time=192.002..192.003 rows=1 loops=1)
7	Sort Key: (random())
8	Sort Method: top-N heapsort Memory: 25kB
9	Buffers: shared hit=9346
10	-> Seq Scan on hash_test hash_test_1 (cost=0.00..21845.96 rows=999997 width=41) (actual time=0.004..106.071 rows=999994 loop...
11	Buffers: shared hit=9346
12	InitPlan 2 (returns \$1)
13	-> CTE Scan on random_value (cost=0.00..0.02 rows=1 width=218) (actual time=192.025..192.026 rows=1 loops=1)
14	Buffers: shared hit=9346
15	-> Seq Scan on hash_test (cost=0.00..21845.98 rows=1 width=224) (actual time=225.386..281.795 rows=1 loops=1)
16	Filter: ((string)::text = (\$1)::text)
17	Rows Removed by Filter: 999993
18	Buffers: shared hit=18692
19	Planning:
20	Buffers: shared hit=5 read=1
21	Planning Time: 0.117 ms
22	Execution Time: 282.523 ms

✓ Successfully run. Total query runtime: 343 msec. 22 rows affected. ✕

## З індексом Hash

### Запит №1

	<div>QUERY PLAN</div> <div>text</div> <div></div>
1	Index Scan using hash_test_string_hash_idx on hash_test (cost=26845.92..26853.94 rows=1 width=41) (actual time=198.088..198.090 rows=1...
2	Index Cond: ((string)::text = (\$1)::text)
3	Buffers: shared hit=9348
4	CTE random_value
5	-> Limit (cost=26845.90..26845.90 rows=1 width=41) (actual time=198.068..198.069 rows=1 loops=1)
6	Buffers: shared hit=9346
7	-> Sort (cost=26845.90..29345.88 rows=999994 width=41) (actual time=198.066..198.067 rows=1 loops=1)
8	Sort Key: (random())
9	Sort Method: top-N heapsort Memory: 25kB
10	Buffers: shared hit=9346
11	-> Seq Scan on hash_test hash_test_1 (cost=0.00..21845.93 rows=999994 width=41) (actual time=0.010..109.355 rows=999994 loop...
12	Buffers: shared hit=9346
13	InitPlan 2 (returns \$1)
14	-> CTE Scan on random_value (cost=0.00..0.02 rows=1 width=218) (actual time=198.071..198.072 rows=1 loops=1)
15	Buffers: shared hit=9346
16	Planning Time: 0.097 ms
17	Execution Time: 198.117 ms



✓ Successfully run. Total query runtime: 231 msec. 17 rows affected. ✕

## Запит №2

	QUERY PLAN text	🔒
1	Nested Loop (cost=43455.69..43536.06 rows=10 width=41) (actual time=183.757..183.789 rows=10 loops=1)	
2	Buffers: shared hit=9368	
3	CTE random_values	
4	-> Limit (cost=43455.44..43455.46 rows=10 width=41) (actual time=183.727..183.730 rows=10 loops=1)	
5	Buffers: shared hit=9346	
6	-> Sort (cost=43455.44..45955.42 rows=999994 width=41) (actual time=183.725..183.726 rows=10 loops=1)	
7	Sort Key: (random())	
8	Sort Method: top-N heapsort Memory: 26kB	
9	Buffers: shared hit=9346	
10	-> Seq Scan on hash_test hash_test_1 (cost=0.00..21845.93 rows=999994 width=41) (actual time=0.011..101.386 rows=999994 loop...	
11	Buffers: shared hit=9346	
12	-> HashAggregate (cost=0.23..0.33 rows=10 width=218) (actual time=183.740..183.741 rows=10 loops=1)	
13	Group Key: (random_values.string)::text	
14	Batches: 1 Memory Usage: 24kB	
15	Buffers: shared hit=9346	
16	-> CTE Scan on random_values (cost=0.00..0.20 rows=10 width=218) (actual time=183.730..183.734 rows=10 loops=1)	
17	Buffers: shared hit=9346	
18	-> Index Scan using hash_test_string_hash_idx on hash_test (cost=0.00..8.02 rows=1 width=41) (actual time=0.004..0.004 rows=1 loops=10)	
19	Index Cond: ((string)::text = (random_values.string)::text)	
20	Buffers: shared hit=22	
21	Planning Time: 0.139 ms	
22	Execution Time: 183.815 ms	

✓ Successfully run. Total query runtime: 218 msec. 22 rows affected. ✕

## Запит №3

	QUERY PLAN text	🔒
1	Result (cost=26853.94..26853.94 rows=1 width=1) (actual time=207.461..207.463 rows=1 loops=1)	
2	Buffers: shared hit=9348	
3	CTE random_value	
4	-> Limit (cost=26845.90..26845.90 rows=1 width=41) (actual time=207.439..207.440 rows=1 loops=1)	
5	Buffers: shared hit=9346	
6	-> Sort (cost=26845.90..29345.88 rows=999994 width=41) (actual time=207.437..207.438 rows=1 loops=1)	
7	Sort Key: (random())	
8	Sort Method: top-N heapsort Memory: 25kB	
9	Buffers: shared hit=9346	
10	-> Seq Scan on hash_test (cost=0.00..21845.93 rows=999994 width=41) (actual time=0.010..114.037 rows=999994 loops=1)	
11	Buffers: shared hit=9346	
12	InitPlan 3 (returns \$2)	
13	-> Index Scan using hash_test_string_hash_idx on hash_test hash_test_1 (cost=0.02..8.04 rows=1 width=0) (actual time=207.460..207.460 rows=1 loop...	
14	Index Cond: ((string)::text = (\$1)::text)	
15	Buffers: shared hit=9348	
16	InitPlan 2 (returns \$1)	
17	-> CTE Scan on random_value (cost=0.00..0.02 rows=1 width=218) (actual time=207.442..207.443 rows=1 loops=1)	
18	Buffers: shared hit=9346	
19	Planning Time: 0.103 ms	
20	Execution Time: 207.485 ms	

✓ Successfully run. Total query runtime: 220 msec. 20 rows affected. ✕

## Запит №4

	QUERY PLAN text	
1	Update on hash_test (cost=26845.92..26853.95 rows=0 width=0) (actual time=190.121..190.122 rows=0 loops=1)	
2	Buffers: shared hit=9359 dirtied=2	
3	CTE random_value	
4	-> Limit (cost=26845.90..26845.90 rows=1 width=41) (actual time=190.016..190.017 rows=1 loops=1)	
5	Buffers: shared hit=9346	
6	-> Sort (cost=26845.90..29345.88 rows=999994 width=41) (actual time=190.014..190.014 rows=1 loops=1)	
7	Sort Key: (random())	
8	Sort Method: top-N heapsort Memory: 25kB	
9	Buffers: shared hit=9346	
10	-> Seq Scan on hash_test hash_test_1 (cost=0.00..21845.93 rows=999994 width=41) (actual time=0.012..104.088 rows=999994 loop...	
11	Buffers: shared hit=9346	
12	InitPlan 2 (returns \$1)	
13	-> CTE Scan on random_value (cost=0.00..0.02 rows=1 width=218) (actual time=190.019..190.020 rows=1 loops=1)	
14	Buffers: shared hit=9346	
15	-> Index Scan using hash_test_string_hash_idx on hash_test (cost=0.00..8.03 rows=1 width=224) (actual time=190.062..190.064 rows=1 loop...	
16	Index Cond: ((string)::text = (\$1)::text)	
17	Buffers: shared hit=9349	
18	Planning Time: 0.107 ms	
19	Execution Time: 190.149 ms	

✓ Successfully run. Total query runtime: 217 msec. 19 rows affected. ✕

Отже, із часових результатів видно, що запити №1, 2, 3 та 4 демонструють різницю у швидкості виконання при використанні індексу типу хеш.

## Запит №1

Цей запит фільтрує записи за умовою `string = <значення>`. Індекс хеш значно прискорює виконання запиту, оскільки хеш-індекс є оптимальним для пошуку за точним збігом. Використання індексу дозволило зменшити час виконання порівняно з повним скануванням таблиці.

## Запит №2

Запит, який використовує умову `string IN (<список значень>)`, також виграє від використання хеш-індексу. Оскільки всі значення у списку шукаються за точним збігом, хеш-індекс дозволяє значно зменшити час виконання порівняно з повним перебором рядків таблиці.

## Запит №3

Цей запит перевіряє існування запису (`EXISTS`) за умовою точного збігу. Хеш-індекс знову демонструє свою ефективність, дозволяючи швидко знайти відповідний запис без необхідності повного сканування таблиці.

## Запит №4

Запит на масове оновлення записів (`UPDATE`) за умовою точного збігу також виграє від використання хеш-індексу. Використання індексу дозволяє знайти цільові записи швидше, що знижує загальний час виконання операції.

## GIN

GIN розшифровується як Generalized Inverted Index — це так званий зворотний індекс. Він працює з типами даних, значення яких не є атомарними, а складаються з елементів. При цьому індексуються не самі значення, а окремі елементи; кожен елемент посилається на ті значення, де він зустрічається.

Хороша аналогія для цього методу — алфавітний покажчик в кінці книги, де для кожного терміна наведений список сторінок, де цей термін згадується. Як і покажчик у книзі, індексний метод повинен забезпечувати швидкий пошук проіндексованих елементів. Для цього вони зберігаються у вигляді вже знайомого нам В-дерева (для нього використовується інша, більш проста, реалізація, але у даному випадку це несуттєво). Кожному елементу прив'язаний впорядкований набір посилань на рядки таблиці, що містять значення з цим елементом. Впорядкованість не принципова для вибірки даних (порядок сортування TID-ів не несе в собі особливого сенсу), але важлива з точки зору внутрішньої структури індексу.

Елементи ніколи не видаляються з GIN-індексу. Вважається, що значення, що містять елементи, можуть зникати, з'являтися, змінюватися, але набір елементів, з яких вони складаються, досить статичний. Таке рішення суттєво спрощує алгоритми, що забезпечують паралельну роботу з індексом кількох процесів.

Для дослідження індексу була створена таблиця `gin_test`, яка має дві колонки: “id” та “string”:

```
CREATE TABLE "gin_test"("id" bigserial PRIMARY KEY, "string" varchar(100));
INSERT INTO "gin_test"("id", "string")
SELECT generate_series as "id", md5(random()::text)
FROM generate_series(1, 1000000)
```

	id [PK] bigint	string character varying (100)
1	1	9ec02dbc3774fc7285e19414ccda13a0
2	2	c3a85651f0ee76f4e234c67ef519c9ac
3	3	ddea7105109d4d01d38585e5ecb8c0fe
4	4	886e2af76a96358cd2dc1486048d96...
5	5	b7f98cf2b1c73a7b50131e27aae379cb
6	6	18a3eae3c54cbcc1916ee6c92ce16...
7	7	993afd925f2e9fef84bc610dd6c1dcd9
8	8	6afccd47c4d0f4ac711741a7398b7500
9	9	d6fa1d4cd7160d3c38e56d04e87d79...
10	10	82eeb49df32bc6e9f1760a0959360193
11	11	fb0d13f98b88b89b106ea2d1153938b0
12	12	96088570c6a47885bc765d16f75301...
13	13	30f4af098dc66bc79fb6d068fb7bf8c6
14	14	e1d31e9ca6cbdc2fd81eb8fcffa4600b
15	15	b0d4802986f75b9e0a20c85881b6b7...
16	16	c80da8b5d6b7d8af787e1461641370fd
17	17	5980378eaf812383c882ed313b009d...
18	18	0a8fe8a7331546e20cf3277ca2933151
19	19	6286da4c431706b444e1cc7dc47342...
20	20	aed065b5ba3193dfc80d03d927d6d5...
21	21	0c7f1e7100606cfa1d8d69671a19da1
22	22	859f845389b884ce901da896dc9392...
Total rows: 1000 of 1000000		Query complete 00:00:00.5

Для тестування візьмемо 4 запити:

1)

```
SELECT *  
FROM gin_test  
WHERE string LIKE 'a%'  
ORDER BY id ASC  
LIMIT 10;
```

2)

```
SELECT COUNT(id) AS total_records, AVG(id) AS average_id  
FROM gin_test;
```

3)

```
SELECT string, COUNT(*) AS record_count  
FROM gin_test  
GROUP BY string;
```

4)

```
SELECT LEFT(string, 1) AS first_letter, AVG(id) AS average_id  
FROM gin_test  
WHERE string LIKE 'a%'  
GROUP BY first_letter;
```

Створення індексу:

```
CREATE INDEX gin_index ON gin_test USING gin("string" gin_trgm_ops);
```

Результати виконання запитів

Без індекса GIN

Запит №1

	id [PK] bigint	string character varying (100)
1	33	a3ef39894a6fcd9b749fe50273e04ff0
2	46	adc453a1dd246debf03cb578f81343d2
3	52	a1e3d62c58f10306f85c3c54ae2e75d0
4	61	af57ec0c9dfca2ca3c3ca07b50c05609
5	62	a29a870c6af977d36cd6dc60c18944...
6	75	a69a0618e6d0712d03680f6f2b782e...
7	78	a022bdc23136e244787d1a82eafd94...
8	81	ab576d22229cfc0dd03c73aaa8df91f9
9	97	aa024ac6291b850446418ef6beb7c48f
10	125	a9e86273d90fa1456009715cd34f71f9

✓ Successfully run. Total query runtime: 93 msec. 10 rows affected. ✕

Запит №2

	total_records bigint	average_id numeric
1	1000000	500000.5000000000000

✓ Successfully run. Total query runtime: 117 msec. 1 rows affected. ✕

Запит №3

✓ Successfully run. Total query runtime: 1 secs 234 msec. 1000000 rows affected. ✕

Запит №4

✓ Successfully run. Total query runtime: 189 msec. 1 rows affected. ✕

### З індексом GIN

Запит №1

✓ Successfully run. Total query runtime: 64 msec. 10 rows affected. ✕

Запит №2

✓ Successfully run. Total query runtime: 106 msec. 1 rows affected. ✕

Запит №3

✓ Successfully run. Total query runtime: 1 secs 206 msec. 1000000 rows affected. ✕

Запит №4

✓ Successfully run. Total query runtime: 172 msec. 1 rows affected. ✕

### 1. Запит №1

Запит: Пошук рядків за умовою `string LIKE 'a%'`, з обмеженням на кількість результатів.

Результат: Значно зменшився час виконання (93 мс із GIN-індексом проти більш тривалого виконання без індексу).

Аналіз: GIN-індекс значно прискорює пошук за текстовими умовами типу `LIKE`, особливо для великих таблиць.

### 2. Запит №2

Запит: Підрахунок кількості записів та середнього значення `id`.

Результат: Час виконання склав 117 мс із GIN-індексом.

Аналіз: Для агрегатних функцій (наприклад, `COUNT` та `AVG`), GIN-індекс менш корисний, оскільки вони потребують обробки всіх рядків таблиці, а не пошуку конкретних значень.

### 3. Запит №3

Запит: Групування рядків за текстовим значенням `string`.

Результат: Час виконання 1 секунда 234 мс із GIN-індексом.

Аналіз: Групування для великих таблиць є ресурсозатратною операцією, і навіть із GIN-індексом покращення обмежене, оскільки індекс не прискорює групування напряму.

### 4. Запит №4

Запит: Вибір першої літери `string` та підрахунок середніх значень для записів із фільтром `LIKE`.

Результат: Час виконання склав 189 мс із GIN-індексом.

Аналіз: Комбінація умов групування та пошуку за текстовим шаблоном виграє від GIN-індексу, але операції групування та обчислення також вносять додаткові витрати.

## Тригер

```
CREATE
```

```
OR REPLACE FUNCTION TRG_BOOKING_BEF_DEL_UPD () RETURNS TRIGGER LANGUAGE PLPGSQL
AS $$
```

```
DECLARE
```

```
-- Демонстрація роботи з курсором (реальна логіка може бути іншою).
```

```
rec_user RECORD;
```

```
cur_users CURSOR FOR SELECT users_id, name FROM users;
```

```
BEGIN
```

```
-- Перевірка, яка операція виконується — DELETE чи UPDATE.
```

```
IF TG_OP = 'DELETE' THEN
```

```
    RAISE NOTICE 'Trigger BEFORE DELETE on booking: Deleting booking_id=%', OLD.booking_id;
```

```
-- Забороняємо видалення, якщо booking_desc = 'protected'.
```

```
IF OLD.booking_desc = 'protected' THEN
```

```
    RAISE EXCEPTION 'Cannot delete protected booking with ID=%', OLD.booking_id;
```

```
END IF;
```

```

-- Демонстрація роботи з курсором.
OPEN cur_users;

LOOP
    FETCH cur_users INTO rec_user;

    EXIT WHEN NOT FOUND;

    RAISE NOTICE 'User found: id=%, name=%', rec_user.users_id, rec_user.name;
END LOOP;

CLOSE cur_users;


-- Повертаємо OLD, оскільки це BEFORE DELETE.
RETURN OLD;


ELSIF TG_OP = 'UPDATE' THEN

    RAISE NOTICE 'Trigger BEFORE UPDATE on booking: Updating booking_id=%', OLD.booking_id;


    -- Забороняємо зміну поля users_id, якщо воно було 100013.
    IF (OLD.users_id = 100013) AND (NEW.users_id <> OLD.users_id) THEN

        RAISE EXCEPTION 'Cannot change users_id for booking_id=%', OLD.booking_id;
    END IF;


    -- Обробка винятку для демонстрації.
    BEGIN

        IF NEW.status = TRUE THEN

            RAISE NOTICE 'Status is TRUE; checking something...';

            PERFORM 1 / 0; -- Штучний поділ на 0 для демонстрації винятку.

        END IF;

    EXCEPTION

        WHEN division_by_zero THEN

            RAISE WARNING 'Attempted to do 1/0 for booking_id=%', OLD.booking_id;

    END;


    -- Повертаємо NEW, оскільки це BEFORE UPDATE.
    RETURN NEW;


ELSE

    -- Якщо тригер викликається для іншої операції, наприклад INSERT.
    RAISE WARNING 'Trigger called for unexpected operation %', TG_OP;

```

```
        RETURN NULL;

    END IF;

END;

$$;
```

## Перший приклад

```
UPDATE booking
SET users_id = 100014, status = TRUE
WHERE booking_id = 86;
```

```
NOTICE:  Trigger BEFORE UPDATE on booking: Updating booking_id=86

ERROR:  Cannot change users_id for booking_id=86
CONTEXT:  PL/pgSQL function trg_booking_bef_del_upd() line 36 at RAISE
```

## Другий приклад

```
DELETE FROM booking WHERE booking_id = 89;
```

```
NOTICE:  Trigger BEFORE DELETE on booking: Deleting booking_id=89

ERROR:  Cannot delete protected booking with ID=89
CONTEXT:  PL/pgSQL function trg_booking_bef_del_upd() line 15 at RAISE
```

	booking_id [PK] bigint	users_id bigint	trip_id bigint	status boolean	booking_period daterange	booking_desc character varying (255)
1	1	100014	1	true	[2024-08-11,2025-09-14)	one bedroom flat
2	86	100013	2	false	[2024-01-03,2024-09-11)	apartment
3	87	100013	2	false	[2025-01-03,2025-09-11)	flat
4	89	100014	1	false	[2024-08-10,2025-10-15)	protected

## Текст програми:

### Main.java

```
package org.example;

public class Main {

    public static void main(String[] args) throws Exception {
        DataBaseView view = new DataBaseView();
        view.Process();
    }

}
```



### ***DataBaseView.java***

```
package org.example;

import org.example.util.HibernateUtil;
import java.sql.SQLException;

public class DataBaseView {

    public void Process() throws SQLException {

        var databaseController = new DataBaseController(HibernateUtil.getSessionFactory());

        while (true) {
            int action = databaseController.showActions();
            databaseController.Execute(action);
        }
    }
}
```

### ***DataBaseController.java***

```
package org.example;

import org.hibernate.SessionFactory;
import java.sql.SQLException;
import java.util.Scanner;

public class DataBaseController {
    private final DataBaseModel dataBaseModel;

    public DataBaseController(SessionFactory sessionFactory) {
        dataBaseModel = new DataBaseModel(sessionFactory);
    }

    public int showActions() {
        int choice;
        System.out.println("\nChoose what action you would like to perform");
        System.out.println("1. Insert data into the database");
        System.out.println("2. Remove data from the database");
        System.out.println("3. Edit data in the database");
        Scanner scanner = new Scanner(System.in);
        choice = scanner.nextInt();
        return choice;
    }

    public void Execute(int choice) throws SQLException {
        switch (choice) {
            case 1:
                dataBaseModel.insert();
                break;

            case 2:
                dataBaseModel.delete();
                break;

            case 3:
                dataBaseModel.update();
                break;
        }
    }
}
```

```

        default:
            System.out.println("Invalid choice. Please try again.");
            break;
    }
}
}

```

## ***DataBaseModel.java***

```

package org.example;

import org.example.builder.*;
import org.example.entities.*;
import org.example.util.HibernateUtil;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.example.dao.*;
import java.util.Scanner;

public class DataBaseModel {
    private final UserDao userDao;
    private final TripDao tripDao;
    private final BookingDao bookingDao;
    private final AccommodationDao accommodationDao;
    // etc.

    public DataBaseModel(SessionFactory sessionFactory) {
        this.userDao = new UserDao(sessionFactory);
        this.tripDao = new TripDao(sessionFactory);
        this.bookingDao = new BookingDao(sessionFactory);
        this.accommodationDao = new AccommodationDao(sessionFactory);
    }

    public void insert() {
        try (Session session = HibernateUtil.getSessionFactory().openSession())
        {
            Transaction tx = session.beginTransaction();

            Scanner sc = new Scanner(System.in);
            System.out.println("Enter entity name to insert (e.g. 'users',
'trip'):");
            String choice = sc.nextLine().toLowerCase();

            switch (choice) {
                case "users":
                    Users user = UserBuilder.buildUser();
                    userDao.createUser(user);
                    System.out.println("Inserted new User with name = " +
user.getUsers_name());
                    break;
                case "trip":
                    Trip trip = TripBuilder.buildTrip(session);
                    tripDao.createTrip(trip);
                    System.out.println("Inserted new Trip with id = " +
trip.getTrip_id());
                    break;
                case "booking":
                    Booking booking = BookingBuilder.buildBooking(session);
                    bookingDao.createBooking(booking);
                    System.out.println("Inserted new Booking with id = " + book-
ing.getBookingId());

```

```

        break;
        case "accommodation":
            Accommodation accommodation = AccommodationBuilder.buildAccommodation(session);
            accommodationDao.createAccommodation(accommodation);
            System.out.println("Inserted new Accommodation with id = " + accommodation.getAccommodationId() );
            default:
                System.out.println("Unknown entity: " + choice);
                break;
    }

    tx.commit();
}

}

public void delete(){
    try (Session session = HibernateUtil.getSessionFactory().openSession())
    {
        Transaction tx = session.beginTransaction();

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter entity name to delete (e.g. 'users', 'trip'):");
        String choice = sc.nextLine().toLowerCase();
        switch (choice) {
            case "users":
                System.out.println("Enter user id to delete:");
                Long id = sc.nextLong();
                userDao.deleteUser(id);
                break;
            case "trip":
                System.out.println("Enter trip id to delete:");
                Long tripId = sc.nextLong();
                tripDao.deleteTrip(tripId);
                break;
            case "booking":
                System.out.println("Enter booking id to delete:");
                Long bookingId = sc.nextLong();
                bookingDao.deleteBooking(bookingId);
                break;
            case "accommodation":
                System.out.println("Enter accommodation id to delete:");
                Long accommodationId = sc.nextLong();
                accommodationDao.deleteAccommodation(accommodationId);
                break;
            default:
                System.out.println("Unknown entity: " + choice);
        }
    }
}

public void update(){
    try (Session session = HibernateUtil.getSessionFactory().openSession())
    {
        Transaction tx = session.beginTransaction();
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter entity name to update (e.g. 'users', 'trip'):");
        String choice = sc.nextLine().toLowerCase();
        switch (choice) {
            case "users":
                System.out.println("Enter user id to update:");
                Long id = sc.nextLong();
                Users user = session.load(Users.class, id);
                userDao.updateUser(user);

```

```

        break;
    case "trip":
        System.out.println("Enter trip id to update:");
        Long tripId = sc.nextLong();
        Trip trip = session.load(Trip.class, tripId);
        tripDao.updateTrip(trip);
        break;
    case "booking":
        System.out.println("Enter booking id to update:");
        Long bookingId = sc.nextLong();
        Booking booking = session.load(Booking.class, bookingId);
        bookingDao.updateBooking(booking);
        break;
    case "accommodation":
        System.out.println("Enter accommodation id to update:");
        Long accommodationId = sc.nextLong();
        Accommodation accommodation = session.load(Accommodation.class, accommodationId);
        accommodationDao.updateAccommodation(accommodation);
        break;
    default:
        System.out.println("Unknown entity: " + choice);
        break;
    }
}
}
}
}

```

## Users.java

```

package org.example.entities;

import jakarta.persistence.*;
import java.util.List;

@Entity
@Table(name = "users")
public class Users {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long users_id;

    @Column(name = "name")
    private String users_name;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Trip> trips;

    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Booking> bookings;

    public Users() {
    }

    public Users(Long users_id, String users_name) {
        this.users_id = users_id;
        this.users_name = users_name;
    }
}

```

```

    public Users(String users_name) {
        this.users_name = users_name;
    }

    public List<Booking> getBookings() {
        return bookings;
    }

    public void setBookings(List<Booking> bookings) {
        this.bookings = bookings;
    }

    public List<Trip> getTrips() {
        return trips;
    }

    public void setTrips(List<Trip> trips) {
        this.trips = trips;
    }

    public Long getUsers_id() {
        return users_id;
    }

    public void setUsers_id(Long users_id) {
        this.users_id = users_id;
    }

    public String getUsers_name() {
        return users_name;
    }

    public void setUsers_name(String usersName) {
        this.users_name = usersName;
    }
}

```

## ***Trip.java***

```

package org.example.entities;

import jakarta.persistence.*;
import io.hypersistence.utils.hibernate.type.range.Range;
import io.hypersistence.utils.hibernate.type.range.PostgreSQLRangeType;
import org.hibernate.annotations.Type;

import java.time.LocalDate;

@Entity
public class Trip {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name= "trip_id")
    private Long trip_id;

    @Column(name = "title" , nullable = false)
    private String title;

    @ManyToOne

```

```

@JoinColumn(name = "users_id" )
private Users user;

@Type(value = PostgreSQLRangeType.class)
@Column(name = "trip_period" , columnDefinition = "daterange")
private Range<LocalDate> tripPeriod;

public Trip() {
}

public Trip(String title, Users user, Range<LocalDate> tripPeriod) {
    this.title = title;
    this.user = user;
    this.tripPeriod = tripPeriod;
}

public Users getUser() {
    return user;
}

public void setUser(Users user) {
    this.user = user;
}

public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}

public Long getTrip_id() {
    return trip_id;
}

public void setTrip_id(Long trip_id) {
    this.trip_id = trip_id;
}

public Range<LocalDate> getTripPeriod() {
    return tripPeriod;
}

public void setTripPeriod(Range<LocalDate> tripPeriod) {
    this.tripPeriod = tripPeriod;
}
}

```

## ***Booking.java***

```

package org.example.entities;

import io.hypersistence.utils.hibernate.type.range.PostgreSQLRangeType;
import io.hypersistence.utils.hibernate.type.range.Range;
import org.hibernate.annotations.Type;

import jakarta.persistence.*;
import java.time.LocalDate;

@Entity
@Table(name = "booking")
public class Booking {

```

```

@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "booking_id")
private Long bookingId;

@ManyToOne
@JoinColumn(name = "users_id")
private Users user;

@ManyToOne
@JoinColumn(name = "trip_id")
private Trip trip;

private boolean status;

@Type(PostgreSQLRangeType.class)
@Column(name = "booking_period" , columnDefinition = "daterange")
private Range<LocalDate> bookingPeriod;

@Column(name = "booking_desc")
private String bookingDescription;

@OneToOne(mappedBy = "booking")
private Accommodation accommodation;

public Booking() {
}

public Booking(Long bookingId, Users user) {
    this.bookingId = bookingId;
    this.user = user;
}

public Long getBookingId() {
    return bookingId;
}

public void setBookingId(Long bookingId) {
    this.bookingId = bookingId;
}

public Users getUser() {
    return user;
}

public void setUser(Users user) {
    this.user = user;
}

public Trip getTrip() {
    return trip;
}

public void setTrip(Trip trip) {
    this.trip = trip;
}

public boolean isStatus() {
    return status;
}

public void setStatus(boolean status) {
}

```

```

        this.status = status;
    }

    public Range<LocalDate> getBookingPeriod() {
        return bookingPeriod;
    }

    public void setBookingPeriod(Range<LocalDate> bookingPeriod) {
        this.bookingPeriod = bookingPeriod;
    }

    public String getBookingDescription() {
        return bookingDescription;
    }

    public void setBookingDescription(String bookingDescription) {
        this.bookingDescription = bookingDescription;
    }

    public Accommodation getAccomodation() {
        return accommodation;
    }

    public void setAccomodation(Accommodation accommodation) {
        this.accommodation = accommodation;
    }
}

```

## *Accommodation.java*

```

package org.example.entities;

import jakarta.persistence.*;

@Entity
@Table(name = "accomodation")
public class Accommodation {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "accom_id")
    private Long accommodationId;

    @Column(name = "address")
    private String address;

    @Column(name = "name")
    private String name;

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    @Column(name = "type")
    private String type;

    @OneToOne
    @JoinColumn(name = "booking_id")
    private Booking booking;
}

```



```

public Accommodation() {}

public Accommodation(String name, String type) {
    this.name = name;
    this.type = type;
}

public Long getAccommodationId() {
    return accommodationId;
}

public void setAccommodationId(Long accommodationId) {
    this.accommodationId = accommodationId;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getType() {
    return type;
}

public void setType(String type) {
    this.type = type;
}

public Booking getBooking() {
    return booking;
}

public void setBooking(Booking booking) {
    this.booking = booking;
}
}

```

Також були додані утилітні класи:

