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

Факультет прикладної математики
Кафедра програмного забезпечення комп’ютерних систем

Лабораторна робота №3

з дисципліни “Бази даних”

“Засоби оптимізації роботи СУБД PostgreSQL”

Виконав
студент ІІІ курсу
групи КП-83
Симонюк Володимир Павлович
варіант № 18

Зарахована
“____” “_____” 20___ р.
викладачем
Радченко К. О.

Київ 2020

Мета роботи:

Здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL шляхом впровадження ORM (об'єктно-орієнтованої моделі).

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.
Індекси за варіантом - **BTree**, **GIN**.

3. Розробити тригер бази даних PostgreSQL.
Тригери за варіантом - **after update**, **insert**.

Хід роботи:
Завдання №2:

BTree

```
CREATE INDEX customerIdTreeIndex ON “customer” using btree (customer_id);
```

Звертання до таблиці (без індекса):

```
EXPLAIN ANALYZE SELECT customer_id, name, email FROM "customer"
```

Output | Explain | Messages | Notifications

QUERY PLAN text
Seq Scan on customer (cost=0.00..18.09 rows=1009 width=68) (actual tim...
Planning Time: 0.041 ms
Execution Time: 0.199 ms

Звертання до таблиці (з індексом):

```
EXPLAIN ANALYZE SELECT customer_id, name, email FROM "customer"
```

Output | Explain | Messages | Notifications

QUERY PLAN text
Seq Scan on customer (cost=0.00..18.09 rows=1009 width=68) (actual time=0.007..0.131 rows=1009 loops=1)
Planning Time: 0.200 ms
Execution Time: 0.173 ms

Результат:

```
SELECT customer_id, name, email FROM "customer"
```

Output | Explain | Messages | Notifications

customer_id [PK] integer	name text	email text
4	6eevNUGvKP	Z1J7yz@gmail.com
3	valadya	sos

……

Звертання до таблиці з використанням фільтра (без індекса):

```
EXPLAIN ANALYZE SELECT customer_id, name, email FROM "customer"
WHERE customer_id > 100 and customer_id < 500
```

Output | Explain | Messages | Notifications

QUERY PLAN text
Bitmap Heap Scan on customer (cost=4.33..11.92 rows=5 width=68) ...
Recheck Cond: ((customer_id > 100) AND (customer_id < 500))
Heap Blocks: exact=4
-> Bitmap Index Scan on customer_pkey (cost=0.00..4.33 rows=5 wi...
Index Cond: ((customer_id > 100) AND (customer_id < 500))
Planning Time: 0.056 ms
Execution Time: 0.154 ms

Звертання до таблиці з використанням фільтра (з індексом):

```
EXPLAIN ANALYZE SELECT customer_id, name, email FROM "customer"
WHERE customer_id > 100 and customer_id < 500
```

Output | Explain | Messages | Notifications

QUERY PLAN text
Bitmap Heap Scan on customer (cost=4.33..11.92 rows=5 width=68) (actual time=0.120..0.163 rows=399 loops=1)
Recheck Cond: ((customer_id > 100) AND (customer_id < 500))
Heap Blocks: exact=4
-> Bitmap Index Scan on customeridtreeindex (cost=0.00..4.33 rows=5 width=0) (actual time=0.107..0.108 rows=399 loops=1)
Index Cond: ((customer_id > 100) AND (customer_id < 500))
Planning Time: 0.080 ms
Execution Time: 0.199 ms

Результат:

```
SELECT customer_id, name, email FROM "customer"
WHERE customer_id > 100 and customer_id < 500
```

Output | Explain | Messages | Notifications

customer_id [PK] integer	name text	email text
101	ea7856ea53	0fafae@gmail.com
102	acc6a958a3	40df08@gmail.com
103	e701d3e5bb	536627@gmail.com

……

Звертання до таблиці (без індекса):

```
EXPLAIN ANALYZE SELECT customer_id, name, email FROM "customer"
```

Output | Explain | Messages | Notifications

QUERY PLAN text
Seq Scan on customer (cost=0.00..18.09 rows=1009 width=68) (actual tim...
Planning Time: 0.041 ms
Execution Time: 0.199 ms

Звертання до таблиці (з індексом):

```
EXPLAIN ANALYZE SELECT customer_id, name, email FROM "customer"
```

Output | Explain | Messages | Notifications

QUERY PLAN text
Seq Scan on customer (cost=0.00..18.09 rows=1009 width=68) (actual time=0.007..0.131 rows=1009 loops=1)
Planning Time: 0.200 ms
Execution Time: 0.173 ms

Результат:

```
SELECT customer_id, name, email FROM "customer"
```

Output | Explain | Messages | Notifications

customer_id [PK] integer	name text	email text
4	6eevNUGvKP	Z1J7yz@gmail.com
3	valadya	sos

…….

Звертання до таблиці з використанням агрегатної функції (без індекса):

```
EXPLAIN ANALYZE SELECT COUNT(customer_id) FROM "customer"
```

Output | Explain | Messages | Notifications

QUERY PLAN text
Aggregate (cost=20.61..20.62 rows=1 width=8) (actual time=0.160..0.160 rows=1 loops=1)
-> Seq Scan on customer (cost=0.00..18.09 rows=1009 width=4) (actual time=0.008..0.089 rows=1009 loops=1)
Planning Time: 0.066 ms
Execution Time: 0.184 ms

Звертання до таблиці з використанням агрегатної функції (з індексом):

```
EXPLAIN ANALYZE SELECT COUNT(customer_id) FROM "customer"
```

Output | Explain | Messages | Notifications

QUERY PLAN text
Aggregate (cost=20.61..20.62 rows=1 width=8) (actual time=0.171..0.172 rows=1 loops=1)
-> Seq Scan on customer (cost=0.00..18.09 rows=1009 width=4) (actual time=0.009..0.090 rows=1009 loops=1)
Planning Time: 0.046 ms
Execution Time: 0.190 ms

Результат:

```
SELECT COUNT(customer_id) FROM "customer"
```

Output | Explain | Messages | Notifications

count bigint
1009

Звертання до таблиці з використанням групування (без індекса):

```
EXPLAIN ANALYZE SELECT name FROM "customer"
GROUP BY name
```

Output | Explain | Messages | Notifications

QUERY PLAN text
HashAggregate (cost=20.61..22.61 rows=200 width=32) (actual time=0.407..0.608 rows=1009 loops=1)
Group Key: name
-> Seq Scan on customer (cost=0.00..18.09 rows=1009 width=32) (actual time=0.007..0.091 rows=1009 loops=1)
Planning Time: 0.077 ms
Execution Time: 0.685 ms

Звертання до таблиці з використанням групування (з індексом):

```
EXPLAIN ANALYZE SELECT name FROM "customer"
GROUP BY name
```

Output | Explain | Messages | Notifications

QUERY PLAN text
HashAggregate (cost=20.61..22.61 rows=200 width=32) (actual time=0.451..0.593 rows=1009 loops=1)
Group Key: name
-> Seq Scan on customer (cost=0.00..18.09 rows=1009 width=32) (actual time=0.010..0.109 rows=1009 loops=1)
Planning Time: 0.052 ms
Execution Time: 0.650 ms

Результат:

```
SELECT name FROM "customer"
GROUP BY name
```

Output | Explain | Messages | Notifications

name text
9459d110ed
3231dc4f59
e38538ec56

.....

Як і очікувалось, доданий BTree індекс на колонку customer_id, що є PrimaryKey даної сутності, ніяк не впливає на продуктивність запитів, адже саме цей індекс використовує PrimaryKey ‘під капотом’.

GIN

```
ALTER TABLE "product" ADD COLUMN ts_pname tsvector;
UPDATE "product" SET ts_pname = to_tsvector (name) WHERE true;
CREATE INDEX ginProductNameIndex ON "product" USING gin(ts_pname);
```

Після додавання даного індексу ми повинні спостерігати прискорення пошуку по імені конкретного продукту. Впровадження даного індексу є доречним в силу використання приладів, де покупці за назвою продукту генерують цінники.

Практика показала відповідність теорії, при збільшенні часу планування вдвічі, швидкість пошуку об’єкта з фіксованим значенням проіндексованої колонки, зменшилась втричі!

Звертання до таблиці з використанням фільтра (без індекса):

```
EXPLAIN ANALYZE SELECT product_id, name, cost, description FROM "product"
WHERE name LIKE 'tomato'
```

Output | Explain | Messages | Notifications

QUERY PLAN text
Seq Scan on product (cost=0.00..17.88 rows=3 width=100) (actual time=0.114..0.115 rows=1 loops=1)
Filter: (name ~~ 'tomato'::text)
Rows Removed by Filter: 999
Planning Time: 0.042 ms
Execution Time: 0.131 ms

Звертання до таблиці з використанням фільтра (з індексом):

```
EXPLAIN ANALYZE SELECT product_id, name, cost, description FROM "product"
WHERE to_tsquery('tomato') @@ ts_pname
```

Output | Explain | Messages | Notifications

QUERY PLAN text
Bitmap Heap Scan on product (cost=8.29..22.45 rows=5 width=100) (actual time=0.022..0.023 rows=1 loops=1)
Filter: (to_tsquery('tomato'::text) @@ ts_pname)
Heap Blocks: exact=1
-> Bitmap Index Scan on ginproductnameindex (cost=0.00..8.29 rows=5 width=0) (actual time=0.013..0.013 rows=1 loops=1)
Index Cond: (ts_pname @@ to_tsquery('tomato'::text))
Planning Time: 0.087 ms
Execution Time: 0.043 ms

Результат:

```
SELECT product_id, name, cost, description FROM "product"
WHERE name LIKE 'tomato'
```

Output | Explain | Messages | Notifications

product_id [PK] integer	name text	cost numeric	description text
8	tomato	113	6b4a7e89d919...

Звертання до таблиці з використанням агрегатної функції (без індекса):

```
EXPLAIN ANALYZE SELECT COUNT(product_id)
FROM "product"
WHERE name LIKE 'tomato'
```

Output | Explain | Messages | Notifications

QUERY PLAN text
Aggregate (cost=17.88..17.89 rows=1 width=8) (actual time=0.119..0.119 rows=1 loops=1)
-> Seq Scan on product (cost=0.00..17.88 rows=3 width=4) (actual time=0.115..0.116 rows=1 loops=1)
Filter: (name ~~ 'tomato'::text)
Rows Removed by Filter: 999
Planning Time: 0.049 ms
Execution Time: 0.139 ms

Звертання до таблиці з використанням агрегатної функції (з індексом):

```
EXPLAIN ANALYZE SELECT COUNT(product_id)
FROM "product"
WHERE to_tsquery('tomato') @@ ts_pname
```

Output | Explain | Messages | Notifications

QUERY PLAN text
Aggregate (cost=22.46..22.47 rows=1 width=8) (actual time=0.033..0.034 rows=1 loops=1)
-> Bitmap Heap Scan on product (cost=8.29..22.45 rows=5 width=4) (actual time=0.029..0.030 rows=1 loops=1)
Filter: (to_tsquery('tomato'::text) @@ ts_pname)
Heap Blocks: exact=1
-> Bitmap Index Scan on ginproductnameindex (cost=0.00..8.29 rows=5 width=0) (actual time=0.021..0.021 rows=1 loops=1
Index Cond: (ts_pname @@ to_tsquery('tomato'::text))
Planning Time: 0.090 ms
Execution Time: 0.058 ms

Результат:

```
SELECT COUNT(product_id)
FROM "product"
WHERE name LIKE 'tomato'
```

Output | Explain | Messages

count bigint
1

Завдання №3:

After update. При зміні ім’я клієнта, автоматично змінюємо значення колонки імені на його замовленнях:

```
CREATE OR REPLACE FUNCTION afterUpdateCustomer()
        RETURNS TRIGGER
        LANGUAGE plpgsql
        AS $$
DECLARE
    orders_ CURSOR IS
        SELECT *
        FROM "order"
        WHERE customer_id = NEW.customer_id;
BEGIN
    FOR order_ IN orders_ LOOP
        UPDATE "order"
        SET customer_name = NEW.name
        WHERE order_id = order_.order_id;
    end loop;
    return NEW;
END;
$$;

-- declaring func on trigger
CREATE TRIGGER after_update_customer
AFTER UPDATE
ON "customer"
FOR EACH ROW
EXECUTE PROCEDURE after_update_customer();
```

Результат до оновлення:

```
UPDATE "customer"
SET name = 'valadya'
WHERE "customer".customer_id = 3;

SELECT order_id, customer_name
FROM public."order"
WHERE customer_id = 3;
```

Output | Explain | Messages | Notifications

order_id [PK] integer	customer_name text
3	valadya
6	valadya
13	valadya

Результат після оновлення:

```
UPDATE "customer"
SET name = 'gym-star'
WHERE "customer".customer_id = 3;

SELECT order_id, customer_name
FROM public."order"
WHERE customer_id = 3;
```

Output | Explain | Messages | Notifications

order_id [PK] integer	customer_name text
3	gym-star
6	gym-star
13	gym-star

After insert. Якщо ім’я нового користувача - ‘lucker’ - для нього створюється купон зі значенням 25

```
CREATE OR REPLACE FUNCTION after_insert_customer()
       RETURNS TRIGGER
       LANGUAGE plpgsql
       AS $$
BEGIN
    IF NEW.name = 'lucker' THEN
               INSERT INTO "coupon" (value)
               VALUES 25;
    return NEW;
END;
$$;

CREATE TRIGGER after_insert_customer
  AFTER INSERT
  ON public."customer"
  FOR EACH ROW
  EXECUTE PROCEDURE after_insert_customer();
```