



НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря
Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

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

Лабораторна робота №1

з дисципліни

«Бази даних і засоби управління»

**Тема: «Засоби оптимізації роботи СУБД
PostgreSQL»**

Виконала: студентка III курсу

ФПМ групи КВ-84

Величко М. М.

Перевірив:

Київ – 2020

Завдання роботи:

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.

Варіант 8:

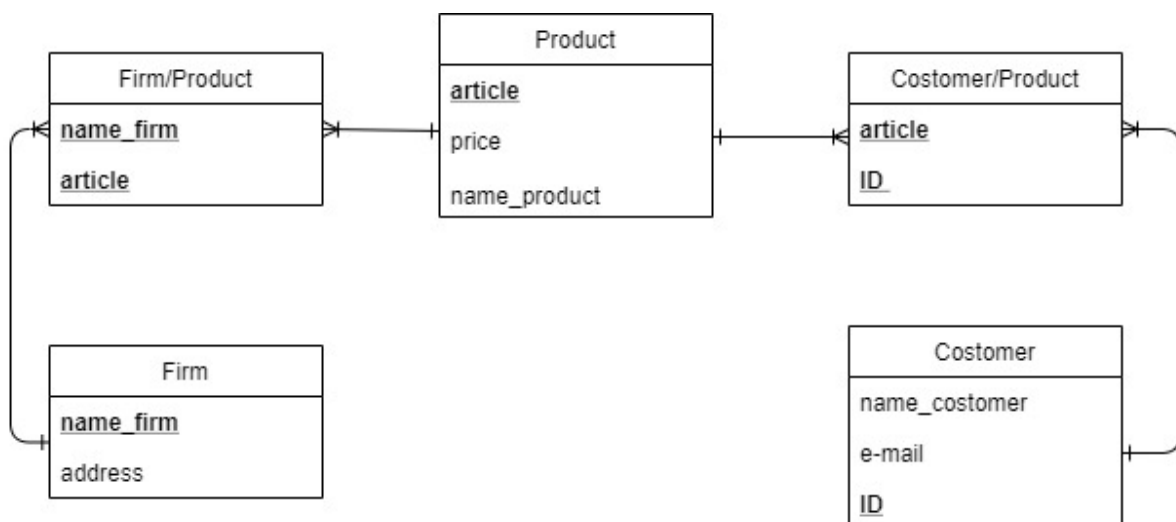
8	<i>BTree, GIN</i>	<i>after insert, update</i>
---	-------------------	-----------------------------

Вимоги до звітування щодо пунктів 1-3 завдання:

- для завдання №1: схему бази даних у вигляді таблиць і зв’язків між ними, а також класи ORM і зв’язки між ними, що відповідають таблицям бази даних. Навести приклади запитів у вигляді ORM.
- для завдання №2: команди створення індексів, тексти, результати і час виконання запитів SQL, пояснити чому індекси прискорюють (або не прискорюють) швидкість виконання запитів.
- для завдання №3: команди, що ініціюють виконання тригера, текст тригера та скріншоти зі змінами у таблицях бази даних.

Пункт №1 завдання:

Схема бази даних:



Класи ORM і зв'язки між ними:

```
class Costumer(Base):
    __tablename__ = 'costumer'
    id = Column(Integer, primary_key=True)
    name_costumer = Column(String(100))
    email = Column(String(100))
    costumer_product = relationship('CostumerProduct')

    def __repr__(self):
        return "<Costumer(id='{id}', name='{name}', email='{email}')" \
            .format(self.id, self.name_costumer, self.email)

class Firm(Base):
    __tablename__ = 'firm'
    name_firm = Column(String(30), primary_key=True)
    address = Column(String(30))
    firm_product = relationship("FirmProduct")
    def __repr__(self):
        return "<Firm(name_firm='{name_firm}', address='{address}')" \
            .format(self.name_firm, self.address)
```

```
class Product(Base):
    __tablename__ = 'product'
    article = Column(Integer, primary_key=True)
    name_product = Column(String(30))
    price = Column(Float)
    costumer_product = relationship("CostumerProduct")
    firm_product = relationship("FirmProduct")
    def __repr__(self):
        return "<Firm(article='{article}', name_product='{name_product}', price='{price}')" \
            .format(self.article, self.name_product, self.price)

class CostumerProduct(Base):
    __tablename__ = 'costumer_product'
    __table_args__ = {'extend_existing': True}
    id = Column(Integer, ForeignKey('costumer.id'), primary_key=True)
    article = Column(Integer, ForeignKey('product.article'), primary_key=True)
    def __repr__(self):
        return "<CostumerProduct(id='{id}', article='{article}')" \
            .format(self.id, self.article)
```

```

class FirmProduct(Base):
    __tablename__ = 'costumer_product'
    __table_args__ = {'extend_existing': True}
    name_firm = Column(String(100), ForeignKey('firm.name_firm'), primary_key=True)
    article = Column(Integer, ForeignKey('product.article'), primary_key=True)

    def __repr__(self):
        return "<CostumerProduct(name_firm='{ }', article='{ }'" \
            .format(self.name_firm, self.article)

```

Приклади запитів у вигляді ORM:

```

def create(s, table, parameter_1, parameter_2):
    try:
        arg = None
        if table == 1:
            arg = Costumer(name_costumer=parameter_1, email=parameter_2)
        elif table == 2:
            arg = Firm(name_firm=parameter_1, address=parameter_2)
        elif table == 3:
            arg = Product(name_product=parameter_1, price=parameter_2)
        elif table == 4:
            arg = CostumerProduct(id=parameter_1, article=parameter_2)
        elif table == 5:
            arg = FirmProduct(name_firm=parameter_1, article=parameter_2)
        s.add(arg)
        s.commit()
    except exc.SQLAlchemyError:
        s.rollback()
        print("Error")

```

```
def edit(s, table, param1, param2, param3):
    x = None
    if table == 1:
        x = s.query(Costumer).get(param3)
        x.name_costumer = param1
        x.email = param2
    elif table == 2:
        x = s.query(Firm).get(param3)
        x.name_firm = param1
        x.address = param2
    elif table == 3:
        x = s.query(Product).get(param3)
        x.name_product = param1
        x.price = param2
    elif table == 4:
        x = s.query(CostumerProduct).get(param3)
        x.id = param1
        x.article = param2
    elif table == 5:
        x = s.query(FirmProduct).get(param3)
        x.name_firm = param1
        x.article = param2
    s.add(x)
    s.commit()
    print("Error")
```

```
def delete(s, table, param):
    try:
        x = None
        if table == 1:
            x = s.query(Costumer).get(param)
        elif table == 2:
            x = s.query(Firm).get(param)
        elif table == 3:
            x = s.query(Product).get(param)
        elif table == 4:
            x = s.query(CostumerProduct).get(param)
        elif table == 5:
            x = s.query(FirmProduct).get(param)
        s.delete(x)
        s.commit()
    except exc.SQLAlchemyError:
        s.rollback()
        print("Error")
```

```
def print_table(s, table):
    if table == 1:
        for cost in s.query(Costumer).all():
            print(cost)
    elif table == 2:
        for firm in s.query(Firm).all():
            print(firm)
    elif table == 3:
        for prod in s.query(Product).all():
            print(prod)
    elif table == 4:
        for cost_prod in s.query(CostumerProduct).all():
            print(cost_prod)
    elif table == 5:
        for cost_firm in s.query(FirmProduct).all():
            print(cost_firm)
```

Ілюстрація виконання запитів:

Додавання даних до таблиць:

Додавання до costumer:

```
pythonProject2 - Model.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help

pythonProject2 Model.py
Controller.py Model.py View.py
Run: Controller
Choose table:
Customer - press 1
Firm - press 2
Product - press 3
Customer_Product - press 4
Firm_Product - press 5

1
Enter customer name

Run
Enter customer e-mail

r@example
Enter:
1 - create
2 - delete
3 - edit
4 - get
5 - exit

4
Choose table:
Customer - press 1
```

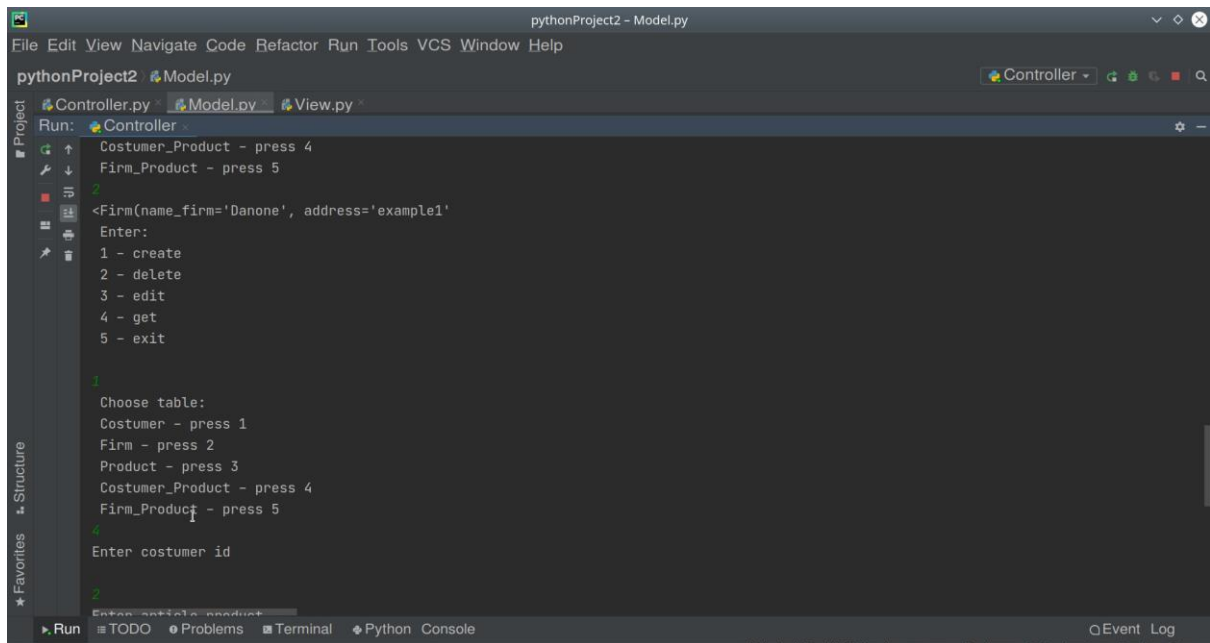
```
pythonProject2 - Model.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help

pythonProject2 Model.py
Controller.py Model.py View.py
Run: Controller
3 - exit

4
Choose table:
Customer - press 1
Firm - press 2
Product - press 3
Customer_Product - press 4
Firm_Product - press 5

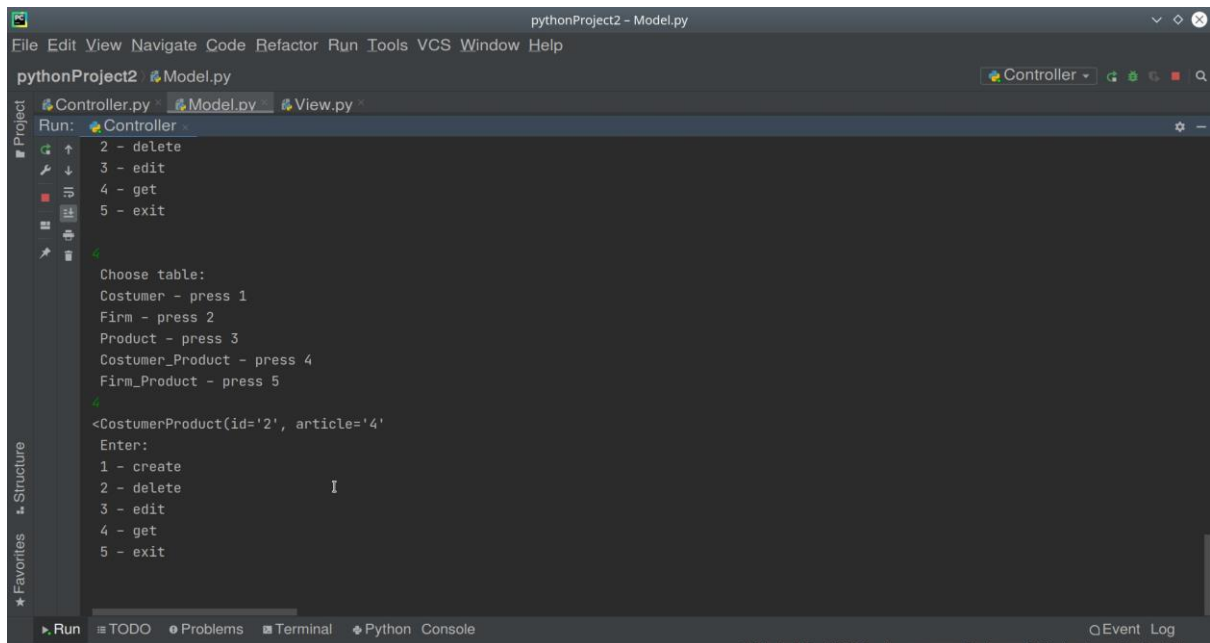
1
<Customer(id='2', name='Dan', email=dan@example
>
<Customer(id='110022', name='Jim', email=jim@example>
<Customer(id='110024', name='f', email=f>
<Customer(id='110023', name='h', email=h>
<Customer(id='110025', name='e', email=>
<Customer(id='110026', name='r', email=r>
<Customer(id='110027', name='Ron', email=r@example>
Enter:
1 - create
2 - delete
3 - edit
4 - get
```

Додавання до costumer_product:



```
pythonProject2 - Model.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help

pythonProject2 Model.py
Controller.py Model.py View.py
Run: Controller
Costumer_Product - press 4
Firm_Product - press 5
2
<Firm(name_firm='Danone', address='example1')
Enter:
1 - create
2 - delete
3 - edit
4 - get
5 - exit
1
Choose table:
Costumer - press 1
Firm - press 2
Product - press 3
Costumer_Product - press 4
Firm_Product - press 5
4
Enter costumer id
2
Enter article product
```



```
pythonProject2 - Model.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help

pythonProject2 Model.py
Controller.py Model.py View.py
Run: Controller
2 - delete
3 - edit
4 - get
5 - exit
4
Choose table:
Costumer - press 1
Firm - press 2
Product - press 3
Costumer_Product - press 4
Firm_Product - press 5
4
<CustomerProduct(id='2', article='4')
Enter:
1 - create
2 - delete
3 - edit
4 - get
5 - exit
```

Обробка виключної ситуації(додавання рядка в таблицю firm зі вже існуючим ім'ям фірми)


```
pythonProject2 - Model.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help

pythonProject2 Model.py
Controller.py Model.py View.py
Run: Controller
1 - create
2 - delete
3 - edit
4 - get
5 - exit

4
Choose table:
Customer - press 1
Firm - press 2
Product - press 3
Customer_Product - press 4
Firm_Product - press 5

2
<Firm(name_firm='Danone', address='example1')
Enter:
1 - create
2 - delete
3 - edit
4 - get
5 - exit
```

```
pythonProject2 - Model.py
File Edit View Navigate Code Refactor Run Tools VCS Window Help

pythonProject2 Model.py
Controller.py Model.py View.py
Run: Controller
Choose table:
Customer - press 1
Firm - press 2
Product - press 3
Customer_Product - press 4
Firm_Product - press 5

2
Enter firm name
Danone
Enter firm address
d
Error
Enter:
1 - create
2 - delete
3 - edit
4 - get
5 - exit
```

Видано повідомлення про помилку.

Пункт №2 завдання:

Команди створення індексів, тексти, результати і час виконання запитів SQL:

GIN index:

Сворено та заповнено таблицю на 100000 елементів, з типом даних атрибутів, що дозволяють роботу з **GIN** індексом:

The screenshot shows the pgAdmin interface with the 'test' table selected in the left sidebar. The table structure is as follows:

id	name	massege
123	CX	'cx':1
124	UA	'ua':1
125	KN	'kn':1
126	DR	'dr':1
127	OG	'og':1
128	WA	'wa':1
129	FL	'fl':1

Створення індексу:

The screenshot shows the pgAdmin interface with the following query in the Query Editor:

```
1 create index on test using gin(massege)
```

The Data Output section displays the query plan:

```

QUERY PLAN
text
1  Finalize Aggregate (cost=17519.60..17519.61 rows=1 width=8) (actual time=65.256..66.878 rows=1 loops=1)
2  -> Gather (cost=17519.49..17519.60 rows=1 width=8) (actual time=64.924..66.869 rows=1 loops=1)
3  Workers Planned: 1

```

The Messages section shows: "CREATE INDEX" and "Query returned successfully in 120 msec."

Запити без використання індексу

The screenshot shows the pgAdmin interface with the following query in the Query Editor:

```
1 explain analyze select * from test where massege @@ to_tsquery('ex:*')
```

The Data Output section displays the query plan:

```

QUERY PLAN
text
1  Gather (cost=1000.00..17540.18 rows=210 width=21) (actual time=0.690..48.372 rows=138 loops=1)
2  Workers Planned: 1
3  Workers Launched: 1
4  -> Parallel Seq Scan on test (cost=0.00..16519.18 rows=124 width=21) (actual time=0.680..45.243 rows=69 loops=2)
5  Filter: (massege @@ to_tsquery('ex*':text))
6  Rows Removed by Filter: 49931
7  Planning Time: 0.094 ms
8  Execution Time: 48.395 ms

```

pgAdmin

File Object Tools Help

Browser Extensions Foreign Data Wrappers Languages Schemas (1) public Collations Domains FTS Configuration FTS Dictionary FTS Parsers FTS Template Foreign Tables Functions Materialized Views Procedures Sequences Tables (1) test Columns id name message

test/postgres@LocalServer

Query Editor Query History

```
1 explain analyze select * from test where message @@ to_tsquery('ex:*') order by name
```

Data Output

QUERY PLAN
text
1 Gather Merge (cost=17523.50..17537.76 rows=124 width=21) (actual time=71.642..73.737 rows=138 loops=1)
2 Workers Planned: 1
3 Workers Launched: 1
4 -> Sort (cost=16523.49..16523.80 rows=124 width=21) (actual time=69.571..69.577 rows=69 loops=2)
5 Sort Key: name
6 Sort Method: quicksort Memory: 31kB
7 Worker 0: Sort Method: quicksort Memory: 29kB
8 -> Parallel Seq Scan on test (cost=0.00..16519.18 rows=124 width=21) (actual time=0.808..69.461 rows=69 loops=2)

pgAdmin

File Object Tools Help

Browser Extensions Foreign Data Wrappers Languages Schemas (1) public Collations Domains FTS Configuration FTS Dictionary FTS Parsers FTS Template Foreign Tables Functions Materialized Views Procedures Sequences Tables (1) test Columns id name message

test/postgres@LocalServer

Query Editor Query History

```
1 explain analyze select sum(id) from test where message @@ to_tsquery('ex:*')
```

Data Output

QUERY PLAN
text
1 Finalize Aggregate (cost=17519.60..17519.61 rows=1 width=8) (actual time=49.872..55.586 rows=1 loops=1)
2 -> Gather (cost=17519.49..17519.60 rows=1 width=8) (actual time=49.850..55.553 rows=2 loops=1)
3 Workers Planned: 1
4 Workers Launched: 1
5 -> Partial Aggregate (cost=16519.49..16519.50 rows=1 width=8) (actual time=47.944..47.944 rows=1 loops=2)
6 -> Parallel Seq Scan on test (cost=0.00..16519.18 rows=124 width=4) (actual time=0.660..47.896 rows=69 loops=2)
7 Filter: (message @@ to_tsquery('ex:*'))
8 Rows Removed by Filter: 49931

Successfully run. Total query runtime: 120 msec. 10 rows affected.

pgAdmin

File Object Tools Help

Browser Extensions Foreign Data Wrappers Languages Schemas (1) public Collations Domains FTS Configuration FTS Dictionary FTS Parsers FTS Template Foreign Tables Functions Materialized Views Procedures Sequences Tables (1) test Columns id name message

test/postgres@LocalServer

Query Editor Query History

```
1 explain analyze select count(message) from test where message @@ to_tsquery('ex:*')
```

Data Output

QUERY PLAN
text
1 Finalize Aggregate (cost=17519.60..17519.61 rows=1 width=8) (actual time=65.256..66.878 rows=1 loops=1)
2 -> Gather (cost=17519.49..17519.60 rows=1 width=8) (actual time=64.924..66.869 rows=2 loops=1)
3 Workers Planned: 1
4 Workers Launched: 1
5 -> Partial Aggregate (cost=16519.49..16519.50 rows=1 width=8) (actual time=63.476..63.477 rows=1 loops=2)
6 -> Parallel Seq Scan on test (cost=0.00..16519.18 rows=124 width=14) (actual time=0.990..63.406 rows=69 loops=2)
7 Filter: (message @@ to_tsquery('ex:*'))
8 Rows Removed by Filter: 49931

Аналогічні запити з використанням індексу:

pgAdmin interface showing the Query Editor and Data Output for a query on the 'test' table.

Query Editor:

```
1 explain analyze select * from test where massege @@ to_tsquery('ex:*')
```

Data Output:

QUERY PLAN
1 Bitmap Heap Scan on test (cost=53.88..633.91 rows=210 width=21) (actual time=0.050..0.146 rows=138 loops=1)
2 Recheck Cond: (massege @@ to_tsquery('ex*':text))
3 Heap Blocks: exact=122
4 -> Bitmap Index Scan on test_massege_idx (cost=0.00..53.82 rows=210 width=0) (actual time=0.040..0.040 rows=138 loops=1)
5 Index Cond: (massege @@ to_tsquery('ex*':text))
6 Planning Time: 0.080 ms
7 Execution Time: 0.162 ms

pgAdmin interface showing the Query Editor and Data Output for a query on the 'test' table, ordered by name.

Query Editor:

```
1 explain analyze select * from test where massege @@ to_tsquery('ex:*') order by name
```

Data Output:

QUERY PLAN
1 Sort (cost=642.01..642.54 rows=210 width=21) (actual time=0.213..0.219 rows=138 loops=1)
2 Sort Key: name
3 Sort Method: quicksort Memory: 35kB
4 -> Bitmap Heap Scan on test (cost=53.88..633.91 rows=210 width=21) (actual time=0.050..0.199 rows=138 loops=1)
5 Recheck Cond: (massege @@ to_tsquery('ex*':text))
6 Heap Blocks: exact=122
7 -> Bitmap Index Scan on test_massege_idx (cost=0.00..53.82 rows=210 width=0) (actual time=0.041..0.041 rows=138 loops=1)
8 Index Cond: (massege @@ to_tsquery('ex*':text))

pgAdmin interface showing the Query Editor and Data Output for a query on the 'test' table, counting the number of rows.

Query Editor:

```
1 explain analyze select count(massege) from test where massege @@ to_tsquery('ex:*')
```

Data Output:

QUERY PLAN
1 Aggregate (cost=634.44..634.45 rows=1 width=8) (actual time=0.180..0.180 rows=1 loops=1)
2 -> Bitmap Heap Scan on test (cost=53.88..633.91 rows=210 width=14) (actual time=0.053..0.152 rows=138 loops=1)
3 Recheck Cond: (massege @@ to_tsquery('ex*':text))
4 Heap Blocks: exact=122
5 -> Bitmap Index Scan on test_massege_idx (cost=0.00..53.82 rows=210 width=0) (actual time=0.043..0.043 rows=138 loops=1)
6 Index Cond: (massege @@ to_tsquery('ex*':text))
7 Planning Time: 0.126 ms
8 Execution Time: 0.199 ms

The screenshot shows the pgAdmin interface with a query executed: `explain analyze select sum(id) from test where massege @@ to_tsquery('ex:*')`. The 'Data Output' section displays the 'QUERY PLAN' for this query.

Step	Operation	Cost	Rows	Width	Actual Time	Loops
1	Aggregate	(cost=634.44..634.45 rows=1 width=8)	1	8	(actual time=0.193..0.193 rows=1 loops=1)	1
2	Bitmap Heap Scan on test	(cost=53.88..633.91 rows=210 width=4)	210	4	(actual time=0.049..0.182 rows=138 loops=1)	1
3	Recheck Cond: (massege @@ to_tsquery('ex*':text))					
4	Heap Blocks: exact=122					
5	Bitmap Index Scan on test_massege_idx	(cost=0.00..53.82 rows=210 width=0)	210	0	(actual time=0.039..0.040 rows=138 loops=1)	1
6	Index Cond: (massege @@ to_tsquery('ex*':text))					
7	Planning Time	0.108 ms				
8	Execution Time	0.210 ms				

BTREE index:

Створено таблицю та заповнено 100000 елементів:

The screenshot shows the pgAdmin interface with a query executed: `SELECT * FROM public.test2 ORDER BY id ASC`. The 'Data Output' section displays the results of the query.

	id [PK] integer	value integer
1	1	4
2	2	13
3	3	83
4	4	10
5	5	25
6	6	73
7	7	24
8	8	15
9	9	79
10	10	49
11	11	78
12	12	73

Successfully run. Total query runtime: 495 msec. 100000 rows affected.

Створення індексу:

The screenshot shows the pgAdmin interface with a query executed: `create index on test2 using btree(value)`. The 'Messages' section displays the result of the query.

CREATE INDEX
Query returned successfully in 259 msec.

Приклади запитів без використання індексу:

pgAdmin 127.0.0.1:35313/browser/ 120%

File Object Tools Help

Browser

- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Schemas (1)
 - public
 - Collations
 - Domains
 - FTS Configuration
 - FTS Dictionary
 - FTS Parsers
 - FTS Template
 - Foreign Tables
 - Functions
 - Materialized Views
 - Procedures
 - Sequences
 - Tables (2)
 - test
 - test2
 - Trigger Functions
 - Types
 - Views
- Login/Group Roles (10)

Dashboard Properties Statistics SQL Dependencies Dependents firm/postgres... test/postgres... test/postgres...

test/postgres@LocalServer

Explain Notifications Scratch Pad

Query Editor Query History

```
1 explain analyze select * from test2 where value < 5
```

Data Output

QUERY PLAN

text

1	Seq Scan on test2 (cost=0.00..1693.00 rows=4940 width=8) (actual time=0.012..7.877 rows=5045 loops=1)
2	Filter: (value < 5)
3	Rows Removed by Filter: 94955
4	Planning Time: 0.036 ms
5	Execution Time: 8.095 ms

pgAdmin 127.0.0.1:35313/browser/ 120%

File Object Tools Help

Browser

- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Schemas (1)
 - public
 - Collations
 - Domains
 - FTS Configuration
 - FTS Dictionary
 - FTS Parsers
 - FTS Template
 - Foreign Tables
 - Functions
 - Materialized Views
 - Procedures
 - Sequences
 - Tables (2)
 - test
 - test2
 - Trigger Functions
 - Types
 - Views
- Login/Group Roles (10)

Dashboard Properties Statistics SQL Dependencies Dependents firm/postgres... test/postgres... test/postgres...

test/postgres@LocalServer

Explain Notifications Scratch Pad

Query Editor Query History

```
1 explain analyze select * from test2 where value < 5 order by value
```

Data Output

QUERY PLAN

text

1	Sort (cost=1996.08..2008.43 rows=4940 width=8) (actual time=10.973..11.234 rows=5045 loops=1)
2	Sort Key: value
3	Sort Method: quicksort Memory: 429kB
4	-> Seq Scan on test2 (cost=0.00..1693.00 rows=4940 width=8) (actual time=0.010..10.183 rows=5045 loops=1)
5	Filter: (value < 5)
6	Rows Removed by Filter: 94955
7	Planning Time: 0.045 ms
8	Execution Time: 11.366 ms

pgAdmin 127.0.0.1:35313/browser/ 120%

File Object Tools Help

Browser

- Catalogs
- Event Triggers
- Extensions
- Foreign Data Wrappers
- Languages
- Schemas (1)
 - public
 - Collations
 - Domains
 - FTS Configuration
 - FTS Dictionary
 - FTS Parsers
 - FTS Template
 - Foreign Tables
 - Functions
 - Materialized Views
 - Procedures
 - Sequences
 - Tables (2)
 - test
 - test2
 - Trigger Functions
 - Types
 - Views
- Login/Group Roles (10)

Dashboard Properties Statistics SQL Dependencies Dependents firm/postgres... test/postgres... test/postgres...

test/postgres@LocalServer

Explain Notifications Scratch Pad

Query Editor Query History

```
1 explain analyze select sum(id) from test2 where value < 5
```

Data Output

QUERY PLAN

text

1	Aggregate (cost=1705.35..1705.36 rows=1 width=8) (actual time=9.878..9.883 rows=1 loops=1)
2	-> Seq Scan on test2 (cost=0.00..1693.00 rows=4940 width=4) (actual time=0.014..9.547 rows=5045 loops=1)
3	Filter: (value < 5)
4	Rows Removed by Filter: 94955
5	Planning Time: 0.070 ms
6	Execution Time: 9.923 ms

pgAdmin interface showing a query execution result. The query is:

```
1 explain analyze select count(value) from test2 where value < 5
```

The Data Output section shows the Query Plan:

Step	Operation	Cost	Actual Time	Rows	Width	Loops
1	Aggregate	(cost=1705.35..1705.36 rows=1 width=8)	(actual time=8.993..8.994 rows=1 loops=1)			
2	-> Seq Scan on test2	(cost=0.00..1693.00 rows=4940 width=4)	(actual time=0.008..8.700 rows=5045 loops=1)			
3	Filter: (value < 5)					
4	Rows Removed by Filter: 94955					
5	Planning Time: 0.082 ms					
6	Execution Time: 9.015 ms					

Successfully run. Total query runtime: 95 msec. 6 rows affected.

Результати запитів з використанням індексу:

pgAdmin interface showing two query execution results using an index.

Query 1:

```
1 explain analyze select * from test2 where value < 5
```

The Data Output section shows the Query Plan:

Step	Operation	Cost	Actual Time	Rows	Width	Loops
1	Bitmap Heap Scan on test2	(cost=94.70..599.45 rows=4940 width=8)	(actual time=0.354..1.717 rows=5045 loops=1)			
2	Recheck Cond: (value < 5)					
3	Heap Blocks: exact=443					
4	-> Bitmap Index Scan on test2_value_idx	(cost=0.00..93.47 rows=4940 width=0)	(actual time=0.315..0.316 rows=5045 loops=1)			
5	Index Cond: (value < 5)					
6	Planning Time: 0.055 ms					
7	Execution Time: 1.871 ms					

Query 2:

```
1 explain analyze select * from test2 where value < 5 order by value
```

The Data Output section shows the Query Plan:

Step	Operation	Cost	Actual Time	Rows	Width	Loops
1	Sort	(cost=902.53..914.88 rows=4940 width=8)	(actual time=2.164..2.621 rows=5045 loops=1)			
2	Sort Key: value					
3	Sort Method: quicksort Memory: 429kB					
4	-> Bitmap Heap Scan on test2	(cost=94.70..599.45 rows=4940 width=8)	(actual time=0.263..1.677 rows=5045 loops=1)			
5	Recheck Cond: (value < 5)					
6	Heap Blocks: exact=443					
7	-> Bitmap Index Scan on test2_value_idx	(cost=0.00..93.47 rows=4940 width=0)	(actual time=0.226..0.226 rows=5045 loops=1)			
8	Index Cond: (value < 5)					
9	Planning Time: 0.054 ms					
10	Execution Time: 2.769 ms					

The top screenshot shows the pgAdmin interface with the following query in the Query Editor:

```
1 explain analyze select sum(id) from test2 where value < 5
```

The Data Output shows the following query plan:

QUERY PLAN
1 Aggregate (cost=611.80..611.81 rows=1 width=8) (actual time=1.723..1.724 rows=1 loops=1)
2 -> Bitmap Heap Scan on test2 (cost=94.70..599.45 rows=4940 width=4) (actual time=0.241..1.440 rows=5045 loops=1)
3 Recheck Cond: (value < 5)
4 Heap Blocks: exact=443
5 -> Bitmap Index Scan on test2_value_idx (cost=0.00..93.47 rows=4940 width=0) (actual time=0.204..0.204 rows=5045 loops=1)
6 Index Cond: (value < 5)
7 Planning Time: 0.046 ms
8 Execution Time: 1.746 ms

The bottom screenshot shows the pgAdmin interface with the following query in the Query Editor:

```
1 explain analyze select count(value) from test2 where value < 5
```

The Data Output shows the following query plan:

QUERY PLAN
1 Aggregate (cost=611.80..611.81 rows=1 width=8) (actual time=1.933..1.934 rows=1 loops=1)
2 -> Bitmap Heap Scan on test2 (cost=94.70..599.45 rows=4940 width=4) (actual time=0.338..1.599 rows=5045 loops=1)
3 Recheck Cond: (value < 5)
4 Heap Blocks: exact=443
5 -> Bitmap Index Scan on test2_value_idx (cost=0.00..93.47 rows=4940 width=0) (actual time=0.291..0.291 rows=5045 loops=1)
6 Index Cond: (value < 5)
7 Planning Time: 0.066 ms
8 Execution Time: 1.959 ms

Висновки:

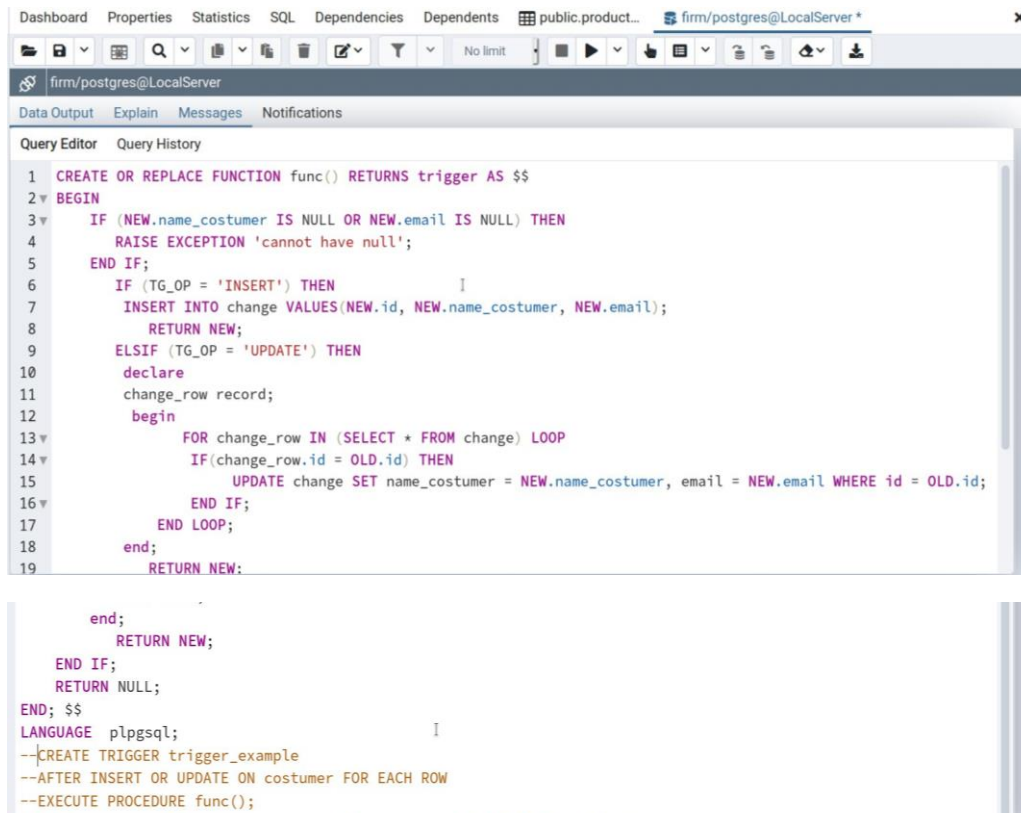
Як можна побачити з результатів в обох випадка використання індексів пришвидшує виконання запитів. Індеси є ефективними коли в таблиці є достатня кількість запитів.

Btree використовується для даних які можна відсортувати, Дані в індексі впорядковані по неспаданню а сторінки одного рівня пов'язані між собою двонаправленим списком. Тому отримати впорядкований набір даних ми можемо, просто проходячи по списку в одну або в іншу сторону, не повертаючись кожен раз до кореня. Древа є збалансованими, тому пошук будь-яких значень займає приблизно однаковий час.

Gin використовується для не атомарних даних. При цьому індексуються не власними значення, а окремі елементи; кожен елемент посилається на ті значення, в яких він зустрічається, що можна порівняти з алфавітним покажчиком, тому gin доцільно використовувати для прискорення повнотекстового пошуку.

Пункт №3 завдання:

Текст тригера:



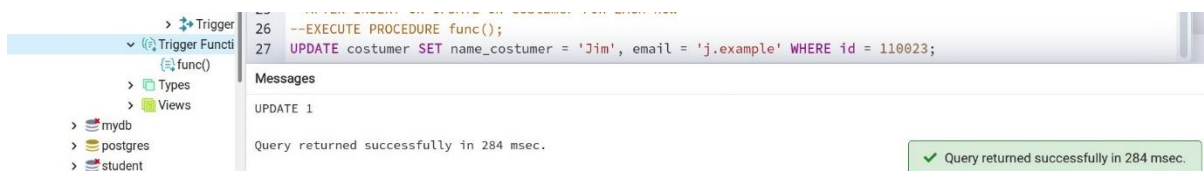
```
1 CREATE OR REPLACE FUNCTION func() RETURNS trigger AS $$
2 BEGIN
3     IF (NEW.name_costumer IS NULL OR NEW.email IS NULL) THEN
4         RAISE EXCEPTION 'cannot have null';
5     END IF;
6     IF (TG_OP = 'INSERT') THEN
7         INSERT INTO change VALUES(NEW.id, NEW.name_costumer, NEW.email);
8         RETURN NEW;
9     ELSIF (TG_OP = 'UPDATE') THEN
10        declare
11        change_row record;
12        begin
13            FOR change_row IN (SELECT * FROM change) LOOP
14                IF(change_row.id = OLD.id) THEN
15                    UPDATE change SET name_costumer = NEW.name_costumer, email = NEW.email WHERE id = OLD.id;
16                END IF;
17            END LOOP;
18        end;
19        RETURN NEW;
20    END IF;
21    RETURN NULL;
22 END; $$
23 LANGUAGE plpgsql;
--CREATE TRIGGER trigger_example
--AFTER INSERT OR UPDATE ON costumer FOR EACH ROW
--EXECUTE PROCEDURE func();
```

Тригер спрацьовує після вставки або редагування даних в таблиці costumer.

Після вставка тригер записує данні в таблицю change, при редагуванні редагує данні з відповідним id в таблиці change. Якщо не задано ім'я або електронна пошта спрацьовує виключення, адже атрибути таблиці change не можуть бути NULL.

Команди, що ініціюють виконання тригера та зміни в таблицях після їх виконання:

Редагування:



```
26 --EXECUTE PROCEDURE func();
27 UPDATE costumer SET name_costumer = 'Jim', email = 'j.example' WHERE id = 110023;
```

Messages

UPDATE 1

Query returned successfully in 284 msec.

Вставка:

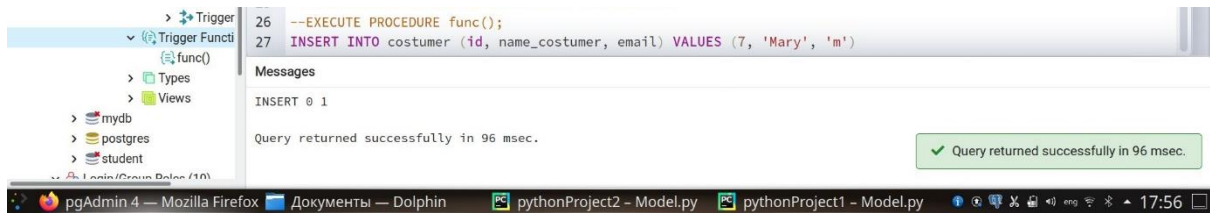


Таблица costumer:

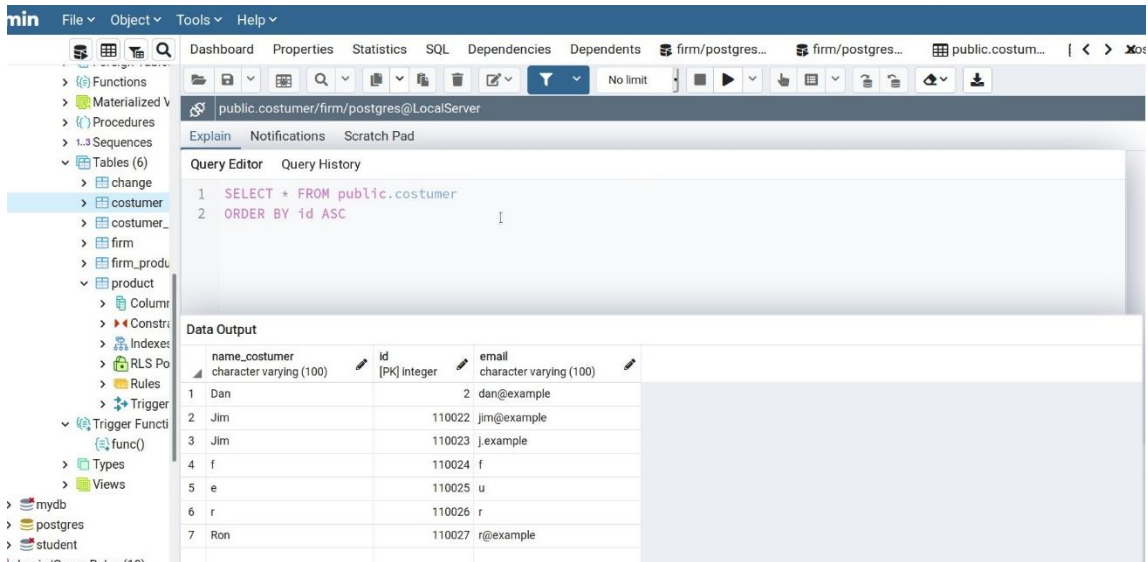


Таблица change:

