



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 групи КВ-23

Паламарчук Максим

Телеграм: https://t.me/Maxim_Pal

Перевірив: Петрашенко А.В.

Київ – 2024

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

Завдання роботи полягає у наступному:

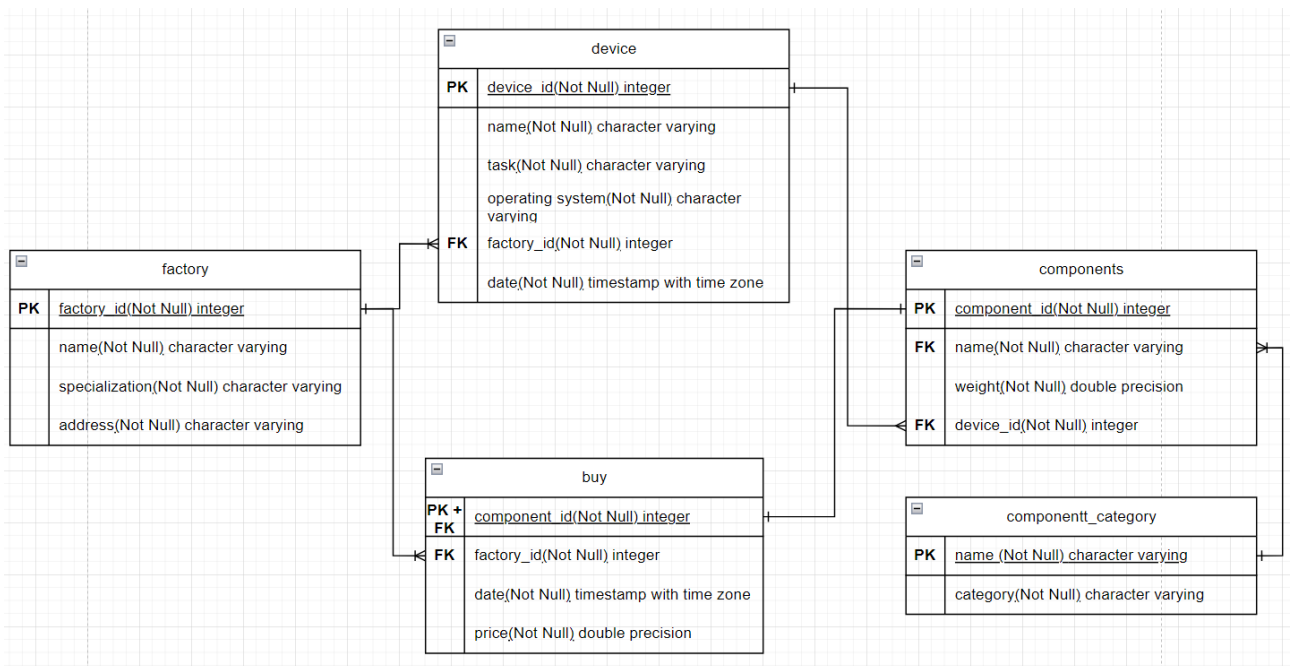
1. Перетворити модуль “Модель” з шаблону MVC РГР у вигляд об’єктно-реляційної проєкції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.
4. Навести приклади та проаналізувати рівні ізоляції транзакцій у PostgreSQL.

Варіант 20

20	GIN, BRIN	after insert, update
----	-----------	----------------------

Виконання роботи

Логічна модель (схема) “Система управління та аналізу даних в галузі робототехніки”



Класи ORM зі зв'язками

```
class Factory(Base):
    __tablename__ = "factory"

    factory_id = Column(Integer, primary_key=True, nullable=False)
    name = Column(String(30), nullable=False)
    specialization = Column(String(60), nullable=False)
    address = Column(String(70), unique=True, nullable=False)

    devices = relationship("Device", back_populates="factory")
    buys = relationship("Buy", back_populates="factory")

class Device(Base):
    __tablename__ = "device"

    device_id = Column(Integer, primary_key=True, nullable=False)
    name = Column(String(20), nullable=False)
    task = Column(String(200), nullable=False)
    operating_system = Column(String(20), nullable=False)
    factory_id = Column(Integer, ForeignKey('factory.factory_id'), nullable=False)
    date = Column(DateTime, nullable=False)

    factory = relationship("Factory", back_populates="devices")
    components = relationship("Components", back_populates="device")

class Components(Base):
    __tablename__ = "components"

    component_id = Column(Integer, primary_key=True, nullable=False)
    name = Column(String(20), ForeignKey('component_category.name'), nullable=False)
    weight = Column(DOUBLE_PRECISION, nullable=False)
    device_id = Column(Integer, ForeignKey('device.device_id'))

    device = relationship("Device", back_populates="components")
    buys = relationship("Buy", back_populates="component")
    component_category = relationship("Component_Category",
back_populates="component")

class Component_Category(Base):
    __tablename__ = "component_category"

    name = Column(String(20), primary_key=True, nullable=False)
    category = Column(String(30), nullable=False)

    name = Column(String(20), primary_key=True, nullable=False)
    category = Column(String(30), nullable=False)

    component = relationship("Components", back_populates="component_category")

class Buy(Base):
    __tablename__ = "buy"

    component_id = Column(Integer, ForeignKey('components.component_id'),
primary_key=True, nullable=False)
    factory_id = Column(Integer, ForeignKey('factory.factory_id'), nullable=False)
    date = Column(DateTime, nullable=False)
    price = Column(DOUBLE_PRECISION, nullable=False)

    factory = relationship("Factory", back_populates="buys")
    component = relationship("Components", back_populates="buys")
```

Приклади запитів у вигляді ORM

Фрагмент програми для введення даних в таблицю:

```
def add_row(self, attributes, attributes_name, table):
    session = self.Session()

    table = Base.metadata.tables[table.lower()]

    new_row = {}
    for column_name, column_value in zip(attributes_name, attributes):
        if column_name in table.columns:
            new_row[column_name] = column_value

    session.execute(table.insert().values(new_row))
    session.commit()
```

Фрагмент програми для видалення даних з таблиці:

```
def delete_row(self, row_id, PK, table):
    session = self.Session()

    table = Base.metadata.tables[table.lower()]
    condition = getattr(table.c, PK) == row_id
    session.execute(table.delete().where(condition))

    session.commit()
```

Фрагмент програми для оновлення даних в таблиці:

```
def update_row(self, row_id, PK, attributes, attributes_name, table):
    session = self.Session()

    table = Base.metadata.tables[table.lower()]

    update_values = {}
    for column_name, column_value in zip(attributes_name, attributes):
        if column_name in table.columns:
            update_values[column_name] = column_value

    condition = getattr(table.c, PK) == row_id
    session.execute(table.update().where(condition).values(update_values))

    session.commit()
```

Фрагмент програми для генерування даних в таблиці:

```
def random_table(self, counts, table):
    session = self.Session()

    query = text(f"CALL random_{table.lower()}(:counts)")
    session.execute(query, {'counts': counts})

    session.commit()
```

Фрагмент програми для пошуку даних у таблиці:

```
def get_DeviceOfFactory(self, FK):
    session = self.Session()

    start_time = time.time()

    factory = session.query(Factory).filter_by(factory_id=FK).first()

    end_time = time.time()
    duration = (end_time - start_time) * 1000

    device_counts = {}
    for device in factory.devices:
        device_counts[device.name] = device_counts.get(device.name, 0) + 1

    rows = [
        (factory.factory_id, factory.name, factory.address, device_name, count)
        for device_name, count in device_counts.items()
    ]
    column_names = ["factory_id", "factory_name", "address", "device_name", "count"]
    return rows, column_names, duration

def get_ComponentsOfDevice(self, FK):
    session = self.Session()

    start_time = time.time()

    device = session.query(Device).filter_by(device_id=FK).first()

    end_time = time.time()
    duration = (end_time - start_time) * 1000

    component_averages = {}
    for component in device.components:
        if component.name not in component_averages:
            component_averages[component.name] = []
        component_averages[component.name].append(component.weight)

    rows = [
        (device.device_id, device.name, component_name, sum(weights) / len(weights))
        for component_name, weights in component_averages.items()
    ]
    column_names = ["device_id", "device_name", "component_name", "avg_weight"]
    return rows, column_names, duration

def get_BuyOfComponents(self, first_date, second_date, FK):
    session = self.Session()

    start_time = time.time()

    factory = session.query(Factory).filter_by(factory_id=FK).first()

    end_time = time.time()
    duration = (end_time - start_time) * 1000

    rows = [
        (factory.factory_id, factory.name, buy.component.component_id,
        buy.component.name, buy.date, buy.price)
        for buy in factory.buys
        if first_date <= buy.date <= second_date
    ]
    column_names = ["factory_id", "factory_name", "component_id", "component_name",
    "date", "price"]
    return rows, column_names, duration
```

Фрагмент програми для отримання імен стовпчиків таблиці:

```
def get_attributes(self, table):  
    return Base.metadata.tables[table.lower()].columns.keys()
```

Створення індексів

GIN

```
CREATE INDEX idx_gin_name ON components USING GIN (name gin_trgm_ops);
```

BRIN

```
CREATE INDEX idx_brin_name ON components USING BRIN(name)
```

Приклад 1: Просте фільтрування

Без індексів:

Query Query History

```
1 SET enable_seqscan = on;  
2 EXPLAIN ANALYSE  
3 SELECT * FROM components  
4 WHERE name LIKE '%r%'  
5 LIMIT 15;
```

Data Output Messages Notifications

≡+ 📄 ▼ 📋 ▼ 🗑️ 🗄️ ⬇️ ⚡ SQL

	QUERY PLAN	
	text	🔒
1	Limit (cost=0.00..0.31 rows=15 width=30) (actual time=0.022..0.025 rows=15 loops=1)	
2	-> Seq Scan on components (cost=0.00..228265.25 rows=11000100 width=30) (actual time=0.020..0.023 rows=15 loop...	
3	Filter: ((name)::text ~~ '%r%':text)	
4	Planning Time: 0.218 ms	
5	Execution Time: 0.045 ms	

3 индексом GIN:

Query

Query History

1

SET enable_seqscan = off;

2

EXPLAIN ANALYSE

3

SELECT * FROM components

4

WHERE name LIKE '%r%'

5

LIMIT 15;

Data Output

Messages

Notifications

≡

+

📄

▼

📋

▼

🗑️

🔄

📥

📤

↶

SQL

QUERY PLAN

text

🔒

1

Limit (cost=152106.41..152106.72 rows=15 width=30) (actual time=1388.610..1388.614 rows=15 loops=1)

2

-> Bitmap Heap Scan on components (cost=152106.41..380371.66 rows=11000100 width=30) (actual time=1388.609..1388.612 rows=15 loops=1)

3

Recheck Cond: ((name)::text ~~ '%r%':text)

4

Heap Blocks: lossy=1

5

-> Bitmap Index Scan on idx_gin_name (cost=0.00..149356.39 rows=11000100 width=0) (actual time=1382.126..1382.126 rows=7017688 loop...

6

Index Cond: ((name)::text ~~ '%r%':text)

7

Planning Time: 0.102 ms

8

Execution Time: 1389.185 ms

З індексом BRIN:

Query

Query History

1

2

3

4

5

6

7

8

SET enable_seqscan = off;

EXPLAIN ANALYSE

SELECT * FROM components

WHERE name LIKE '%r%'

LIMIT 15;

Data Output

Messages

Notifications

SQL

QUERY PLAN

text

1

Limit (cost=10000000000.00..10000000000.31 rows=15 width=30) (actual time=0.011..0.013 rows=15 loops=1)

2

-> Seq Scan on components (cost=10000000000.00..10000228265.25 rows=11000100 width=30) (actual time=0.010..0.012 rows=15 loop...

3

Filter: ((name)::text ~~ '%r%':text)

4

Planning Time: 0.061 ms

5

Execution Time: 0.023 ms

Приклад 2: Агрегатні функції

Без індексів:

Query

Query History

1

2

3

4

5

SET enable_seqscan = on;

EXPLAIN ANALYSE

SELECT COUNT(*) FROM components

WHERE name = 'Servo motors 6'

QUERY PLAN

text

1

Finalize Aggregate (cost=149224.84..149224.85 rows=1 width=8) (actual time=327.836..331.498 rows=1 loops=1)

2

-> Gather (cost=149224.62..149224.84 rows=2 width=8) (actual time=327.756..331.492 rows=3 loops=1)

3

Workers Planned: 2

4

Workers Launched: 2

5

-> Partial Aggregate (cost=148224.62..148224.64 rows=1 width=8) (actual time=300.559..300.559 rows=1 loops=3)

6

-> Parallel Seq Scan on components (cost=0.00..148056.19 rows=67375 width=0) (actual time=0.226..298.027 rows=56452 loop...

7

Filter: ((name)::text = 'Servo motors 6':text)

8

Rows Removed by Filter: 3610248

9

Planning Time: 0.636 ms

10

Execution Time: 331.517 ms

З індексом GIN:

Query Query History	
1	SET enable_seqscan = off;
2	
3	EXPLAIN ANALYSE
4	SELECT COUNT(*) FROM components
5	WHERE name = 'Servo motors 6'

QUERY PLAN	
	text
1	Finalize Aggregate (cost=142820.76..142820.77 rows=1 width=8) (actual time=577.324..583.392 rows=1 loops=1)
2	-> Gather (cost=142820.55..142820.76 rows=2 width=8) (actual time=576.967..583.385 rows=3 loops=1)
3	Workers Planned: 2
4	Workers Launched: 2
5	-> Partial Aggregate (cost=141820.55..141820.56 rows=1 width=8) (actual time=548.293..548.293 rows=1 loops=3)
6	-> Parallel Bitmap Heap Scan on components (cost=12947.75..141652.11 rows=67375 width=0) (actual time=316.785..545.638 rows=56452 loop...
7	Recheck Cond: ((name)::text = 'Servo motors 6'::text)
8	Rows Removed by Index Recheck: 1311719
9	Heap Blocks: exact=16129 lossy=12259
10	-> Bitmap Index Scan on idx_gin_name (cost=0.00..12907.32 rows=161701 width=0) (actual time=340.092..340.092 rows=169355 loops=1)
11	Index Cond: ((name)::text = 'Servo motors 6'::text)
12	Planning Time: 0.120 ms
13	Execution Time: 583.519 ms

З індексом BRIN:

Query Query History	
1	SET enable_seqscan = off;
2	
3	EXPLAIN ANALYSE
4	SELECT COUNT(*) FROM components
5	WHERE name = 'Servo motors 6'

QUERY PLAN	
	text
1	Finalize Aggregate (cost=140821.77..140821.78 rows=1 width=8) (actual time=423.344..429.456 rows=1 loops=1)
2	-> Gather (cost=140821.55..140821.76 rows=2 width=8) (actual time=423.253..429.451 rows=3 loops=1)
3	Workers Planned: 2
4	Workers Launched: 2
5	-> Partial Aggregate (cost=139821.55..139821.56 rows=1 width=8) (actual time=392.761..392.762 rows=1 loops=3)
6	-> Parallel Bitmap Heap Scan on components (cost=77.83..139653.12 rows=67375 width=0) (actual time=0.908..390.128 rows=56452 loop...
7	Recheck Cond: ((name)::text = 'Servo motors 6'::text)
8	Rows Removed by Index Recheck: 3610248
9	Heap Blocks: lossy=34005
10	-> Bitmap Index Scan on idx_brin_name (cost=0.00..37.40 rows=6489785 width=0) (actual time=2.582..2.582 rows=907640 loops=1)
11	Index Cond: ((name)::text = 'Servo motors 6'::text)
12	Planning Time: 0.106 ms
13	Execution Time: 429.488 ms

Приклад 3: Групування

Без індексів:

1	SET enable_seqscan = on;
2	
3	EXPLAIN ANALYSE
4	SELECT name, device_id, COUNT(*) FROM components
5	WHERE name = 'Servo motors 6'
6	GROUP BY name, device_id

QUERY PLAN	text	
5	Workers Launched: 2	
6	-> Partial GroupAggregate (cost=153459.64..154638.70 rows=67375 width=26) (actual time=304.923..323.950 rows=55070 loops=3)	
7	Group Key: device_id	
8	-> Sort (cost=153459.64..153628.07 rows=67375 width=18) (actual time=304.915..310.887 rows=56452 loops=3)	
9	Sort Key: device_id	
10	Sort Method: external merge Disk: 1704kB	
11	Worker 0: Sort Method: quicksort Memory: 3920kB	
12	Worker 1: Sort Method: quicksort Memory: 3953kB	
13	-> Parallel Seq Scan on components (cost=0.00..148056.19 rows=67375 width=18) (actual time=0.252..289.955 rows=56452 loop...	
14	Filter: ((name)::text = 'Servo motors 6'::text)	
15	Rows Removed by Filter: 3610248	
16	Planning Time: 0.095 ms	
17	Execution Time: 410.371 ms	

З індексом GIN:

1	SET enable_seqscan = off;
2	
3	EXPLAIN ANALYSE
4	SELECT name, device_id, COUNT(*) FROM components
5	WHERE name = 'Servo motors 6'
6	GROUP BY name, device_id

-> Sort (cost=147055.56..147224.00 rows=67375 width=18) (actual time=575.162..580.277 rows=56452 loops=3)
Sort Key: device_id
Sort Method: external merge Disk: 1704kB
Worker 0: Sort Method: quicksort Memory: 3916kB
Worker 1: Sort Method: quicksort Memory: 3955kB
-> Parallel Bitmap Heap Scan on components (cost=12947.75..141652.11 rows=67375 width=18) (actual time=328.803..562.188 rows=56452 loo...
Recheck Cond: ((name)::text = 'Servo motors 6'::text)
Rows Removed by Index Recheck: 1311719
Heap Blocks: exact=14965 lossy=12185
-> Bitmap Index Scan on idx_gin_name (cost=0.00..12907.32 rows=161701 width=0) (actual time=351.406..351.406 rows=169355 loops=1)
Index Cond: ((name)::text = 'Servo motors 6'::text)
Planning Time: 0.783 ms
Execution Time: 670.091 ms

З індексом BRIN:

```

1 SET enable_seqscan = off;
2
3 ▼ EXPLAIN ANALYSE
4 SELECT name, device_id, COUNT(*) FROM components
5 WHERE name = 'Servo motors 6'
6 GROUP BY name, device_id

```

	-> Sort (cost=145056.57..145225.00 rows=67375 width=18) (actual time=428.104..433.298 rows=56452 loops=3)
	Sort Key: device_id
	Sort Method: external merge Disk: 1752kB
	Worker 0: Sort Method: quicksort Memory: 3881kB
	Worker 1: Sort Method: quicksort Memory: 3901kB
	-> Parallel Bitmap Heap Scan on components (cost=77.83..139653.12 rows=67375 width=18) (actual time=0.968..414.598 rows=56452 loop...
	Recheck Cond: ((name)::text = 'Servo motors 6'::text)
	Rows Removed by Index Recheck: 3610248
	Heap Blocks: lossy=33132
	-> Bitmap Index Scan on idx_brin_name (cost=0.00..37.40 rows=6489785 width=0) (actual time=2.777..2.777 rows=907640 loops=1)
	Index Cond: ((name)::text = 'Servo motors 6'::text)
	Planning Time: 0.150 ms
	Execution Time: 527.151 ms

Приклад 4: Сортування

Без індексів:

```

1 SET enable_seqscan = on;
2
3 ▼ EXPLAIN ANALYSE
4 SELECT * FROM components
5 WHERE name = 'Servo motors 6'
6 ORDER BY device_id
7

```

9	-> Parallel Seq Scan on components (cost=0.00..148056.19 rows=67375 width=30) (actual time=0.211..317.074 rows=56452 loop...
10	Filter: ((name)::text = 'Servo motors 6'::text)
11	Rows Removed by Filter: 3610248
12	Planning Time: 0.538 ms
13	Execution Time: 395.369 ms

З індексом GIN:

```

1 SET enable_seqscan = off;
2
3 ▼ EXPLAIN ANALYSE
4 SELECT * FROM components
5 WHERE name = 'Servo motors 6'
6 ORDER BY device_id

```

	Rows Removed by Index Recheck: 1311719
	Heap Blocks: exact=16923 lossy=12079
	-> Bitmap Index Scan on idx_gin_name (cost=0.00..12907.32 rows=161701 width=0) (actual time=335.421..335.422 rows=169355 loops=1)
	Index Cond: ((name)::text = 'Servo motors 6'::text)
	Planning Time: 0.131 ms
	Execution Time: 646.622 ms

З індексом BRIN:

```
1 SET enable_seqscan = off;
2
3 EXPLAIN ANALYSE
4 SELECT * FROM components
5 WHERE name = 'Servo motors 6'
6 ORDER BY device_id
```

-> Parallel Bitmap Heap Scan on components (cost=77.83..139653.12 rows=67375 width=30) (actual time=1.029..394.168 rows=56452 loop...)	
Recheck Cond: ((name)::text = 'Servo motors 6'::text)	
Rows Removed by Index Recheck: 3610248	
Heap Blocks: lossy=33399	
-> Bitmap Index Scan on idx_brin_name (cost=0.00..37.40 rows=6489785 width=0) (actual time=2.921..2.922 rows=907640 loops=1)	
Index Cond: ((name)::text = 'Servo motors 6'::text)	
Planning Time: 0.126 ms	
Execution Time: 484.010 ms	

Приклад 5: Операції з'єднання

Без індексів:

```
1 SET enable_seqscan = on;
2
3 EXPLAIN ANALYSE
4 SELECT c.component_id, c.name, c.device_id, COUNT(*) FROM components c
5 JOIN device d ON c.device_id = d.device_id
6 WHERE c.name = 'Servo motors 6'
7 GROUP BY c.component_id
```

-> Parallel Hash (cost=148056.19..148056.19 rows=67375 width=22) (actual time=313.898..313.898 rows=56452 loops=3)	
Buckets: 262144 Batches: 1 Memory Usage: 11360kB	
-> Parallel Seq Scan on components c (cost=0.00..148056.19 rows=67375 width=22) (actual time=0.201..300.520 rows=56452 loops=3)	
Filter: ((name)::text = 'Servo motors 6'::text)	
Rows Removed by Filter: 3610248	
Planning Time: 2.375 ms	
Execution Time: 588.678 ms	

З індексом GIN:

```
1 SET enable_seqscan = off;
2
3 EXPLAIN ANALYSE
4 SELECT c.component_id, c.name, c.device_id, COUNT(*) FROM components c
5 JOIN device d ON c.device_id = d.device_id
6 WHERE c.name = 'Servo motors 6'
7 GROUP BY c.component_id
```

!9	-> Parallel Bitmap Heap Scan on components c (cost=12947.75..141652.11 rows=67375 width=22) (actual time=316.120..578.114 rows=56452 loo...
!0	Recheck Cond: ((name)::text = 'Servo motors 6'::text)
!1	Rows Removed by Index Recheck: 1311719
!2	Heap Blocks: exact=16105 lossy=12365
!3	-> Bitmap Index Scan on idx_gin_name (cost=0.00..12907.32 rows=161701 width=0) (actual time=341.849..341.849 rows=169355 loops=1)
!4	Index Cond: ((name)::text = 'Servo motors 6'::text)
!5	Planning Time: 0.290 ms
!6	Execution Time: 788.486 ms

З індексом BRIN:

1	SET enable_seqscan = off;
2	
3	EXPLAIN ANALYSE
4	SELECT c.component_id, c.name, c.device_id, COUNT(*) FROM components c
5	JOIN device d ON c.device_id = d.device_id
6	WHERE c.name = 'Servo motors 6'
7	GROUP BY c.component_id
20	Recheck Cond: ((name)::text = 'Servo motors 6'::text)
21	Rows Removed by Index Recheck: 3610248
22	Heap Blocks: lossy=33692
23	-> Bitmap Index Scan on idx_brin_name (cost=0.00..37.40 rows=6489785 width=0) (actual time=2.195..2.195 rows=907640 loops=1)
24	Index Cond: ((name)::text = 'Servo motors 6'::text)
25	Planning Time: 0.183 ms
26	Execution Time: 599.479 ms

Результати:

Виконання запитів без індексів у всіх приклад виявилися швидшими чим з індексами GIN і BRIN. А з індексом BRIN запити трохи швидше виконалися чим з GIN. Такі результати виникли оскільки запити із Seq Scan ефективні, коли в результаті є велика кількість рядків, які підходять під задану умову. З індексом GIN запит виконується довше за Seq Scan і BRIN, оскільки багато рядків підходять під умову. GIN є ефективним для складних умов, за якими знаходяться невелика кількість рядків. BRIN призначений для колонок з лінійною залежністю, але він виявився ефективнішим за GIN.

Розробка тригерів

Створення функції для тригера:

```
CREATE OR REPLACE FUNCTION insert_update_trigger()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        RAISE NOTICE 'Inserted factory with ID: %', OLD.factory_id;
    ELSIF TG_OP = 'UPDATE' THEN
        RAISE NOTICE 'Updated factory with ID: %', NEW.factory_id;
    END IF;

    RETURN NEW;
EXCEPTION
    WHEN OTHERS THEN
        RAISE NOTICE 'An error occurred in insert_update_trigger trigger: %s',
SQLERRM;
        RETURN NULL;
END;
$$ LANGUAGE plpgsql;
```

Створення тригера для таблиці фабрика:

```
CREATE TRIGGER insert_update_trigger_t
AFTER INSERT OR UPDATE ON factory
FOR EACH ROW
EXECUTE FUNCTION insert_update_trigger();
```

Додавання рядка в таблицю фабрика:

Query	Query History
1	INSERT INTO factory (name, specialization, address)
2	VALUES ('factory_trigger', 'spec_trigger', 'address_trigger')

Data Output	Messages	Notifications
ПОВІДОМЛЕННЯ: Inserted factory with ID: <NULL>		
INSERT 0 1		
Query returned successfully in 37 msec.		

Звідси видно, що тригер спрацював, оскільки вивелося повідомлення про вставку.

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	1	Boston Dynamics	industrial robots for assembly and processing	Address 10
2	2	ABB Robotics	mobile robots with advanced maneuvering capabilities	Address 2
3	3	Boston Dynamics	industrial robots for assembly and processing	Address 1
4	4	Boston Dynamics	industrial robots for assembly and processing	Address 0
5	6	KUKA Robotics	industrial robots for automation of production process...	Address 5
6	7	ABB Robotics	industrial robots for automation of production process...	Address 3
7	10	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 7
8	13	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 6
9	15	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 9
10	25	ABB Robotics	industrial robots for assembly and processing	Address 4
11	27	factory_trigger	spec_trigger	address_trigger

Зміна рядка в таблиці фабрика:

Query

Query History

1

▼

UPDATE factory

2

SET name = 'newvalue_factory',

3

specialization = 'newvalue_spec',

4

address = 'newvalue_address'

5

WHERE factory_id = 27;

Data Output

Messages

Notifications

ПОВІДОМЛЕННЯ: Updated factory with ID: 27

UPDATE 1

Query returned successfully in 34 msec.

Звідси видно, що тригер спрацював, оскільки вивелося повідомлення про зміну.

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	1	Boston Dynamics	industrial robots for assembly and processing	Address 10
2	2	ABB Robotics	mobile robots with advanced maneuvering capabilities	Address 2
3	3	Boston Dynamics	industrial robots for assembly and processing	Address 1
4	4	Boston Dynamics	industrial robots for assembly and processing	Address 0
5	6	KUKA Robotics	industrial robots for automation of production process...	Address 5
6	7	ABB Robotics	industrial robots for automation of production process...	Address 3
7	10	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 7
8	13	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 6
9	15	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 9
10	25	ABB Robotics	industrial robots for assembly and processing	Address 4
11	27	newvalue_factory	newvalue_spec	newvalue_address

Використання рівнів ізоляції

READ COMMITTED

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	1	Boston Dynamics	industrial robots for assembly and processing	Address 10
2	2	ABB Robotics	mobile robots with advanced maneuvering capabilities	Address 2
3	3	Boston Dynamics	industrial robots for assembly and processing	Address 1
4	4	Boston Dynamics	industrial robots for assembly and processing	Address 0
5	6	KUKA Robotics	industrial robots for automation of production process...	Address 5
6	7	ABB Robotics	industrial robots for automation of production process...	Address 3
7	10	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 7
8	13	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 6
9	15	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 9
10	25	ABB Robotics	industrial robots for assembly and processing	Address 4
11	27	newvalue_factory	newvalue_spec	newvalue_address
12	28	name5	spec6	adrs5

Вікно 1:

```
1 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
2
3 BEGIN;
4
5 UPDATE factory SET name = 'AAAA' WHERE factory_id = 28;
6
7
```

ПОВІДОМЛЕННЯ: Updated factory with ID: 28
UPDATE 1

Query returned successfully in 34 msec.

Вікно 2:

Query Query History

```
1 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
2
3 BEGIN;
4
5 SELECT * FROM factory WHERE factory_id = 28;
6
7
```

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	28	name5	spec6	adrs5

Звідси видно, що без фіксації змін у першому вікні, у другому вікні немає змін і назва фабрики лишилося таким самим.

Вікно 1:

Query Query History

```
1 SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
2
3 COMMIT;
4
5
```

COMMIT

Query returned successfully in 35 msec.

Вікно 2:

1	SET TRANSACTION ISOLATION LEVEL READ COMMITTED;			
2				
3				
4				
5	SELECT * FROM factory WHERE factory_id = 28;			
6				

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	28	AAAA	spec6	adrs5

Тепер видно, що після фіксації у першому вікні, у другому вікні відображаються зміни, назва фабрики змінилася.

REPEATABLE READ

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	1	Boston Dynamics	industrial robots for assembly and processing	Address 10
2	2	ABB Robotics	mobile robots with advanced maneuvering capabilities	Address 2
3	3	Boston Dynamics	industrial robots for assembly and processing	Address 1
4	4	Boston Dynamics	industrial robots for assembly and processing	Address 0
5	6	KUKA Robotics	industrial robots for automation of production process...	Address 5
6	7	ABB Robotics	industrial robots for automation of production process...	Address 3
7	10	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 7
8	13	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 6
9	15	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 9
10	25	ABB Robotics	industrial robots for assembly and processing	Address 4
11	27	newvalue_factory	newvalue_spec	newvalue_address
12	28	BBBB	spec6	adrs5

Вікно 1:

1	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;			
2				
3				
4	BEGIN;			
5				
6	SELECT * FROM factory WHERE factory_id = 28;			

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	28	BBBB	spec6	adrs5

Вікно 2:

```
3 BEGIN;
4
5 UPDATE factory SET name = 'CCCCC' WHERE factory_id = 28;
6
7 COMMIT;
8
```

ПОВІДОМЛЕННЯ: Updated factory with ID: 28
COMMIT

Query returned successfully in 34 msec.

Змінюємо таблицю з фіксацією

Вікно 1:

```
6 SELECT * FROM factory WHERE factory_id = 28;
7
8 COMMIT;
```

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	28	BBBB	spec6	adrs5

Видно, що навіть після фіксації змін у другому вікні, у першому вікні не відображаються зміни.

SERIALIZABLE

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	1	Boston Dynamics	industrial robots for assembly and processing	Address 10
2	2	ABB Robotics	mobile robots with advanced maneuvering capabilities	Address 2
3	3	Boston Dynamics	industrial robots for assembly and processing	Address 1
4	4	Boston Dynamics	industrial robots for assembly and processing	Address 0
5	6	KUKA Robotics	industrial robots for automation of production process...	Address 5
6	7	ABB Robotics	industrial robots for automation of production process...	Address 3
7	10	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 7
8	13	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 6
9	15	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 9
10	25	ABB Robotics	industrial robots for assembly and processing	Address 4
11	27	newvalue_factory	newvalue_spec	newvalue_address
12	29	ser_factory	ser_spec	ser_address

Вікно 1:

```
1 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
2  
3  
4 BEGIN;  
5  
6 DELETE FROM factory WHERE factory_id = 29;  
7
```

DELETE 1

Query returned successfully in 212 msec.

Видаляємо рядок без фіксації зміни.

Вікно 2:

Query Query History

```
1 SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
2  
3  
4 BEGIN;  
5  
6 ▾ INSERT INTO factory (name, specialization, address)  
7 VALUES ('ser_factory', 'ser_spec', 'ser_address');  
8  
9 COMMIT;  
10
```

Total rows: 0 of 0 Waiting for the query to complete... 00:00:37.227 Ln 10, Col 1

Як видно без фіксації зміни у першому вікні, у другому вікні запит не може виконатися.

Вікно 1:

```
8 COMMIT;
```

COMMIT

Query returned successfully in 34 msec.

Робимо фіксацію зміни.

Вікно 2:

ПОВІДОМЛЕННЯ: Inserted factory with ID: <NULL>

COMMIT

Query returned successfully in 1 min 18 secs.

	factory_id [PK] integer	name character varying (30)	specialization character varying (60)	address character varying (70)
1	1	Boston Dynamics	industrial robots for assembly and processing	Address 10
2	2	ABB Robotics	mobile robots with advanced maneuvering capabilities	Address 2
3	3	Boston Dynamics	industrial robots for assembly and processing	Address 1
4	4	Boston Dynamics	industrial robots for assembly and processing	Address 0
5	6	KUKA Robotics	industrial robots for automation of production process...	Address 5
6	7	ABB Robotics	industrial robots for automation of production process...	Address 3
7	10	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 7
8	13	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 6
9	15	Boston Dynamics	mobile robots with advanced maneuvering capabilities	Address 9
10	25	ABB Robotics	industrial robots for assembly and processing	Address 4
11	27	newvalue_factory	newvalue_spec	newvalue_address
12	31	ser_factory	ser_spec	ser_address

Тепер у другому вікні запит зміг виконатися, оскільки відбулася фіксація змін у першому вікні.