

**Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп’ютерних
систем**

**Лабораторна робота №2
з дисципліни
“Бази даних і засоби управління”
Тема: “Засоби оптимізації роботи СУБД PostgreSQL”**

Виконав: Воеводін Ілля Петрович
Студент групи КВ-84
Перевірів(ла): _____

Постановка задачі для варіанту 9

Завдання роботи полягає у наступному:

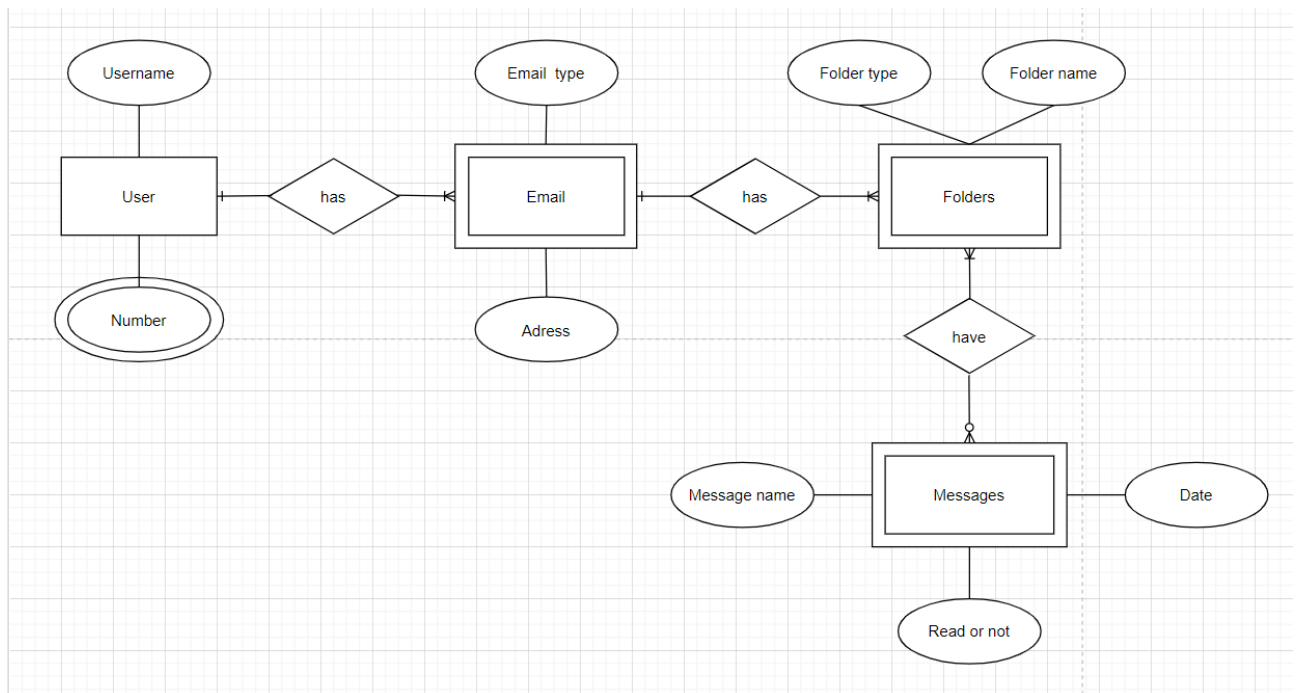
1. Перетворити модуль “Модель” з шаблону MVC лабораторної роботи №2 у вигляд об’єктно-реляційної проекції (ORM).
2. Створити та проаналізувати різні типи індексів у PostgreSQL.
3. Розробити тригер бази даних PostgreSQL.

Вимоги до пункту завдання №1

Для перетворення функцій, що реалізують запити до об’єктної бази даних, необхідно встановити бібліотеку sqlalchemy, налаштувати програму на роботу з ORM, розробити класи-сутності для об’єктів-сутностей, представлених відповідними таблицями БД та пов’язаних зв’язками 1:M, M:M та 1:1 виконати опис схеми бази даних. Особливу увагу приділити контролю зовнішніх зв’язків між таблицями засобами ORM.

Замінити виклики запитів мовою SQL на відповідні запити засобами SQLAlchemy по роботі з об’єктами. Обов’язковим є реалізація вставки, вилучення та редагування екземплярів класів-сутностей. Розробка запитів на генерацію даних та пошук екземплярів класів-сутностей вітається, але не є обов’язковою.

Інтерфейси функцій (вхідні та вихідні аргументи функцій модуля “Модель”) мають залишитись без змін.



Вимоги до пункту завдання №2

Відповідно до варіанту індексування продемонструвати на прикладах запитів SQL SELECT підвищення швидкодії їх виконання з використанням індексів, а

також пояснити чому для деяких випадків індексування використовувати недоцільно. При цьому для наочного представлення слід використати функцію генерування рандомізованих даних з лабораторної роботи №2, створивши необхідну кількість тестових даних. Навести 4-5 прикладів запитів SELECT (із виведенням результуючих даних), що містять фільтрацію, агрегатні функції, групування та сортування (у необхідних комбінаціях).

Вимоги до пункту завдання №3

Створити тригер бази даних PostgreSQL відповідно до варіанта. Тригерна функція має включати обробку запису, що модифікується (вставляється або вилучається), умовні оператори, курсорні цикли та обробку виключних ситуацій. Виконати відлагодження тригера при різних вхідних даних, навівши 2-3 приклади його використання.

9	<i>BTree, BRIN</i>	<i>before delete, update</i>
---	--------------------	------------------------------

Результати

- Вимоги до пункту завдання №1

```
class Users(Base):
    __tablename__ = 'users'
    user_id = Column(Integer, primary_key=True)
    username = Column(String)
    phone_number = Column(Integer)

    def __repr__(self):
        return f"Users (user_id='{self.user_id}', username='{self.username}', phone_number='{self.phone_number}')"

class Email(Base):
    __tablename__ = 'email'
    email_id = Column(Integer, primary_key=True)
    user_id = Column(Integer, ForeignKey('users.user_id'))
    email_address = Column(String)
    email_type = Column(String)

    def __repr__(self):
        return f"Email (email_id='{self.email_id}', users_id='{self.user_id}', email_address='{self.email_address}', " \
            f"email_type='{self.email_type}')"

class Folders(Base):
    __tablename__ = 'folders'
    folders_id = Column(Integer, primary_key=True)
    email_id = Column(Integer, ForeignKey('email.email_id'))
    folder_name = Column(String)
    folder_type = Column(String)

    def __repr__(self):
        return f"Folders (folders_id='{self.folders_id}', email_id='{self.email_id}', " \
            f"folder_name='{self.folder_name}', folder_type='{self.folder_type}')"

class FolMes(Base):
    __tablename__ = 'folders_messages'
    folders_id = Column(Integer, primary_key=True, unique=False)
    messages_id = Column(Integer, ForeignKey('messages.messages_id'), unique=False)

    def __repr__(self):
        return f"Folders_messages (folders_id='{self.folders_id}', messages_id='{self.messages_id}')"

class Messages(Base):
    __tablename__ = 'messages'
    messages_id = Column(Integer, primary_key=True)
    read_or_not = Column(Boolean)
    message_name = Column(String)
    message_date = Column(Date)

    def __repr__(self):
        return f"Messages (messages_id='{self.messages_id}', read_or_not='{self.read_or_not}', " \
            f"message_name='{self.message_name}', message_date='{self.message_date}')"

```

```

def create_item(table_name, *value):
    keys = get_all_keys(table_name)

    i = 0
    for key in keys:
        d = getattr(table_name, key).type
        if type(d) is Integer:
            d = 0
        elif type(d) is Boolean:
            d = False
        elif type(d) is Date:
            d = DT.datetime.strptime('1,1,2020', '%d,%m,%Y').date()
        elif type(d) is String:
            d = 'str'
        if type(d) != type(value[i]):
            raise mvc_exc.ValueTypeError(f"Value: <{value[i]}> must be the same type as key!")
        i = i + 1

    with session_scope() as s:
        check_id = {keys[0]: value[0]}
        check = s.query(table_name).filter_by(**check_id).all()

        if not check:
            check_items = {keys[i]: value[i] for i in range(1, len(keys))}
        else:
            raise mvc_exc.ItemAlreadyStored(
                f"Can't create item with {keys[0]} = {value[0]} because it is already exists")

        check = s.query(table_name).filter_by(**check_items).all()
        if not check:
            create_it = {}
            i = 0
            for key in keys:
                create_it[key] = value[i]
                i = i + 1
        else:
            raise mvc_exc.ItemAlreadyStored(f"Can't create item because all data for its id {keys[0]} = {value[0]} "
                f"already exists")

        s.add(table_name(**create_it))

```

```

def read_item(table_name, **name):
    with session_scope() as s:
        keys = get_all_keys(table_name)
        for key, value in name.items():
            f = False

            for item in keys:
                if key == item:
                    f = True
                    d = getattr(table_name, key).type
                    if type(d) is Integer:
                        d = 0
                    elif type(d) is Boolean:
                        d = False
                    elif type(d) is Date:
                        d = DT.datetime.strptime('1,1,2020', '%d,%m,%Y').date()
                    elif type(d) is String:
                        d = 'str'
                    if type(d) != type(value):
                        raise mvc_exc.ValueTypeError(f"Value: <{value}> must be the same type as key!")

            if f is False:
                raise mvc_exc.KeyNameError(f"Key: <{key}> doesn't exist in table '{table_name.__tablename__}'!")

        read_it = s.query(table_name).filter_by(**name).all()

        if not read_it:
            raise mvc_exc.ItemNotStored(f"{name.items()} not stored")

        for rows in read_it:
            print(rows)

def read_items(table_name):
    with session_scope() as s:
        keys = get_all_keys(table_name)
        read_all = s.query(table_name).order_by(getattr(table_name, keys[0]).asc()).all()

        if not read_all:
            print(f"Table {table_name} is empty")
        for row in read_all:
            print(row)

```

```

def update_item(table_name, new_value, **name):
    with session_scope() as s:
        keys = get_all_keys(table_name)
        for key, value in name.items():
            f = False

            for item in keys:
                if key == item:
                    f = True
                    d = getattr(table_name, key).type
                    if type(d) is Integer:
                        d = 0
                    elif type(d) is Boolean:
                        d = False
                    elif type(d) is Date:
                        d = DT.datetime.strptime('1,1,2020', '%d,%m,%Y').date()
                    elif type(d) is String:
                        d = 'str'

                    if type(d) != type(value):
                        raise mvc_exc.ValueTypeError(f"Value: <{value}> must be the same type as key!")
                    elif type(d) != type(new_value):
                        raise mvc_exc.ValueTypeError(f"Value: <{new_value}> must be the same type as key!")

            if f is False:
                raise mvc_exc.KeyNameError(f"Key: <{key}> doesn't exist in table '{table_name.__tablename__}'!")

        try:
            update_it = s.query(table_name).filter_by(**name).one()
        except Exception:
            raise mvc_exc.ItemNotStored(f"Can't update {name.items()} because it is not stored")

        for key, value in name.items():
            setattr(update_it, key, new_value)
            s.add(update_it)

```

```

def delete_items(table_name, **name):
    with session_scope() as s:
        keys = get_all_keys(table_name)
        for key, value in name.items():
            f = False

            for item in keys:
                if key == item:
                    f = True
                    d = getattr(table_name, key).type
                    if type(d) is Integer:
                        d = 0
                    elif type(d) is Boolean:
                        d = False
                    elif type(d) is Date:
                        d = DT.datetime.strptime('1,1,2020', '%d,%m,%Y').date()
                    elif type(d) is String:
                        d = 'str'

                    if type(d) != type(value):
                        raise mvc_exc.ValueTypeError(f"Value: <{value}> must be the same type as key!")

            if f is False:
                raise mvc_exc.KeyNameError(f"Key: <{key}> doesn't exist in table '{table_name.__tablename__}'!")

        try:
            delete_it = s.query(table_name).filter_by(**name).one()
        except Exception:
            raise mvc_exc.ItemNotStored(f"Can't delete {name.items()} because it is not stored")

        s.delete(delete_it)

```

Вимоги до пункту завдання №2

Запит 1:

Index

Data Output	Explain	Messages	Notifications	Query Editor	Query History
QUERY PLAN text				<pre>1 DROP INDEX IF EXISTS eml_idx; 2 CREATE INDEX eml_idx ON email(email_id); 3 4 --set enable_indexscan to on; 5 EXPLAIN ANALYZE SELECT count(*) FROM email 6 WHERE (email_id BETWEEN 20020 AND 50050) AND email_type LIKE '%A%' 7 AND email_address LIKE '%1%' 8 GROUP BY email_id;</pre>	
1	GroupAggregate (cost=0.29..1293.28 rows=616 width=12) (actual time=0.091..6.911 rows=608 loop...				
2	Group Key: email_id				
3	-> Index Scan using eml_idx on email (cost=0.29..1284.04 rows=616 width=4) (actual time=0.068..6...				
4	Index Cond: ((email_id >= 20020) AND (email_id <= 50050))				
5	Filter: (((email_type)::text ~~ '%A% '::text) AND ((email_address)::text ~~ '%1% '::text))				
6	Rows Removed by Filter: 29423				
7	Planning Time: 1.140 ms				
8	Execution Time: 6.955 ms				

BRIN

Data Output	Messages	Notifications	Explain	Query Editor	Query History
QUERY PLAN text				<pre>1 DROP INDEX IF EXISTS eml_brin_idx; 2 CREATE INDEX eml_brin_idx ON email USING brin(email_id); 3 4 set enable_indexscan to off; 5 EXPLAIN ANALYZE SELECT count(*) FROM email 6 WHERE (email_id BETWEEN 20020 AND 50050) AND email_type LIKE '%A%' 7 AND email_address LIKE '%1%' 8 GROUP BY email_id;</pre>	
1	HashAggregate (cost=1444.30..1450.46 rows=616 width=12) (actual time=9.735..9.806 rows=608 loop...				
2	Group Key: email_id				
3	-> Bitmap Heap Scan on email (cost=12.22..1441.22 rows=616 width=4) (actual time=1.691..9.343 ro...				
4	Recheck Cond: ((email_id >= 20020) AND (email_id <= 50050))				
5	Rows Removed by Index Recheck: 31047				
6	Filter: (((email_type)::text ~~ '%A% '::text) AND ((email_address)::text ~~ '%1% '::text))				
7	Rows Removed by Filter: 29423				
8	Heap Blocks: lossy=384				
9	-> Bitmap Index Scan on eml_brin_idx (cost=0.00..12.06 rows=40000 width=0) (actual time=0.041....				
10	Index Cond: ((email_id >= 20020) AND (email_id <= 50050))				
11	Planning Time: 1.078 ms				
12	Execution Time: 9.875 ms				

Запит 2:

Index

Data Output	Explain	Messages	Notifications	Query Editor	Query History
QUERY PLAN text				<pre>1 DROP INDEX IF EXISTS folder_idx; 2 CREATE INDEX folder_idx ON folders(folders_id); 3 4 --set enable_indexscan to on; 5 EXPLAIN ANALYZE SELECT max(email_id) FROM folders 6 WHERE folders_id < 10000 AND folder_type LIKE 'L_' 7 GROUP BY email_id;</pre>	
1	HashAggregate (cost=393.64..397.88 rows=424 width=8) (actual time=2.655....				
2	Group Key: email_id				
3	-> Index Scan using folder_idx on folders (cost=0.29..391.51 rows=425 width=...				
4	Index Cond: (folders_id < 10000)				
5	Filter: (((folder_type)::text ~~ 'L_ '::text)				
6	Rows Removed by Filter: 9585				
7	Planning Time: 1.441 ms				
8	Execution Time: 2.734 ms				

BRIN

Data Output	Messages	Notifications	Explain	Query Editor	Query History
QUERY PLAN text				<pre>1 DROP INDEX IF EXISTS fld_brin_idx; 2 CREATE INDEX fld_brin_idx ON folders USING brin(folders_id); 3 4 set enable_indexscan to off; 5 EXPLAIN ANALYZE SELECT max(email_id) FROM folders 6 WHERE folders_id < 10000 AND folder_type LIKE 'L_' 7 8 GROUP BY email_id;</pre>	
1	HashAggregate (cost=992.25..996.49 rows=424 width=8) (actual time=2.704..2.849 row...				
2	Group Key: email_id				
3	-> Bitmap Heap Scan on folders (cost=12.14..990.13 rows=425 width=4) (actual time=0...				
4	Recheck Cond: (folders_id < 10000)				
5	Rows Removed by Index Recheck: 7595				
6	Filter: (((folder_type)::text ~~ 'L_ '::text)				
7	Rows Removed by Filter: 9585				
8	Heap Blocks: lossy=128				
9	-> Bitmap Index Scan on fld_brin_idx (cost=0.00..12.03 rows=16666 width=0) (actua...				
10	Index Cond: (folders_id < 10000)				
11	Planning Time: 1.387 ms				
12	Execution Time: 2.948 ms				

Запит 3:

Index

Data Output	Explain	Messages	Notifications	Query Editor	Query History
QUERY PLAN text				<pre>1 DROP INDEX IF EXISTS user_idx; 2 CREATE INDEX user_idx ON users(user_id); 3 4 --set enable_indexscan to on; 5 EXPLAIN ANALYZE SELECT * FROM users 6 WHERE user_id >= 80000 AND (phone_number BETWEEN 400000 AND 600000) 7 AND username LIKE '_F' 8 9 ORDER BY user_id ASC;</pre>	
1	Index Scan using user_idx on users (cost=0.29..851.12 rows=238 width=11) (actual time=0.222..5.9..)				
2	Index Cond: (user_id >= 80000)				
3	Filter: ((phone_number >= 400000) AND (phone_number <= 600000) AND ((username)::text ~ '_F':				
4	Rows Removed by Filter: 19839				
5	Planning Time: 1.828 ms				
6	Execution Time: 5.964 ms				

BRIN

Data Output	Messages	Notifications	Explain	Query Editor	Query History
QUERY PLAN text				<pre>1 DROP INDEX IF EXISTS usr_brin_idx; 2 CREATE INDEX usr_brin_idx ON users USING brin(user_id); 3 4 set enable_indexscan to off; 5 EXPLAIN ANALYZE SELECT * FROM users 6 WHERE user_id >= 80000 AND (phone_number BETWEEN 400000 AND 600000) 7 AND username LIKE '_F' 8 9 ORDER BY user_id ASC;</pre>	
1			Sort (cost=1362.52..1363.11 rows=238 width=11) (actual time=3.382..3.387 rows=161 lo...		
2			Sort Key: user_id		
3			Sort Method: quicksort Memory: 32kB		
4			-> Bitmap Heap Scan on users (cost=12.12..1353.12 rows=238 width=11) (actual time=...		
5			Recheck Cond: (user_id >= 80000)		
6			Rows Removed by Index Recheck: 8969		
7			Filter: ((phone_number >= 400000) AND (phone_number <= 600000) AND ((username...		
8			Rows Removed by Filter: 19839		
9			Heap Blocks: lossy=157		
10			-> Bitmap Index Scan on usr_brin_idx (cost=0.00..12.06 rows=40000 width=0) (actu...		
11			Index Cond: (user_id >= 80000)		
12			Planning Time: 0.919 ms		
13			Execution Time: 3.414 ms		

Запит 4:

Index

Data Output	Explain	Messages	Notifications	Query Editor	Query History
QUERY PLAN text				<pre>1 DROP INDEX IF EXISTS mes_idx; 2 CREATE INDEX mes_idx ON messages(messages_id); 3 4 set enable_seqscan to off; 5 EXPLAIN ANALYZE SELECT min(message_date) FROM messages 6 WHERE messages_id < 90000 AND read_or_not = True 7 AND message_name LIKE '%DF%' 8 9 GROUP BY messages_id;</pre>	
1	GroupAggregate (cost=0.29..3290.23 rows=4 width=8) (actual time=0.139..19.870 rows=211 loops...				
2	Group Key: messages_id				
3	-> Index Scan using mes_idx on messages (cost=0.29..3290.17 rows=4 width=8) (actual time=0.0...				
4	Index Cond: (messages_id < 90000)				
5	Filter: (read_or_not AND ((message_name)::text ~ '%DF%':text))				
6	Rows Removed by Filter: 89788				
7	Planning Time: 1.005 ms				
8	Execution Time: 19.922 ms				

BRIN

Data Output	Messages	Notifications	Explain	Query Editor	Query History
QUERY PLAN text				<pre>1 DROP INDEX IF EXISTS mes_brin_idx; 2 CREATE INDEX mes_brin_idx ON messages USING brin(messages_id); 3 4 set enable_seqscan to off; 5 6 EXPLAIN ANALYZE SELECT min(message_date) FROM messages 7 WHERE messages_id < 90000 AND read_or_not = True 8 AND message_name LIKE '%DF%' 9 10 GROUP BY messages_id;</pre>	
1			GroupAggregate (cost=2053.19..2053.26 rows=4 width=8) (actual time=14.581..14.646 rows=211 loo...		
2			Group Key: messages_id		
3			-> Sort (cost=2053.19..2053.20 rows=4 width=8) (actual time=14.575..14.583 rows=211 loops=1)		
4			Sort Key: messages_id		
5			Sort Method: quicksort Memory: 34kB		
6			-> Bitmap Heap Scan on messages (cost=12.16..2053.15 rows=4 width=8) (actual time=0.057..1...		
7			Recheck Cond: (messages_id < 90000)		
8			Rows Removed by Index Recheck: 4665		
9			Filter: (read_or_not AND ((message_name)::text ~ '%DF%':text))		
10			Rows Removed by Filter: 89788		
11			Heap Blocks: lossy=512		
12			-> Bitmap Index Scan on mes_brin_idx (cost=0.00..12.16 rows=99999 width=0) (actual time=0...		
13			Index Cond: (messages_id < 90000)		
14			Planning Time: 0.942 ms		
15			Execution Time: 14.683 ms		

Before delete trigger

```
DROP TABLE IF EXISTS deleted_users;  
DROP TRIGGER IF EXISTS delete_trigger ON users;
```

```
CREATE TABLE deleted_users (  
    user_id INT NOT NULL UNIQUE,  
    username VARCHAR NOT NULL,  
    phone_number INT NOT NULL  
);
```

```
CREATE OR REPLACE FUNCTION delete_it()  
    RETURNS TRIGGER  
    LANGUAGE PLPGSQL  
    AS  
    $$  
BEGIN  
    IF OLD.user_id > 10 THEN  
        INSERT INTO deleted_users(user_id,username,phone_number)  
        VALUES(OLD.user_id,OLD.username,OLD.phone_number);  
    END IF;  
  
    RETURN OLD;  
END;  
$$;
```

```
CREATE TRIGGER delete_trigger  
    BEFORE DELETE  
    ON users  
    FOR EACH ROW  
    EXECUTE PROCEDURE delete_it();
```

```
1  DROP TABLE IF EXISTS deleted_users;  
2  DROP TRIGGER IF EXISTS delete_trigger ON users;  
3  
4  CREATE TABLE deleted_users (  
5      user_id INT NOT NULL UNIQUE,  
6      username VARCHAR NOT NULL,  
7      phone_number INT NOT NULL  
8  );  
9  
10 CREATE OR REPLACE FUNCTION delete_it()  
11     RETURNS TRIGGER  
12     LANGUAGE PLPGSQL  
13     AS  
14     $$  
15 BEGIN  
16     IF OLD.user_id > 10 THEN  
17         INSERT INTO deleted_users(user_id,username,phone_number)  
18         VALUES(OLD.user_id,OLD.username,OLD.phone_number);  
19     END IF;  
20  
21     RETURN OLD;  
22 END;  
23 $$;  
24  
25 CREATE TRIGGER delete_trigger  
26     BEFORE DELETE  
27     ON users  
28     FOR EACH ROW  
29     EXECUTE PROCEDURE delete_it();
```

Перевірка роботи тригера:





```
-----| Menu |-----
1. Read 1 item
2. Read table
3. Create item
4. Update item
5. Delete item
6. Input random data in tables
7. SQL query

Select option by number...
5
Choose table and input it's name:
< users >, < email >, < folders >, < folders_messages >, < messages >
users
===== users TABLE =====
['user_id', 'username', 'phone_number']
users (user_id='1', username='AF', phone_number='88055')
users (user_id='4', username='BB', phone_number='659761')
users (user_id='5', username='MN', phone_number='824091')
users (user_id='6', username='GG', phone_number='474651')
users (user_id='7', username='CC', phone_number='573121')
users (user_id='9', username='SD', phone_number='537356')
users (user_id='10', username='GY', phone_number='625091')
users (user_id='11', username='FA', phone_number='549064')
Choose key and value to delete:

Input key:
username
Input value:
FA
===== users TABLE =====
['user_id', 'username', 'phone_number']
users (user_id='1', username='AF', phone_number='88055')
users (user_id='4', username='BB', phone_number='659761')
users (user_id='5', username='MN', phone_number='824091')
users (user_id='6', username='GG', phone_number='474651')
users (user_id='7', username='CC', phone_number='573121')
users (user_id='9', username='SD', phone_number='537356')
users (user_id='10', username='GY', phone_number='625091')

Process finished with exit code 0
```

Log-таблиця deleted_users з видаленими даними таблиці users

	Data Output	Explain	Messages	Notifications
	 user_id integer	 username character varying	 phone_number integer	
1	13	SS		102469
2	11	FA		549064

```

-----| Menu |-----
1. Read 1 item
2. Read table
3. Create item
4. Update item
5. Delete item
6. Input random data in tables
7. SQL query





Select option by number...
5
Choose table and input it's name:
< users >, < email >, < folders >, <folders_messages >, < messages >
users
===== users TABLE =====
['user_id', 'username', 'phone_number']
users (user_id='1', username='AF', phone_number='88055')
users (user_id='4', username='BB', phone_number='659761')
users (user_id='5', username='MN', phone_number='824091')
users (user_id='6', username='GG', phone_number='474651')
users (user_id='7', username='CC', phone_number='573121')
users (user_id='9', username='SD', phone_number='537356')
users (user_id='10', username='GY', phone_number='625091')
Choose key and value to delete:

Input key:
user_id
Input value:
10
===== users TABLE =====
['user_id', 'username', 'phone_number']
users (user_id='1', username='AF', phone_number='88055')
users (user_id='4', username='BB', phone_number='659761')
users (user_id='5', username='MN', phone_number='824091')
users (user_id='6', username='GG', phone_number='474651')
users (user_id='7', username='CC', phone_number='573121')
users (user_id='9', username='SD', phone_number='537356')

Process finished with exit code 0

```

Table deleted_users has the same data because user_id <10

Data Output	Explain	Messages	Notifications
 user_id integer	 username character varying	 phone_number integer	
1	13	SS	102469
2	11	FA	549064

Update trigger

```
DROP TABLE IF EXISTS update_id_log;  
DROP TRIGGER IF EXISTS update_trigger ON messages;
```

```
CREATE TABLE update_id_log(  
    messages_id INT NOT NULL,  
    message_name VARCHAR NOT NULL  
);
```

```
CREATE OR REPLACE FUNCTION update_it()  
    RETURNS TRIGGER  
    LANGUAGE PLPGSQL  
    AS  
    $$  
    DECLARE  
        up_curs CURSOR FOR  
            SELECT * FROM messages;  
    BEGIN  
        FOR mes IN up_curs LOOP  
            IF mes.read_or_not = False AND mes.message_date < now() THEN  
                INSERT INTO update_id_log(messages_id,message_name)  
                    VALUES (mes.messages_id,mes.message_name);  
            END IF;  
        END LOOP;  
        RETURN NEW;  
    END;  
    $;
```

```
CREATE TRIGGER update_trigger  
    BEFORE UPDATE  
    ON messages  
    FOR EACH STATEMENT  
    EXECUTE PROCEDURE update_it();
```

```

1 DROP TABLE IF EXISTS update_id_log;
2 DROP TRIGGER IF EXISTS update_trigger ON messages;
3
4 CREATE TABLE update_id_log(
5     messages_id INT NOT NULL,
6     message_name VARCHAR NOT NULL
7 );
8
9 CREATE OR REPLACE FUNCTION update_it()
10 RETURNS TRIGGER
11 LANGUAGE PLPGSQL
12 AS
13 $$
14 DECLARE
15     up_curs CURSOR FOR
16     SELECT * FROM messages;
17 BEGIN
18     FOR mes IN up_curs LOOP
19         IF mes.read_or_not = False AND mes.message_date < now() THEN
20             INSERT INTO update_id_log(messages_id,message_name)
21             VALUES (mes.messages_id,mes.message_name);
22         END IF;
23     END LOOP;
24     RETURN NEW;
25 END;
26 $$;
27
28 CREATE TRIGGER update_trigger
29 BEFORE UPDATE
30 ON messages
31 FOR EACH STATEMENT
32 EXECUTE PROCEDURE update_it();

```

Перевірка роботи тригера:

```

-----| Menu |-----
1. Read 1 item
2. Read table
3. Create item
4. Update item
5. Delete item
6. Input random data in tables
7. SQL query

Select option by number...
4
Choose table and input it's name:
< users >, < email >, < folders >, < folders_messages >, < messages >
messages
===== messages TABLE =====
['messages_id', 'read_or_not', 'message_name', 'message_date']
messages (messages_id='1', read_or_not='True', message_name='LFSY', message_date='2019-07-24')
messages (messages_id='2', read_or_not='False', message_name='FIFF', message_date='2020-12-22')
messages (messages_id='3', read_or_not='False', message_name='YYUS', message_date='2019-09-22')
messages (messages_id='4', read_or_not='False', message_name='YVXH', message_date='2019-05-11')
messages (messages_id='5', read_or_not='False', message_name='UKWV', message_date='2019-10-23')
messages (messages_id='6', read_or_not='True', message_name='HXPQ', message_date='2018-02-23')
messages (messages_id='7', read_or_not='True', message_name='ICUR', message_date='2017-05-06')
messages (messages_id='8', read_or_not='True', message_name='GLMM', message_date='2019-03-03')
messages (messages_id='9', read_or_not='False', message_name='AAEE', message_date='2018-07-16')
Choose key and value to update:

Input key:
message_date
Input value:
2020-12-22
Input new value:
2018-12-18
===== messages TABLE =====
['messages_id', 'read_or_not', 'message_name', 'message_date']
messages (messages_id='1', read_or_not='True', message_name='LFSY', message_date='2019-07-24')
messages (messages_id='2', read_or_not='False', message_name='FIFF', message_date='2018-12-30')
messages (messages_id='3', read_or_not='False', message_name='YYUS', message_date='2019-09-22')
messages (messages_id='4', read_or_not='False', message_name='YVXH', message_date='2019-05-11')
messages (messages_id='5', read_or_not='False', message_name='UKWV', message_date='2019-10-23')
messages (messages_id='6', read_or_not='True', message_name='HXPQ', message_date='2018-02-23')
messages (messages_id='7', read_or_not='True', message_name='ICUR', message_date='2017-05-06')
messages (messages_id='8', read_or_not='True', message_name='GLMM', message_date='2019-03-03')
messages (messages_id='9', read_or_not='False', message_name='AAEE', message_date='2018-07-16')

Process finished with exit code 0

```

Таблиця update_id_log після UPDATE з даними які мають message_date < now() та не були прочитані

Data Output		Explain	Messages	Notifications
	messages_id integer		message_name character varying	
1		3	YYUS	
2		4	YVXH	
3		5	UKWV	
4		9	AAEE	