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”
тема “Засоби оптимізації роботи СУБД PostgreSQL”

Виконав
студент ІІ курсу
групи КП-93
Філенко Богдан Миколайович
(*прізвище, ім’я, по батькові*)
Варіант 25

Перевірив
“____”“_____” 20___ р.
викладач
Петрашенко Андрій Васильович
(*прізвище, ім’я, по батькові*)

Київ 2020

Мета роботи

Здобуття практичних навичок використання засобів оптимізації СУБД PostgreSQL.

Постановка завдання

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.

Завдання 1

Перетворений код:

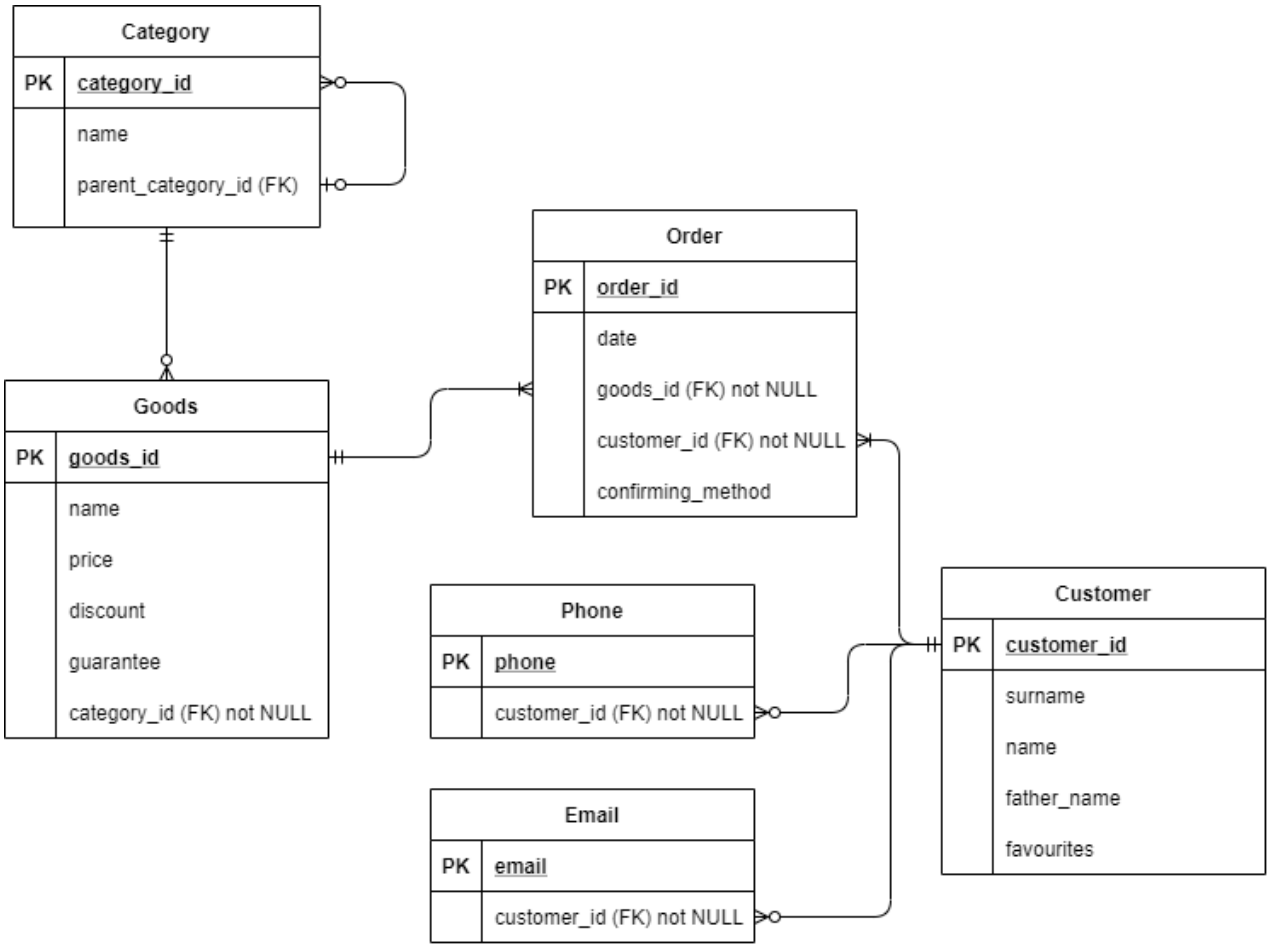
```python
from category import Category
from customer import Customer
from email import Email
from phone import Phone
from goods import Goods
from base import Base
from order import Order
from base import Base, Session, engine

import psycopg2
import query_parser

def iterator(mes):
    for i in range(10):
        mes += "chr(trunc(65+random()*25)::int) || "
    return mes


class Model:
    # ========== ctor ==========
    def __init__(self):
        Base.metadata.create_all(engine)
        self.session = Session()
        self.conn = psycopg2.connect("dbname='shop' user='postgres' host='localhost' password='3497279088'")
        self.curs = self.conn.cursor()


    # ========== Goods table ==========
    def read_goods_by_pk(self, goods_pk):
        return self.session.query(Goods).filter(Goods.goods_id == goods_pk).one()

    def insert_goods(self, goods):
        self.session.add(Goods(goods[0], goods[1], goods[2], goods[3], goods[4], self.session.query(Category)
                               .filter(Category.category_id == goods[4]).one()))
        self.session.commit()

    def update_goods(self, goods):
        self.session.query(Goods).filter(Goods.goods_id == goods[0]) \
            .update({'name': goods[1], 'price': goods[2], 'discount': goods[3], 'guarantee': goods[4],
                     'category_id': goods[5]})
        self.session.commit()

    def delete_goods(self, goods_start_id, goods_end_id):
        self.session.query(Goods).filter(Goods.goods_id >= goods_start_id).filter(Goods.goods_id <= goods_end_id)\
            .delete()
        self.session.commit()

    def generate_goods(self, goods_counter):
        message = "SELECT "
        for i in range(2):
            message = iterator(message)
        message += 'chr(trunc(65+random()*25)::int), trunc(10000+random()*999999)::int, random()*30, 24, ' \
                   '(SELECT category_id FROM "Category" order by random() limit 1) from generate_series(1, {})' \
            .format(goods_counter)
        self.curs.execute('INSERT INTO "Goods" (name, price, discount, guarantee, category_id) {}'.format(message))
        self.conn.commit()
```

```python

    # ========== Customers table ==========
    def read_customer_by_pk(self, customer_pk):
        return self.session.query(Customer).filter(Customer.customer_id == customer_pk).one()

    def insert_customer(self, customer):
        self.session.add(Customer(customer[0], customer[1], customer[2], customer[3]))
        self.session.commit()

    def update_customer(self, customer):
        self.session.query(Customer).filter(Customer.customer_id == customer[0]) \
        .update({'surname': customer[1], 'name': customer[2], 'father_name': customer[3], 'favourites': customer[4]})
        self.session.commit()

    def delete_customer(self, customer_start_id, customer_end_id):
        self.session.query(Customer).filter(Customer.customer_id >= customer_start_id) \
                            .filter(Customer.customer_id <= customer_end_id).delete()
        self.session.commit()

    def generate_customers(self, customers_number):
        message = "SELECT "
        message = iterator(message)
        message += "chr(trunc(65+random()*25)::int) as surname, "
        message = iterator(message)
        message += "chr(trunc(65+random()*25)::int) as name, "
        message = iterator(message)
        message += "chr(trunc(65+random()*25)::int) as father_name "
        self.curs.execute('INSERT INTO "Customer" (surname, name, father_name, favourites) {},'
                          .format(message, customers_number))
        self.conn.commit()

    # ========== Phone table ==========
    def read_phone_by_pk(self, phone_pk):
        return self.session.query(Phone).filter(Phone.phone == phone_pk).one()

    def insert_phone(self, phone):
        self.session.add(Phone(phone[0], phone[1], self.session.query(Customer)
                               .filter(Customer.customer_id == phone[1]).one()))
        self.session.commit()

    def update_phone(self, phone):
        self.session.query(Phone).filter(Phone.phone == phone[0]) \
            .update({'phone': phone[1], 'customer_id': phone[2]})
        self.session.commit()

    def delete_phone(self, phone):
        self.session.query(Phone).filter(Phone.phone.ilike(phone)).delete()
        self.session.commit()

    def generate_phone(self, phone_counter):
        self.curs.execute('INSERT INTO "Phone" SELECT ' + "'+'" + ' || text(trunc(100000000+random()*999999999)::int), '
                                                              '(SELECT customer_id FROM "Customer" order by random() limit 1) FROM generate_series(1, {})'.
                          format(phone_counter))
        self.conn.commit()

    # ========== Email table ==========
    def read_email_by_pk(self, email_pk):
        return self.session.query(Email).filter(Email.email == email_pk).one()

```

```
115
116     def insert_email(self, email):
117         self.session.add(Email(email[0], email[1], self.session.query(Customer)
118                                .filter(Customer.customer_id == email[1]).one()))
119         self.session.commit()
120
121     def update_email(self, email):
122         self.session.query(Email).filter(Email.email == email[0]) \
123             .update({'email': email[1], 'customer_id': email[2]})
124         self.session.commit()
125
126     def delete_email(self, email):
127         self.session.query(Email).filter(Email.email == email).delete()
128         self.session.commit()
129
130     def generate_emails(self, emails_counter):
131         message = "SELECT "
132         for i in range(2):
133             message = iterator(message)
134         message += "'@gmail.com'"
135         self.curs.execute('INSERT INTO "Email" {}, (SELECT customer_id FROM "Customer" '
136                           'order by random() limit 1) FROM generate_series(1, {})'.
137                           format(message, emails_counter))
138         self.conn.commit()
139
140     # ========== Category table ==========
141     def read_category_by_pk(self, category_pk):
142         return self.session.query(Category).filter(Category.category_id == category_pk).one()
143
144     def insert_category(self, category):
145         if category[1] != '':

145         if category[1] != '':
146             self.session.add(Category(category[0], category[1]))
147         else:
148             self.session.add(Category(category[0]))
149         self.session.commit()
150
151     def update_category(self, category):
152         self.session.query(Category).filter(Category.category_id == category[0]) \
153             .update({'name': category[1], 'parent_category_id': category[2]})
154         self.session.commit()
155
156     def delete_category(self, category_start_id, category_end_id):
157         self.session.query(Category).filter(Category.category_id >= category_start_id) \
158                             .filter(Category.category_id <= category_end_id).delete()
159         self.session.commit()
160
161     def generate_categories(self, categories_counter):
162         message = "SELECT "
163         for i in range(2):
164             message = iterator(message)
165         message += "chr(trunc(65+random()*25)::int), null"
166         self.curs.execute('INSERT INTO "Category" (name, parent_category_id) {} FROM generate_series(1, {})'
167                           .format(message, categories_counter))
168         self.conn.commit()
169
170     # ========== Order table ==========
171     def read_order_by_pk(self, order_pk):
172         return self.session.query(Order).filter(Order.order_id == order_pk).one()
173
174     def insert_order(self, order):
175         self.session.add(Order(order[0], order[1], order[2], order[3], self.session.query(Goods)
```

```
        self.session.add(Order(order[0], order[1], order[2], order[3], self.session.query(Goods)
                               .filter(Goods.goods_id == order[1]).one(), self.session.query(Customer)
                               .filter(Customer.customer_id == order[2]).one()))
        self.session.commit()

    def update_order(self, order):
        self.session.query(Order).filter(Order.order_id == order[0]) \
            .update({'date': order[1], 'goods_id': order[2], 'customer_id': order[3], 'confirming_method': order[4]})
        self.session.commit()

    def delete_order(self, order_start_id, order_end_id):
        self.session.query(Order).filter(Order.order_id >= order_start_id).filter(Order.order_id <= order_end_id)\
            .delete()
        self.session.commit()

    def generate_orders(self, orders_number):
        message = "SELECT timestamp '2008-01-10 20:00:00' + " \
                  "random() * (timestamp '2020-12-31 23:00:00' - timestamp '2008-01-10 20:00:00'), " \
                  '(SELECT goods_id FROM "Goods" order by random() limit 1), ' \
                  '(SELECT customer_id FROM "Customer" order by random() limit 1), ' + "'phone'"
        self.curs.execute('INSERT INTO "Order" '
                          '(date, goods_id, customer_id, confirming_method) {} from generate_series(1, {})'
                          .format(message, orders_number))
        self.conn.commit()

    # ========== Find ==========
    def find_entities(self, query):
        pass
        try:
            message = "SELECT * FROM \"{}\" WHERE ".format(query[0])
            message += query_parser.QueryParser.parse_query(query)

            message = message.rstrip("and ")
            self.curs.execute(message)
            return self.curs.fetchall()
        except Exception as ex:
            raise ex
        finally:
            self.conn.rollback()
```

Схема бази даних:



Класи ORM:

Клас Category

```python
from base import Base
from sqlalchemy import Column, Integer, REAL, String, ForeignKey

class Category(Base):
    __tablename__ = 'Category'

    category_id = Column(Integer, primary_key=True)
    name = Column(String)
    parent_category_id = Column(Integer, ForeignKey('Category.category_id'))

    def __init__(self, name, parent_category_id=None):
        self.name = name
        self.parent_category_id = parent_category_id
```

Клас Customer

```python
from base import Base
from sqlalchemy import Column, Integer, String, VARCHAR, ARRAY
class Customer(Base):
    __tablename__ = 'Customer'

    customer_id = Column(Integer, primary_key=True)
    surname = Column(String)
    name = Column(String)
    father_name = Column(String)
    favourites = Column(ARRAY(VARCHAR(100)))

    def __init__(self, surname, name, father_name, favourites):
        self.surname = surname
        self.name = name
        self.father_name = father_name
        self.favourites = favourites
```

Клас Email

```python
from base import Base
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship


class Email(Base):
    __tablename__ = 'Email'

    email = Column(String, primary_key=True)
    customer_id = Column(Integer, ForeignKey('Customer.customer_id'))
    customer = relationship('Customer', backref='Email')

    def __init__(self, email, customer_id, customer):
        self.email = email
        self.customer_id = customer_id
        self.customer = customer
```

Клас Goods

```python
from base import Base
from sqlalchemy import Column, Integer, REAL, String, ForeignKey
from sqlalchemy.orm import relationship


class Goods(Base):
    __tablename__ = 'Goods'

    goods_id = Column(Integer, primary_key=True)
    name = Column(String, unique=True)
    price = Column(REAL)
    discount = Column(REAL)
    guarantee = Column(Integer)
    category_id = Column(Integer, ForeignKey('Category.category_id'))
    category = relationship('Category', backref="Goods")

    def __init__(self, name, price, discount, guarantee, category_id, category):
        self.name = name
        self.price = price
        self.discount = discount
        self.guarantee = guarantee
        self.category_id = category_id
        self.category = category
```

Клас Order

```python
from base import Base
from sqlalchemy import Column, Integer, String, ForeignKey, Date
from sqlalchemy.orm import relationship


class Order(Base):
    __tablename__ = 'Order'

    order_id = Column(Integer, primary_key=True)
    date = Column(Date)
    goods_id = Column(Integer, ForeignKey('Goods.goods_id'))
    goods = relationship('Goods', backref='Order')
    customer_id = Column(Integer, ForeignKey('Customer.customer_id'))
    customer = relationship('Customer', backref='Order')
    confirming_method = Column(String)

    def __init__(self, date, goods_id, customer_id, confirming_method, goods, customer):
        self.date = date
        self.goods_id = goods_id
        self.customer_id = customer_id
        self.confirming_method = confirming_method
        self.goods = goods
        self.customer = customer
```

Клас Phone

```python
from base import Base
from sqlalchemy import Column, Integer, String, ForeignKey
from sqlalchemy.orm import relationship

class Phone(Base):
    __tablename__ = 'Phone'

    phone = Column(String, primary_key=True)
    customer_id = Column(Integer, ForeignKey('Customer.customer_id'))
    customer = relationship('Customer', backref='Phone')

    def __init__(self, phone, customer_id, customer):
        self.phone = phone
        self.customer_id = customer_id
        self.customer = customer
```

Приклади запитів:

Додавання сутності Customer

```
Enter command: post/customers
Enter surname: Bill
Enter name: Gates
Enter father name: Allan
Enter favourites: Samsung,Apple
Done! ['Bill', 'Gates', 'Allan', ['Samsung', 'Apple']] was inserted!
Enter command:
```

Оновлення сутності Customer

```
Enter command: update/customers/1100005
Enter surname (Enter empty row to skip): Gates
Enter name (Enter empty row to skip): Bill
Enter father name (Enter empty row to skip):
Enter favourites (Enter empty row to skip):
Customer with ID 1100005 was successfully updated
[1100005, 'Gates', 'Bill', 'Allan', ['Samsung', 'Apple']]
Enter command:
```

Видалення Customer`a

```
Enter command: delete/customers/1100005
All customers in ID range [1100005, 1100005] was successfully deleted
Enter command:
```

Додавання сутності Goods

```
Enter command: post/goods
Enter name: Aser Nitro
Enter price: 20000
Enter discount: 0
Enter guarantee: 24
Enter category ID: 16
Done! ['Aser Nitro', '20000', '0', '24', '16'] was inserted!
Enter command:
```

Видалення сутності Category

```
Enter command: delete/categories/16
All categories in ID range [16, 16] was successfully deleted
Enter command:
```

Видалення сутності Goods

```
Enter command: delete/goods/100034
All goods in ID range [100034, 100034] was successfully deleted
Enter command:
```

Завдання 2

Приклади запитів до та після індексування:

1)

```
1  explain analyze select * from "Goods" where name = 'Aser' and price < 30000
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Seq Scan on "Goods" (cost=0.00..2435.09 rows=1 width=41) (actual time=0.040..31.887 rows=1 loops=1)
2	Filter: ((price < '30000'::double precision) AND ((name)::text = 'Aser'::text))
3	Rows Removed by Filter: 100005
4	Planning Time: 1.293 ms
5	Execution Time: 31.911 ms

```
1  explain analyze select * from "Goods" where name = 'Aser' and price < 30000
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Index Scan using goods_index on "Goods" (cost=0.42..8.44 rows=1 width=41) (actual time=0.207..0.210 rows=1 loops=1)
2	Index Cond: (((name)::text = 'Aser'::text) AND (price < '30000'::double precision))
3	Planning Time: 2.636 ms
4	Execution Time: 0.246 ms

2)

```
1  explain analyze select * from "Goods" order by name asc
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Sort (cost=13316.92..13566.94 rows=100006 width=41) (actual time=1221.836..1390.448 rows=100006 loops=1)
2	Sort Key: name
3	Sort Method: external merge Disk: 5296kB
4	-> Seq Scan on "Goods" (cost=0.00..1935.06 rows=100006 width=41) (actual time=0.041..20.153 rows=100006 loops=1)
5	Planning Time: 0.175 ms
6	Execution Time: 1405.784 ms

```
1  explain analyze select * from "Goods" order by name asc
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Index Scan using goods_index on "Goods" (cost=0.42..7680.40 rows=100006 width=41) (actual time=0.111..153.831 rows=100006 loops=1)
2	Planning Time: 0.233 ms
3	Execution Time: 160.340 ms

3)

```
1  explain analyze select favourites, customer_id from "Customer" group by favourites, customer_id
2  having favourites @> cast(array ['Samsung', 'Apple'] as varchar(100)[])
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	HashAggregate (cost=2671.17..2707.80 rows=3663 width=45) (actual time=109.541..110.406 rows=1951 loops=1)
2	Group Key: customer_id
3	-> Seq Scan on "Customer" (cost=0.00..2662.01 rows=3663 width=45) (actual time=26.702..106.837 rows=1951 loops=1)
4	Filter: (favourites @> '{Samsung,Apple}'::character varying(100)[])
5	Rows Removed by Filter: 98050
6	Planning Time: 1.552 ms
7	Execution Time: 110.861 ms

```
1  explain analyze select favourites, customer_id from "Customer" group by favourites, customer_id
2  having favourites @> cast(array ['Samsung', 'Apple'] as varchar(100)[])
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	HashAggregate (cost=1519.33..1555.96 rows=3663 width=45) (actual time=9.507..10.283 rows=1951 loops=1)
2	Group Key: customer_id
3	-> Bitmap Heap Scan on "Customer" (cost=52.39..1510.17 rows=3663 width=45) (actual time=4.927..7.608 rows=1951 loops=1)
4	Recheck Cond: (favourites @> '{Samsung,Apple}'::character varying(100)[])
5	Heap Blocks: exact=1074
6	-> Bitmap Index Scan on customers_index (cost=0.00..51.47 rows=3663 width=0) (actual time=4.480..4.481 rows=1951 loops=1)
7	Index Cond: (favourites @> '{Samsung,Apple}'::character varying(100)[])
8	Planning Time: 85.268 ms
9	Execution Time: 10.996 ms

4)

```
1  explain analyze select count(favourites) from "Customer" group by favourites
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	HashAggregate (cost=2912.02..2913.02 rows=100 width=49) (actual time=126.981..126.998 rows=101 loops=1)
2	Group Key: favourites
3	-> Seq Scan on "Customer" (cost=0.00..2412.01 rows=100001 width=41) (actual time=0.035..18.286 rows=100001 loops=1)
4	Planning Time: 0.185 ms
5	Execution Time: 127.067 ms

```
1  explain analyze select count(favourites) from "Customer" group by favourites
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	HashAggregate (cost=2912.02..2913.02 rows=100 width=49) (actual time=114.298..114.317 rows=101 loops=1)
2	Group Key: favourites
3	-> Seq Scan on "Customer" (cost=0.00..2412.01 rows=100001 width=41) (actual time=0.028..16.104 rows=100001 loops=1)
4	Planning Time: 2.403 ms
5	Execution Time: 114.367 ms

Індексування недоречно використовувати у випадках, коли таблиця бази даних містить малу кількість даних (<10000), оскільки на використання індексу також затрачаються ресурси, отже, у даному випадку доцільно використовувати звичайне послідовне сканування. Також, індекси рекомендовано використовувати на полях, які є незмінними протягом часу.

У перших трьох випадках індекси прискорюють виконання запитів, оскільки доступ до полів відбувається не через послідовний пошук, а за допомогою відповідного індекса. У останньому випадку система не використала індекси, оскільки на етапі, описаному в 3-му рядку запит повертає усі рядки із таблиці, отже, він не є ефективним.

Команди створення індексів:

BTree:

```
1  create index goods_index on "Goods" using btree(name, price)
```

GIN:

```
1  create index customers_index on "Customer" using gin(favourites)
```

Завдання 3

Текст тригера:

```
DECLARE
    goods "Goods"%ROWTYPE;
BEGIN
    IF(TG_OP='INSERT') THEN
        IF (SELECT COUNT(*) FROM public."Category" WHERE name = NEW.name) > 1 THEN
            RAISE EXCEPTION 'The category % is already added',NEW.name;
        ELSE
            RETURN NEW;
        END IF;
    ELSE
        FOR goods IN SELECT * FROM public."Goods" WHERE category_id IS NULL LOOP
            IF OLD.parent_category_id IS NULL THEN
                DELETE FROM public."Goods" WHERE goods_id = goods.goods_id;
            ELSE
                UPDATE "Goods" SET category_id = OLD.parent_category_id WHERE goods_id = goods.goods_id;
            END IF;
        END LOOP;
    END IF;
    RETURN OLD;
END
```

Ініціювання тригера:

Insert

```
insert into "Category" (name, parent_category_id) values('Aser', 1)

```

Data Output Explain Messages Notifications

```
ERROR: ОШИБКА:  The category Aser is already added
CONTEXT:  функция PL/pgSQL trigger_function(), строка 7, оператор RAISE


SQL state: P0001
```

Delete

```
delete from "Category" where category_id = 17
```

Data Output Explain Messages Notifications

```
DELETE 1

Query returned successfully in 252 msec.
```

Дані таблиці Goods до видалення:

	goods_id [PK] integer	name character varying (200)	price real	discount real	guarantee integer	category_id integer
1	2	Aser	10000	0	24	17
2	32	Aser Aspire	10000	0	24	17
3	33	Aser Aspire	10000	0	24	17

Дані таблиці Category до видалення:

	category_id [PK] integer	name character varying (100)	parent_category_id integer
1	1	Laptops	[null]
2	2	Phones	[null]
3	17	Aser	1

Дані відповідних таблиць після видалення:

	goods_id [PK] integer	name character varying (200)	price real	discount real	guarantee integer	category_id integer
1	2	Aser	10000	0	24	1
2	32	Aser Aspire	10000	0	24	1
3	33	Aser Aspire	10000	0	24	1

	category_id [PK] integer	name character varying (100)	parent_category_id integer
1	1	Laptops	[null]
2	2	Phones	[null]

Даний тригер відповідає за те, щоб після видалення сутності із таблиці Category, полям category_id присвоїлося значення parent_category_id видаленої категорії. Якщо parent_category_id = null, то усі товари видаляються зі сховища.

Аби даний функціонал було реалізовано, а усіх вимог до тригера дотримано, було додано допоміжний тригер before delete, який прибирав зв’язок по FK:

```
DECLARE
    goods "Goods"%ROWTYPE;
BEGIN
    UPDATE public."Goods" SET category_id = NULL WHERE category_id = OLD.category_id;
    RETURN OLD;
END

```