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 №2

з дисципліни “Бази даних”

тема “**Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL**”

Виконав
студент ІІ курсу
групи КП-93
Варіант 23

Філенко Богдан Миколайович

Перевірив
“--” “вересня” 2020р.
викладач
Петрашенко Андрій Васильович

Київ 2020

Мета роботи

Здобуття вмінь програмування прикладних додатків баз даних

Постановка завдання

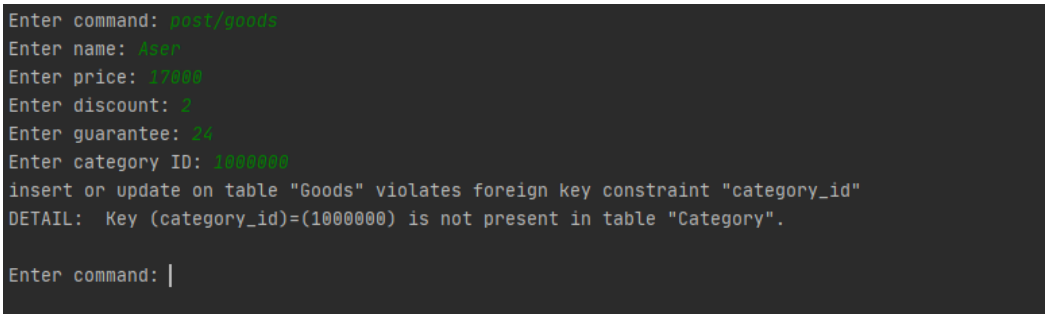
1. Реалізувати функції внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Посилання на Git репозиторій:

https://github.com/AAndromedAA/databases/tree/main/lab2

Завдання 1

Приклади обробки помилок при уведенні даних:



```
Enter command: post/orders
Enter date: 2020-10-10
Enter goods ID: 20000000
Enter customer ID: 2
Enter confirming method: phone
insert or update on table "Order" violates foreign key constraint "goods_id"
DETAIL:  Key (goods_id)=(20000000) is not present in table "Goods".

Enter command:
```

```
Enter command: post/categories
Enter category name: Aser
Enter parent category ID: 10000000
insert or update on table "Category" violates foreign key constraint "parent_category_id"
DETAIL:  Key (parent_category_id)=(10000000) is not present in table "Category".

Enter command:
```

При видаленні даних із таблиць баз даних помилок не було знайдено, так як при спробі видалення сутності за неіснуючим ідентифікатором помилка не виникає. Крім того, програма контролює відповідність ключів батьківських та підлеглих таблиць, тому при видаленні сутності із батьківської таблиці, видаляються записи із підлеглої таблиці із відповідними зовнішніми ключами.

Приклади валідації даних при введенні:

```
Enter command: post/customers
Enter surname: 111
Enter name: Gfgfdg
Enter father name: gfdgfdgfd
Incorrect type of entered values
Enter command:
```

```
Enter command: post/orders
Enter date: 2020-10-10
Enter goods ID: 5
Enter customer ID: 2
Enter confirming method: none
Incorrect type of entered values
Enter command:
```

Завдання 2

Ілюстрації зі згенерованими даними таблиць:

```
SELECT *
    FROM public."Customer";
```

Data Output

	customer_id [PK] integer	surname character varying (50)	name character varying (50)	father_name character varying (50)
26	100049	GTAVIKKCVYC	IBOVOTPHTAY	YQBMRFGDGBV
27	100050	HQIVLHNXXSY	BOWPTMRXBUI	LECHQQFKGCS
28	100051	EIDOYUEDMFW	IYCIQDQIFYQ	JWWLDESAIAT
29	100052	TNQAHSDCHGI	EGOPVTJHIGM	URWEKYRYELC
30	100053	HVKTGUOPKMB	TFDUFWMRCLL	IPLPJSBGWVX
31	100054	XUEQRGHUOWJ	JSVROLXAQLT	NPHAFSRJCUW
32	100055	WCHLFWOAWQV	FUGKCCURKNJ	HEUNDFLQSTG
33	100056	TMLDKSAQPRT	GOUWJPUVYHA	XMEFADTVANF
34	100057	BGNEKCKEHWM	RSQXWTXBCJD	HYTPLLBOEAM
35	100058	MMYTBJQWSGB	RWBUMXTOGUF	FTAKSVXCVDM
36	100059	VKRGYFIMFTW	UINTQOODYDA	QKUIRTGDTFV
37	100060	YUNIAKKQTEQ	BQXSBXNPVAW	FBRCLHULCET
38	100061	PALPQHAYBOQ	NYNPIHDQAWT	PQCHTTSGHXB
39	100062	WRXGBSPRFGT	YCSXNWPBLFJ	OJMIJBJSGTG

Query History | Data Output | Explain | Messages | Notifications

```
SELECT *
    FROM public."Goods";
```

Data Output

	goods_id [PK] integer	name character varying (100)	price integer	discount real	guarantee integer	category_id integer
96	400120	IXYBEFJUOHONQGCDAMWBE	123188	1.92692	24	13
97	400121	HGWVYOSFNYWHEBCDLPIEK	978723	13.1015	24	13
98	400122	OTXWANWOUDARJFQJFRMAU	854809	22.3805	24	13
99	400123	CBNDKRFETJPOPTLQMCFRM	620888	23.0698	24	13
100	400124	TTXYTMDUIFLFNQHAQIUEC	666629	24.8274	24	13
101	400125	JEWNVPWTEGHEDRWUMJK...	225967	4.34347	24	13
102	400126	BNHHGMOHUVNDILFPCNMII	172573	13.339	24	13
103	400127	BGQBXLHRKNWVUYIKAIGEW	11618	8.24791	24	13
104	400128	GTEXDDQATWVAKPWVWDO...	222063	4.23378	24	13
105	400129	QGBTFOUFFTDTCEVTTBEUX	247587	11.8876	24	13
106	400130	FSFPUKMRAPJRVSOJHQUCA	362508	27.4883	24	13
107	400131	LARDCMVFDHJRRUFKBKNEY	199018	27.1405	24	13
108	400132	EIOCXLTAKWXJFPBTQOUDD	36477	4.73928	24	13
109	400133	GYBHVLMAOMRNCGRXYEJDF	225533	6.54888	24	13

Відповідні SQL запити:

```
INSERT INTO public."Customer" (surname, name, father_name)
SELECT chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) ||
       chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) ||
       chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) ||
       chr(trunc(65+random()*25)::int) as surname, chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) ||
       chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) ||
       chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) ||
       chr(trunc(65+random()*25)::int) as name, chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) ||
       chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) ||
       chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) ||
       chr(trunc(65+random()*25)::int) as father_name from generate_series(1, 100000)
```

```
INSERT INTO public."Goods" (name, price, discount, guarantee, category_id)
SELECT chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) ||
       chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) ||
       chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) || chr(trunc(65+random()*25)::int) ||
       chr(trunc(65+random()*25)::int) as name, trunc(10000+random()*999999)::int as price, random()*30 as discount, 24 as quarantee,
       (SELECT category_id FROM "Category" order by random() limit 1) as category_id from generate_series(1, 100000)
```

Завдання 3

Пошук у таблиці Goods за заданими атрибутами:

```
Enter command: find/Goods?price=10000_to_17000&discount=0_to_10&name=GO
(411921, 'WHCUNBJQNIBVRFYGOFOMW', 15818, 4.90358, 24, 13)
(434366, 'KLHNQPNRFPJRGOTEWIXWL', 15782, 9.93106, 24, 13)
(437641, 'VXCWUFGOBANFOSRXPOCCA', 16861, 8.0736, 24, 13)
(459128, 'JWAGGOSLUQECOOJPIULPF', 10189, 8.44092, 24, 13)
(463866, 'GXJBWMLJDQXKFYBXHXXGO', 15350, 0.470911, 24, 13)
(468129, 'QVNGOVEHSSNAFTOAOYPSF', 16694, 6.65761, 24, 13)
(473790, 'PAMPKWSIUIPIJKGGGOGIO', 12881, 3.56254, 24, 13)
(488553, 'VDPVUXPUJGOIWJMQXPVJE', 14871, 6.32803, 24, 13)
(490052, 'GOOBGQPTWPMDPNCGLWDAC', 10567, 9.88253, 24, 13)
Operation time --- 38.03873062133789 ms.
Enter command:
```

```
SELECT * FROM public."Goods" WHERE price BETWEEN 10000 and 17000 and discount BETWEEN 0 and 10 and name LIKE '%GO%'
```

ut

_id teger	name character varying (100)	price integer	discount real	guarantee integer	category_id integer
411921	WHCUNBJQNIBVRFYGOFOMW	15818	4.90358	24	13
434366	KLHNQPNRFPJRGOTEWIXWL	15782	9.93106	24	13
437641	VXCWUFGOBANFOSRXPOCCA	16861	8.0736	24	13
459128	JWAGGOSLUQECOOJPIULPF	10189	8.44092	24	13
463866	GXJBWMLJDQXKFYBXHXXGO	15350	0.470911	24	13
468129	QVNGOVEHSSNAFTOAOYPSF	16694	6.65761	24	13
473790	PAMPKWSIUIPIJKGGGOGIO	12881	3.56254	24	13
488553	VDPVUXPUJGOIWJMQXPVJE	14871	6.32803	24	13
490052	GOOBGQPTWPMDPNCGLWD	10567	9.88253	24	13

Пошук у таблиці Customer за заданими атрибутами:

```
Enter command: find/Customer?surname=NT&name=FM&father_name=MN
Operation time --- 32.97901153564453 ms.
Enter command:
```

```
1  SELECT * FROM public."Customer" WHERE surname LIKE '%NT%' and name LIKE '%FM%' and father_name LIKE '%MN%'
```

Output

customer_id [PK] integer	surname character varying (50)	name character varying (50)	father_name character varying (50)

Пошук у таблиці Order за заданими атрибутами:

```
Enter command: find/Order?date=2002-01-01_to_2020-10-10&customer_id=1_to_3&confirming_method=phone
(2, datetime.date(2020, 1, 5), 2, 1, 'phone')
(4, datetime.date(2018, 2, 21), 1, 3, 'phone')
Operation time --- 5.99360466003418 ms.
Enter command:
```

```
1  SELECT * FROM public."Order" WHERE date BETWEEN timestamp '2002-01-01' and timestamp '2020-10-10' and
2  customer_id BETWEEN 1 and 3 and confirming_method LIKE '%phone%'
```

Data Output

	order_id [PK] integer	date date	goods_id integer	customer_id integer	confirming_method character varying (10)
1	2	2020-01...	2	1	phone
2	4	2018-02...	1	3	phone

Завдання 4

Model:

lab2_model.py

```python
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):
        self.conn = psycopg2.connect("dbname='lab1' user='postgres' host='localhost' password='3497279088'")
        self.curs = self.conn.cursor()

    # ========== Goods table ==========
    def read_goods_by_pk(self, goods_pk):
        self.curs.execute('SELECT * FROM "Goods" WHERE goods_id = {}'.format(goods_pk))
        return self.curs.fetchall()

    def insert_goods(self, goods):
        try:
            self.curs.execute('INSERT INTO "Goods" (name, price, discount, guarantee, category_id) '
                              'VALUES (\'%s\', %d, %f, %d, %d)' %
                              (goods[0], int(goods[1]), float(goods[2]), int(goods[3]), int(goods[4])))
            self.conn.commit()
```

```
        except Exception as ex:
            raise ex
        finally:
            self.conn.rollback()

    def update_goods(self, goods):
        try:
            self.curs.execute('UPDATE "Goods" SET name=\'{}\', price={}, discount={}, guarantee={}, category_id={} '
                              'WHERE goods_id={};'.
                              format(goods[1], int(goods[2]), float(goods[3]), int(goods[4]), int(goods[5]), goods[0]))
            self.conn.commit()
        except Exception as ex:
            raise ex
        finally:
            self.conn.rollback()

    def delete_goods(self, goods_start_id, goods_end_id):
        for table in ["Order", "Goods"]:
            self.curs.execute('DELETE FROM "{}" WHERE goods_id >= {} and goods_id <= {}'
                              .format(table, goods_start_id, goods_end_id))
            self.conn.commit()
        self.conn.rollback()

    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()

    # ========== Customers table ==========
    def read_customer_by_pk(self, customer_pk):
        self.curs.execute('SELECT * FROM "Customer" WHERE customer_id = {}'.format(customer_pk))
        return self.curs.fetchall()

    def insert_customer(self, customer):
        try:
            self.curs.execute('INSERT INTO "Customer" (surname, name, father_name) '
                              'VALUES (\'%s\', \'%s\', \'%s\')' % (customer[0], customer[1], customer[2]))
            self.conn.commit()
        except Exception as ex:
            raise ex
        finally:
            self.conn.rollback()

    def update_customer(self, customer):
        try:
            self.curs.execute('UPDATE "Customer" SET surname=\'{}\', name=\'{}\', father_name=\'{}\' '
                              'WHERE customer_id={};'.
                              format(customer[1], customer[2], customer[3], customer[0]))
            self.conn.commit()
        except Exception as ex:
            raise ex
        finally:
            self.conn.rollback()

    def delete_customer(self, customer_start_id, customer_end_id):
```

```
        for table in ["Order", "Phone", "Email", "Customer"]:
            self.curs.execute('DELETE FROM "{}" WHERE customer_id >= {} and customer_id <= {}'
                              .format(table, customer_start_id, customer_end_id))
            self.conn.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) {} from generate_series(1, {})'
                          .format(message, customers_number))
        self.conn.commit()

    # ========== Phone table ==========
    def read_phone_by_pk(self, phone_pk):
        self.curs.execute('SELECT * FROM "Phone" WHERE phone = \'{}\''.format(phone_pk))
        return self.curs.fetchall()

    def insert_phone(self, phone):
        try:
            self.curs.execute('INSERT INTO "Phone" (phone, customer_id) '
                              'VALUES (\'%s\', %d)' % (phone[0], int(phone[1])))
            self.conn.commit()
        except Exception as ex:
            raise ex
        finally:

            self.conn.rollback()

    def update_phone(self, phone):
        try:
            self.curs.execute('UPDATE "Phone" SET phone=\'{}\', customer_id={} '
                              'WHERE phone=\'{}\';'.
                              format(phone[1], int(phone[2]), phone[0]))
            self.conn.commit()
        except Exception as ex:
            raise ex
        finally:
            self.conn.rollback()

    def delete_phone(self, phone):
        self.curs.execute('DELETE FROM "Phone" WHERE phone=\'{}\''.format(phone))
        self.conn.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):
        self.curs.execute('SELECT * FROM "Email" WHERE email = \'{}\''.format(email_pk))
        return self.curs.fetchall()

    def insert_email(self, email):
        try:
```

```
            self.curs.execute('INSERT INTO "Email" (email, customer_id) '
                              'VALUES (\'%s\', %d);' % (email[0], int(email[1])))
            self.conn.commit()
        except Exception as ex:
            raise ex
        finally:
            self.conn.rollback()

    def update_email(self, email):
        try:
            self.curs.execute('UPDATE "Email" SET email=\'{}\', customer_id={} '
                              'WHERE email=\'{}\';'.
                              format(email[1], int(email[2]), email[0]))
            self.conn.commit()
        except Exception as ex:
            raise ex
        finally:
            self.conn.rollback()

    def delete_email(self, email):
        self.curs.execute('DELETE FROM "Email" WHERE email=\'{}\''.format(email))
        self.conn.commit()

    def generate_emails(self, emails_counter):
        message = "SELECT "
        for i in range(2):
            message = iterator(message)
        message += "'@gmail.com'"
        self.curs.execute('INSERT INTO "Email" {}, (SELECT customer_id FROM "Customer" '
                          'order by random() limit 1) FROM generate_series(1, {})'.
                          format(message, emails_counter))
        self.conn.commit()

    # ========== Category table ==========
    def read_category_by_pk(self, category_pk):
        self.curs.execute('SELECT * FROM "Category" WHERE category_id = {}'.format(category_pk))
        return self.curs.fetchall()

    def insert_category(self, category):
        try:
            self.curs.execute('INSERT INTO "Category" (name, parent_category_id) '
                              'VALUES (\'%s\', %d);' % (category[0], int(category[1])))
            self.conn.commit()
        except Exception as ex:
            raise ex
        finally:
            self.conn.rollback()

    def update_category(self, category):
        try:
            self.curs.execute('UPDATE "Category" SET name=\'{}\', parent_category_id={} '
                              'WHERE category_id={};'.
                              format(category[1], int(category[2]), int(category[0])))
            self.conn.commit()
        except Exception as ex:
            raise ex
        finally:
            self.conn.rollback()

    def delete_category(self, category_start_id, category_end_id):
```

```
        for table in ["Goods", "Category"]:
            self.curs.execute('DELETE FROM "{}" WHERE category_id >= {} and category_id <= {}'
                              .format(table, category_start_id, category_end_id))
            self.conn.commit()

    def generate_categories(self, categories_counter):
        message = "SELECT "
        for i in range(2):
            message = iterator(message)
        message += "chr(trunc(65+random()*25)::int), null"
        self.curs.execute('INSERT INTO "Category" (name, parent_category_id) {} FROM generate_series(1, {})'
                          .format(message, categories_counter))
        self.conn.commit()

    # ========== Order table ==========
    def read_order_by_pk(self, order_pk):
        self.curs.execute('SELECT * FROM "Order" WHERE order_id = {}'.format(order_pk))
        return self.curs.fetchall()

    def insert_order(self, order):
        try:
            self.curs.execute('INSERT INTO "Order" (date, goods_id, customer_id, confirming_method) '
                              'VALUES (\'%s\', %d, %d, \'%s\')'
                              % (order[0], int(order[1]), int(order[2]), order[3]))
            self.conn.commit()
        except Exception as ex:
            raise ex
        finally:
            self.conn.rollback()


    def update_order(self, order):
        try:
            self.curs.execute('UPDATE "Order" SET date=\'{}\', goods_id={}, customer_id={}, confirming_method=\'{}\' '
                              'WHERE order_id={};'.
                              format(order[1], int(order[2]), int(order[3]), order[4], int(order[0])))
            self.conn.commit()
        except Exception as ex:
            raise ex
        finally:
            self.conn.rollback()

    def delete_order(self, order_start_id, order_end_id):
        self.curs.execute('DELETE FROM "Order" WHERE order_id >= {} and order_id <= {}'
                          .format(order_start_id, order_end_id))
        self.conn.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):
        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()
```

View:

lab2_view.py

```python
from model import Model
import time


class View:
    def __init__(self):
        self.model = Model()

    def insert_goods(self, item):
        self.model.insert_goods(item)
        print("Done! {} was inserted!".format(item))

    def insert_customer(self, item):
        self.model.insert_customer(item)
        print("Done! {} was inserted!".format(item))

    def insert_phone(self, item):
        self.model.insert_phone(item)
        print("Done! {} was inserted!".format(item))

    def insert_email(self, item):
        self.model.insert_email(item)
        print("Done! {} was inserted!".format(item))

    def insert_order(self, item):
        self.model.insert_order(item)
        print("Done! {} was inserted!".format(item))

    def insert_category(self, item):
        self.model.insert_category(item)

        print("Done! {} was inserted!".format(item))

    def update_goods(self, item):
        self.model.update_goods(item)
        print("Goods with ID {} was successfully updated\n{}".format(item[0], item))

    def update_category(self, item):
        self.model.update_category(item)
        print("Goods with ID {} was successfully updated\n{}".format(item[0], item))

    def update_customer(self, item):
        self.model.update_customer(item)
        print("Customer with ID {} was successfully updated\n{}".format(item[0], item))

    def update_order(self, item):
        self.model.update_order(item)
        print("Order with ID {} was successfully updated\n{}".format(item[0], item))

    def update_phone(self, item):
        self.model.update_phone(item)
        print("Phone {} was successfully updated\n{}".format(item[0], item))

    def update_email(self, item):
        self.model.update_email(item)
        print("Email {} was successfully updated\n{}".format(item[0], item))

    def delete_phone(self, item_pk):
        self.model.delete_phone(item_pk)
        print("Phone {} was successfully deleted".format(item_pk))

```

```
    def delete_email(self, item_pk):
        self.model.delete_email(item_pk)
        print("Email {} was successfully deleted".format(item_pk))

    def delete_customers(self, item_start_pk, item_end_pk):
        self.model.delete_customer(item_start_pk, item_end_pk)
        print("All customers in ID range [{}, {}] was successfully deleted".format(item_start_pk, item_end_pk))

    def delete_goods(self, item_start_pk, item_end_pk):
        self.model.delete_goods(item_start_pk, item_end_pk)
        print("All goods in ID range [{}, {}] was successfully deleted".format(item_start_pk, item_end_pk))

    def delete_orders(self, item_start_pk, item_end_pk):
        self.model.delete_order(item_start_pk, item_end_pk)
        print("All orders in ID range [{}, {}] was successfully deleted".format(item_start_pk, item_end_pk))

    def delete_categories(self, item_start_pk, item_end_pk):
        self.model.delete_category(item_start_pk, item_end_pk)
        print("All categories in ID range [{}, {}] was successfully deleted".format(item_start_pk, item_end_pk))

    def generate_goods(self, items_counter):
        self.model.generate_goods(items_counter)
        print("{} random goods was successfully generated".format(items_counter))

    def generate_customers(self, items_counter):
        self.model.generate_customers(items_counter)
        print("{} random customers was successfully generated".format(items_counter))

    def generate_categories(self, items_counter):
        self.model.generate_categories(items_counter)

       print("{} random categories was successfully generated".format(items_counter))

    def generate_orders(self, items_counter):
        self.model.generate_orders(items_counter)
        print("{} random orders was successfully generated".format(items_counter))

    def generate_phones(self, items_counter):
        self.model.generate_phone(items_counter)
        print("{} random phones was successfully generated".format(items_counter))

    def generate_emails(self, items_counter):
        self.model.generate_emails(items_counter)
        print("{} random phones was successfully generated".format(items_counter))

    def find_items(self, tables):
        time_before = time.time()
        items = self.model.find_entities(tables)
        time_after = time.time()
        for item in items:
            print(item)
        print("Operation time --- {} ms.".format((time_after-time_before)*1000))
```

Controller:

```
from view import View
import model

inp_requests = dict({"goods": ["Enter name", "Enter price", "Enter discount", "Enter guarantee",
                               "Enter category ID"],
                     "customer": ["Enter surname", "Enter name", "Enter father name"],
                     "phone": ["Enter phone number", "Enter customer ID"],
                     "email": ["Enter email", "Enter customer ID"],
                     "order": ["Enter date", "Enter goods ID", "Enter customer ID", "Enter confirming method"],
                     "category": ["Enter category name", "Enter parent category ID"]})


def validate(option, item):
    if option == "goods":
        return True if (item[0].isalnum() and item[1].isdigit() and item[2].isdigit() and item[3].isdigit() and
                        item[4].isdigit()) else False
    if option == "customer":
        return True if (item[0].isalpha() and item[1].isalpha() and item[2].isalpha()) else False
    if option == "phone":
        return True if item[1].isdigit() else False
    if option == "email":
        return True if item[1].isdigit() else False
    if option == "order":
        return True if (item[1].isdigit() and item[2].isdigit() and (item[3] == 'phone' or item[3] == 'email'))\
            else False
    if option == "category":
        return True if item[1].isdigit() else False


class Controller:

    def __init__(self):
        self.view = View()
        self.mod = model.Model()

    def insert_item(self, option):
        global inp_requests
        item = list()
        for request in inp_requests[option]:
            item.append(input(request+": "))
        if option == "goods":
            if validate(option, item):
                self.view.insert_goods(item)
            else:
                raise Exception("Incorrect type of entered values")
        if option == "customer":
            if validate(option, item):
                self.view.insert_customer(item)
            else:
                raise Exception("Incorrect type of entered values")
        if option == "phone":
            if validate(option, item):
                self.view.insert_phone(item)
            else:
                raise Exception("Incorrect type of entered values")
        if option == "email":
            if validate(option, item):
                self.view.insert_email(item)
            else:
                raise Exception("Incorrect type of entered values")
        if option == "order":
```

```
            if validate(option, item):
                self.view.insert_order(item)
            else:
                raise Exception("Incorrect type of entered values")
        if option == "category":
            if validate(option, item):
                self.view.insert_category(item)
            else:
                raise Exception("Incorrect type of entered values")

    def update_item(self, option, item_pk):
        if not item_pk.isdecimal():
            raise Exception("\'{}\' is not a decimal id".format(item_pk))
        global inp_requests
        item = list()
        for request in inp_requests[option]:
            item.append(input(request+" (Enter empty row to skip): "))
        new_item = list()
        if option == "goods":
            curr_item = self.mod.read_goods_by_pk(int(item_pk))
            new_item.append(curr_item[0][0])
            for i in range(1, 6):
                if item[i-1] != "":
                    new_item.append(item[i-1])
                else:
                    new_item.append(curr_item[0][i])
            if validate(option, new_item):
                self.view.update_goods(new_item)
            else:
                raise Exception("Incorrect type of entered values")

        if option == "customer":
            curr_item = self.mod.read_customer_by_pk(int(item_pk))
            new_item.append(curr_item[0][0])
            for i in range(1, 4):
                if item[i-1] != "":
                    new_item.append(item[i-1])
                else:
                    new_item.append(curr_item[0][i])
            if validate(option, new_item):
                self.view.update_customer(new_item)
            else:
                raise Exception("Incorrect type of entered values")
        if option == "phone":
            curr_item = self.mod.read_phone_by_pk(item_pk)
            new_item.append(curr_item[0][0])
            for i in range(1, 3):
                if item[i-1] != "":
                    new_item.append(item[i-1])
                else:
                    new_item.append(curr_item[0][i-1])
            if validate(option, new_item):
                self.view.update_phone(new_item)
            else:
                raise Exception("Incorrect type of entered values")
        if option == "email":
            curr_item = self.mod.read_email_by_pk(item_pk)
            new_item.append(curr_item[0][0])
            for i in range(1, 3):
                if item[i-1] != "":
                    new_item.append(item[i-1])
```

```
                else:
                    new_item.append(curr_item[0][i-1])
            if validate(option, new_item):
                self.view.update_email(new_item)
            else:
                raise Exception("Incorrect type of entered values")
        if option == "order":
            curr_item = self.mod.read_order_by_pk(int(item_pk))
            new_item.append(curr_item[0][0])
            for i in range(1, 5):
                if item[i-1] != "":
                    new_item.append(item[i-1])
                else:
                    new_item.append(curr_item[0][i])
            if validate(option, new_item):
                self.view.update_order(new_item)
            else:
                raise Exception("Incorrect type of entered values")
        if option == "category":
            curr_item = self.mod.read_category_by_pk(int(item_pk))
            new_item.append(curr_item[0][0])
            for i in range(1, 3):
                if item[i-1] != "":
                    new_item.append(item[i-1])
                else:
                    new_item.append(curr_item[0][i])
            if validate(option, new_item):
                self.view.update_category(new_item)
            else:
                raise Exception("Incorrect type of entered values")


    def delete_items(self, option, item_start_pk, item_end_pk=0):
        if not item_start_pk.isdecimal() or not str(item_end_pk).isdecimal():
            raise Exception("\'{}\' or \'{}\' is not a decimal id".format(item_start_pk, item_end_pk))
        if item_end_pk == 0:
            item_end_pk = item_start_pk
        if option == "goods":
            self.view.delete_goods(item_start_pk, item_end_pk)
        if option == "customer":
            self.view.delete_customers(item_start_pk, item_end_pk)
        if option == "category":
            self.view.delete_categories(item_start_pk, item_end_pk)
        if option == "order":
            self.view.delete_orders(item_start_pk, item_end_pk)
        if option == "phone":
            self.view.delete_phone(item_start_pk)
        if option == "email":
            self.view.delete_email(item_start_pk)

    def generate_items(self, option, items_number):
        if not items_number.isdecimal():
            raise Exception("\'{}\' is not a decimal".format(items_number))
        if option == "goods":
            self.view.generate_goods(items_number)
        if option == "customer":
            self.view.generate_customers(items_number)
        if option == "category":
            self.view.generate_categories(items_number)
        if option == "order":
            self.view.generate_orders(items_number)

         if option == "phone":
             self.view.generate_phones(items_number)
         if option == "email":
             self.view.generate_emails(items_number)

     def find_items(self, subcommand):
         query = subcommand.split('?')
         self.view.find_items(query)
```