



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 II курсу
групи КП-93
Варіант 23

Філенко Богдан Миколайович

Перевірів
“--” “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 видалення 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 введенні даних:

```
Enter command: post/goods
Enter name: Acer
Enter price: 17000
Enter discount: 2
Enter guarantee: 24
Enter category ID: 1000000
insert or update on table "Goods" violates foreign key constraint "category_id"
DETAIL: Key (category_id)=(1000000) is not present in table "Category".
Enter command: |
```

```
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: Asen
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: 67gfdg
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:

1	SELECT *
2	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	IBOVOTPHAY	YQBMRFGDGBV
27	100050	HQIVLHNXXSY	BOWPTMRXBUI	LECHQQFKGCS
28	100051	EIDOUYEDMFV	IYCIQDQIFYQ	JWWLDESAIAT
29	100052	TNQAHSDCCHI	EGOPVTJHIGM	URWEKYRYELC
30	100053	HVKTGUOPKMB	TFDUFWMRCLL	IPLPJSBGWVX
31	100054	XUEQRGHUOWJ	JSVROLXAQLT	NPHAFSRJCUW
32	100055	WCHLFWOAWQV	FUGKCCURKNJ	HEUNDFLQSTG
33	100056	TMLDKSAQPR	GOUWJPUVYHA	XMEFADTVANF
34	100057	BGNEKCKEHWM	RSQXWTXBCJD	HYTPLLBOEAM
35	100058	MMYTBQWSGB	RWBUMXTOGUF	FTAKSVXCVDIM
36	100059	VKRGYFIMFTW	UINTQOODYDA	QKUIRTGDTFV
37	100060	YUNIAKKQTEQ	BQXSBXNPAW	FBRLHLULCET
38	100061	PALPQHAYBOQ	NYNPIHDQAWT	PQCHTTSGHXB
39	100062	WRXGBSPRFGT	YCSXNWPBLFJ	OJMIJBSJGTG
40	100063	MMYTBQWSGB	RWBUMXTOGUF	FTAKSVXCVDIM

Query History Data Output Explain Messages Notifications

1	SELECT *
2	FROM public."Goods";

Data Output						
	goods_id [PK] integer	name character varying (100)	price integer	discount real	guarantee integer	category_id integer
96	400120	IXYBEFUOHONQGCDAWBE	123188	1.92692	24	13
97	400121	HGWVYOSFNWYHEBCDLPIEK	978723	13.1015	24	13
98	400122	OTXWANWOUJARJFQJFRMAU	854809	22.3805	24	13
99	400123	CBNDKRFTJPOPTLQMCFRM	620888	23.0698	24	13
100	400124	TTXYTMDUIFLFNQHAQUIEC	666629	24.8274	24	13
101	400125	JEWNVPTWEGHEDRWUMJK...	225967	4.34347	24	13
102	400126	BNHHGMOHUVNDILFPCNMII	172573	13.339	24	13
103	400127	BQBXLHRKNWVUYIKAIGEW	11618	8.24791	24	13
104	400128	GTEXDDQATWVAKPWVWDO...	222063	4.23378	24	13
105	400129	QGBTFOUFFDTCEVTTBEUX	247587	11.8876	24	13
106	400130	FSFPUKMRAPJRVSOJHQUCA	362508	27.4883	24	13
107	400131	LARDCMVFDHJRRUFKBKNEY	199018	27.1405	24	13
108	400132	EIOCKLTAKWJFFBTQOUD	36477	4.73928	24	13
109	400133	GVBHVLMAOMRNCGRXYEJDF	225533	6.54888	24	13
110	400134	YFBNBUDJHJLJANSEFQOXT	653407	8.73143	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) 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) 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 guarantee,
(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=60
(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, 'QVNGOVEHSSNAFTOA0YPSF', 16694, 6.65761, 24, 13)
(473790, 'PAMPKWSIUIPIJKGGGOGIO', 12881, 3.56254, 24, 13)
(488553, 'VDPVUXPUJG0IWJMQXPVJE', 14871, 6.32803, 24, 13)
(490052, 'GO0BGQPTWMPDPNCGLWDAC', 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 '%60%'
```

Output

Jd teger	name character varying (100)	price integer	discount real	guarantee integer	category_id integer
411921	WHCUNBJQNIBVRFYGOFOMW	15818	4.90358		24
434366	KLHNQPNRFPJRGOTEWIXWL	15782	9.93106		24
437641	VXCWUFGOBANFOSRXPOCCA	16861	8.0736		24
459128	JWAGGOSLUQECOOJPIULPF	10189	8.44092		24
463866	GXJBWMLJDQXKFYBXHXXGO	15350	0.470911		24
468129	QVNGOVEHSSNAFTOA0YPSF	16694	6.65761		24
473790	PAMPKWSIUIPIJKGGGOGIO	12881	3.56254		24
488553	VDPVUXPUJG0IWJMQXPVJE	14871	6.32803		24
490052	GO0BGQPTWMPDPNCGLWD	10567	9.88253		24

Пошук у таблиці *Customer* за заданими атрибутами:

```
Enter command: Find/Customer?surname=N&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

```
1 import psycopg2
2 import query_parser
3
4
5 def iterator(mes):
6     for i in range(10):
7         mes += "chr(trunc(65+random()*25)::int) || "
8     return mes
9
10
11 class Model:
12     # ===== ctor =====
13     def __init__(self):
14         self.conn = psycopg2.connect("dbname='lab1' user='postgres' host='localhost' password='3497279088'")
15         self.curs = self.conn.cursor()
16
17     # ===== Goods table =====
18     def read_goods_by_pk(self, goods_pk):
19         self.curs.execute('SELECT * FROM "Goods" WHERE goods_id = {}'.format(goods_pk))
20         return self.curs.fetchall()
21
22     def insert_goods(self, goods):
23         try:
24             self.curs.execute('INSERT INTO "Goods" (name, price, discount, guarantee, category_id) '
25                               'VALUES (\'%s\', %d, %f, %d, %d)' %
26                               (goods[0], int(goods[1]), float(goods[2]), int(goods[3]), int(goods[4])))
27             self.conn.commit()
```

```

28         except Exception as ex:
29             raise ex
30         finally:
31             self.conn.rollback()
32
33     def update_goods(self, goods):
34         try:
35             self.curs.execute('UPDATE "Goods" SET name=\'{}\'\', price={}, discount={}, guarantee={}, category_id={} '
36                               'WHERE goods_id={};'.
37                               format(goods[1], int(goods[2]), float(goods[3]), int(goods[4]), int(goods[5]), goods[0]))
38             self.conn.commit()
39         except Exception as ex:
40             raise ex
41         finally:
42             self.conn.rollback()
43
44     def delete_goods(self, goods_start_id, goods_end_id):
45         for table in ["Order", "Goods"]:
46             self.curs.execute('DELETE FROM "{}" WHERE goods_id >= {} and goods_id <= {}'.
47                               .format(table, goods_start_id, goods_end_id))
48             self.conn.commit()
49         self.conn.rollback()
50
51     def generate_goods(self, goods_counter):
52         message = "SELECT "
53         for i in range(2):
54             message = iterator(message)
55             message += 'chr(trunc(65+random()*25)::int), trunc(10000+random()*999999)::int, random()*30, 24, ' \
56                       '(SELECT category_id FROM "Category" order by random() limit 1) from generate_series(1, {})'\
57                       .format(goods_counter)
58
59         self.curs.execute('INSERT INTO "Goods" (name, price, discount, guarantee, category_id) {}'.format(message))
60         self.conn.commit()
61
62     # ===== Customers table =====
63     def read_customer_by_pk(self, customer_pk):
64         self.curs.execute('SELECT * FROM "Customer" WHERE customer_id = {}'.format(customer_pk))
65         return self.curs.fetchall()
66
67     def insert_customer(self, customer):
68         try:
69             self.curs.execute('INSERT INTO "Customer" (surname, name, father_name) '
70                               'VALUES (\'{}s\', \'{}s\', \'{}s\')' % (customer[0], customer[1], customer[2]))
71             self.conn.commit()
72         except Exception as ex:
73             raise ex
74         finally:
75             self.conn.rollback()
76
77     def update_customer(self, customer):
78         try:
79             self.curs.execute('UPDATE "Customer" SET surname=\'{}\'\', name=\'{}\'\', father_name=\'{}\'\ '
80                               'WHERE customer_id={};'.
81                               format(customer[1], customer[2], customer[3], customer[0]))
82             self.conn.commit()
83         except Exception as ex:
84             raise ex
85         finally:
86             self.conn.rollback()
87
88     def delete_customer(self, customer_start_id, customer_end_id):

```

```

88     for table in ["Order", "Phone", "Email", "Customer"]:
89         self.curs.execute('DELETE FROM "{}" WHERE customer_id >= {} and customer_id <= {}'.format(table, customer_start_id, customer_end_id))
90         self.conn.commit()
91
92
93     def generate_customers(self, customers_number):
94         message = "SELECT "
95         message = iterator(message)
96         message += "chr(trunc(65+random()*25)::int) as surname, "
97         message = iterator(message)
98         message += "chr(trunc(65+random()*25)::int) as name, "
99         message = iterator(message)
100        message += "chr(trunc(65+random()*25)::int) as father_name "
101        self.curs.execute('INSERT INTO "Customer" (surname, name, father_name) {} from generate_series(1, {})'.format(message, customers_number))
102        self.conn.commit()
103
104
105    # ===== Phone table =====
106    def read_phone_by_pk(self, phone_pk):
107        self.curs.execute('SELECT * FROM "Phone" WHERE phone = \'{}\'''.format(phone_pk))
108        return self.curs.fetchall()
109
110    def insert_phone(self, phone):
111        try:
112            self.curs.execute('INSERT INTO "Phone" (phone, customer_id) '
113                              'VALUES (\'%s\', %d)' % (phone[0], int(phone[1])))
114            self.conn.commit()
115        except Exception as ex:
116            raise ex
117        finally:
118            self.conn.rollback()
119
120    def update_phone(self, phone):
121        try:
122            self.curs.execute('UPDATE "Phone" SET phone=\'{}\'''.format(phone[0]),
123                              'WHERE phone=\'{}\'''.format(phone[1]),
124                              'format(phone[1], int(phone[2]), phone[0]))')
125            self.conn.commit()
126        except Exception as ex:
127            raise ex
128        finally:
129            self.conn.rollback()
130
131    def delete_phone(self, phone):
132        self.curs.execute('DELETE FROM "Phone" WHERE phone=\'{}\'''.format(phone))
133        self.conn.commit()
134
135    def generate_phone(self, phone_counter):
136        self.curs.execute('INSERT INTO "Phone" SELECT ' + "'+' + ' || text(trunc(100000000+random()*999999999)::int), '
137                          '(SELECT customer_id FROM "Customer" order by random() limit 1) FROM generate_series(1, {})'.format(phone_counter))
138        self.conn.commit()
139
140
141    # ===== Email table =====
142    def read_email_by_pk(self, email_pk):
143        self.curs.execute('SELECT * FROM "Email" WHERE email = \'{}\'''.format(email_pk))
144        return self.curs.fetchall()
145
146    def insert_email(self, email):
147        try:

```



```

148         self.curs.execute('INSERT INTO "Email" (email, customer_id) '
149                             'VALUES (\'%s\', %d);' % (email[0], int(email[1])))
150         self.conn.commit()
151     except Exception as ex:
152         raise ex
153     finally:
154         self.conn.rollback()
155
156     def update_email(self, email):
157         try:
158             self.curs.execute('UPDATE "Email" SET email=\'{}\',' , customer_id={} '
159                                 'WHERE email=\'{}\';'.
160                                 format(email[1], int(email[2]), email[0]))
161             self.conn.commit()
162         except Exception as ex:
163             raise ex
164         finally:
165             self.conn.rollback()
166
167     def delete_email(self, email):
168         self.curs.execute('DELETE FROM "Email" WHERE email=\'{}\';'.format(email))
169         self.conn.commit()
170
171     def generate_emails(self, emails_counter):
172         message = "SELECT "
173         for i in range(2):
174             message = iterator(message)
175             message += "@gmail.com"
176         self.curs.execute('INSERT INTO "Email" {}, (SELECT customer_id FROM "Customer" '
177                             'order by random() limit 1) FROM generate_series(1, {})'.
178                             format(message, emails_counter))
179         self.conn.commit()
180
181     # ===== Category table =====
182     def read_category_by_pk(self, category_pk):
183         self.curs.execute('SELECT * FROM "Category" WHERE category_id = {}'.format(category_pk))
184         return self.curs.fetchall()
185
186     def insert_category(self, category):
187         try:
188             self.curs.execute('INSERT INTO "Category" (name, parent_category_id) '
189                                 'VALUES (\'%s\', %d);' % (category[0], int(category[1])))
190             self.conn.commit()
191         except Exception as ex:
192             raise ex
193         finally:
194             self.conn.rollback()
195
196     def update_category(self, category):
197         try:
198             self.curs.execute('UPDATE "Category" SET name=\'{}\',' , parent_category_id={} '
199                                 'WHERE category_id={};'.
200                                 format(category[1], int(category[2]), int(category[0])))
201             self.conn.commit()
202         except Exception as ex:
203             raise ex
204         finally:
205             self.conn.rollback()
206
207     def delete_category(self, category_start_id, category_end_id):

```

```

208     for table in ["Goods", "Category"]:
209         self.curs.execute('DELETE FROM "{}" WHERE category_id >= {} and category_id <= {}'.format(table, category_start_id, category_end_id))
210         self.conn.commit()
211
212
213     def generate_categories(self, categories_counter):
214         message = "SELECT "
215         for i in range(2):
216             message = iterator(message)
217             message += "chr(trunc(65+random()*25)::int), null"
218             self.curs.execute('INSERT INTO "Category" (name, parent_category_id) {} FROM generate_series(1, {})'.format(message, categories_counter))
219             self.conn.commit()
220
221
222     # ===== Order table =====
223     def read_order_by_pk(self, order_pk):
224         self.curs.execute('SELECT * FROM "Order" WHERE order_id = {}'.format(order_pk))
225         return self.curs.fetchall()
226
227     def insert_order(self, order):
228         try:
229             self.curs.execute('INSERT INTO "Order" (date, goods_id, customer_id, confirming_method) '
230                               'VALUES (\'%s\', %d, %d, \'%s\')'
231                               % (order[0], int(order[1]), int(order[2]), order[3]))
232             self.conn.commit()
233         except Exception as ex:
234             raise ex
235         finally:
236             self.conn.rollback()
237
238
239     def update_order(self, order):
240         try:
241             self.curs.execute('UPDATE "Order" SET date=\'{}\'.format(query[0])
242                               'WHERE order_id={}'.format(order[1], int(order[2]), int(order[3]), order[4], int(order[0]))
243             self.conn.commit()
244         except Exception as ex:
245             raise ex
246         finally:
247             self.conn.rollback()
248
249     def delete_order(self, order_start_id, order_end_id):
250         self.curs.execute('DELETE FROM "Order" WHERE order_id >= {} and order_id <= {}'.format(order_start_id, order_end_id))
251         self.conn.commit()
252
253
254     def generate_orders(self, orders_number):
255         message = "SELECT timestamp '2008-01-10 20:00:00' + " \
256                 "random() * (timestamp '2020-12-31 23:00:00' - timestamp '2008-01-10 20:00:00'), " \
257                 '(SELECT goods_id FROM "Goods" order by random() limit 1), ' \
258                 '(SELECT customer_id FROM "Customer" order by random() limit 1), ' + "'phone'"
259         self.curs.execute('INSERT INTO "Order" '
260                           '(date, goods_id, customer_id, confirming_method) {} from generate_series(1, {})'.format(message, orders_number))
261         self.conn.commit()
262
263
264     # ===== Find =====
265     def find_entities(self, query):
266         try:
267             message = "SELECT * FROM \"{}\" WHERE ".format(query[0])
268             message += query_parser.QueryParser.parse_query(query)
269             message = message.rstrip("and ")
270             self.curs.execute(message)
271             return self.curs.fetchall()
272         except Exception as ex:
273             raise ex
274         finally:
275             self.conn.rollback()

```

View:

```
1  from model import Model
2  import time
3
4
5  class View:
6      def __init__(self):
7          self.model = Model()
8
9      def insert_goods(self, item):
10         self.model.insert_goods(item)
11         print("Done! {} was inserted!".format(item))
12
13     def insert_customer(self, item):
14         self.model.insert_customer(item)
15         print("Done! {} was inserted!".format(item))
16
17     def insert_phone(self, item):
18         self.model.insert_phone(item)
19         print("Done! {} was inserted!".format(item))
20
21     def insert_email(self, item):
22         self.model.insert_email(item)
23         print("Done! {} was inserted!".format(item))
24
25     def insert_order(self, item):
26         self.model.insert_order(item)
27         print("Done! {} was inserted!".format(item))
28
29     def insert_category(self, item):
30         self.model.insert_category(item)
31
32         print("Done! {} was inserted!".format(item))
33
34     def update_goods(self, item):
35         self.model.update_goods(item)
36         print("Goods with ID {} was successfully updated\n{}".format(item[0], item))
37
38     def update_category(self, item):
39         self.model.update_category(item)
40         print("Goods with ID {} was successfully updated\n{}".format(item[0], item))
41
42     def update_customer(self, item):
43         self.model.update_customer(item)
44         print("Customer with ID {} was successfully updated\n{}".format(item[0], item))
45
46     def update_order(self, item):
47         self.model.update_order(item)
48         print("Order with ID {} was successfully updated\n{}".format(item[0], item))
49
50     def update_phone(self, item):
51         self.model.update_phone(item)
52         print("Phone {} was successfully updated\n{}".format(item[0], item))
53
54     def update_email(self, item):
55         self.model.update_email(item)
56         print("Email {} was successfully updated\n{}".format(item[0], item))
57
58     def delete_phone(self, item_pk):
59         self.model.delete_phone(item_pk)
60         print("Phone {} was successfully deleted".format(item_pk))
```

```
61     def delete_email(self, item_pk):
62         self.model.delete_email(item_pk)
63         print("Email {} was successfully deleted".format(item_pk))
64
65     def delete_customers(self, item_start_pk, item_end_pk):
66         self.model.delete_customer(item_start_pk, item_end_pk)
67         print("All customers in ID range [{}, {}] was successfully deleted".format(item_start_pk, item_end_pk))
68
69     def delete_goods(self, item_start_pk, item_end_pk):
70         self.model.delete_goods(item_start_pk, item_end_pk)
71         print("All goods in ID range [{}, {}] was successfully deleted".format(item_start_pk, item_end_pk))
72
73     def delete_orders(self, item_start_pk, item_end_pk):
74         self.model.delete_order(item_start_pk, item_end_pk)
75         print("All orders in ID range [{}, {}] was successfully deleted".format(item_start_pk, item_end_pk))
76
77     def delete_categories(self, item_start_pk, item_end_pk):
78         self.model.delete_category(item_start_pk, item_end_pk)
79         print("All categories in ID range [{}, {}] was successfully deleted".format(item_start_pk, item_end_pk))
80
81     def generate_goods(self, items_counter):
82         self.model.generate_goods(items_counter)
83         print("{} random goods was successfully generated".format(items_counter))
84
85     def generate_customers(self, items_counter):
86         self.model.generate_customers(items_counter)
87         print("{} random customers was successfully generated".format(items_counter))
88
89     def generate_categories(self, items_counter):
90         self.model.generate_categories(items_counter)
91
92         print("{} random categories was successfully generated".format(items_counter))
93
94     def generate_orders(self, items_counter):
95         self.model.generate_orders(items_counter)
96         print("{} random orders was successfully generated".format(items_counter))
97
98     def generate_phones(self, items_counter):
99         self.model.generate_phone(items_counter)
100         print("{} random phones was successfully generated".format(items_counter))
101
102     def generate_emails(self, items_counter):
103         self.model.generate_emails(items_counter)
104         print("{} random phones was successfully generated".format(items_counter))
105
106     def find_items(self, tables):
107         time_before = time.time()
108         items = self.model.find_entities(tables)
109         time_after = time.time()
110         for item in items:
111             print(item)
112         print("Operation time --- {} ms.".format((time_after-time_before)*1000))
```

Controller:

```
1  from view import View
2  import model
3
4  inp_requests = dict({"goods": ["Enter name", "Enter price", "Enter discount", "Enter guarantee",
5                                "Enter category ID"],
6                       "customer": ["Enter surname", "Enter name", "Enter father name"],
7                       "phone": ["Enter phone number", "Enter customer ID"],
8                       "email": ["Enter email", "Enter customer ID"],
9                       "order": ["Enter date", "Enter goods ID", "Enter customer ID", "Enter confirming method"],
10                      "category": ["Enter category name", "Enter parent category ID"]})
11
12
13  def validate(option, item):
14      if option == "goods":
15          return True if (item[0].isalnum() and item[1].isdigit() and item[2].isdigit() and item[3].isdigit() and
16                          item[4].isdigit()) else False
17      if option == "customer":
18          return True if (item[0].isalpha() and item[1].isalpha() and item[2].isalpha()) else False
19      if option == "phone":
20          return True if item[1].isdigit() else False
21      if option == "email":
22          return True if item[1].isdigit() else False
23      if option == "order":
24          return True if (item[1].isdigit() and item[2].isdigit() and (item[3] == 'phone' or item[3] == 'email'))\
25                          else False
26      if option == "category":
27          return True if item[1].isdigit() else False
28
29
30  class Controller:
31
32      def __init__(self):
33          self.view = View()
34          self.mod = model.Model()
35
36      def insert_item(self, option):
37          global inp_requests
38          item = list()
39          for request in inp_requests[option]:
40              item.append(input(request+": "))
41          if option == "goods":
42              if validate(option, item):
43                  self.view.insert_goods(item)
44              else:
45                  raise Exception("Incorrect type of entered values")
46          if option == "customer":
47              if validate(option, item):
48                  self.view.insert_customer(item)
49              else:
50                  raise Exception("Incorrect type of entered values")
51          if option == "phone":
52              if validate(option, item):
53                  self.view.insert_phone(item)
54              else:
55                  raise Exception("Incorrect type of entered values")
56          if option == "email":
57              if validate(option, item):
58                  self.view.insert_email(item)
59              else:
60                  raise Exception("Incorrect type of entered values")
61          if option == "order":
```

```

61         if validate(option, item):
62             self.view.insert_order(item)
63         else:
64             raise Exception("Incorrect type of entered values")
65     if option == "category":
66         if validate(option, item):
67             self.view.insert_category(item)
68         else:
69             raise Exception("Incorrect type of entered values")
70
71     def update_item(self, option, item_pk):
72         if not item_pk.isdecimal():
73             raise Exception("\'{}\'' is not a decimal id".format(item_pk))
74         global inp_requests
75         item = list()
76         for request in inp_requests[option]:
77             item.append(input(request+" (Enter empty row to skip): "))
78         new_item = list()
79         if option == "goods":
80             curr_item = self.mod.read_goods_by_pk(int(item_pk))
81             new_item.append(curr_item[0][0])
82             for i in range(1, 6):
83                 if item[i-1] != "":
84                     new_item.append(item[i-1])
85                 else:
86                     new_item.append(curr_item[0][i])
87             if validate(option, new_item):
88                 self.view.update_goods(new_item)
89             else:
90                 raise Exception("Incorrect type of entered values")
91
92         if option == "customer":
93             curr_item = self.mod.read_customer_by_pk(int(item_pk))
94             new_item.append(curr_item[0][0])
95             for i in range(1, 4):
96                 if item[i-1] != "":
97                     new_item.append(item[i-1])
98                 else:
99                     new_item.append(curr_item[0][i])
100             if validate(option, new_item):
101                 self.view.update_customer(new_item)
102             else:
103                 raise Exception("Incorrect type of entered values")
104         if option == "phone":
105             curr_item = self.mod.read_phone_by_pk(item_pk)
106             new_item.append(curr_item[0][0])
107             for i in range(1, 3):
108                 if item[i-1] != "":
109                     new_item.append(item[i-1])
110                 else:
111                     new_item.append(curr_item[0][i-1])
112             if validate(option, new_item):
113                 self.view.update_phone(new_item)
114             else:
115                 raise Exception("Incorrect type of entered values")
116         if option == "email":
117             curr_item = self.mod.read_email_by_pk(item_pk)
118             new_item.append(curr_item[0][0])
119             for i in range(1, 3):
120                 if item[i-1] != "":

```

```

121         else:
122             new_item.append(curr_item[0][i-1])
123         if validate(option, new_item):
124             self.view.update_email(new_item)
125         else:
126             raise Exception("Incorrect type of entered values")
127     if option == "order":
128         curr_item = self.mod.read_order_by_pk(int(item_pk))
129         new_item.append(curr_item[0][0])
130         for i in range(1, 5):
131             if item[i-1] != "":
132                 new_item.append(item[i-1])
133             else:
134                 new_item.append(curr_item[0][i])
135         if validate(option, new_item):
136             self.view.update_order(new_item)
137         else:
138             raise Exception("Incorrect type of entered values")
139     if option == "category":
140         curr_item = self.mod.read_category_by_pk(int(item_pk))
141         new_item.append(curr_item[0][0])
142         for i in range(1, 3):
143             if item[i-1] != "":
144                 new_item.append(item[i-1])
145             else:
146                 new_item.append(curr_item[0][i])
147         if validate(option, new_item):
148             self.view.update_category(new_item)
149         else:
150             raise Exception("Incorrect type of entered values")
151
152     def delete_items(self, option, item_start_pk, item_end_pk=0):
153         if not item_start_pk.isdecimal() or not str(item_end_pk).isdecimal():
154             raise Exception("{}\{}\' or \{}\{}\' is not a decimal id".format(item_start_pk, item_end_pk))
155         if item_end_pk == 0:
156             item_end_pk = item_start_pk
157         if option == "goods":
158             self.view.delete_goods(item_start_pk, item_end_pk)
159         if option == "customer":
160             self.view.delete_customers(item_start_pk, item_end_pk)
161         if option == "category":
162             self.view.delete_categories(item_start_pk, item_end_pk)
163         if option == "order":
164             self.view.delete_orders(item_start_pk, item_end_pk)
165         if option == "phone":
166             self.view.delete_phone(item_start_pk)
167         if option == "email":
168             self.view.delete_email(item_start_pk)
169
170     def generate_items(self, option, items_number):
171         if not items_number.isdecimal():
172             raise Exception("{}\{}\' is not a decimal".format(items_number))
173         if option == "goods":
174             self.view.generate_goods(items_number)
175         if option == "customer":
176             self.view.generate_customers(items_number)
177         if option == "category":
178             self.view.generate_categories(items_number)
179         if option == "order":
180             self.view.generate_orders(items_number)
181
182         if option == "phone":
183             self.view.generate_phones(items_number)
184         if option == "email":
185             self.view.generate_emails(items_number)
186
187     def find_items(self, subcommand):
188         query = subcommand.split('?')
189         self.view.find_items(query)

```