

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ**

**Отчет о лабораторной работе №2.21 по дисциплине основы
программной инженерии**

Выполнил:

XXX XXX XXX,
2 курс, группа ПИЖ-б-о-20-1,

Проверил:

Доцент кафедры
прикладной математики и
компьютерной
безопасности, Воронкин Р.А.

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2022 г.

ВЫПОЛНЕНИЕ:

Примеры из МУ:

```
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 import sqlite3
6 from sqlite3 import Error
7
8
9 def sql_connection():
10     try:
11         con = sqlite3.connect(':memory:')
12         print("Connection is established: Database is created in memory")
13     except Error:
14         print(Error)
15     finally:
16         con.close()
17
18
19 ▶ if __name__ == "__main__":
20     sql_connection()
21
```

Run: e1 x

C:\Python37\python.exe C:/Users/student-09-525/PycharmProjects/OPI_9/Examples/e1.py

Connection is established: Database is created in memory

Process finished with exit code 0

```
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 import sqlite3
6 from sqlite3 import Error
7
8
9 def sql_connection():
10     try:
11         con = sqlite3.connect('mydatabase.db')
12         return con
13     except Error:
14         print(Error)
15     return None
16
17
18 def sql_table(con):
19     cursor_obj = con.cursor()
20     cursor_obj.execute(
21         """
22         CREATE TABLE employees (
23             id integer PRIMARY KEY,
24             name text,
25             salary real,
26             department text,
27             position text,
28             hireDate text)
29         """
30     )
31     con.commit()
32
33
34 ▶ if __name__ == "__main__":
35     con = sql_connection()
36     sql_table(con)
37
```

Таблица: employees

id	name	salary	department	position	hireDate
Фи...	Фил...	Фил...	Фильтр	Фильтр	Фильтр

```

1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 import sqlite3
6
7 con = sqlite3.connect('mydatabase.db')
8
9
10 def sql_insert(con, entities):
11     cursor_obj = con.cursor()
12     cursor_obj.execute(
13         """
14         INSERT INTO employees VALUES(1, 'John', 700, 'HR',
15         'Manager', '2017 - 01 - 04')
16         """
17     )
18     cursor_obj.execute(
19         """
20         INSERT INTO employees(id, name, salary, department, position, hireDate)
21         VALUES(?, ?, ?, ?, ?, ?)
22         """
23     )
24     entities
25     con.commit()
26
27
28 ▶ if __name__ == "__main__":
29     entities = (2, 'Andrew', 800, 'IT', 'Tech', '2018-02-06')
30     sql_insert(con, entities)
31

```

Таблица:  employees 

	id	name	salary	department	position	hireDate
	Фи...	Фильтр	Фил...	Фильтр	Фильтр	Фильтр
1	1	John	700.0	HR	Manager	2017 - 01 - 04
2	2	Andrew	800.0	IT	Tech	2018-02-06

```

1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 import sqlite3
6
7
8 con = sqlite3.connect('mydatabase.db')
9
10
11 def sql_update(con):
12     cursor_obj = con.cursor()
13     cursor_obj.execute(
14         """UPDATE employees SET name = 'Rogers' where id = 2"""
15     )
16     con.commit()
17
18
19 ▶ if __name__ == "__main__":
20     sql_update(con)
21

```

	id	name	salary	department	position	hireDate
	Фи...	Фил...	Фил...	Фильтр	Фильтр	Фильтр
1	1	John	700.0	HR	Manager	2017 - 01 - 04
2	2	Rogers	800.0	IT	Tech	2018-02-06

```
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 import sqlite3
6
7
8 con = sqlite3.connect('mydatabase.db')
9
10
11 def sql_fetch(con):
12     cursor_obj = con.cursor()
13     cursor_obj.execute("SELECT * FROM employees")
14     rows = cursor_obj.fetchall()
15     for row in rows:
16         print(row)
17
18
19 ▶ if __name__ == "__main__":
20     sql_fetch(con)
21
```

Run: e5 ×

▶ C:\Python37\python.exe C:/Users/student-09-525/PycharmProjects/OPI_9/Examples/e5.py
(1, 'John', 700.0, 'HR', 'Manager', '2017 - 01 - 04')
(2, 'Rogers', 800.0, 'IT', 'Tech', '2018-02-06')

```
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 import sqlite3
6
7
8 con = sqlite3.connect('mydatabase.db')
9
10
11 def sql_fetch(con):
12     cursor_obj = con.cursor()
13     cursor_obj.execute(
14         "SELECT id, name FROM employees WHERE salary ≥ 800.0"
15     )
16     rows = cursor_obj.fetchall()
17     for row in rows:
18         print(row)
19
20
21 ▶ if __name__ == "__main__":
22     sql_fetch(con)
23
```

Run: e5 ×

▶ C:\Python37\python.exe C:/Users/student-09-525/PycharmProjects/OPI_9/Examples/e5.py
(1, 'John', 700.0, 'HR', 'Manager', '2017 - 01 - 04')
(2, 'Rogers', 800.0, 'IT', 'Tech', '2018-02-06')

```
1 ▶ #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4
5 import sqlite3
6
7
8 con = sqlite3.connect('mydatabase.db')
9
10
11 def sql_fetch(con):
12     cursor_obj = con.cursor()
13     cursor_obj.execute(
14         "SELECT name from sqlite_master where type='table'"
15     )
16     print(cursor_obj.fetchall())
17
18 ▶ if __name__ == "__main__":
19     sql_fetch(con)
20
```

Run: e7 ×

▶ C:\Python37\python.exe C:/Users/student-09-525/PycharmProjects/OPI_9/Examples/e7.py
[('employees',)]

```

1 ▶ #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3
4
5   import sqlite3
6
7   ▶ if __name__ == "__main__":
8       con = sqlite3.connect('mydatabase.db')
9
10      cursor_obj = con.cursor()
11      cursor_obj.execute(
12          "CREATE TABLE IF NOT EXISTS projects(id INTEGER, name TEXT)"
13      )
14      data = [
15          (1, "Ridesharing"),
16          (2, "Water Purifying"),
17          (3, "Forensics"),
18          (4, "Botany")
19      ]
20      cursor_obj.executemany("INSERT INTO projects VALUES (?, ?)", data)
21      con.commit()
22

```




Таблица: projects

	id	name
	Фи...	Фильтр
1	1	Ridesharing
2	2	Water Purifying
3	3	Forensics
4	4	Botany

```

1 ▶ #!/usr/bin/env python3
2   # -*- coding: utf-8 -*-
3
4
5   import ...
6
7
8
9 ▶   if __name__ == "__main__":
10      con = sqlite3.connect('mydatabase.db')
11
12      cursor_obj = con.cursor()
13      cursor_obj.execute(
14          """
15      CREATE TABLE IF NOT EXISTS assignments(
16      id INTEGER, name TEXT, date DATE
17      )
18      """
19      )
20      data = [
21          (1, "Ridesharing", datetime.date(2017, 1, 2)),
22          (2, "Water Purifying", datetime.date(2018, 3, 4))
23      ]
24      cursor_obj.executemany("INSERT INTO assignments VALUES(?, ?, ?)", data)
25      con.commit()
26

```

Таблица:  assignments  

	id	name	date
	Фи...	Фильтр	Фильтр
1	1	Ridesharing	2017-01-02
2	2	Water Purifying	2018-03-04

```

1  ▶  #!/usr/bin/env python3
2      # -*- coding: utf-8 -*-
3
4
5  import argparse
6  import sqlite3
7  import typing as t
8  from pathlib import Path
9
10
11  def display_workers(staff: t.List[t.Dict[str, t.Any]]) → None:
12      """
13      Отобразить список работников.
14      """
15      # Проверить, что список работников не пуст.
16      if staff:
17          # Заголовок таблицы.
18          line = '+-{}-+-{}-+-{}-+-{}-+'.format(
19              '-' * 4,
20              '-' * 30,
21              '-' * 20,
22              '-' * 8
23          )
24          print(line)
25          print(
26              '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
27                  "No",
28                  "Ф.И.О.",
29                  "Должность",
30                  "Год"
31              )
32          )
33          print(line)
34          # Вывести данные о всех сотрудниках.
35          for idx, worker in enumerate(staff, 1):
36              print(
37                  '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
38                      idx,
39                      worker.get('name', ''),
40                      worker.get('post', ''),
41                      worker.get('year', 0)
42                  )
43              )
44              print(line)
45      else:
46          print("Список работников пуст.")
47
48
49  def create_db(database_path: Path) → None:
50      """
51      Создать базу данных.
52      """
53      conn = sqlite3.connect(database_path)
54      cursor = conn.cursor()
55      # Создать таблицу с информацией о должностях.
56      cursor.execute(
57          """
58          CREATE TABLE IF NOT EXISTS posts (
59              post_id INTEGER PRIMARY KEY AUTOINCREMENT,
60              post_title TEXT NOT NULL
61          )
62          """
63      )
64      # Создать таблицу с информацией о работниках.

```

```

65     cursor.execute(
66         """
67         CREATE TABLE IF NOT EXISTS workers (
68             worker_id INTEGER PRIMARY KEY AUTOINCREMENT,
69             worker_name TEXT NOT NULL,
70             post_id INTEGER NOT NULL,
71             worker_year INTEGER NOT NULL,
72             FOREIGN KEY(post_id) REFERENCES posts(post_id)
73         )
74         """
75     )
76     conn.close()
77
78
79     def add_worker(
80         database_path: Path,
81         name: str,
82         post: str,
83         year: int
84     ) → None:
85         """
86         Добавить работника в базу данных.
87         """
88         conn = sqlite3.connect(database_path)
89         cursor = conn.cursor()
90         # Получить идентификатор должности в базе данных.
91         # Если такой записи нет, то добавить информацию о новой должности.
92         cursor.execute(
93             """
94             SELECT post_id FROM posts WHERE post_title = ?
95             """
96             ,
97             (post,)
98         )
99         row = cursor.fetchone()
100         if row is None:
101             cursor.execute(
102                 """
103                 INSERT INTO posts (post_title) VALUES (?)
104                 """
105                 ,
106                 (post,)
107             )
108             post_id = cursor.lastrowid
109         else:
110             post_id = row[0]
111             # Добавить информацию о новом работнике.
112             cursor.execute(
113                 """
114                 INSERT INTO workers (worker_name, post_id, worker_year)
115                 VALUES (?, ?, ?)
116                 """
117                 ,
118                 (name, post_id, year)
119             )
120             conn.commit()
121             conn.close()
122
123     def select_all(database_path: Path) → t.List[t.Dict[str, t.Any]]:
124         """
125         Выбрать всех работников.
126         """
127         conn = sqlite3.connect(database_path)
128         cursor = conn.cursor()
129         cursor.execute(
130             """

```



```

129         SELECT workers.worker_name, posts.post_title, workers.worker_year
130         FROM workers
131         INNER JOIN posts ON posts.post_id = workers.post_id
132         """
133     )
134     rows = cursor.fetchall()
135     conn.close()
136     return [
137         {
138             "name": row[0],
139             "post": row[1],
140             "year": row[2],
141         }
142         for row in rows
143     ]
144
145
146 def select_by_period(
147     database_path: Path, period: int
148 ) → t.List[t.Dict[str, t.Any]]:
149     """
150     Выбрать всех работников с периодом работы больше заданного.
151     """
152     conn = sqlite3.connect(database_path)
153     cursor = conn.cursor()
154     cursor.execute(
155         """
156         SELECT workers.worker_name, posts.post_title, workers.worker_year
157         FROM workers
158         INNER JOIN posts ON posts.post_id = workers.post_id
159         WHERE (strftime('%Y', date('now')) - workers.worker_year) ≥ ?
160         """,
161         (period,)
162     )
163     rows = cursor.fetchall()
164     conn.close()
165     return [
166         {
167             "name": row[0],
168             "post": row[1],
169             "year": row[2],
170         }
171         for row in rows
172     ]
173
174
175 def main(command_line=None):
176     # Создать родительский парсер для определения имени файла.
177     file_parser = argparse.ArgumentParser(add_help=False)
178     file_parser.add_argument(
179         "--db",
180         action="store",
181         required=False,
182         default=str(Path.home() / "workers.db"),
183         help="The database file name"
184     )
185     # Создать основной парсер командной строки.
186     parser = argparse.ArgumentParser("workers")
187     parser.add_argument(
188         "--version",
189         action="version",
190         version=f"%(prog)s 0.1.0"
191     )
192     subparsers = parser.add_subparsers(dest="command")

```

Индивидуальное задание:

```

196         parents=[file_parser],
197         help="Add a new worker"
198     )
199     add.add_argument(
200         "-n",
201         "--name",
202         action="store",
203         required=True,
204         help="The worker's name"
205     )
206     add.add_argument(
207         "-p",
208         "--post",
209         action="store",
210         help="The worker's post"
211     )
212     add.add_argument(
213         "-y",
214         "--year",
215         action="store",
216         type=int,
217         required=True,
218         help="The year of hiring"
219     )
220     # Создать субпарсер для отображения всех работников.
221     _ = subparsers.add_parser(
222         "display",
223         parents=[file_parser],
224         help="Display all workers"
225     )
226     # Создать субпарсер для выбора работников.
227     select = subparsers.add_parser(
228         "select",
229         parents=[file_parser],
230         help="Select the workers"
231     )
232     select.add_argument(
233         "-p",
234         "--period",
235         action="store",
236         type=int,
237         required=True,
238         help="The required period"
239     )
240     # Выполнить разбор аргументов командной строки.
241     args = parser.parse_args(command_line)
242     # Получить путь к файлу базы данных.
243     db_path = Path(args.db)
244     create_db(db_path)
245     # Добавить работника.
246     if args.command == "add":
247         add_worker(db_path, args.name, args.post, args.year)
248     # Отобразить всех работников.
249     elif args.command == "display":
250         display_workers(select_all(db_path))
251     # Выбрать требуемых работников.
252     elif args.command == "select":
253         display_workers(select_by_period(db_path, args.period))
254     pass
255
256
257 ▶ if __name__ == "__main__":
258     main()
259

```

PS C:\Users\student-09-525\PycharmProjects\OPI_9\Examples> python .\example.py add --db="workers.db" --name="Mqbcjb" --post="QEkhds" --year="2018"

PS C:\Users\student-09-525\PycharmProjects\OPI_9\Examples> python .\example.py add --db="workers.db" --name="Knqmcn" --post="Ejhiqjxk" --year="2015"

PS C:\Users\student-09-525\PycharmProjects\OPI_9\Examples> python .\example.py display --db="workers.db"

No	Ф.И.О.	Должность	Год
1	Mqbcjb	QEkhds	2018
2	Knqmcn	Ejhiqjxk	2015

PS C:\Users\student-09-525\PycharmProjects\OPI_9\Examples> python .\example.py select --db="workers.db" --period=12

Список работников пуст.

PS C:\Users\student-09-525\PycharmProjects\OPI_9\Examples> python .\example.py select --db="workers.db" --period=5

No	Ф.И.О.	Должность	Год
1	Knqmcn	Ejhiqjxk	2015

PS C:\Users\student-09-525\PycharmProjects\OPI_9\Examples> █

```

1  ▶ #!/usr/bin/env python3
2  # -*- coding: utf-8 -*-
3
4
5  import argparse
6  import sqlite3
7  import typing as t
8  from pathlib import Path
9
10
11 def display_flights(flights: t.List[t.Dict[str, t.Any]]) → None:
12     """
13     Отобразить список рейсов.
14     """
15     if flights:
16         # Заголовок таблицы.
17         line = '+-{}-+-{}-+-{}-+-{}-+'.format(
18             '-' * 4,
19             '-' * 30,
20             '-' * 20,
21             '-' * 15
22         )
23         print(line)
24         print(
25             '| {:^4} | {:^30} | {:^20} | {:^15} |'.format(
26                 "No",
27                 "Пункт назначения",
28                 "Номер рейса",
29                 "Тип самолёта"
30             )
31         )
32         print(line)
33         for idx, flight in enumerate(flights, 1):
34             print(
35                 '| {:>4} | {:<30} | {:<20} | {:<15} |'.format(
36                     idx,
37                     flight.get('flight_destination', ''),
38                     flight.get('flight_number', ''),
39                     flight.get('airplane_type', 0)
40                 )
41             )
42             print(line)
43     else:
44         print("Список рейсов пуст.")
45
46
47 def create_db(database_path: Path) → None:
48     """
49     Создать базу данных.
50     """
51     conn = sqlite3.connect(database_path)
52     cursor = conn.cursor()
53     cursor.execute(
54         """
55         CREATE TABLE IF NOT EXISTS flight_numbers (
56             num_id INTEGER PRIMARY KEY AUTOINCREMENT,
57             num_title TEXT NOT NULL
58         )
59         """
60     )
61     cursor.execute(
62         """
63         CREATE TABLE IF NOT EXISTS flights (
64             flight_id INTEGER PRIMARY KEY AUTOINCREMENT,

```

```

65         flight_destination TEXT NOT NULL,
66         num_id INTEGER NOT NULL,
67         airplane_type TEXT NOT NULL,
68         FOREIGN KEY(num_id) REFERENCES flight_numbers(num_id)
69     )
70     """
71 )
72 conn.close()
73
74
75 def add_flight(
76     database_path: Path,
77     flight_destination: str,
78     flight_number: str,
79     airplane_type: int
80 ) → None:
81     """
82     Добавить рейс в базу данных.
83     """
84     conn = sqlite3.connect(database_path)
85     cursor = conn.cursor()
86     cursor.execute(
87         """
88         SELECT num_id FROM flight_numbers WHERE num_title = ?
89         """,
90         (flight_number,)
91     )
92     row = cursor.fetchone()
93     if row is None:
94         cursor.execute(
95             """
96             INSERT INTO flight_numbers (num_title) VALUES (?)
97             """,
98             (flight_number,)
99         )
100         num_id = cursor.lastrowid
101     else:
102         num_id = row[0]
103     cursor.execute(
104         """
105         INSERT INTO flights (flight_destination, num_id, airplane_type)
106         VALUES (?, ?, ?)
107         """,
108         (flight_destination, num_id, airplane_type)
109     )
110     conn.commit()
111     conn.close()
112
113
114 def select_all(database_path: Path) → t.List[t.Dict[str, t.Any]]:
115     """
116     Выбрать все рейсы.
117     """
118     conn = sqlite3.connect(database_path)
119     cursor = conn.cursor()
120     cursor.execute(
121         """SELECT flights.flight_destination, flight_numbers.num_title,
122         flights.airplane_type FROM flights INNER JOIN flight_numbers ON
123         flight_numbers.num_id = flights.num_id """
124     )
125     rows = cursor.fetchall()
126     conn.close()
127     return [
128         {

```

```

129         "flight_destination": row[0],
130         "flight_number": row[1],
131         "airplane_type": row[2],
132     }
133     for row in rows
134 ]
135
136
137 def select_flights(
138     database_path: Path, air_type: str
139 ) → t.List[t.Dict[str, t.Any]]:
140     """
141     Выбрать все рейсы заданного типа.
142     """
143     conn = sqlite3.connect(database_path)
144     cursor = conn.cursor()
145     cursor.execute(
146         """
147         SELECT flights.flight_destination, flight_numbers.num_title,
148         flights.airplane_type
149         FROM flights
150         INNER JOIN flight_numbers ON flight_numbers.num_id = flights.num_id
151         WHERE flights.airplane_type = ?
152         """,
153         (air_type,)
154     )
155     rows = cursor.fetchall()
156     conn.close()
157     return [
158         {
159             "flight_destination": row[0],
160             "flight_number": row[1],
161             "airplane_type": row[2],
162         }
163         for row in rows
164     ]
165
166
167 def main(command_line=None):
168     # Создать родительский парсер для определения имени файла.
169     file_parser = argparse.ArgumentParser(add_help=False)
170     file_parser.add_argument(
171         "--db",
172         action="store",
173         required=False,
174         default=str(Path.home() / "workers.db"),
175         help="The database file name"
176     )
177     # Создать основной парсер командной строки.
178     parser = argparse.ArgumentParser("workers")
179     parser.add_argument(
180         "--version",
181         action="version",
182         version=f"%(prog)s 0.1.0"
183     )
184     subparsers = parser.add_subparsers(dest="command")
185     add = subparsers.add_parser(
186         "add",
187         parents=[file_parser],
188         help="Add a new flight"
189     )
190     add.add_argument(
191         "-d",
192         "--dest",

```

ОТВЕТЫ НА ВОПРОСЫ

```

193         action="store",
194         required=True,
195         help="Flight destination"
196     )
197     add.add_argument(
198         "-n",
199         "--flight_num",
200         action="store",
201         help="The flight number"
202     )
203     add.add_argument(
204         "-t",
205         "--type",
206         action="store",
207         type=str,
208         required=True,
209         help="The airplane type"
210     )
211     _ = subparsers.add_parser(
212         "display",
213         parents=[file_parser],
214         help="Display all flights"
215     )
216     select = subparsers.add_parser(
217         "select",
218         parents=[file_parser],
219         help="Select the flights"
220     )
221     select.add_argument(
222         "-T",
223         "--type",
224         action="store",
225         type=str,
226         required=True,
227         help="The required type"
228     )
229     # Выполнить разбор аргументов командной строки.
230     args = parser.parse_args(command_line)
231     # Получить путь к файлу базы данных.
232     db_path = Path(args.db)
233     create_db(db_path)
234     if args.command == "add":
235         add_flight(db_path, args.dest, args.flight_num, args.type)
236     elif args.command == "display":
237         display_flights(select_all(db_path))
238     elif args.command == "select":
239         display_flights(select_flights(db_path, args.type))
240     pass
241
242
243 ► if __name__ == "__main__":
244     main()
245
PS C:\Users\student-09-525\PycharmProjects\0PI_9\Individual> python .\individual.py add --db="flights.db" --dest="Chicago" --flight_num="N146" --type="Military"
PS C:\Users\student-09-525\PycharmProjects\0PI_9\Individual> python .\individual.py add --db="flights.db" --dest="Chicago" --flight_num="N146" --type="Passenger"
PS C:\Users\student-09-525\PycharmProjects\0PI_9\Individual> python .\individual.py add --db="flights.db" --dest="Chicago" --flight_num="F1346" --type="Passenger"
PS C:\Users\student-09-525\PycharmProjects\0PI_9\Individual> python .\individual.py display --db="flights.db"
+-----+-----+-----+-----+
| No | Пункт назначения | Номер рейса | Тип самолёта |
+-----+-----+-----+-----+
| 1 | Miami | M1245 | Passenger |
+-----+-----+-----+-----+
| 2 | Egypt | EG436 | Sanitary |
+-----+-----+-----+-----+
| 3 | Chicago | N146 | Military |
+-----+-----+-----+-----+
| 4 | Chicago | N146 | Passenger |
+-----+-----+-----+-----+
| 5 | Chicago | F1346 | Passenger |
+-----+-----+-----+-----+
PS C:\Users\student-09-525\PycharmProjects\0PI_9\Individual> python .\individual.py select --db="flights.db" --type="Passenger"
+-----+-----+-----+-----+
| No | Пункт назначения | Номер рейса | Тип самолёта |
+-----+-----+-----+-----+
| 1 | Miami | M1245 | Passenger |
+-----+-----+-----+-----+
| 2 | Chicago | N146 | Passenger |
+-----+-----+-----+-----+
| 3 | Chicago | F1346 | Passenger |
+-----+-----+-----+-----+
PS C:\Users\student-09-525\PycharmProjects\0PI_9\Individual>

```

1. Каково назначение модуля sqlite3 ?

Непосредственно модуль sqlite3 – это API к СУБД SQLite. Своего рода адаптер, который переводит команды, написанные на Питоне, в команды, которые понимает SQLite. Как и наоборот, доставляет ответы от SQLite в python-программу. Модуль sqlite3 содержит много классов, функций и констант. Их перечень можно посмотреть с помощью функции `dir()`.

2. Как выполняется соединение с базой данных SQLite3? Что такое курсор базы данных?

Чтобы использовать SQLite3 в Python, прежде всего, вам нужно будет импортировать модуль `sqlite3`, а затем создать объект соединения, который соединит нас с базой данных и позволит нам выполнять операторы SQL. Объект соединения создается с помощью функции `connect()`. Вызов функции `connect()` приводит к созданию объекта-экземпляра от класса `Connection`.

Этот объект обеспечивает связь с файлом базы данных, представляет конкретную БД в программе. Для взаимодействия с базой данных SQLite3 в Python необходимо создать объект `cursor`. Вы можете создать его с помощью метода `cursor()`. Курсор SQLite3 – это метод объекта соединения. Для выполнения инструкций SQLite3 сначала устанавливается соединение, а затем создается объект курсора с использованием объекта соединения следующим образом

3. Как подключиться к базе данных SQLite3, находящейся в оперативной памяти компьютера?

Мы также можем создать базу данных в оперативной памяти с помощью функции `:memory:` with the `connect`. Такая база данных называется базой данных в памяти.

4. Как корректно завершить работу с базой данных SQLite3?

`con.close()` для высвобождения памяти.

5. Как осуществляется вставка данных в таблицу базы данных SQLite3?

Чтобы вставить данные в таблицу, используется оператор `INSERT INTO`.

6. Как осуществляется обновление данных таблицы базы данных

SQLite3?

Чтобы обновить данные в таблице, просто создайте соединение, затем создайте объект курсора с помощью соединения и, наконец, используйте оператор UPDATE в методе execute (). Предположим, что мы хотим обновить имя сотрудника, чей идентификатор равен 2. Для обновления мы будем использовать оператор UPDATE , а для сотрудника, чей идентификатор равен 2. Мы будем использовать предложение WHERE в качестве условия для выбора этого сотрудника.

```
def sql_update(con):  
    cursor_obj = con.cursor()  
    cursor_obj.execute(  
        "UPDATE employees SET name = 'Rogers' where id = 2"  
    )  
    con.commit()
```

7. Как осуществляется выборка данных из базы данных SQLite3?

Оператор SELECT используется для выбора данных из определенной таблицы. Если вы хотите выбрать все столбцы данных из таблицы, вы можете использовать звездочку (*).

8. Каково назначение метода rowcount?

SQLite3 rowcount используется для возврата количества строк, которые были затронуты или выбраны последним выполненным SQL-запросом. Когда мы используем rowcount с оператором SELECT , будет возвращено значение -1, поскольку количество выбранных строк неизвестно до тех пор, пока все они не будут извлечены.

9. Как получить список всех таблиц базы данных SQLite3?

Чтобы перечислить все таблицы в базе данных SQLite3, вы должны запросить данные из таблицы sqlite_master, а затем использовать fetchall() для получения результатов из инструкции SELECT . sqlite_master – это главная таблица в SQLite3, которая хранит все таблицы.

10. Как выполнить проверку существования таблицы как при ее

добавлении, так и при ее удалении?

Чтобы проверить, не существует ли таблица уже, мы используем IF NOT EXISTS с оператором CREATE TABLE следующим образом:

```
CREATE TABLE IF NOT EXISTS table_name (column1, column2, ..., columnN)
```

```
DROP TABLE IF EXISTS table_name
```

11. Как выполнить массовую вставку данных в базу данных SQLite3?

Метод executemany можно использовать для вставки нескольких строк одновременно.

```
cursor_obj.execute(
```

```
"CREATE TABLE IF NOT EXISTS projects(id INTEGER, name TEXT)"  
)
```

```
data = [
```

```
(1, "Ridesharing"),
```

```
(2, "Water Purifying"),
```

```
(3, "Forensics"),
```

```
(4, "Botany")
```

```
]
```

```
cursor_obj.executemany("INSERT INTO projects VALUES (?, ?)", data)
```

```
con.commit()
```

12. Как осуществляется работа с датой и временем при работе с базами

данных SQLite3

В базе данных Python SQLite3 мы можем легко хранить дату или время, импортируя модуль datetime .

```
cursor_obj.execute(
```

```
"""
```

```
CREATE TABLE IF NOT EXISTS assignments(
```

```
id INTEGER, name TEXT, date DATE
```

```
)
```

```
"""
```

```
)
```

```
data = [
```

```
(1, "Ridesharing", datetime.date(2017, 1, 2)),
```

```
(2, "Water Purifying", datetime.date(2018, 3, 4))
```

```
]
```

```
cursor_obj.executemany("INSERT INTO assignments VALUES(?, ?, ?)",  
data)
```

```
con.commit()
```