Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития

Кафедра инфокоммуникаций

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №2.21**

**дисциплины «Анализ данных»**

**Вариант 13**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | Выполнил:  Иващенко Олег Андреевич  2 курс, группа ИВТ-б-о-22-1,  09.03.02 «Информационные и вычислительные машины», направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |  |
|  | | Руководитель практики:  Воронкин Роман Александрович, доцент кафедры инфокоммуникаций  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |  |
|  |  | |  | |

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_ Дата защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ставрополь, 2024 г.

**Тема**: «Взаимодействие с базами данных SQLite3 с помощью языка программирования Python»

**Цель**: Изучить взаимодействие с базами данных SQLite3 при помощи ЯП Python.

Порядок выполнения работы

Пример 1. Для примера 1 лабораторной работы 2.17 реализовать возможность хранения данных в базе данных SQLite3.

Листинг 1 – Код примера

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  import argparse  import sqlite3  import typing as t  from pathlib import Path  def display\_workers(staff: t.List[t.Dict[str, t.Any]]) -> None:  """  Отобразить список работников.  """  # Проверить, что список работников не пуст.  if staff:  # Заголовок таблицы.  line = '+-{}-+-{}-+-{}-+-{}-+'.format(  '-' \* 4,  '-' \* 30,  '-' \* 20,  '-' \* 8  )  print(line)  print(  '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(  "№",  "Ф.И.О.",  "Должность",  "Год"  )  )  print(line)  # Вывести данные о всех сотрудниках.  for idx, worker in enumerate(staff, 1):  print(  '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(  idx,  worker.get('name', ''),  worker.get('post', ''),  worker.get('year', 0)  )  )  print(line)  else:  print("Список работников пуст.")  def create\_db(database\_path: Path) -> None:  """  Создать базу данных.  """  conn = sqlite3.connect(database\_path)  cursor = conn.cursor()  # Создать таблицу с информацией о должностях.  cursor.execute(  """  CREATE TABLE IF NOT EXISTS posts (  post\_id INTEGER PRIMARY KEY AUTOINCREMENT,  post\_title TEXT NOT NULL  )  """  )  # Создать таблицу с информацией о работниках.  cursor.execute(  """  CREATE TABLE IF NOT EXISTS workers (  worker\_id INTEGER PRIMARY KEY AUTOINCREMENT,  worker\_name TEXT NOT NULL,  post\_id INTEGER NOT NULL,  worker\_year INTEGER NOT NULL,  FOREIGN KEY(post\_id) REFERENCES posts(post\_id)  )  """  )  conn.close()  def add\_worker(  database\_path: Path,  name: str,  post: str,  year: int  ) -> None:  """  Добавить работника в базу данных.  """  conn = sqlite3.connect(database\_path)  cursor = conn.cursor()  # Получить идентификатор должности в базе данных.  # Если такой записи нет, то добавить информацию о новой должности.  cursor.execute(  """  SELECT post\_id FROM posts WHERE post\_title = ?  """,  (post,)  )  row = cursor.fetchone()  if row is None:  cursor.execute(  """  INSERT INTO posts (post\_title) VALUES (?)  """,  (post,)  )  post\_id = cursor.lastrowid  else:  post\_id = row[0]  # Добавить информацию о новом работнике.  cursor.execute(  """  INSERT INTO workers (worker\_name, post\_id, worker\_year)  VALUES (?, ?, ?)  """,  (name, post\_id, year)  )    conn.commit()  conn.close()  def select\_all(database\_path: Path) -> t.List[t.Dict[str, t.Any]]:  """  Выбрать всех работников.  """  conn = sqlite3.connect(database\_path)  cursor = conn.cursor()    cursor.execute(  """  SELECT workers.worker\_name, posts.post\_title, workers.worker\_year  FROM workers  INNER JOIN posts ON posts.post\_id = workers.post\_id  """  )  rows = cursor.fetchall()  conn.close()  return [  {  "name": row[0],  "post": row[1],  "year": row[2],  }  for row in rows  ]  def select\_by\_period(  database\_path: Path, period: int  ) -> t.List[t.Dict[str, t.Any]]:  """  Выбрать всех работников с периодом работы больше заданного.  """  conn = sqlite3.connect(database\_path)  cursor = conn.cursor()  cursor.execute(  """  SELECT workers.worker\_name, posts.post\_title, workers.worker\_year  FROM workers  INNER JOIN posts ON posts.post\_id = workers.post\_id  WHERE (strftime('%Y', date('now')) - workers.worker\_year) >= ?  """,  (period,)  )  rows = cursor.fetchall()  conn.close()  return [  {  "name": row[0],  "post": row[1],  "year": row[2],  }  for row in rows  ]  def main(command\_line=None):  # Создать родительский парсер для определения имени файла.  file\_parser = argparse.ArgumentParser(add\_help=False)  file\_parser.add\_argument(  "--db",  action="store",  required=False,  default=str(Path.home() / "workers.db"),  help="The database file name"  )  # Создать основной парсер командной строки.  parser = argparse.ArgumentParser("workers")  parser.add\_argument(  "--version",  action="version",  version="%(prog)s 0.1.0"  )  subparsers = parser.add\_subparsers(dest="command")  # Создать субпарсер для добавления работника.  add = subparsers.add\_parser(  "add",  parents=[file\_parser],  help="Add a new worker"  )  add.add\_argument(  "-n",  "--name",  action="store",  required=True,  help="The worker's name"  )  add.add\_argument(  "-p",  "--post",  action="store",  help="The worker's post"  )  add.add\_argument(  "-y",  "--year",  action="store",  type=int,  required=True,  help="The year of hiring"  )  # Создать субпарсер для отображения всех работников.  \_ = subparsers.add\_parser(  "display",  parents=[file\_parser],  help="Display all workers"  )  # Создать субпарсер для выбора работников.  select = subparsers.add\_parser(  "select",  parents=[file\_parser],  help="Select the workers"  )  select.add\_argument(  "-P",  "--period",  action="store",  type=int,  required=True,  help="The required period"  )  # Выполнить разбор аргументов командной строки.  args = parser.parse\_args(command\_line)  # Получить путь к файлу базы данных.  db\_path = Path(args.db)  create\_db(db\_path)  # Добавить работника.  if args.command == "add":  add\_worker(db\_path, args.name, args.post, args.year)  # Отобразить всех работников.  elif args.command == "display":  display\_workers(select\_all(db\_path))  # Выбрать требуемых рааботников.  elif args.command == "select":  display\_workers(select\_by\_period(db\_path, args.period))  pass  if \_\_name\_\_ == "\_\_main\_\_":  main() |

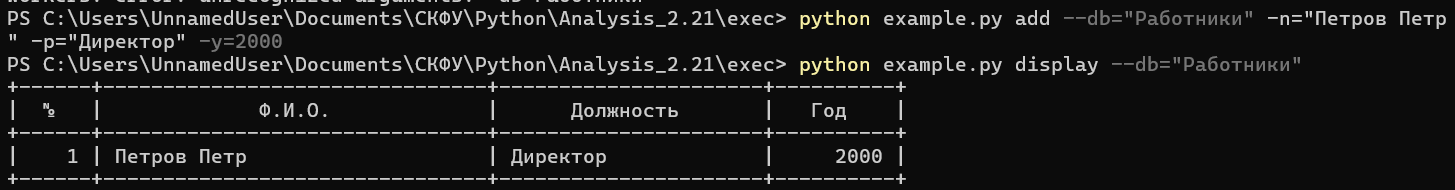


Рисунок 1.1 – Добавление и вывод новой записи в базу данных

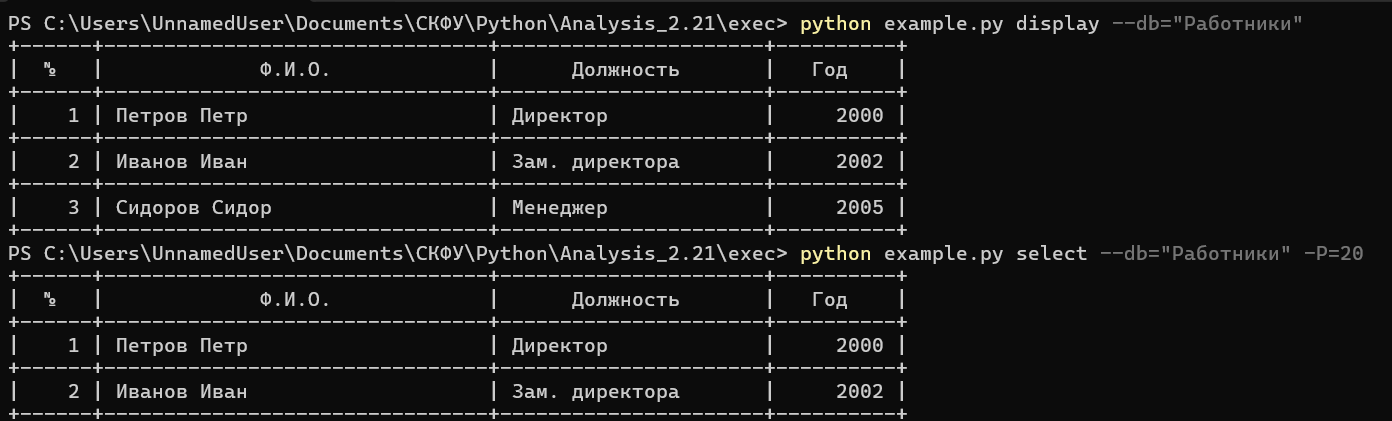


Рисунок 1.2 – Выборка записей из базы данных

Индивидуальное задание. Для своего варианта лабораторной работы 2.17 необходимо реализовать хранение данных в базе данных SQLite3. Информация в базе данных должна храниться не менее чем в двух таблицах.

Листинг 2 – Код программы

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  import json  from datetime import datetime  import argparse  import os.path  from pathlib import Path  import sqlite3  import typing as t  def print\_help():  """  Функция вывода доступных пользователю команд  """  print("list - вывод всех добавленных записей")  print("add - добавление новых записей")  print("find - найти запись по фамилии")  print("exit - завершение работы программы")  def add\_worker(database\_path: Path, name: str, phone: str, year: int) -> None:  """  Функция добавления новой записи, возвращает запись  """  connection = sqlite3.connect(database\_path)  cursor = connection.cursor()  cursor.execute(  """  INSERT INTO workers (worker\_name, phone\_number, worker\_year)  VALUES (?, ?, ?)  """,  (name, phone, year)  )  connection.commit()  connection.close()  def print\_list(staff: t.List[t.Dict[str, t.Any]]) -> None:  """  Функция выводит на экран список всех существующих записей  """  if staff:  line = '+-{}-+-{}-+-{}-+-{}-+'.format(  '-' \* 4,  '-' \* 30,  '-' \* 20,  '-' \* 8  )  print(line)  print(  '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(  "№",  "Ф.И.О.",  "Номер телефона",  "Год"  )  )  print(line)  for idx, worker in enumerate(staff, 1):  print(  '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(  idx,  worker.get('name', ''),  worker.get('phone\_number', ''),  worker.get('year', 0)  )  )  print(line)  else:  print("Список работников пуст.")  def create\_db(database\_path: Path) -> None:  """  Создание базы данных  """  connection = sqlite3.connect(database\_path)  cursor = connection.cursor()  cursor.execute(  """  CREATE TABLE IF NOT EXISTS workers (  worker\_id INTEGER PRIMARY KEY AUTOINCREMENT,  worker\_name TEXT NOT NULL,  phone\_number TEXT NOT NULL,  worker\_year INTEGER NOT NULL  )  """  )  connection.close()  def select\_all(database\_path: Path) -> t.List[t.Dict[str, t.Any]]:  """  Выбор всех записей из базы данных  """  connection = sqlite3.connect(database\_path)  cursor = connection.cursor()  cursor.execute(  """  SELECT workers.worker\_name, workers.phone\_number, workers.worker\_year  FROM workers  """  )  rows = cursor.fetchall()  connection.close()  return [  {  "name": row[0],  "phone\_number": row[1],  "year": row[2],  }  for row in rows  ]  def select\_by\_period(  database\_path: Path,  period: int) -> t.List[t.Dict[str, t.Any]]:  """  Выборка по периоду  """  connection = sqlite3.connect(database\_path)  cursor = connection.cursor()  cursor.execute(  """  SELECT workers.worker\_name, workers.worker\_phone\_number, workers.worker\_year  FROM workers  WHERE (strftime('%Y', date('now')) - workers.worker\_year) >= ?  """,  (period,)  )  rows = cursor.fetchall()  connection.close()  return [  {  "name": row[0],  "phone\_number": row[1],  "year": row[2],  }  for row in rows  ]  def main(command\_line=None):  file\_parser = argparse.ArgumentParser(add\_help=False)  file\_parser.add\_argument(  "--db",  action="store",  required=False,  default=str(Path.home() / "workers.db"),  help="Название файла базы даанных"  )  parser = argparse.ArgumentParser("workers")  parser.add\_argument(  "--version",  action="version",  version="%(prog)s 0.1.0"  )  subparsers = parser.add\_subparsers(dest="command")  add = subparsers.add\_parser(  "add",  parents=[file\_parser],  help="Add a new worker"  )  add.add\_argument(  "-n",  "--name",  action="store",  required=True,  help="Имя работника"  )  add.add\_argument(  "-p",  "--phone",  action="store",  help="Номер телефона работника"  )  add.add\_argument(  "-d",  "--date",  action="store",  required=True,  help="Дата нанятия"  )  \_ = subparsers.add\_parser(  "display",  parents=[file\_parser],  help="Вывести на экран всех работников"  )  select = subparsers.add\_parser(  "select",  parents=[file\_parser],  help="Выборка работников"  )  select.add\_argument(  "-p",  "--period",  action="store",  type=int,  required=True,  help="Требуемый период"  )  args = parser.parse\_args(command\_line)  db\_path = Path(args.db)  create\_db(db\_path)  if args.command == "add":  add\_worker(db\_path, args.name, args.phone, args.date)  elif args.command == "display":  print\_list(select\_all(db\_path))  elif args.command == "select":  print\_list(select\_by\_period(db\_path, args.period))  pass  if \_\_name\_\_ == "\_\_main\_\_":  """  Основная программа  """  main() |

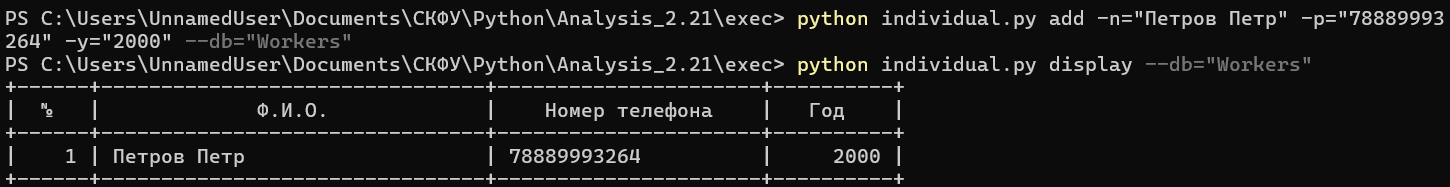


Рисунок 2.1 – Создание базы данных, добавление первой записи и вывод всех записей на экран

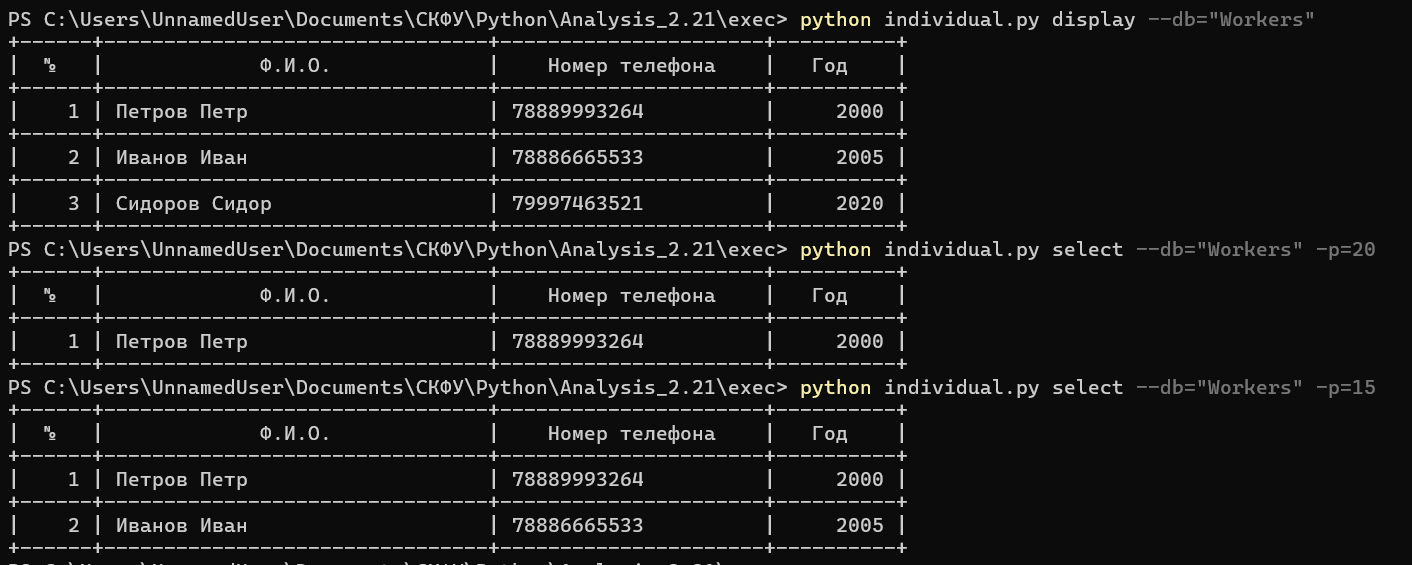


Рисунок 2.2 – Выборка записей из базы данных

Листинг 3 – Код программы dif.py

|  |
| --- |
|  |

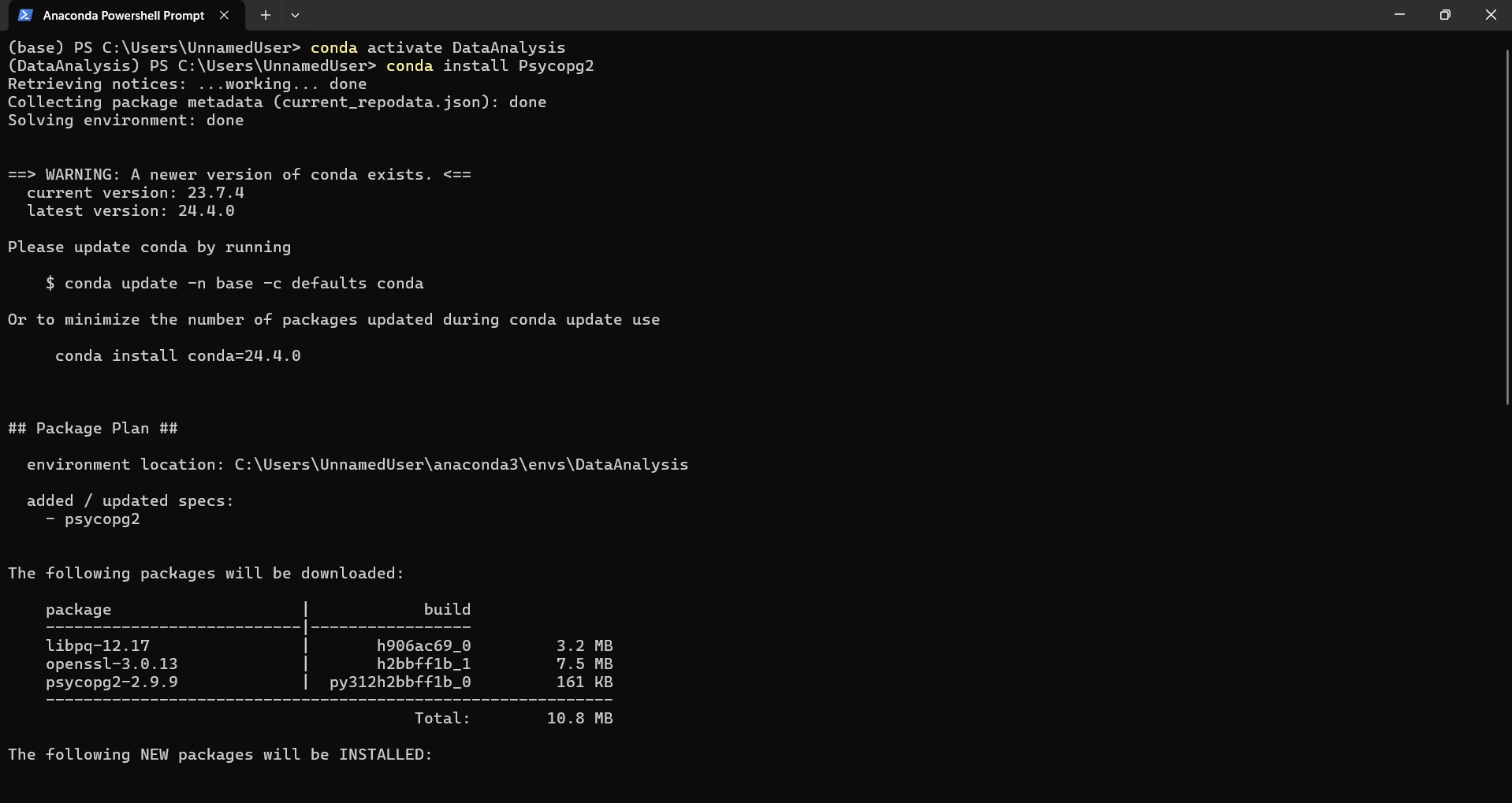


Рисунок 3.1 - Установка пакета PostgreSQL

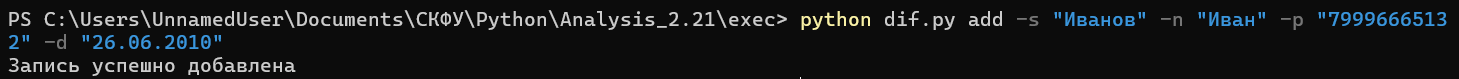


Рисунок 3.2 – Добавление новой записи в таблицу

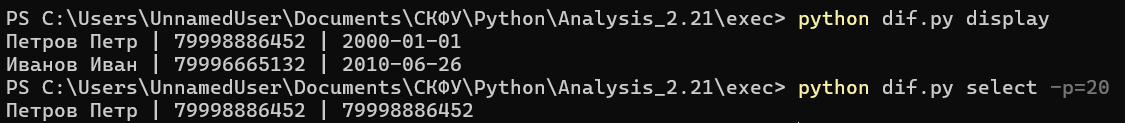


Рисунок 3.3 – Вывод списка сотрудников и выборка

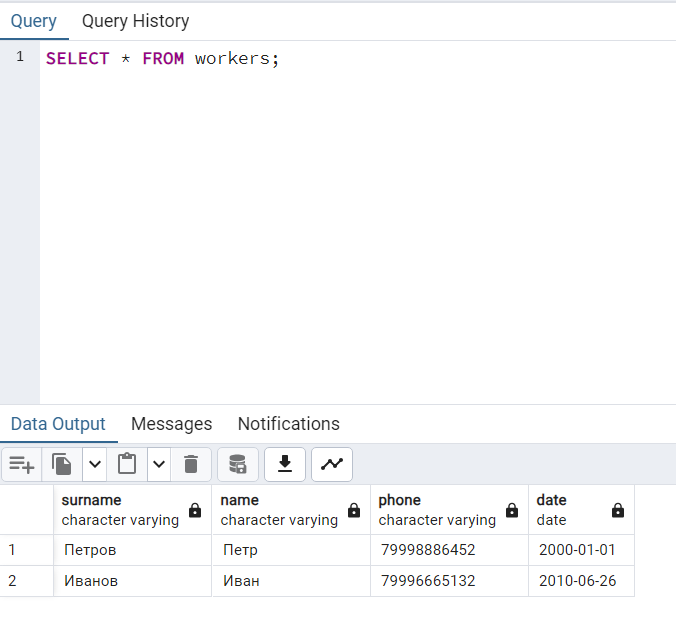


Рисунок 3.4 – Данные в таблице PostgreSQL

Контрольные вопросы

**Выводы**: В процессе выполнения лабораторной работы были изучены методы взаимодействия баз данных SQLite3 с помощью языка программирования Python. Были проработаны примеры лабораторной работы, выполнено индивидуальное задание и задание повышенной сложности.