УРОК 37. РАБОТА С БАЗАМИ ДАННЫХ ИЗ ИНТЕРФЕЙСА PYTHON

ПОДКЛЮЧЕНИЕ К БАЗЕ ДАННЫХ

Библиотека sqlite3 (или соответствующая библиотека для другой БД) предоставляет возможность взаимодействия с базой данных SQLite из Python. Она включена в стандартную библиотеку Python, поэтому нет необходимости устанавливать дополнительные модули. Для работы с другими базами данных, необходимо установить соответствующую библиотеку, например, psycopg2 для PostgreSQL или mysql-connector-python для MySQL.

Нам понадобится подключаться к СУБД MySQL, поэтому придется установить дополнительный модуль командой:

```
Python
pip install mysql-connector-python
```

Для подключения к базе данных нужно импортировать установленный модуль, создать переменную-словарь для описания конфигурации подключения, затем установить соединение:

```
Python
import mysql.connector

dbconfig = {'host': 'ich-db.ccegls0svc9m.eu-central-1.rds.amazonaws.com',
      'user': 'ich1',
      'password': 'password',
      'database': 'example'}

connection = mysql.connector.connect(**dbconfig)
```

РАБОТА С БАЗОЙ ДАННЫХ

Курсор (cursor) является объектом, с помощью которого выполняются операции с базой данных.

Он представляет собой интерфейс для отправки запросов и получения результатов. Курсор создается из соединения (connection) с помощью метода cursor():

```python
cursor = connection.cursor()
```

По завершению работы с БД курсор и соединение нужно закрывать, чтобы избежать непредвиденных ошибок и снижения производительности :

```python
cursor.close()
connection.close()
```

Метод execute используется для выполнения команд SQL.

Он принимает строку с SQL-запросом в качестве аргумента.
Пример использования:

```python
cursor.execute("SELECT * FROM users")
```

При работе с базой данных может потребоваться извлечение данных из результата запроса. Для этого можно использовать методы fetchone, fetchall или fetchmany.
Например:

```python
cursor.execute("SELECT * FROM users")
result = cursor.fetchall()

for row in result:
  print(row)
```

Метод fetchall возвращает все строки результата в виде списка, где каждая строка представлена кортежем или списком.

Метод execute может также принимать параметры, которые могут быть переданы в SQL-запрос с использованием специального синтаксиса.

Например:

```python
name = "John"
age = 25
cursor.execute("INSERT INTO users (name, age) VALUES (%s, %s)", (name, age))
```

Метод executemany позволяет выполнить несколько операций с разными значениями параметров.

Он принимает два аргумента: строку с SQL-запросом и последовательность кортежей или списков, содержащих значения параметров. Пример:


```python
data = [("John", 25), ("Alice", 30), ("Bob", 35)]
cursor.executemany("INSERT INTO users (name, age) VALUES (%s, %s)", data)
```

В данном примере будет выполнено несколько операций INSERT для добавления пользователей в таблицу users.

📎 **Метод commit применяется после выполнения операций, которые изменяют данные в базе данных, чтобы сохранить изменения.**

Он вызывается на объекте соединения (connection).
Пример:

```python
connection.commit()
```

После вызова commit, изменения будут фиксированы в базе данных.

В библиотеке mysql.connector есть возможность использовать параметры с именованными заполнителями в SQL-запросах. Вместо плейсхолдеров %s можно использовать имена, которые будут заменены соответствующими значениями при выполнении запроса.

```python
data = {"name": "John", "age": 25}
cursor.execute("INSERT INTO users (name, age) VALUES (%(name)s, %(age)s)", data)
```

Работа с базой данных может быть выполнена в рамках транзакции. Транзакция представляет собой логическую группу операций, которые должны быть выполнены все или ни одна. Для выполнения транзакций можно использовать методы commit и rollback. Если все операции в транзакции прошли успешно, вызов commit фиксирует

изменения. В случае возникновения ошибки или отката транзакции вызывается метод rollback, который отменяет все выполненные операции.

```python
try:
  connection = mysql.connector.connect(**dbconfig)
  cursor = connection.cursor()

  # Выполнение операций в рамках транзакции

  connection.commit()
except:
  connection.rollback()
Finally:
  cursor.close()
  connection.close()
```

Для обработки ошибок, связанных с базой данных, также можно использовать блок try-except. В случае возникновения исключения, можно обработать ошибку.

```python
try:
  cursor.execute("INSERT INTO users (name, age) VALUES (?, ?)", ("John", 25))
  connection.commit()
except sqlite3.Error as e:
  print("Ошибка базы данных:", e)
  connection.rollback()
```

В блоке except можно указать различные действия для обработки ошибок, например, вывод сообщения об ошибке или запись в журнал.

При работе с большими объемами данных или сложными запросами может потребоваться разделение логики работы с базой данных на несколько функций или классов. Это позволяет улучшить читаемость и обеспечить модульность кода.


```python
def insert_user(cursor, name, age):
  cursor.execute("INSERT INTO users (name, age) VALUES (%s, %s)", (name, age))

def get_users(cursor):
  cursor.execute("SELECT * FROM users")
  return cursor.fetchall()
```

В данном примере выделены две функции: insert_user для добавления пользователя и get_users для получения списка пользователей.

ПРАКТИЧЕСКАЯ РАБОТА

1. Написать программу, которая запрашивает у пользователя возраст и выводит все строки таблицы users, где возраст больше указанного.
2. В базе данных ich_edit три таблицы. Users с полями (id, name, age), Products с полями (pid, prod, quantity) и Sales с полями (sid, id, pid).
   Программа должна вывести все покупки каждого пользователя.

ПОЛЕЗНЫЕ МАТЕРИАЛЫ

1. Python, введение в БД
2. Руководство по работе с базами данных в Python