Лабораторная работа №13 Работа с PostgreSQL из Python

Цель работы:

Освоить базовые операции подключения к PostgreSQL и выполнения SQL-запросов из Python с использованием библиотеки psycopg2.

Программное обеспечение: Python 3.x, psycopg2, PostgreSQL
Время: 2 академических часа

Теоретическая часть

1. Установка библиотеки psycopg2

```
python3 -m venv venv
source venv/bin/activate
pip install psycopg2
```

2. Основные классы и методы psycopg2

- **psycopg2.connect()** - подключение к базе данных
- **connection.cursor()** - создание курсора
- **cursor.execute()** - выполнение SQL-запроса
- **cursor.fetchone()** - получение одной строки результата
- **cursor.fetchall()** - получение всех строк результата
- **connection.commit()** - подтверждение транзакции
- **connection.rollback()** - откат транзакции
- **cursor.close(), connection.close()** - закрытие соединения

Подготовка базы данных

SQL-скрипт для создания тестовой базы

```
-- Создайте базу данных (выполните в pgAdmin или DBeaver)
CREATE DATABASE students_db;
```

```sql
-- Подключитесь к базе данных students_db и выполните:
CREATE TABLE students (
    id SERIAL PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    age INTEGER,
    group_name VARCHAR(20),
    average_grade DECIMAL(3,2)
);

CREATE TABLE subjects (
    id SERIAL PRIMARY KEY,
    subject_name VARCHAR(100) NOT NULL,
    hours INTEGER
);

CREATE TABLE grades (
    id SERIAL PRIMARY KEY,
    student_id INTEGER REFERENCES students(id),
    subject_id INTEGER REFERENCES subjects(id),
    grade INTEGER CHECK (grade BETWEEN 1 AND 5),
    exam_date DATE DEFAULT CURRENT_DATE
);

-- Заполнение таблиц тестовыми данными
INSERT INTO students (first_name, last_name, age, group_name,
average_grade) VALUES
('Иван', 'Иванов', 20, 'ИТ-101', 4.2),
('Мария', 'Петрова', 21, 'ИТ-101', 4.5),
('Алексей', 'Сидоров', 19, 'ИТ-102', 3.8),
('Ольга', 'Кузнецова', 20, 'ИТ-102', 4.7),
('Дмитрий', 'Попов', 22, 'ИТ-103', 4.0);

INSERT INTO subjects (subject_name, hours) VALUES
('Математика', 120),
('Программирование', 150),
('Базы данных', 90),
('Физика', 100);

INSERT INTO grades (student_id, subject_id, grade, exam_date) VALUES
```

```
(1, 1, 5, '2024-01-15'),
(1, 2, 4, '2024-01-20'),
(2, 1, 5, '2024-01-15'),
(2, 2, 5, '2024-01-20'),
(3, 1, 3, '2024-01-15'),
(3, 3, 4, '2024-02-10'),
(4, 2, 5, '2024-01-20'),
(4, 3, 5, '2024-02-10'),
(5, 4, 4, '2024-02-05');
```

Практические задания

Задание 1: Базовое подключение к базе данных

Создайте файл `database_connection.py`:

```
import psycopg2
from psycopg2 import Error

try:
    # Подключение к базе данных
    connection = psycopg2.connect(
        host="192.168.1.14",
        database="postgres",
        user="postgres",
        password="123",
        options="-c search_path=my_schema" # my_schema замените на
название вашей схемы
    )

    print("Успешное подключение к базе данных!")

    # Создание курсора
    cursor = connection.cursor()

    # Выполнение простого запроса
    cursor.execute("SELECT version();")

    # Получение результата
    db_version = cursor.fetchone()
```

```
    print(f"Версия PostgreSQL: {db_version[0]}")

except Error as e:
    print(f"Ошибка при подключении: {e}")

finally:
    # Закрытие соединения
    if connection:
        cursor.close()
        connection.close()
        print("Соединение закрыто")
```

Задание 2: Чтение данных (SELECT)

Создайте файл `select_data.py`:

```
import psycopg2
from psycopg2 import Error

try:
    connection = psycopg2.connect(
        host="192.168.1.14",
        database="postgres",
        user="postgres",
        password="123",
        options="-c search_path=my_schema"
    )

    cursor = connection.cursor()

    # 1. Получить всех студентов
    print("=== Все студенты ===")
    cursor.execute("SELECT * FROM students ORDER BY id")
    students = cursor.fetchall()

    for student in students:
        print(f"ID: {student[0]}, Имя: {student[1]} {student[2]}, "
              f"Возраст: {student[3]}, Группа: {student[4]}, Средний
балл: {student[5]}")

    # 2. Получить студентов с высоким средним баллом
```

```
    print("\n=== Студенты со средним баллом > 4.0 ===")
    cursor.execute("""
        SELECT first_name, last_name, average_grade
        FROM students
        WHERE average_grade > 4.0
        ORDER BY average_grade DESC
    """)

    high_gpa_students = cursor.fetchall()
    for student in high_gpa_students:
        print(f"{student[0]} {student[1]}: {student[2]}")

    # 3. Получить количество студентов в каждой группе
    print("\n=== Количество студентов по группам ===")
    cursor.execute("""
        SELECT group_name, COUNT(*) as student_count
        FROM students
        GROUP BY group_name
        ORDER BY group_name
    """)

    group_stats = cursor.fetchall()
    for group in group_stats:
        print(f"Группа {group[0]}: {group[1]} студентов")

except Error as e:
    print(f"Ошибка: {e}")

finally:
    if connection:
        cursor.close()
        connection.close()
```

Задание 3: Добавление данных (INSERT)

Создайте файл `insert_data.py`:

```
import psycopg2
from psycopg2 import Error

try:
```

```
    connection = psycopg2.connect(
        host="192.168.1.14",
        database="postgres",
        user="postgres",
        password="123",
        options="-c search_path=my_schema"
    )

    cursor = connection.cursor()

    # Данные нового студента
    new_student = ('Екатерина', 'Смирнова', 21, 'ИТ-103', 4.3)

    # SQL запрос с параметрами
    insert_query = """
        INSERT INTO students (first_name, last_name, age, group_name,
average_grade)
        VALUES (%s, %s, %s, %s, %s)
        RETURNING id
    """

    cursor.execute(insert_query, new_student)

    # Получаем ID нового студента
    student_id = cursor.fetchone()[0]

    # Подтверждаем изменения
    connection.commit()

    print(f"Добавлен новый студент с ID: {student_id}")
    print(f"Данные: {new_student}")

    # Проверяем, что студент добавлен
    cursor.execute("SELECT COUNT(*) FROM students")
    total_students = cursor.fetchone()[0]
    print(f"Всего студентов в базе: {total_students}")

    # Список новых студентов
    new_students = [
        ('Андрей', 'Васильев', 20, 'ИТ-101', 4.1),
        ('Наталья', 'Федорова', 19, 'ИТ-102', 4.6),
```

```
        ('Сергей', 'Николаев', 22, 'ИТ-103', 3.9)
    ]

    insert_query = """
        INSERT INTO students (first_name, last_name, age, group_name,
average_grade)
        VALUES (%s, %s, %s, %s, %s)
    """

    cursor.executemany(insert_query, new_students)
    connection.commit()

    print(f"Добавлено {len(new_students)} новых студентов")

except Error as e:
    print(f"Ошибка: {e}")
    if connection:
        connection.rollback()

finally:
    if connection:
        cursor.close()
        connection.close()
```

Задание 4: Обновление данных (UPDATE)

Создайте файл `update_data.py`:

```
import psycopg2
from psycopg2 import Error

try:
    connection = psycopg2.connect(
        host="192.168.1.14",
        database="postgres",
        user="postgres",
        password="123",
        options="-c search_path=my_schema"
    )

    cursor = connection.cursor()
```

```
    # Показать текущие данные
    print("=== Текущие данные студента с ID=3 ===")
    cursor.execute("SELECT * FROM students WHERE id = 3")
    student = cursor.fetchone()
    print(f"До обновления: {student}")

    # Обновление данных
    update_query = """
        UPDATE students
        SET average_grade = %s
        WHERE id = %s
    """

    new_grade = 4.2
    student_id = 3

    cursor.execute(update_query, (new_grade, student_id))
    connection.commit()

    print(f"Обновлен средний балл студента ID={student_id} на
{new_grade}")

    # Показать обновленные данные
    cursor.execute("SELECT * FROM students WHERE id = 3")
    updated_student = cursor.fetchone()
    print(f"После обновления: {updated_student}")

    # Увеличить всем студентам средний балл на 0.1
    update_query = "UPDATE students SET average_grade = average_grade +
0.1"

    cursor.execute(update_query)
    connection.commit()

    # Показать статистику
    cursor.execute("SELECT COUNT(*) FROM students")
    total = cursor.fetchone()[0]

    cursor.execute("SELECT AVG(average_grade) FROM students")
    avg_grade = cursor.fetchone()[0]
```

```
    print(f"Обновлены оценки {total} студентов")
    print(f"Новый средний балл по всем студентам: {avg_grade:.2f}")

except Error as e:
    print(f"Ошибка: {e}")
    if connection:
        connection.rollback()

finally:
    if connection:
        cursor.close()
        connection.close()
```

Задание 5: Работа с транзакциями

Создайте файл `transactions.py`:

```
import psycopg2
from psycopg2 import Error

try:
    connection = psycopg2.connect(
        host="192.168.1.14",
        database="postgres",
        user="postgres",
        password="123",
        options="-c search_path=my_schema"
    )

    cursor = connection.cursor()

    print("=== Перевод студента в другую группу ===")

    # Начало транзакции
    connection.autocommit = False

    # Показать текущую группу студента
    student_id = 1
    cursor.execute("SELECT first_name, last_name, group_name FROM
students WHERE id = %s", (student_id,))
```

```
    student = cursor.fetchone()
    print(f"Текущие данные: {student[0]} {student[1]} - группа
{student[2]}")

    # Обновить группу
    new_group = 'ИТ-104'
    update_query = "UPDATE students SET group_name = %s WHERE id = %s"
    cursor.execute(update_query, (new_group, student_id))

    # Проверить результат
    cursor.execute("SELECT group_name FROM students WHERE id = %s",
(student_id,))
    updated_group = cursor.fetchone()[0]

    # Подтвердить или откатить транзакцию
    if updated_group == new_group:
        connection.commit()
        print(f"Транзакция подтверждена. Студент переведен в группу
{new_group}")
    else:
        connection.rollback()
        print("Транзакция откачена. Возникла ошибка при обновлении")

    # Безопасный вариант выполнения транзакции
    try:
        # Несколько операций в одной транзакции
        cursor.execute("UPDATE students SET age = age + 1 WHERE id = 1")
        cursor.execute("UPDATE students SET average_grade =
average_grade + 0.5 WHERE id = 1")

        # Проверим, что средний балл не превышает 5.0
        cursor.execute("SELECT average_grade FROM students WHERE id =
1")
        new_grade = cursor.fetchone()[0]

        if new_grade > 5.0:
            raise ValueError("Средний балл не может превышать 5.0")

        connection.commit()
        print("Транзакция успешно завершена")
```

```
    except Exception as e:
        connection.rollback()
        print(f"Ошибка в транзакции: {e}")
        print("Изменения откачены")

except Error as e:
    print(f"Ошибка подключения: {e}")

finally:
    if connection:
        connection.autocommit = True
        cursor.close()
        connection.close()
```

Задание 6: Работа с JOIN-запросами

Создайте файл `join_queries.py`, который выполняет следующие запросы с объединением таблиц:

1. Получить список всех студентов с указанием их оценок по предметам (включая название предмета и дату экзамена).
2. Найти средний балл каждого студента по всем предметам (используйте `GROUP BY` и `JOIN`).
3. Вывести студентов, которые не сдали хотя бы один экзамен (оценка < 3).

Пример вывода для первого запроса:

```
Иван Иванов - Математика: 5 (2024-01-15)
Иван Иванов - Программирование: 4 (2024-01-20)
...
```

Задание 7: Использование агрегатных функций и GROUP BY

Создайте файл `aggregate_functions.py`, который вычисляет следующую статистику:

1. Средний балл по каждому предмету.
2. Количество студентов в каждой группе, у которых средний балл выше 4.0.
3. Предмет с наибольшим количеством проведённых экзаменов.
4. Самый молодой и самый старший студент в каждой группе.

Требование: Используйте агрегатные функции (`AVG` , `COUNT` , `MAX` , `MIN`) и группировку.

Задание 8: Параметризованные запросы с вводом от пользователя

Создайте файл `user_input_queries.py` , который:

1. Запрашивает у пользователя номер группы (например, `ИТ-101`).
2. Выводит всех студентов этой группы.
3. Запрашивает у пользователя минимальный средний балл.
4. Выводит студентов, у которых средний балл выше введённого значения.

Дополнительно: Реализуйте обработку ошибок при вводе некорректных данных (например, несуществующей группы).

Задание 9: Создание простого консольного приложения

Создайте файл `student_manager.py` , который предоставляет текстовое меню:

```
1. Показать всех студентов
2. Добавить нового студента
3. Обновить данные студента
4. Удалить студента
5. Найти студента по фамилии
6. Выход
```

Реализуйте каждую операцию, используя соответствующие SQL-запросы. При удалении или обновлении запрашивайте ID студента. Используйте параметризованные запросы для безопасности.

Контрольные вопросы

1. Что такое psycopg2 и для чего он используется?
2. В чем разница между psycopg2 и psycopg2-binary?
3. Какие основные методы используются для работы с базой данных в psycopg2?
4. Почему важно закрывать соединение с базой данных после работы?

Требования к отчету

1. **Все файлы Python** с выполненными заданиями
2. **Примеры выполнения** (скриншоты или текстовый вывод)
3. **Ответы на контрольные вопросы**
4. **Описание проблем**, возникших при выполнении работы