Защищено:
Гапанюк Ю.Е.

"__"_____2025 г.

Демонстрация:
Гапанюк Ю.Е.

"__"_____2025 г.

Отчет по лабораторной работе № 5 по курсу Парадигмы и конструкции языков программирования ГУИМЦ

Тема работы: "Рубежный контроль представляет собой разработку программы на языке Python.""

5
(количество листов)

ИСПОЛНИТЕЛЬ:

студент группы ИУ5Ц-54Б _____
(подпись)

Щетинин Д.С.

"__"_____2025 г.

Москва, МГТУ - 2025 г.

1. Тема и задание для выполнения рубежного контроля.

Условия рубежного контроля №1 по курсу ПиК ЯП
Тема работы: Рубежный контроль представляет собой разработку программы на языке Python, которая выполняет следующие действия:

1) Необходимо создать два класса данных в соответствии с Вашим вариантом предметной области, которые связаны отношениями один-ко-многим и многие-комногим. Пример классов данных для предметной области Сотрудник-Отдел:

1. Класс «Сотрудник», содержащий поля:
   - ID записи о сотруднике;
   - Фамилия сотрудника;
   - Зарплата (количественный признак);
   - ID записи об отделе. (для реализации связи один-ко-многим)
2. Класс «Отдел», содержащий поля:
   - ID записи об отделе;
   - Наименование отдела.
3. (Для реализации связи многие-ко-многим) Класс «Сотрудники отдела», содержащий поля:
   - ID записи о сотруднике;
   - ID записи об отделе.

2) Необходимо создать списки объектов классов, содержащих тестовые данные (3-5 записей), таким образом, чтобы первичные и вторичные ключи соответствующих записей были связаны по идентификаторам.

3) Необходимо разработать запросы в соответствии с Вашим вариантом. Запросы сформулированы в терминах классов «Сотрудник» и «Отдел», которые используются в примере. Вам нужно перенести эти требования в Ваш вариант предметной области. При разработке запросов необходимо по возможности использовать функциональные возможности языка Python (list/dict comprehensions, функции высших порядков).

Для реализации запроса №2 введите в класс, находящийся на стороне связи «много», произвольный количественный признак, например, «зарплата сотрудника». Результатом рубежного контроля является документ в формате PDF, который содержит текст программы и результаты ее выполнения.

Вариант Д.

1. «Отдел» и «Сотрудник» связаны соотношением один-ко-многим. Выведите список всех сотрудников, у которых фамилия заканчивается на «ов», и названия их отделов.
2. «Отдел» и «Сотрудник» связаны соотношением один-ко-многим. Выведите список отделов со средней зарплатой сотрудников в каждом отделе, отсортированный по средней зарплате (отдельной функции вычисления среднего значения в Python нет, нужно использовать комбинацию функций вычисления суммы и количества значений).
3. «Отдел» и «Сотрудник» связаны соотношением многие-ко-многим. Выведите список всех отделов, у которых название начинается с буквы «А», и список работающих в них сотрудников.

Таблица 1. Варианты предметной области

№ варианта	Класс 1	Класс 2
34	Хранимая процедура	База данных

2. Листинг программы

```
"""
Рубежный контроль №1 по курсу "Программирование и компьютеры"
Вариант 34: Хранимая процедура - База данных
Вариант запросов: Д
"""

class StoredProcedure:
    """Класс "Хранимая процедура" (аналог "Сотрудника")"""
    def __init__(self, id, name, execution_time, db_id):
        self.id = id
        self.name = name
        self.execution_time = execution_time  # время выполнения в мс
        self.db_id = db_id  # для связи один-ко-многим

    def __str__(self):
        return f"{self.name} ({self.execution_time} мс)"


class Database:
    """Класс "База данных" (аналог "Отдела")"""
    def __init__(self, id, name):
        self.id = id
        self.name = name

    def __str__(self):
        return self.name


class ProcedureDatabase:
    """Класс для связи многие-ко-многим"""
    def __init__(self, proc_id, db_id):
        self.proc_id = proc_id
        self.db_id = db_id


def main():
    # Тестовые данные
    databases = [
        Database(1, "AccountingDB"),
        Database(2, "AnalyticsDB"),
        Database(3, "AdminDB"),
        Database(4, "ArchiveDB")
    ]

    procedures = [
        StoredProcedure(1, "calculate_salary", 150, 1),
        StoredProcedure(2, "generate_report", 230, 1),
        StoredProcedure(3, "data_aggregate", 180, 2),  # Изменил название для демонстрации
        StoredProcedure(4, "predict_sales", 320, 2),
        StoredProcedure(5, "backup_data", 420, 3),
        StoredProcedure(6, "validate_data", 380, 3),   # Изменил название для демонстрации
        StoredProcedure(7, "optimize_tables", 270, 4)
    ]

    procedure_databases = [
```

```
    ProcedureDatabase(1, 1),
    ProcedureDatabase(2, 1),
    ProcedureDatabase(3, 2),
    ProcedureDatabase(4, 2),
    ProcedureDatabase(5, 3),
    ProcedureDatabase(6, 3),
    ProcedureDatabase(7, 4),
    ProcedureDatabase(3, 3),  # для связи многие-ко-многим
    ProcedureDatabase(4, 4)   # для связи многие-ко-многим
]

print("="*50)
print("ЗАПРОС 1:")
print("Список процедур с названиями, оканчивающимися на 'ate', и их БД:")
print("="*50)

# Ищем процедуры, оканчивающиеся на "ate"
query1_results = []
for proc in procedures:
    if proc.name.endswith('ate'):
        db = next((d for d in databases if d.id == proc.db_id), None)
        query1_results.append((proc, db))

if not query1_results:
    print("Нет процедур, оканчивающихся на 'ate'")
else:
    for proc, db in query1_results:
        print(f"{proc.name:20} | {db.name if db else 'Нет БД'}")

# Остальные запросы остаются без изменений...
print("\n" + "="*50)
print("ЗАПРОС 2:")
print("Среднее время выполнения процедур по БД (отсортировано):")
print("="*50)
db_stats = {}
for proc in procedures:
    if proc.db_id not in db_stats:
        db_stats[proc.db_id] = {'total': 0, 'count': 0}
    db_stats[proc.db_id]['total'] += proc.execution_time
    db_stats[proc.db_id]['count'] += 1

avg_times = [
    (next(d.name for d in databases if d.id == db_id),
     stats['total']/stats['count'])
    for db_id, stats in db_stats.items()
]

for db_name, avg_time in sorted(avg_times, key=lambda x: x[1]):
    print(f"{db_name:20} | {avg_time:.2f} мс")

print("\n" + "="*50)
print("ЗАПРОС 3:")
print("БД, названия которых начинаются на 'A', и их процедуры:")
print("="*50)
a_dbs = [d for d in databases if d.name.startswith('A')]
for db in a_dbs:
    print(f"\nБД: {db.name}")
    proc_ids = [pd.proc_id for pd in procedure_databases if pd.db_id == db.id]
    db_procs = [p for p in procedures if p.id in proc_ids]
```

```
        for proc in db_procs:
            print(f"  - {proc}")


if __name__ == "__main__":
    main()
```

3. Результаты работы программы

```
C:\Program Files\Python313\python.exe" C:\Users\Денис\Desktop\РК1\RK1.py
==================================================
ЗАПРОС 1:
Список процедур с названиями, оканчивающимися на 'ate', и их БД:
==================================================
data_aggregate       | AnalyticsDB

==================================================
ЗАПРОС 2:
Среднее время выполнения процедур по БД (отсортировано):
==================================================
AccountingDB         | 190.00 мс
AnalyticsDB          | 250.00 мс
ArchiveDB            | 270.00 мс
AdminDB              | 400.00 мс

==================================================
ЗАПРОС 3:
БД, названия которых начинаются на 'A', и их процедуры:
==================================================

БД: AccountingDB
  - calculate_salary (150 мс)
  - generate_report (230 мс)

БД: AnalyticsDB
  - data_aggregate (180 мс)
  - predict_sales (320 мс)

БД: AdminDB
  - data_aggregate (180 мс)
  - backup_data (420 мс)
  - validate_data (380 мс)

БД: ArchiveDB
  - predict_sales (320 мс)
  - optimize_tables (270 мс)
```