Защищено:
Гапанюк Ю.Е.

"__"_____**2025** г.

Демонстрация:
Тошниёзов Ж. Г.
"__"_____**2025** г.

Отчет по рубежному контролю № 2 по курсу Парадигмы и конструкции языков программирования

Вариант №32

5
(количество листов)

студент группы ИУ5Ц-54Б _____
(подпись)

Тошниёзов Ж. Г. "__"_____**2025** г.

Москва, МГТУ - **2025**

1. **Описание задания**

32	Колонка данных	Таблица данных

Вариант предметной области:
Рубежный контроль представляет собой разработку тестов на языке Python. 1) Проведите рефакторинг текста программы рубежного контроля №1 таким образом, чтобы он был пригоден для модульного тестирования. 2) Для текста программы рубежного контроля №1 создайте модульные тесты с применением TDD - фреймворка (3 теста).

2. **Листинг программы**

```
# rk1_refactored.py
class DataColumn:
    """Класс Колонка данных (аналог Сотрудника)"""
    def __init__(self, column_id, name, data_size, table_id):
        self.column_id = column_id
        self.name = name
        self.data_size = data_size  # количественный признак (размер данных в МБ)
        self.table_id = table_id

    def __repr__(self):
        return f"DataColumn({self.column_id}, '{self.name}', {self.data_size},
{self.table_id})"

class DataTable:
    """Класс Таблица данных (аналог Отдела)"""
    def __init__(self, table_id, table_name):
        self.table_id = table_id
        self.table_name = table_name

    def __repr__(self):
        return f"DataTable({self.table_id}, '{self.table_name}')"
```

```
class ColumnTable:
    """Класс для связи многие-ко-многим"""
    def __init__(self, column_id, table_id):
        self.column_id = column_id
        self.table_id = table_id

    def __repr__(self):
        return f"ColumnTable({self.column_id}, {self.table_id})"
```

```
# Модуль данных
class DataModule:
    """Модуль для работы с данными"""
    @staticmethod
    def create_test_data():
        """Создание тестовых данных"""
        tables = [
            DataTable(1, "Users"),
            DataTable(2, "Products"),
            DataTable(3, "Аналитика"),
```

```
            DataTable(4, "Orders"),
            DataTable(5, "Accounts")
        ]

        columns = [
            DataColumn(1, "user_id", 50, 1),
            DataColumn(2, "username", 30, 1),
            DataColumn(3, "user_email", 40, 1),
            DataColumn(4, "product_id", 60, 2),
            DataColumn(5, "product_name", 25, 2),
            DataColumn(6, "price", 15, 2),
            DataColumn(7, "sale_date", 20, 4),
            DataColumn(8, "amount", 10, 4),
            DataColumn(9, "account_id", 35, 5),
            DataColumn(10, "balance", 12, 5),
            DataColumn(11, "analytics_id", 45, 3),
            DataColumn(12, "report_name", 22, 3)
        ]

        column_tables = [
            ColumnTable(1, 1),  # user_id в Users
            ColumnTable(2, 1),  # username в Users
            ColumnTable(3, 1),  # user_email в Users
            ColumnTable(4, 2),  # product_id в Products
            ColumnTable(5, 2),  # product_name в Products
            ColumnTable(6, 2),  # price в Products
            ColumnTable(7, 4),  # sale_date в Orders
            ColumnTable(8, 4),  # amount в Orders
            ColumnTable(9, 5),  # account_id в Accounts
            ColumnTable(10, 5), # balance в Accounts
            ColumnTable(11, 3), # analytics_id в Аналитика
            ColumnTable(12, 3), # report_name в Аналитика
            # Дополнительные связи для многие-ко-многим
            ColumnTable(1, 3),  # user_id также в Аналитика
            ColumnTable(4, 3),  # product_id также в Аналитика
            ColumnTable(7, 3),  # sale_date также в Аналитика
        ]

        return tables, columns, column_tables
```

```
# Модуль запросов
class QueryModule:
    """Модуль для выполнения запросов"""

    @staticmethod
    def query_1(tables, columns):
        """Колонки с названием, оканчивающимся на 'id' и их таблицы"""
        result_lc = [
            (column.name, next(t.table_name for t in tables if t.table_id ==
column.table_id))
            for column in columns
            if column.name.lower().endswith('id')
        ]
```

```
        return result_1c

    @staticmethod
    def query_2(tables, columns):
        """Средний размер данных колонок по таблицам"""
        table_stats = {}

        for column in columns:
            if column.table_id not in table_stats:
                table_stats[column.table_id] = {'total_size': 0, 'count': 0}
            table_stats[column.table_id]['total_size'] += column.data_size
            table_stats[column.table_id]['count'] += 1

        result = []
        for table_id, stats in table_stats.items():
            table = next(t for t in tables if t.table_id == table_id)
            avg_size = stats['total_size'] / stats['count']
            result.append((table.table_name, avg_size))

        result.sort(key=lambda x: x[1])
        return result

    @staticmethod
    def query_3(tables, columns, column_tables):
        """Таблицы с названием на 'A' и их колонки (связь многие-ко-многим)"""
        a_tables = [t for t in tables if t.table_name.startswith('A')]

        result = []
        for table in a_tables:
            table_column_ids = [ct.column_id for ct in column_tables if ct.table_id
== table.table_id]
            table_columns = [col for col in columns if col.column_id in
table_column_ids]

            result.append({
                'table': table.table_name,
                'columns': [col.name for col in table_columns]
            })

        return result
```

```
# Основной модуль
def main():
    """Главная функция приложения"""
    print("=== РУБЕЖНЫЙ КОНТРОЛЬ №1 ===")
    print("Вариант 32: Колонка данных - Таблица данных")
    print()

    # Создание тестовых данных
    tables, columns, column_tables = DataModule.create_test_data()
    query_module = QueryModule()

    print("\n" + "="*50)
```

```
    print("ЗАПРОС 1: Колонки с названием, оканчивающимся на 'id', и их таблицы")
    result1 = query_module.query_1(tables, columns)
    for col_name, table_name in result1:
        print(f"  Колонка: {col_name}, Таблица: {table_name}")

    print("\n" + "="*50)
    print("ЗАПРОС 2: Таблицы со средним размером данных колонок")
    result2 = query_module.query_2(tables, columns)
    for table_name, avg_size in result2:
        print(f"  Таблица: {table_name}, Средний размер: {avg_size:.2f} МБ")

    print("\n" + "="*50)
    print("ЗАПРОС 3: Таблицы с названием на 'A' и их колонки (связь многие-ко-
многим)")
    result3 = query_module.query_3(tables, columns, column_tables)
    for item in result3:
        print(f"  Таблица: {item['table']}")
        print(f"    Колонки: {', '.join(item['columns'])}")
```

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

```
# test_rk1.py
import unittest
from rk1_refactored import DataModule, QueryModule, DataColumn, DataTable,
ColumnTable

class TestRK1Module(unittest.TestCase):
    """Тесты для модулей РК1"""

    def setUp(self):
        """Подготовка тестовых данных"""
        self.tables, self.columns, self.column_tables = DataModule.create_test_data()
        self.query_module = QueryModule()

    def test_query_1_returns_correct_columns(self):
        """Тест запроса 1: возвращает колонки с 'id' в конце"""
        result = self.query_module.query_1(self.tables, self.columns)

        # Проверяем количество найденных колонок
        self.assertEqual(len(result), 4)

        # Проверяем конкретные значения
        expected_columns = ['user_id', 'product_id', 'account_id', 'analytics_id']
        result_columns = [col_name for col_name, _ in result]

        self.assertListEqual(sorted(result_columns), sorted(expected_columns))

        # Проверяем, что каждая колонка имеет связанную таблицу
        for col_name, table_name in result:
            self.assertIsNotNone(col_name)
            self.assertIsNotNone(table_name)
            self.assertTrue(col_name.lower().endswith('id'))
```

```
    def test_query_2_calculates_average_correctly(self):
        """Тест запроса 2: корректно вычисляет среднее значение"""
        result = self.query_module.query_2(self.tables, self.columns)

        # Проверяем количество таблиц
        self.assertEqual(len(result), 5)

        # Проверяем сортировку по возрастанию
        sizes = [size for _, size in result]
        self.assertEqual(sizes, sorted(sizes))

        # Проверяем конкретное вычисление для таблицы "Orders"
        orders_result = [item for item in result if item[0] == "Orders"]
        self.assertEqual(len(orders_result), 1)

        # sale_date(20) + amount(10) = 30 / 2 = 15.0
        table_name, avg_size = orders_result[0]
        self.assertEqual(table_name, "Orders")
        self.assertAlmostEqual(avg_size, 15.0)

    def test_query_3_finds_tables_starting_with_A(self):
        """Тест запроса 3: находит таблицы, начинающиеся с 'A'"""
        result = self.query_module.query_3(self.tables, self.columns,
self.column_tables)

        # Проверяем, что найдена только таблица "Аналитика"
        self.assertEqual(len(result), 1)

        table_info = result[0]
        self.assertEqual(table_info['table'], "Аналитика")

        # Проверяем, что колонки найдены
        self.assertGreater(len(table_info['columns']), 0)

        # Проверяем, что все ожидаемые колонки присутствуют
        expected_columns = ['user_id', 'product_id', 'sale_date', 'analytics_id',
'report_name']
        for col in expected_columns:
            self.assertIn(col, table_info['columns'])

        # Проверяем количество колонок
        self.assertEqual(len(table_info['columns']), 5)
```

```
class TestDataClasses(unittest.TestCase):
    """Тесты для классов данных"""

    def test_datacolumn_creation(self):
        """Тест создания объекта DataColumn"""
        column = DataColumn(1, "test_column", 100, 1)
        self.assertEqual(column.column_id, 1)
        self.assertEqual(column.name, "test_column")
        self.assertEqual(column.data_size, 100)
```

```
        self.assertEqual(column.table_id, 1)

    def test_datatable_creation(self):
        """Тест создания объекта DataTable"""
        table = DataTable(1, "TestTable")
        self.assertEqual(table.table_id, 1)
        self.assertEqual(table.table_name, "TestTable")

    def test_columntable_creation(self):
        """Тест создания объекта ColumnTable"""
        link = ColumnTable(1, 2)
        self.assertEqual(link.column_id, 1)
        self.assertEqual(link.table_id, 2)
```

```
if __name__ == '__main__':
    unittest.main()
```

3. **Результаты работы программы**

```
C:\Users\ToshniezovZG\OneDrive\Desktop\Учёба Баумана\3 курс\Парадгимы и\PCPL_lab_2025\rk_2>py rk1_refactored.py
=== РУБЕЖНЫЙ КОНТРОЛЬ №1 ===
Вариант 32: Колонка данных - Таблица данных


==================================================
ЗАПРОС 1: Колонки с названием, оканчивающимся на 'id', и их таблицы
  Колонка: user_id, Таблица: Users
  Колонка: product_id, Таблица: Products
  Колонка: account_id, Таблица: Accounts
  Колонка: analytics_id, Таблица: Аналитика

==================================================
ЗАПРОС 2: Таблицы со средним размером данных колонок
  Таблица: Orders, Средний размер: 15.00 МБ
  Таблица: Accounts, Средний размер: 23.50 МБ
  Таблица: Products, Средний размер: 33.33 МБ
  Таблица: Аналитика, Средний размер: 33.50 МБ
  Таблица: Users, Средний размер: 40.00 МБ

==================================================
ЗАПРОС 3: Таблицы с названием на 'A' и их колонки (связь многие-ко-многим)
  Таблица: Аналитика
    Колонки: user_id, product_id, sale_date, analytics_id, report_name

C:\Users\ToshniezovZG\OneDrive\Desktop\Учёба Баумана\3 курс\Парадгимы и\PCPL_lab_2025\rk_2>py test_rk1.py
......
----------------------------------------------------------------------
Ran 6 tests in 0.001s

OK
```