Министерство транспорта Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Российский университет транспорта» (МИИТ)

ИНСТИТУТ УПРАВЛЕНИЯ И ЦИФРОВЫХ ТЕХНОЛОГИЙ

Кафедра «Цифровые технологии управления транспортными процессами»

**Отчёт по производственной практике**

**(технологической (проектно-технологической) практике)**

на тему

«Разработка программы для получения целочисленных значений объемов грузов, с сохранением сходимости сумм по группам управляющих параметров»

Выполнил:

студент группы УИС-312 Д.Д. Загинайченко

Проверил:

научный руководитель:

старший преподаватель кафедры ЦТУТП О.О. Нуждин

Москва 2023

|  |
| --- |
| УТВЕРЖДАЮ  Профессор кафедры ЦТУТП  \_\_\_\_\_\_\_\_\_\_\_ Э. К. Лецкий  « » 2023 года |

**ЗАДАНИЕ НА ПРАКТИКУ**

**производственную практику**

**(технологическую (проектно-технологическую) практику)**

студента группы УИС-312 Загинайченко Даниила Дмитриевича

*(ФИО студента полностью)*

*наименование объекта (места прохождения) производственной практики:*

*ЦТУТП*

Наименование темы:

Разработка программы для получения целочисленных значений объемов грузов, с сохранением сходимости сумм по группам управляющих параметров.

Необходимо изучить:

1. библиотеку Google OR-Tools, предназначенную для решения задач оптимизации и линейного программирования;
2. открытое программное обеспечение Apache Airflow для создания, выполнения, мониторинга и оркестровки потоков операций;
3. программный продукт VirtualBox, предназначенный для виртуализации операционных систем;
4. работа в виртуальном дистрибутиве Ubuntu Server (GNU/Linux), основанном на Debian GNU/Linux;
5. библиотеку SQLAlchemy для работы c реляционными СУБД;
6. работа с объектно-реляционной СУБД PostgreSQL.

Необходимо самостоятельно выполнить:

1. разработать программу, которая округляет дробные значения, с сохранением сходимости сумм по группам управляющих параметров.

Основные требования к проделываемой работе:

1. программа должен быть написан на языке программирования Python;
2. данные для обработки должны быть получены из таблицы СУБД Postgresql;
3. полученные значения добавляются к исходным данным, как новый столбец;
4. данные из пункта 3 должны быть записаны в таблицу СУБД Postgresql;
5. сумма по управляющим параметрам считается сохраненной, если она находится в пределах ±1.5 по отношению к изначальной;
6. программа должна запускаться в дистрибутиве Ubuntu;
7. организовать работу программы в качестве процесса работающего по расписанию с помощью Apache Airflow;

Источники информации:

1. Google OR-Tools: [сайт]. – Текст. Изображение: электронные. – URL: https://developers.google.com/optimization/introduction?hl=en (последнее обращение 28.08.2023).
2. Apache Airflow: [сайт]. – Текст. Изображение: электронные. – URL: https://airflow.apache.org/docs/apache-airflow/stable/start.html (последнее обращение 28.08.2023).
3. SQLAlchemy: [сайт]. – Текст. Изображение: электронные. – URL: https://docs.sqlalchemy.org/en/20 (последнее обращение 28.08.2023).
4. Ubuntu Server documentation: [сайт]. – Текст. Изображение: электронные. – URL: https://ubuntu.com/server/docs (последнее обращение 28.08.2023).
5. PostgreSQL: [сайт]. – Текст. Изображение: электронные. – URL: https://www.postgresql.org/docs/15/index.html (последнее обращение 28.08.2023).
6. VirtualBox: [сайт]. – Текст. Изображение: электронные. – URL: https://www.virtualbox.org/wiki/Technical\_documentation (последнее обращение 28.08.2023).

Материалы к защите:

1. заполненный, подписанный и утверждённый лист индивидуального задания на производственную практику;
2. пояснительная записка (отчёт) по производственной практике;
3. презентация о проделанной работе.

|  |  |
| --- | --- |
| Студент группы УИС-312 | Д.Д. Загинайченко |
| Руководитель практики от университета  Старший преподаватель кафедры ЦТУТП | О.О. Нуждин |

**СОДЕРЖАНИЕ**

[**ВВЕДЕНИЕ** 4](#_Toc147922243)

[**РАЗДЕЛ 1. ОБОСНОВАНИЕ ВЫБОРА МЕТОДА РЕШЕНИЯ ЗАДАЧИ** 5](#_Toc147922244)

[**РАЗДЕЛ 2. ВЫБОР ИНСТРУМЕНТОВ И ПОДГОТОВКА ИНФРАСТРУКТУРЫ** 6](#_Toc147922245)

[**2.1 ВЫБОР ИНСТРУМЕНТОВ** 6](#_Toc147922246)

[**2.2 ПОДГОТОВКА ИНФРАСТРУКТУРЫ** 8](#_Toc147922247)

[**2.2.1 НАСТРОЙКА СЕРВЕРА** 8](#_Toc147922248)

[**2.2.2 НАСТРОЙКА БАЗЫ ДАННЫХ** 12](#_Toc147922249)

[**2.2.3 УСТАНОВКА APACHE AIRFLOW И ДРУГИХ БИБЛИОТЕК** 17](#_Toc147922250)

[**РАЗДЕЛ 3. РАЗРАБОТКА И ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ** 19](#_Toc147922251)

[**3.1 РАЗРАБОТКА ПРИЛОЖЕНИЯ** 19](#_Toc147922252)

[**3.2 ТЕСТИРОВАНИЕ ПРОГРАММЫ** 22](#_Toc147922253)

[**ЗАКЛЮЧЕНИЕ** 24](#_Toc147922254)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 25](#_Toc147922255)

[**ПРИЛОЖЕНИЕ А. ЛИСТИНГ DAG-СКРИПТА ОСНОВНОЙ ПРОГРАММЫ:** 26](#_Toc147922256)

[**ПРИЛОЖЕНИЕ Б. ЛИСТИНГ ПРОГРАММЫ ГЕНЕРАЦИИ ВВОДНЫХ ДАННЫХ:** 32](#_Toc147922257)

# **ВВЕДЕНИЕ**

Округление дробных значений является важной операцией в множестве областей, включая финансы, инженерные расчеты и научные исследования. Оно часто применяется для приближения чисел к более удобным или стандартным форматам. В рамках управления грузами, особенно в логистике и снабжении, округление стоимостей или количества товаров может иметь критическое значение для эффективного контроля расходов и оптимизации процессов.

Однако при округлении дробных значений грузов возникает важная проблема - как обеспечить сохранение суммарных значений по группам управляющих параметров. Группы управляющих параметров могут включать в себя различные характеристики, такие как типы грузов, место отгрузки и место выгрузки, характерные для перевозки грузов с помощью железнодорожного транспорта. Важно, чтобы округление не приводило к критическому искажению общего веса грузов внутри каждой группы.

Цель работы заключается в разработке модуля, способного проводить округление дробных значений грузов, при этом гарантируя сохранение суммарных значений по группам управляющих параметров. Этот модуль может служить важным инструментом для логистических компаний, предприятий снабжения и других организаций, управляющих грузами, позволяя им более точно и эффективно управлять ресурсами.

В данной работе будет выбран метод округления, представлен алгоритм разработанного модуля и демонстрация работы модуля.

# **РАЗДЕЛ 1. ОБОСНОВАНИЕ ВЫБОРА МЕТОДА РЕШЕНИЯ ЗАДАЧИ**

При решении данной задачи был выбран метод CP-SAT (Constraint Programming with SAT или программирование с ограничениями и удовлетворением булевых формул). Выбор метода обосновывается следующими факторами:

1. Сохранение сумм по группам УП: основным требованием задачи является сохранение сумм по группам управляющих параметров в пределах ±1.5 относительно изначальных значений после округления дробных значений грузов. В методе CP-SAT используются только целочисленные значения, поэтому необходимо проводить округление сумм до ближайших целочисленных значений, что может привести к погрешности в пределах ±0.5. Затем вводятся ограничения неравенствами, где суммы по группам должны быть в пределах отрезка ±1. Это создает область поиска решений, в пределах которой ожидается нахождение решения, гарантируя сохранение сумм в требуемых пределах. Таким образом, требование о сохранении суммы по группам УП в установленных пределах удовлетворенно.
2. Булевые переменные для округления: метод CP-SAT позволяет использовать булевые переменные (0 или 1) для представления округления вверх или вниз каждого значения, обеспечивая точный контроль над округлением.
3. Ограничения на целые числа: метод CP-SAT исключает дробные значения и гарантирует целочисленность, что соответствует требованию округления до целых чисел.
4. Параллельные вычисления: метод CP-SAT можно эффективно решать с использованием многозадачных вычислений, что позволяет учитывать возможность выбора количества логических ядер процессора для параллельных вычислений, обеспечивая эффективное решение задачи.

Таким образом, выбор метода CP-SAT обеспечивает точное и эффективное решение задачи, удовлетворяя всем требованиям.

# **РАЗДЕЛ 2. ВЫБОР ИНСТРУМЕНТОВ И ПОДГОТОВКА ИНФРАСТРУКТУРЫ**

# **ы**

Для разработки программного модуля, соответствующего требованиям, описанным в задании, были выбраны следующие инструменты и технологии:

1. Язык программирования Python: Python был выбран в качестве основного языка программирования для разработки модуля. Python широко используется для научных вычислений, а также обладает множеством библиотек, удобных для обработки данных, включая Pandas.
2. Google OR-Tools CP-SAT: для решения задачи с ограничениями была выбрана библиотека Google OR-Tools с модулем CP-SAT. Этот инструмент позволяет эффективно моделировать и решать задачи с ограничениями.
3. PostgreSQL: в качестве системы управления базами данных была выбрана PostgreSQL, которая предоставляет надежное хранение и управление данными.
4. SQLAlchemy: для взаимодействия с таблицами СУБД PostgreSQL была выбрана библиотека SQLAlchemy. Она обеспечивает удобный способ работы с базами данных, включая выполнение SQL-запросов, создание и управление таблицами базы данных.
5. VirtualBox: в качестве менеджера виртуальных операционных систем была выбрана VirtualBox. Этот инструмент позволяет создавать и управлять виртуальными машинами, в том числе с установленной Ubuntu Server.
6. Ubuntu Server: виртуальная операционная система Ubuntu Server была выбрана в качестве платформы, на которой будет работать разработанный модуль. Ubuntu Server - это популярный дистрибутив Linux для серверов.
7. Apache Airflow: для организации запуска программы по расписанию был выбран Apache Airflow. Этот инструмент позволяет создавать и управлять рабочими процессами, включая автоматизацию выполнения задач.
8. Pandas: для работы с данными в формате DataFrame была выбрана библиотека Pandas. Она предоставляет мощные средства для анализа и преобразования данных.

Все эти инструменты и технологии в совокупности обеспечивают эффективное и надежное решение поставленной задачи, удовлетворяя всем требованиям и обеспечивая точное управление данными и их обработку.

# **2.2 ПОДГОТОВКА ИНФРАСТРУКТУРЫ**

# **2.2.1 НАСТРОЙКА СЕРВЕРА**

Для установки Ubuntu Server, нужно следовать определенным шагам. На рисунке 1 изображен процесс выбора образа ISO и указания папки, где будет размещен сервер.

На рисунках 2 и 3 демонстрируется выбор характеристик для сервера. Рекомендуется установить максимально доступное количество оперативной памяти и процессорных ядер. Также создается виртуальный жесткий диск, указывается его будущее расположение и оставляется размер диска по умолчанию.

Эти шаги обеспечат корректную установку Ubuntu Server и создание виртуальной среды, на которой можно будет запустить разработанный модуль.

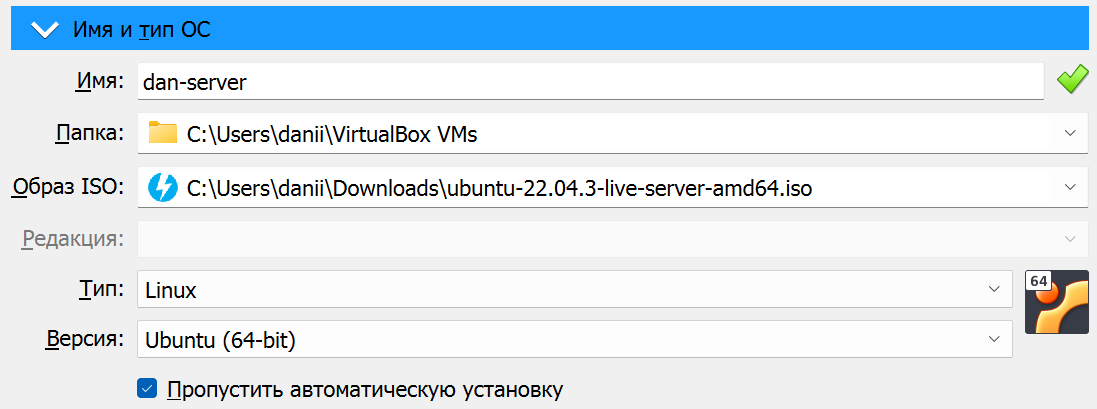
****

Рисунок 1 – Настройка раздела "Имя и тип ОС"

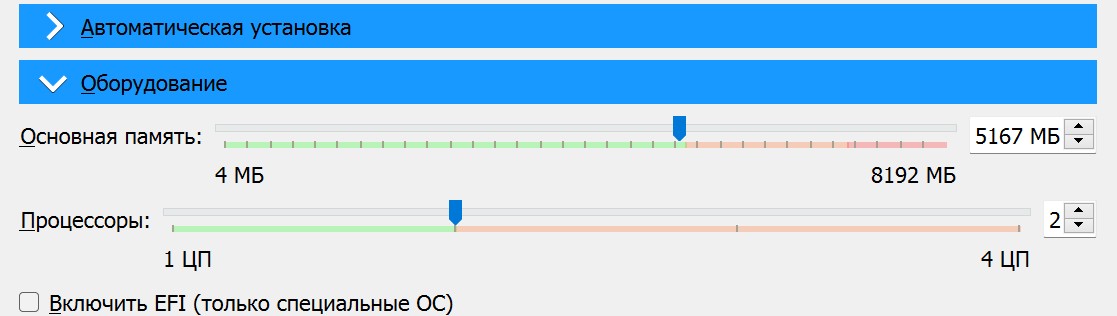


Рисунок 2 – Настройка раздела "Оборудование"

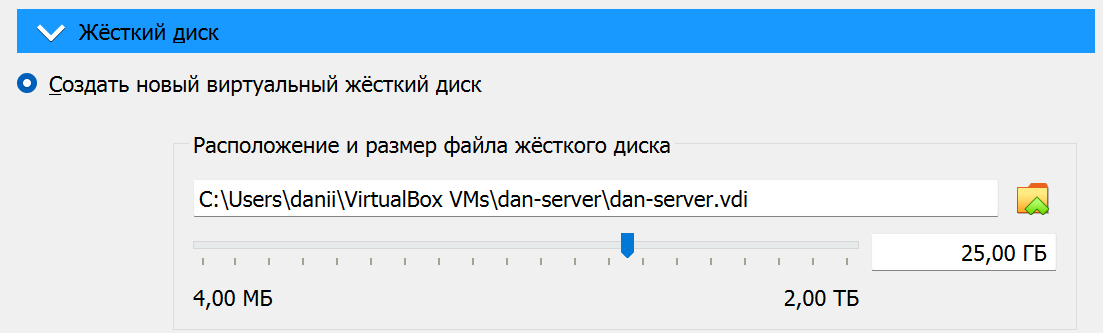


Рисунок 3 – Настройка раздела "Жесткий диск"

После запуска сервера автоматически начнется процесс установки, и рекомендуется оставить все настройки по умолчанию. Во время настройки пользователя, были введены тестовые значения для удобства. Например, имя сервера установлено как "dan20", и пароль "1" (как показано на рисунке 4).

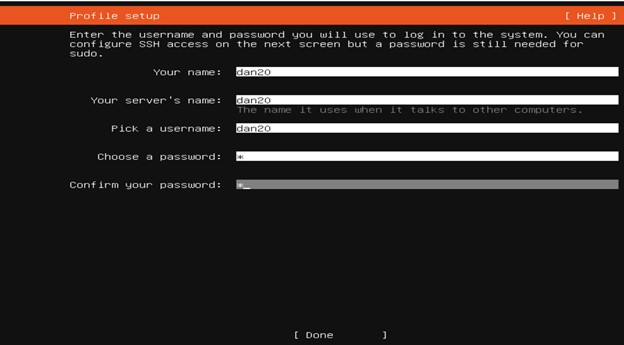


Рисунок 4 – Настройка профиля сервера

После установки сервера необходимо настроить обмен данными между сервером и домашней системой. Для этого были созданы общие папки в разделе "Общие папки" в меню "Настроить" на домашнем экране менеджера VirtualBox. Были выбраны параметры, как на рисунках 5 и 6.

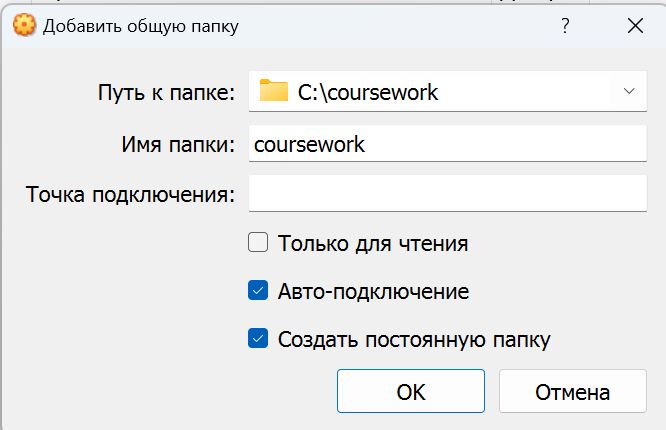


Рисунок 5 – Добавление общей папки

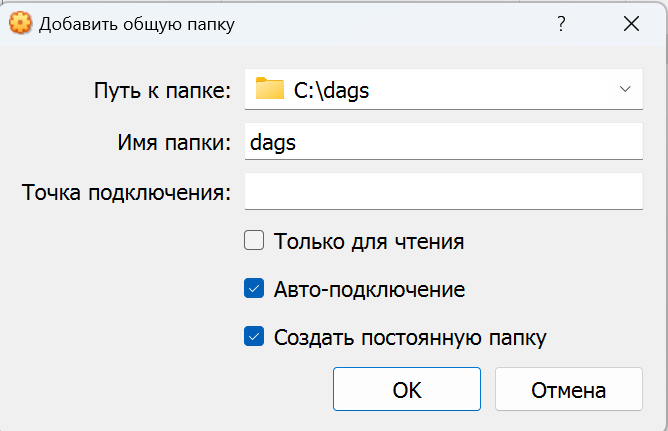


Рисунок 6 – Добавление общей папки

Для установки связи между добавленными папками в домашней системе и созданными папками в виртуальной машине, использованы следующие команды:

1. sudo apt-get install virtualbox-guest-utils – установить гостевые утилиты;
2. sudo mkdir shared – создание папки;
3. sudo mount –t vboxsf coursework shared – создание связи.

Результат выполнения этих команд показано на рисунках 7 и 8, видно, что содержимое папок совпадает, связь была настроена.

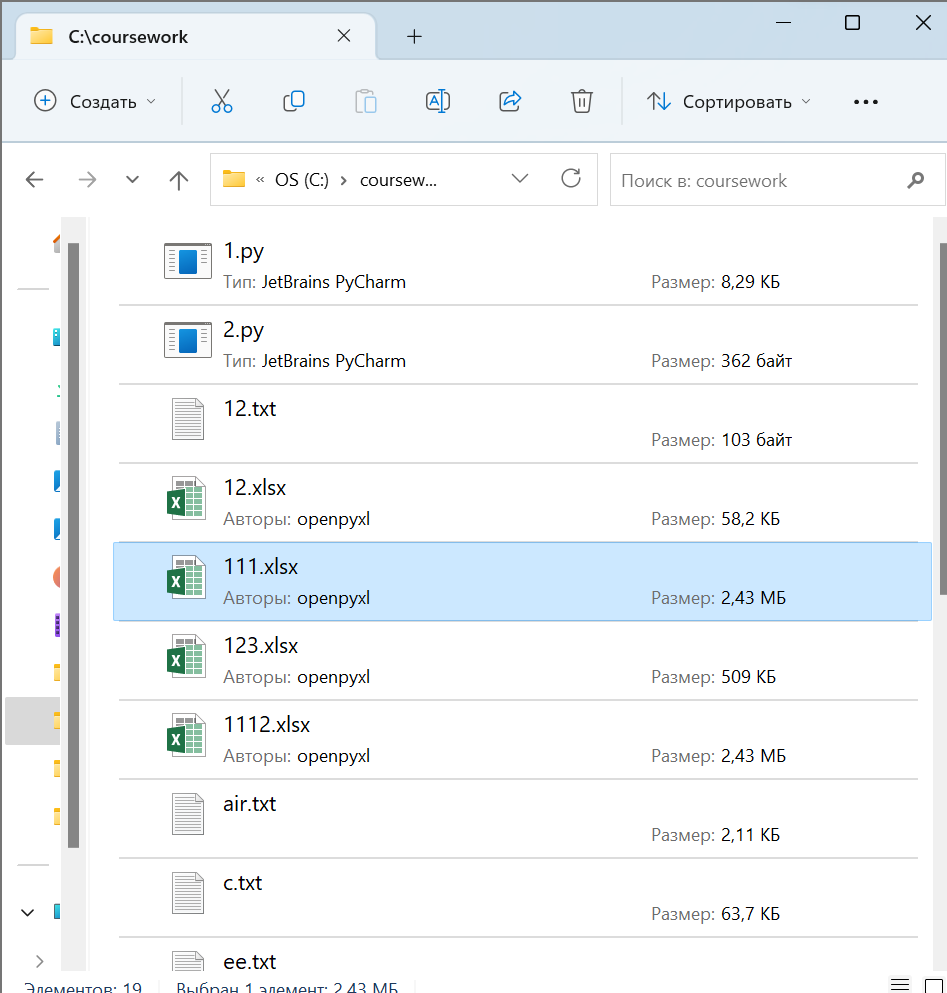


Рисунок 7 – Содержимое папки C:\coursework

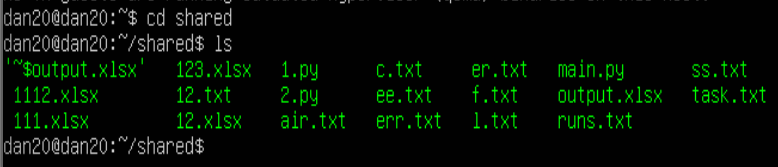


Рисунок 8 – Содержимое папки /shared с помощью команды cd shared

# **2.2.2 НАСТРОЙКА БАЗЫ ДАННЫХ**

Для использования базы данных PostgreSQL на сервере, необходимо сначала установить ее, создать таблицы и заполнить их тестовыми данными. PostgreSQL установлена с помощью следующей команды: sudo apt-get install postgresql postgresql-contrib. Выполнение этой команды показано на рисунке 9.

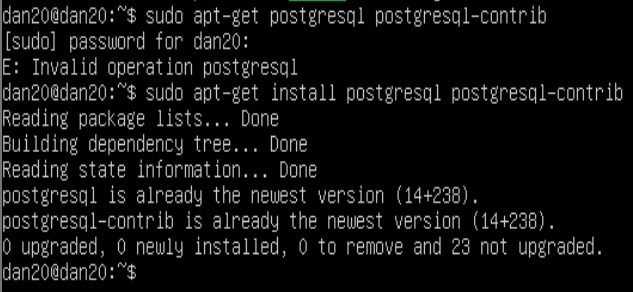


Рисунок 9 –Выполнение установки PostgreSQL

Для программы необходимо создать три таблицы. Одна из них будет использоваться для получения входных данных, и будет создана с помощью программы, которая создаст набор данных, соответствующий ER-модели таблицы, как показано на рисунке 10. Затем программа создаст таблицу и заполнит ее этим набором данных.

В программе для создания набора данных, сначала формируются списки, содержащие все возможные варианты управляющих параметров. Затем указывается необходимое количество значений объемов грузов. Эти значения объемов грузов генерируются случайным образом и следуют равномерному распределению на отрезке от 1 до 100, причем единица измерения считается тонной. Выбор параметров также осуществляется с учетом равномерного распределения. Для каждого значения объема параметры выбираются из соответствующих списков.

ER-модель таблицы входных данных представлена на рисунке 10, а блок-схема программы для генерации набора значений и создания таблицы на ее основе представлена на рисунке 11.



Рисунок 10 – ER-модель таблицы входных данных



Рисунок 11 – Блок-схема алгоритма программы генерации и заполнения таблицы

В самой программе для генерации значений объемов используется метод random.uniform(1, 100), а для выбора параметров из списков - random.choice(parameters\_list\_name). Затем полученные значения записываются в объект данных (DataFrame) и с использованием методов библиотеки SQLAlchemy передаются и сохраняются в таблице PostgreSQL на сервере.

Также для создания строки подключения необходимо знать параметры базы данных. По умолчанию имя пользователя и базы данных – это "postgres", пароль необходимо установить вручную, с помощью команд:

1. sudo -u postgres psql – для входа в PostgreSQL;
2. alter user postgres password '1'; – для установки пароля "1".

IP-адрес сервера по умолчанию 127.0.0.1:5432, и в итоге строка подключения выглядит так: 'postgresql+psycopg2://postgres:1@127.0.0.1:5432/postgres'.

Этот подход позволил создать компактную и читаемую программу, с сохранением быстродействия генерации и записи.

Для запуска этой программы нужно переместить файл с программой в одну из общих папок и загрузить необходимые библиотеки на сервер, что можно сделать, с помощью ввода следующих команд:

1. sudo apt install -y python3 python3-pip python3-venv – установка менеджера пакетов;
2. sudo pip install pandas psycopg2-binary sqlalchemy – установка библиотек;
3. python3 main.py – выполнение программы внутри общей папки.

После выполнения предыдущих шагов и заполнения таблицы данными, вы можете проверить данные в вашей PostgreSQL базе данных на сервере Ubuntu, используя следующие команды:

1. sudo -u postgres psql – для входа в PostgreSQL;
2. select count(\*) from data; - для просмотра количества строк таблицы;
3. select \* from data; - для просмотра всех строк в таблице.

На рисунке 12 можно увидеть количество строк в таблице, а на рисунке 13 представлены некоторые записи из этой таблицы.

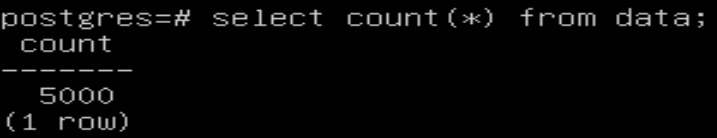


Рисунок 12 – Просмотр количества строк таблицы

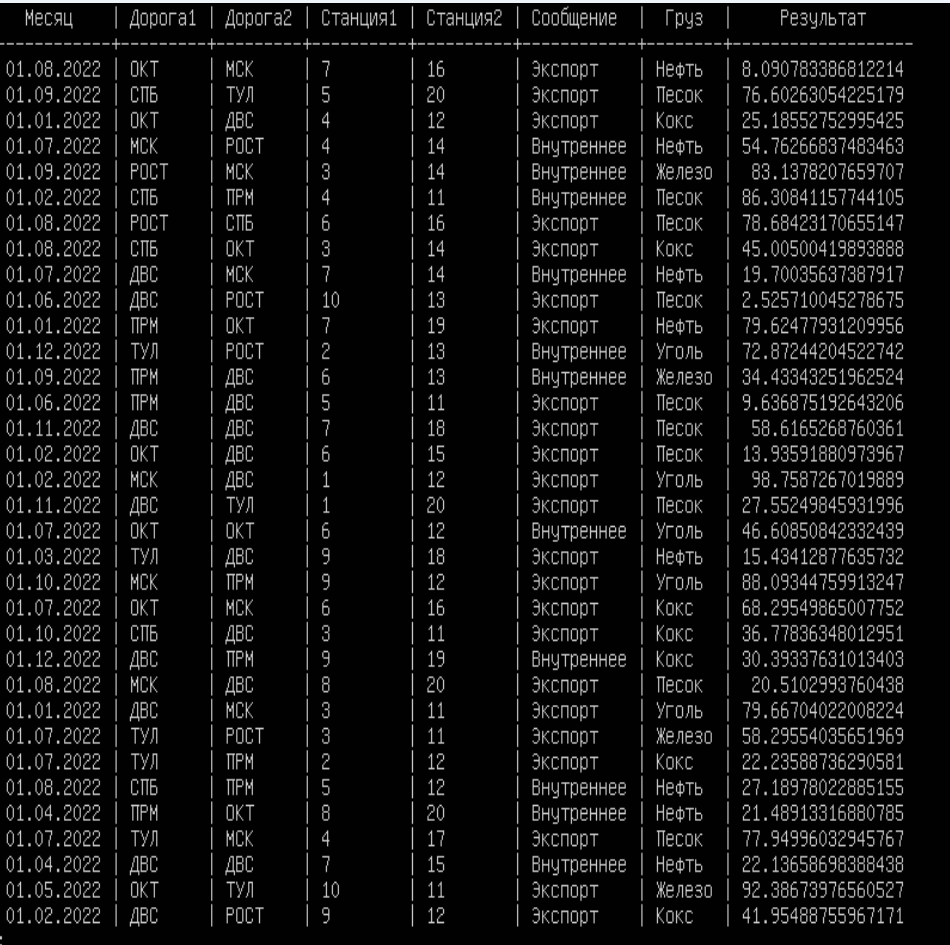


Рисунок 13 – Просмотр строк таблицы

Также будет создана таблица выходных данных, которая будет содержать те же данные, что и таблица входных данных, но с округленными значениями объемов грузов. Не требуется создавать эту таблицу вручную, так как она будет автоматически создана и заполнена из основной программы, используя аналогичные методы и процедуры, описанные при создании таблицы входных данных. ER-модель этой таблицы показана на рисунке 14:



Рисунок 14 ER-модель таблицы выходных данных

Также потребуются таблица, в которую будут заноситься результаты проверки сходимости сумм (sat), времени запуска (start\_time), окончания работы (end\_time) и общего времени работы (all\_time) основной функции. Для этого необходимо выполнить команды:

1. sudo -u postgres psql – для входа в PostgreSQL;
2. CREATE TABLE values (sat BOOLEAN, start\_time timestamp, end\_time timestamp, all\_time interval, num integer); – для создания таблицы с необходимыми столбцами.

ER-модель таблицы указана на рисунке 15:

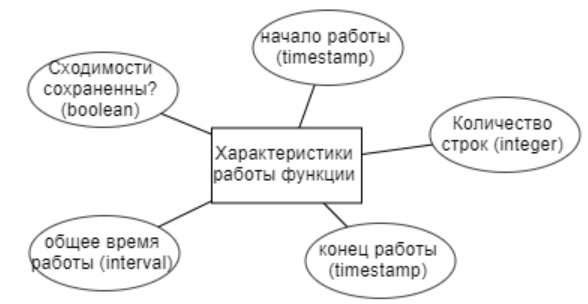


Рисунок 15 ER-модель таблицы характеристик работы основной функции

# **2.2.3 УСТАНОВКА APACHE AIRFLOW И ДРУГИХ БИБЛИОТЕК**

Для установки и инициализации по умолчанию Apache Airflow необходимо выполнить следующие команды:

AIRFLOW\_VERSION=2.7.1

PYTHON\_VERSION="$(python --version | cut -d " " -f 2 | cut -d "." -f 1-2)"

CONSTRAINT\_URL="https://raw.githubusercontent.com/apache/airflow/constraints-${AIRFLOW\_VERSION}/constraints-${PYTHON\_VERSION}.txt"

pip install "apache-airflow==${AIRFLOW\_VERSION}" --constraint "${CONSTRAINT\_URL}" – для установки Apache Airflow;

export PATH=$PATH:/home/dan20/.local/bin – для добавления Apache Airflow в переменную окружения, для запуска команд airflow.

airflow standalone – для инициализации базы данных Apache Airflow.

Для работы программы также необходимо установить библиотеки Numpy и Google OR-Tools. Вы можете выполнить это с помощью команды: pip install numpy ortools kubernetes virtualenv.

После установки Apache Airflow для последующего запуска DAG, необходимо выполнить следующие шаги:

1. создать папку с названием /airflow/dags командой: sudo mkdir /airflow/dags;
2. связать эту папку с ранее добавленной общей папкой в VirtualBox, в которой хранятся ваши DAG-скрипты командой: sudo mount –t vboxsf dags /airflow/dags;
3. переместить Python-скрипт DAG в общую папку на домашней системе.

Эти действия позволят DAG-скрипту отобразиться в списке доступных для выполнения задач в базе данных Apache Airflow.

Проверить это можно с помощью команды: airflow dags list. Результат ее выполнения после этих действий показан на рисунке 16:

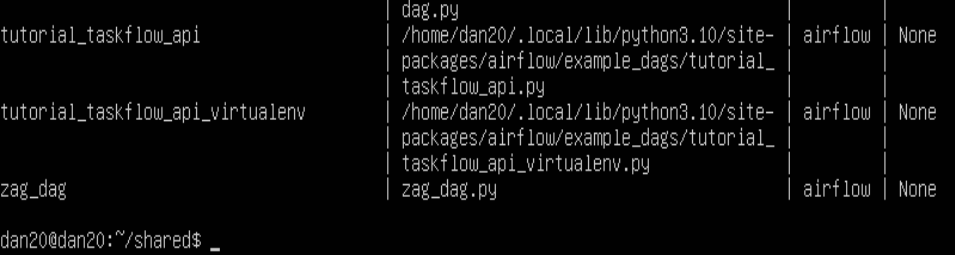


Рисунок 16 – Часть вывода списка DAG-скриптов, включающий пользовательский

# **РАЗДЕЛ 3. РАЗРАБОТКА И ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ**

# **3.1 РАЗРАБОТКА ПРИЛОЖЕНИЯ**

Для разработки программы был создан Apache Airflow DAG (Directed Acyclic Graph или ориентированный ациклический граф), представленный в виде кода Python. Этот DAG позволяет автоматизировать выполнение определенных задач в заданном порядке с определенными интервалами. Хотя обычно в Apache Airflow задачи организуются в виде графа с разделением на подзадачи, в данном случае программа представляет собой линейный процесс без необходимости разделения на подзадачи. Тем не менее, создание DAG всё равно требуется, так как это структурное требование Apache Airflow. В этом DAG содержится всего одна задача, и программа выполняется линейно.

Вот ключевые элементы программы:

1. Создание DAG:

* Сначала были определены параметры по умолчанию для выполнения задач, такие как зависимость от предыдущей задачи, количество попыток выполнения и задержка между попытками.
* Затем был создан объект DAG с указанием его имени, времени начала и конца выполнения, описания и указания параметров.

1. Определение Python-функции:

* Была создана Python-функция my\_python\_function(), которая выполняет основную задачу.
* Внутри этой функции данные извлекаются из базы данных PostgreSQL и загружаются в объект DataFrame для дальнейшей обработки.

1. Группировка, расчет сумм и округление значений:

* Внутри функции данные были сгруппированы по группам управляющих параметров.
* Затем рассчитаны суммы дробных значений объемов грузов для каждой группы, а также найдена сумма всех дробных значений, которые округлены до ближайшего целого значения.

1. Работа с Google OR-Tools:

* Для решения задачи была использована библиотека Google OR-Tools (CP-SAT).
* Создана модель с булевыми переменными и ограничениями на сумму округленных сумм групп и сумм переменных в группе, которое находится в пределах отрезка ±1, чтобы найти первое из возможных решений.
* Решение задачи сохраняется в объект DataFrame.

1. Проверка сходимости и измерение времени выполнения:

* После решения задачи, программа проверяет, сошлись ли суммы округленных значений с суммами изначальных дробных значений для каждой группы. И сохраняется булевый результат проверки.
* Также измеряется время выполнения функции, которое засекается от начала выполнения my\_python\_function() и до записи самого времени в базу данных.

1. Загрузка данных:

* Округленные данные загружаются в базу данных PostgreSQL с созданием или перезаписыванием новой таблицы.
* Результаты проверки сходимости сумм и времена работы программы загружаются в базу данных PostgreSQL с созданием или перезаписыванием новой таблицы.

1. Создание PythonOperator:

* После описания основной функции создается PythonOperator, который выполняет ее и добавляется в DAG в очередь выполнения.

Блок-схема my\_python\_function() представлена на рисунке 17:

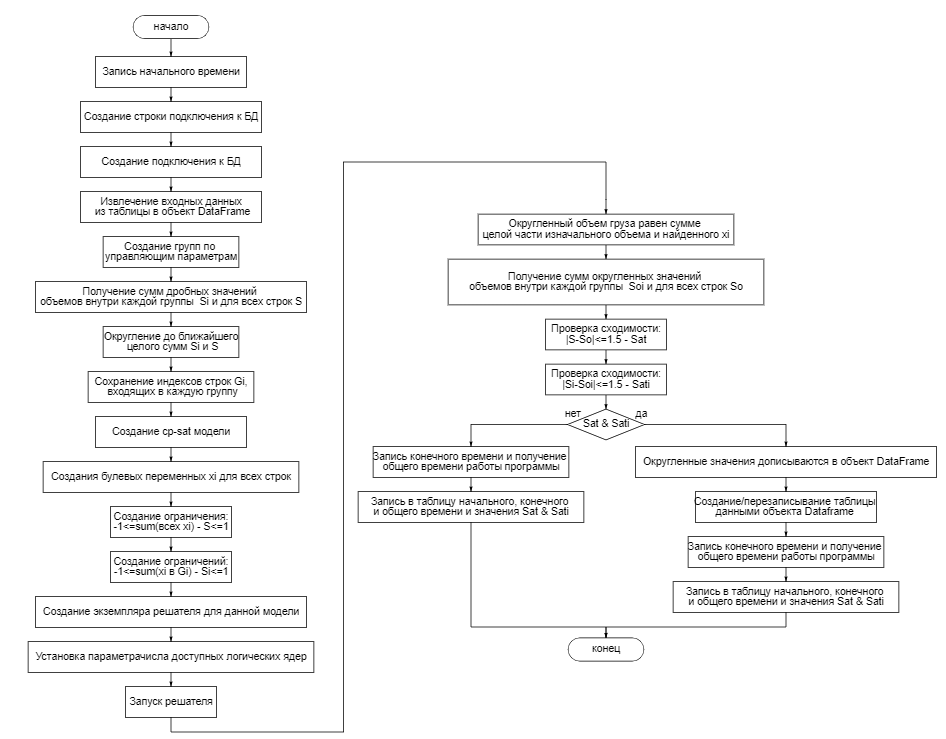


Рисунок 17 Блок-схема алгоритма функции my\_python\_function()

# **3.2 ТЕСТИРОВАНИЕ ПРОГРАММЫ**

В этом разделе программа будет запускаться с использованием Apache Airflow. Между запусками программы по расписанию будут внесены изменения в исходные данные в таблице, и будет проверено, достигли ли суммы определенных значений с помощью просмотра информации из таблицы характеристик.

Для запуска программы по расписанию с использованием Apache Airflow, выполните следующие команды:

1. airflow scheduler –D – для запуска планировщика Apache Airflow в фоновом режиме;
2. airflow dags trigger zag\_dag – для триггера DAG-скрипта, чтобы он появился в очереди на выполнение;
3. airflow dags unpause zag\_dag – для снятия паузы, чтобы DAG-скрипт начал выполняться согласно настроенному расписанию.

Посмотреть информацию о состоянии и характеристиках всех запусков DAG-скрипта можно с помощью команды: airflow dags list-runs -d zag\_dag.

Вывод команды показан на рисунке 18:

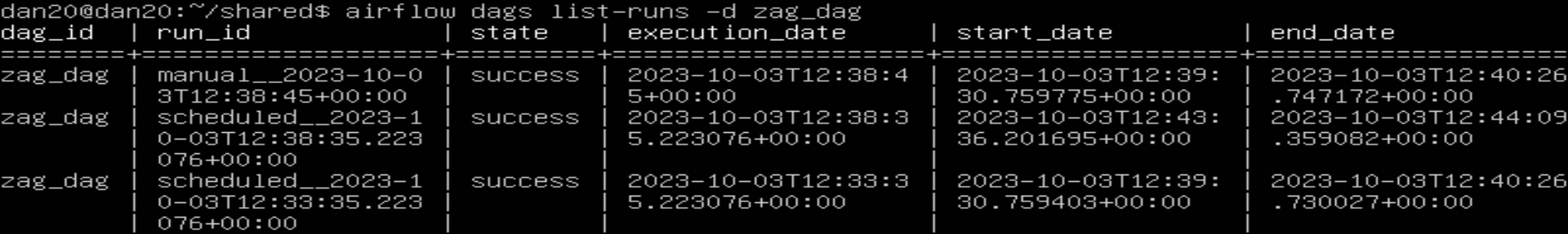


Рисунок 18 Вывод команды airflow dags list-runs zag\_dag

Также после некоторого количества прогонов изменим количество строк в таблице входных данных, изменив количество строк в программе генерации входных данных на 10000 и запустив ее еще раз.

Характеристики всех запусков и сами округленные значения также записываются в базу данных, и посмотреть их можно с помощью команд:

1. sudo -u postgres psql – для входа в PostgreSQL;
2. select \* from values; - для просмотра строк таблицы характеристик;
3. select count(\*) from rounded; - для просмотра количества строк таблицы округленных значений;
4. select \* from rounded; - для просмотра всех строк в таблице округленных значений.

Результаты выполнения на рисунках 19, 20 и 21:

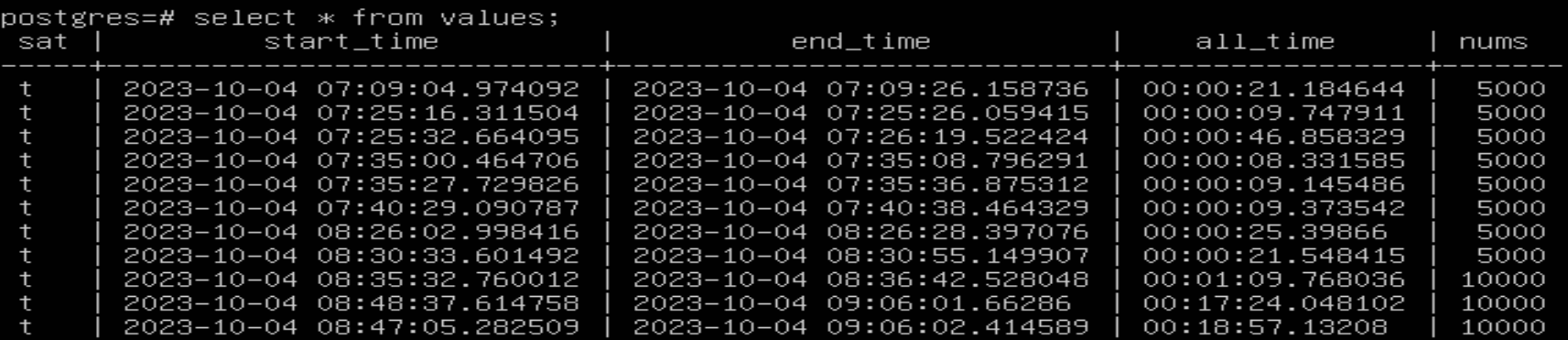


Рисунок 19 Вывод всех строк таблицы характеристик

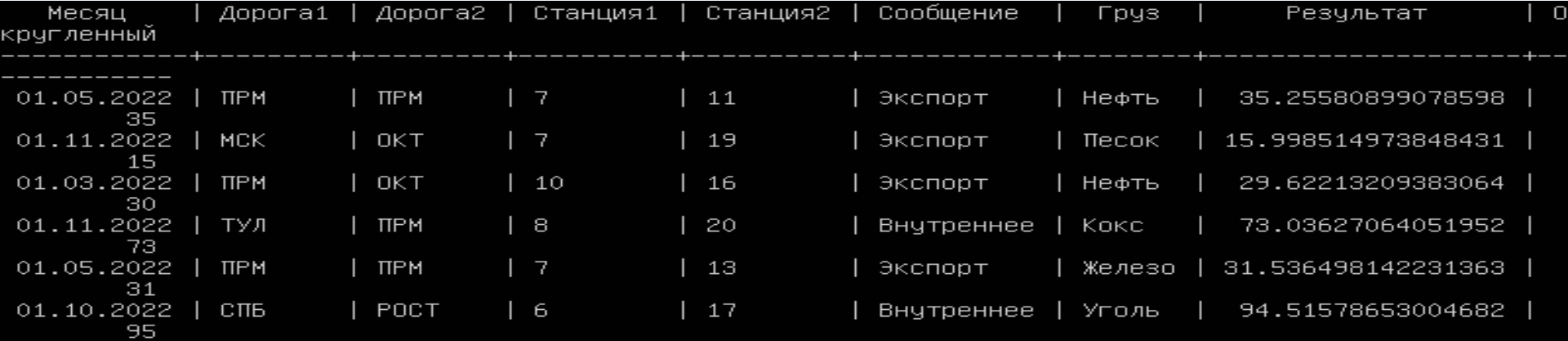


Рисунок 20 Часть вывода всех строк таблицы округленных значений

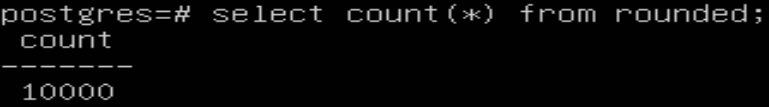


Рисунок 21 Вывод количества строк округленных значений

По значениям столбца sat видно, что для всех прогонов программы сходимость была достигнута и по значениям столбца nums и количества строк в таблице округленных значений, что значения изменились в результате изменения таблицы входных данных, и начиная с 9 прогона количество строк стало 10000.

Для тестирования программы с значениями 20000 и 40000 тысяч сначала будет перезаписана таблица входных данных все так же с помощью запуска программы генерации входных данных и выполнения команды, которая тестирует только основную задачу: airflow tasks test zag\_dag exec.

# **ЗАКЛЮЧЕНИЕ**

В данной работе была разработана и представлена методика решения важной задачи округления дробных значений грузов с сохранением суммарных значений по группам управляющих параметров. Основной целью работы было создание модуля на языке программирования Python, способного автоматически обрабатывать данные из таблицы СУБД PostgreSQL, добавлять округленные значения к исходным данным и записывать их обратно в базу данных.

Модуль, разработанный в рамках исследования, не только успешно выполняет округление, но и обеспечивает сохранение суммарных значений по группам управляющих параметров в пределах ±1.5 относительно изначальных значений. Это гарантирует точность и надежность округления, особенно в логистических и снабженческих приложениях, где сохранение целостности данных имеет критическое значение.

Кроме того, в ходе работы была реализована интеграция с Apache Airflow, что позволяет автоматизировать выполнение модуля по расписанию, что может быть весьма полезно в среде управления грузами и логистики.

Эта технологическая практика призвана улучшить методы управления грузами и оптимизации расходов, предоставляя практикам и специалистам важный инструмент для повышения эффективности и точности данных.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Google OR-Tools: [сайт]. – Текст. Изображение: электронные. – URL: https://developers.google.com/optimization/introduction?hl=en (последнее обращение 28.08.2023).
2. Apache Airflow: [сайт]. – Текст. Изображение: электронные. – URL: https://airflow.apache.org/docs/apache-airflow/stable/start.html (последнее обращение 28.08.2023).
3. SQLAlchemy: [сайт]. – Текст. Изображение: электронные. – URL: https://docs.sqlalchemy.org/en/20 (последнее обращение 28.08.2023).
4. Ubuntu Server documentation: [сайт]. – Текст. Изображение: электронные. – URL: https://ubuntu.com/server/docs (последнее обращение 28.08.2023).
5. PostgreSQL: [сайт]. – Текст. Изображение: электронные. – URL: https://www.postgresql.org/docs/15/index.html (последнее обращение 28.08.2023).
6. VirtualBox: [сайт]. – Текст. Изображение: электронные. – URL: https://www.virtualbox.org/wiki/Technical\_documentation (последнее обращение 28.08.2023).

# **ПРИЛОЖЕНИЕ А. ЛИСТИНГ DAG-СКРИПТА ОСНОВНОЙ ПРОГРАММЫ:**

from airflow import DAG

from airflow.operators.python\_operator import PythonOperator

from datetime import timedelta

import pandas as pd

import numpy as np

import pendulum

from sqlalchemy.types import Text, Float, Integer, Interval

from sqlalchemy import create\_engine, MetaData, Table, Column, DateTime, insert

default\_args = {

'depends\_on\_past': False,

'retries': 1,

'retry\_delay': timedelta(minutes=5),

}

dag = DAG(

'zag\_dag',

start\_date=pendulum.now().subtract(minutes=6),

end\_date=pendulum.now().add(days=7),

description='My Apache Airflow DAG',

default\_args=default\_args,

schedule\_interval=timedelta(minutes=5),

catchup=False,

)

def my\_python\_function():

start\_time = pendulum.now()

table\_name = 'data'

db\_uri = 'postgresql+psycopg2://postgres:1@127.0.0.1:5432/postgres'

engine = create\_engine(db\_uri, echo=True)

df = pd.read\_sql\_table(

table\_name,

con=engine

)

def custom\_round(x):

if x % 1 == 0.5:

return np.ceil(x)

else:

return round(x)

def constraint\_sat(df, group\_indices\_list):

sats = []

for i, group\_indices in enumerate(group\_indices\_list):

subset\_df = df.loc[group\_indices]

diff = abs(subset\_df['Результат'].sum() - subset\_df['Округленный'].sum())

sats.append(diff <= 1.5 and diff >= -1.5)

return all(sats)

df['Дробная'] = df['Результат'].apply(lambda x: x - int(x))

summa = custom\_round(df['Дробная'].sum())

mos = df.groupby('Сообщение').apply(lambda group: group.index.tolist())

r1r2s = df.groupby(['Дорога1', 'Дорога2']).apply(lambda group: group.index.tolist())

mr1s = df.groupby(['Месяц', 'Дорога1']).apply(lambda group: group.index.tolist())

r1ws = df.groupby(['Дорога1', 'Груз']).apply(lambda group: group.index.tolist())

mowms = df.groupby(['Месяц', 'Груз', 'Сообщение']).apply(lambda group: group.index.tolist())

r1s1wms = df.groupby(['Дорога1', 'Станция1', 'Груз', 'Сообщение']).apply(lambda group: group.index.tolist())

def calculate\_sums(df, group\_indices\_list):

sums = []

for i, group\_indices in enumerate(group\_indices\_list):

# Получите подмножество DataFrame с использованием индексов

subset\_df = df.loc[group\_indices]

# Получите сумму значений столбца 'Дробная' для данной группировки

total\_sum = custom\_round(subset\_df['Дробная'].sum())

sums.append(total\_sum)

return sums

sums\_1 = calculate\_sums(df, mos)

sums\_2 = calculate\_sums(df, r1r2s)

sums\_3 = calculate\_sums(df, mr1s)

sums\_4 = calculate\_sums(df, r1ws)

sums\_5 = calculate\_sums(df, mowms)

sums\_6 = calculate\_sums(df, r1s1wms)

def collect\_indices(group\_indices\_list):

indices\_list = []

for group\_indices in group\_indices\_list:

indices\_list.append(group\_indices)

return indices\_list

indices\_list1 = collect\_indices(mos)

indices\_list2 = collect\_indices(r1r2s)

indices\_list3 = collect\_indices(mr1s)

indices\_list4 = collect\_indices(r1ws)

indices\_list5 = collect\_indices(mowms)

indices\_list6 = collect\_indices(r1s1wms)

from ortools.sat.python import cp\_model

model = cp\_model.CpModel()

x = {}

for i in range(len(df)):

x[i] = model.NewIntVar(0, 1, f'x\_{i}')

# Добавление ограничений

total\_x\_sum = sum(x[i] for i in range(len(df)))

model.Add(total\_x\_sum - summa <= 1)

model.Add(total\_x\_sum - summa >= -1)

for indices, total\_sum in zip(indices\_list1, sums\_1):

group\_sum = sum(x[i] for i in indices)

model.Add(group\_sum - total\_sum <= 1)

model.Add(group\_sum - total\_sum >= -1)

for indices, total\_sum in zip(indices\_list2, sums\_2):

group\_sum = sum(x[i] for i in indices)

model.Add(group\_sum - total\_sum <= 1)

model.Add(group\_sum - total\_sum >= -1)

for indices, total\_sum in zip(indices\_list3, sums\_3):

group\_sum = sum(x[i] for i in indices)

model.Add(group\_sum - total\_sum <= 1)

model.Add(group\_sum - total\_sum >= -1)

for indices, total\_sum in zip(indices\_list4, sums\_4):

group\_sum = sum(x[i] for i in indices)

model.Add(group\_sum - total\_sum <= 1)

model.Add(group\_sum - total\_sum >= -1)

for indices, total\_sum in zip(indices\_list5, sums\_5):

group\_sum = sum(x[i] for i in indices)

model.Add(group\_sum - total\_sum <= 1)

model.Add(group\_sum - total\_sum >= -1)

for indices, total\_sum in zip(indices\_list6, sums\_6):

group\_sum = sum(x[i] for i in indices)

model.Add(group\_sum - total\_sum <= 1)

model.Add(group\_sum - total\_sum >= -1)

solver = cp\_model.CpSolver()

solver.parameters.num\_search\_workers = 8

# Решение задачи

status = solver.Solve(model)

df['Округленный'] = 0

for i in range(len(df)):

df.at[i, 'Округленный'] = df.at[i, 'Результат'] - df.at[i, 'Дробная'] + solver.Value(x[i])

sat\_1 = constraint\_sat(df, mos)

sat\_2 = constraint\_sat(df, r1r2s)

sat\_3 = constraint\_sat(df, mr1s)

sat\_4 = constraint\_sat(df, r1ws)

sat\_5 = constraint\_sat(df, mowms)

sat\_6 = constraint\_sat(df, r1s1wms)

diff = abs(df['Результат'].sum() - df['Округленный'].sum())

satisfies\_condition = diff <= 1.5 and diff >= -1.5

sat\_result = all([sat\_1, sat\_2, sat\_3, sat\_4, sat\_5, sat\_6, satisfies\_condition])

df\_without\_fractional = df.drop(columns=["Дробная"])

engine21 = create\_engine(db\_uri, echo=True, pool\_pre\_ping=True)

end\_time = pendulum.now()

all\_time = end\_time - start\_time

user\_table = "values"

metadata = MetaData()

user\_table = Table('values', metadata, Column('sat', DateTime),Column('start\_time', DateTime),Column('end\_time', DateTime),Column('all\_time', Interval),Column('nums', Integer))

stmt = insert(user\_table).values(

sat=sat\_result,

start\_time=start\_time,

end\_time=end\_time,

all\_time=all\_time,

nums=df.shape[0])

with engine21.connect() as conn:

result = conn.execute(stmt)

pass

if sat\_result:

engine1 = create\_engine(db\_uri, echo=True, pool\_pre\_ping=True)

dtype = {

"Месяц": Text,

"Дорога1": Text,

"Дорога2": Text,

"Станция1": Text,

"Станция2": Text,

"Сообщение": Text,

"Груз": Text,

"Результат": Float,

"Округленный": Integer

}

df\_without\_fractional.to\_sql(

'rounded',

engine1,

if\_exists='replace',

index=False,

chunksize=500,

dtype=dtype

)

execute\_task = PythonOperator(

task\_id='exec',

python\_callable=my\_python\_function,

dag=dag,

)

execute\_task

# **ПРИЛОЖЕНИЕ Б. ЛИСТИНГ ПРОГРАММЫ ГЕНЕРАЦИИ ВВОДНЫХ ДАННЫХ:**

import psycopg2

from sqlalchemy import create\_engine

from sqlalchemy.types import Text, Float

import pandas as pd

import random

message = ['Экспорт', 'Внутреннее']

month = ['01.01.2022','01.02.2022','01.03.2022','01.04.2022','01.05.2022','01.06.2022','01.07.2022','01.08.2022','01.09.2022','01.10.2022','01.11.2022','01.12.2022']

road = ['ДВС','ОКТ','СПБ','МСК','ДВС','ПРМ','ТУЛ','РОСТ']

station1 = ['1','2','3','4','5','6','7','8','9','10']

station2 = ['11','12','13','14','15','16','17','18','19','20']

cargo = ['Железо', 'Кокс', 'Песок', 'Нефть', 'Кокс', 'Уголь']

num\_rows = 10000

data = []

for \_ in range(num\_rows):

row = {

'Месяц': random.choice(month),

'Дорога 1': random.choice(road),

'Дорога 2': random.choice(road),

'Станция 1': random.choice(station1),

'Станция 2': random.choice(station2),

'Сообщение': random.choice(message),

'Груз': random.choice(cargo),

'Результат': random.uniform(1, 100)

}

data.append(row)

df = pd.DataFrame(data)

db\_uri = 'postgresql+psycopg2://postgres:1@127.0.0.1:5432/postgres'

engine = create\_engine(db\_uri, echo=True, pool\_pre\_ping=True)

df.to\_sql(

'data',

engine,

if\_exists='replace',

index=False,

chunksize=500,

dtype={

"Месяц": Text,

"Дорога1": Text,

"Дорога2": Text,

"Станция1": Text,

"Станция2": Text,

"Сообщение": Text,

"Груз": Text,

"Результат": Float

}

)