

Министерство науки и высшего образования российской федерации федеральное
государственно автономное образовательное учреждение высшего образования
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(Московский Политех)

КУРСОВОЙ ПРОЕКТ

по курсу: Проектирование и администрирование баз данных

Вариант: № 21

Тема: « Нормализация БД, индексация и разработка десктопного приложения »

Выполнил Ортиков Рамиз Умидович
(ФИО)

Группа 241-326

Проверил(-а) Перепелкина Юлианна Вячеславовна
(ФИО)

Москва

2025

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (МОСКОВСКИЙ
ПОЛИТЕХ)

УТВЕРЖДАЮ
заведующий кафедрой
/ И.О. Фамилия/

« ____ » _____ 20 ____ г.

ЗАДАНИЕ
на выполнение курсовой работы (проекта)

Ортикову Рамизу Умидовичу

(фамилия, имя, отчество обучающегося)

обучающемуся группы 241-326

направления подготовки / специальности/ профессии Информатика и
вычислительная техника 09.03.01 «Системная и программная инженерия»
по дисциплине «Проектирование и администрирование баз данных»

(наименование дисциплины (модуля))

1. Исходные данные к работе (проекту): набор данных о глобальном энергопотреблении
2. Содержание задания по курсовой работе (проекту) – перечень вопросов, подлежащих разработке:

Разрабатываемый вопрос	Объем от всего задания, %	Срок выполнения	Примечание
Раздел 1. Проектирование БД	30	11.06.2025	
1.1. Нормализация исходных данных	15	11.06.2025	3NF
1.2. Создание ER-диаграммы	15	11.06.2025	Диаграмма связей, PlantUML
Раздел 2. Реализация БД	30	13.06.2025	
2.1. Написание SQL-скриптов создания таблиц	15	13.06.2025	PostgreSQL
2.2. Заполнение БД тестовыми данными	15	13.06.2025	psql, ручной импорт
Раздел 3. Анализ производительности	20	15.06.2025	
3.1. EXPLAIN ANALYZE без индексов	10	14.06.2025	SELECT, JOIN
3.2. Оптимизация запросов через индексы	10	15.06.2025	Сравнение
Раздел 4. Разработка интерфейса	20	19.06.2025	
4.1. Создание GUI	10	17.06.2025	Фильтры, график
4.2. Тестирование функционала	10	19.06.2025	Экспорт, сценарии

Руководитель курсовой работы (проекта):

«21» Июня 2025г.

(подпись)

_____ кандидат физ-мат. наук, доцент

Перепёлкина Ю. В.

(Фамилия И.О.)

Дата выдачи задания

Дата сдачи выполненной работы (проекта)

Задание принял к исполнению

«30 » Апреля 2025г.

(дата)

(подпись)

«30 » Апреля 2025г.

«21» Июня 2025г.

Ортиков Р.У.

(Фамилия И.О.)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 ИМПОРТ И ОБРАБОТКА ДАННЫХ В СУБД POSTGRESQL	6
1.1 Описание исходного набора данных.....	6
1.2 Создание базы данных и таблиц	6
1.3 Импорт CSV-файла в таблицу energy_data.....	7
1.4 Заполнение справочной таблицы стран.....	8
1.5 Агрегация и перенос данных в нормализованную структуру.....	8
2 ПРОЕКТИРОВАНИЕ СТРУКТУРЫ БАЗЫ ДАННЫХ И НОРМАЛИЗАЦИЯ	11
2.1 Цели и этапы нормализации	11
2.2 Описание логической модели и связей.....	12
2.3 Описание таблиц: Countries и Energy_Metrics	12
2.4 Построение ER-диаграммы	13
2.5 Преимущества нормализованной структуры.....	14
3 АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ SQL-ЗАПРОСОВ	16
3.1 SELECT с фильтрацией по источнику энергии (до индекса).....	16
3.2 Создание индекса и повторный анализ	17
3.3 Сравнение EXPLAIN ANALYZE до и после индексации	19
3.4 Анализ сложного запроса с JOIN	20
3.5. Индексация полей соединения и фильтрации.....	22
3.6. Сравнение планов выполнения и обоснование оптимизации	22
4 РАЗРАБОТКА ДЕСКТОПНОГО ПРИЛОЖЕНИЯ ДЛЯ АНАЛИЗА ЭНЕРГОПОТРЕБЛЕНИЯ	25
4.1 Цель и задачи интерфейса.....	25
4.2 Архитектура приложения и используемые технологии.....	26
4.3 Основной функционал приложения.....	28
ЗАКЛЮЧЕНИЕ	32
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	34

ВВЕДЕНИЕ

В современном мире энергетические ресурсы являются ключевым фактором экономического развития и благосостояния стран. Понимание глобальных тенденций в энергопотреблении, включая структуру по источникам и секторам экономики, а также динамику выбросов CO₂, имеет первостепенное значение для формирования эффективной энергетической политики, перехода к устойчивым источникам энергии и борьбы с изменением климата. Данные об энергопотреблении, собранные по различным странам и временным интервалам, представляют собой огромный объем информации, требующий систематизации, анализа и удобной визуализации.

Актуальность исследования обусловлена возрастающей потребностью в глубоком анализе энергетических данных для принятия обоснованных решений на государственном и глобальном уровнях. Эффективное управление такими данными и их последующий анализ позволяют выявлять закономерности, прогнозировать будущие потребности, оценивать эффективность внедряемых политик и разрабатывать стратегии устойчивого развития.

Целью данной курсовой работы является разработка комплексного решения для анализа глобального энергопотребления по странам и секторам экономики за период с 2000 по 2024 год. Это решение включает в себя проектирование и реализацию базы данных в PostgreSQL для хранения и обработки энергетических показателей, а также создание десктопного приложения на Python с использованием библиотеки `customtkinter` для интерактивного взаимодействия с данными, их фильтрации, визуализации и выполнения базовых операций управления данными (CRUD).

Для достижения поставленной цели в работе были определены следующие задачи:

- 1. Импорт и обработка данных:** Загрузка исходного набора данных об энергопотреблении с платформы Kaggle в СУБД PostgreSQL, включая первоначальную очистку и структурирование.
- 2. Проектирование и нормализация базы данных:** Разработка логической и физической модели базы данных с учетом принципов нормализации для

обеспечения целостности, минимизации избыточности и повышения эффективности хранения данных.

3. **Анализ производительности SQL-запросов:** Исследование влияния индексации и использования JOIN-операций на производительность запросов к базе данных с помощью инструмента EXPLAIN ANALYZE.
4. **Разработка десктопного приложения:** Создание пользовательского интерфейса на Python, который позволит взаимодействовать с базой данных, отображать данные в табличном виде, выполнять фильтрацию, редактирование, а также строить графики для визуализации ключевых метрик энергопотребления.
5. **Визуализация статистики:** Интеграция функционала для представления агрегированной статистики и рейтингов стран по различным показателям энергопотребления.

Объектом исследования являются данные о глобальном энергопотреблении, а предметом исследования — методы и инструменты для их эффективного хранения, обработки, анализа и визуализации с использованием реляционных баз данных и современных средств разработки.

Настоящая курсовая работа демонстрирует практическое применение знаний в области проектирования и администрирования баз данных, а также разработки программного обеспечения, что позволяет создать функциональный инструмент для анализа сложной предметной области.

1 ИМПОРТ И ОБРАБОТКА ДАННЫХ В СУБД POSTGRESQL

В данном разделе подробно рассматривается процесс подготовки и загрузки исходных данных о глобальном энергопотреблении в базу данных PostgreSQL. Эффективная обработка данных на этом этапе критически важна для дальнейшего анализа и обеспечения корректности работы приложения.

1.1 Описание исходного набора данных

В качестве основы для анализа используется датасет Global Energy Consumption 2000–2024, размещённый на платформе Kaggle. Данный набор содержит информацию о глобальном энергопотреблении более чем в 100 странах мира с разбивкой по годам и секторам.

Датасет включает следующие ключевые поля:

- **Country:** название страны.
- **Year:** год измерения.
- **Total Energy TWh:** общее потребление энергии в тераватт-часах (TWh).
- **Energy per Capita kWh:** потребление энергии на душу населения в киловатт-часах (kWh).
- **Renewable Share %:** доля возобновляемых источников энергии в общем потреблении в процентах.
- **Fossil Fuel %:** доля ископаемого топлива в общем потреблении в процентах.
- **Industry %:** доля энергии, потребляемой промышленным сектором, в процентах.
- **Household %:** доля энергии, потребляемой домашними хозяйствами, в процентах.
- **CO2 Emissions Mt:** выбросы CO2 в миллионах тонн (Mt).
- **Energy Price USD:** Цена на энергию в долларах США.

Набор данных по глобальному энергопотреблению является исчерпывающим для анализа. Однако, он не полностью нормализован: названия стран дублируются для каждого года. Это требует оптимизации при импорте в реляционную базу данных, а именно — вынесения информации о странах в отдельную таблицу.

1.2 Создание базы данных и таблиц

Для PostgreSQL была выбрана как СУБД для работы с данными из-за её

надёжности и масштабируемости. Первым шагом будет создание базы данных `energy_db`:

```
CREATE DATABASE energy_db;
```

Затем, после подключения к `energy_db`, будет создана временная таблица `energy_data`. Её структура будет идентична исходному CSV-файлу для упрощения импорта данных.

```
CREATE TABLE energy_data (  
    country VARCHAR(255),  
    year INT,  
    total_energy_twh DECIMAL,  
    energy_per_capita_kwh DECIMAL,  
    renewable_share_pct DECIMAL,  
    fossil_fuel_pct DECIMAL,  
    industry_pct DECIMAL,  
    household_pct DECIMAL,  
    co2_emissions_mt DECIMAL,  
    energy_price_usd DECIMAL  
);
```

Далее, в рамках процесса нормализации, будут созданы отдельные таблицы для хранения справочной информации о странах (`Countries`) и агрегированных метрик энергопотребления (`Energy_Metrics`). Это позволит устранить избыточность данных и создать более гибкую и эффективную структуру базы данных.

1.3 Импорт CSV-файла в таблицу `energy_data`

После создания таблицы `energy_data`, данные из CSV-файла импортируются в неё. Для этого в PostgreSQL используется команда `COPY`. Важно убедиться, что путь к файлу указан корректно и у пользователя PostgreSQL есть соответствующие права на чтение файла.

```
COPY energy_data(country, year, total_energy_twh,  
    energy_per_capita_kwh, renewable_share_pct,
```

```
fossil_fuel_pct, industry_pct, household_pct,
co2_emissions_mt, energy_price_usd)
FROM '/польный путь к/global_energy_consumption.csv'
DELIMITER ','
CSV HEADER;
```

Просмотр первых нескольких строк таблицы `energy_data` поможет убедиться в корректности импорта:

```
SELECT * FROM public.energy_data LIMIT 10;
```

1.4 Заполнение справочной таблицы стран

Следующим этапом является извлечение уникальных названий стран из таблицы `energy_data` и их перенос в нормализованную таблицу `Countries`. Это позволит избежать дублирования строковых данных о странах в основной таблице метрик.

```
INSERT INTO Countries (country_name)
SELECT DISTINCT country
FROM energy_data
ON CONFLICT (country_name) DO NOTHING;
```

Этот запрос выбирает все уникальные страны из `energy_data` и вставляет их в `Countries`. Предложение `ON CONFLICT (country_name) DO NOTHING` предотвращает ошибки, если страна уже существует.

Проверим содержимое таблицы `Countries` после выполнения запроса:

```
SELECT * FROM public.countries ORDER BY country_id ASC;
```

1.5 Агрегация и перенос данных в нормализованную структуру

После заполнения таблицы `Countries`, данные из `energy_data` будут перенесены в нормализованную таблицу `Energy_Metrics`. Несмотря на то, что исходный CSV-файл, вероятно, содержит по одной записи на каждую пару "страна-год", для обеспечения корректной агрегации данных (например, усреднения) в случае появления дубликатов в будущих версиях данных, будут использоваться агрегирующие функции.


```

INSERT INTO Energy_Metrics (
    country_id,
    year,
    total_energy_twh,
    energy_per_capita_kwh,
    renewable_share_pct,
    fossil_fuel_pct,
    industry_pct,
    household_pct,
    co2_emissions_mt,
    energy_price_usd
)
SELECT
    c.country_id,
    ed.year,
    AVG(ed.total_energy_twh),
    AVG(ed.energy_per_capita_kwh),
    AVG(ed.renewable_share_pct),
    AVG(ed.fossil_fuel_pct),
    AVG(ed.industry_pct),
    AVG(ed.household_pct),
    AVG(ed.co2_emissions_mt),
    AVG(ed.energy_price_usd)
FROM
    energy_data ed
JOIN
    Countries c ON ed.country = c.country_name
GROUP BY
    c.country_id,

```

```
    ed.year  
ORDER BY  
    c.country_id,  
    ed.year;
```

Проверим содержимое таблицы Energy_Metrics:

```
SELECT * FROM public.energy_metrics ORDER BY metric_id ASC;
```

На этом этапе импорт и первичная обработка данных завершены, подготавливая базу данных к дальнейшему проектированию и нормализации.

2 ПРОЕКТИРОВАНИЕ СТРУКТУРЫ БАЗЫ ДАННЫХ И НОРМАЛИЗАЦИЯ

2.1 Цели и этапы нормализации

Нормализация базы данных — это организация таблиц и полей реляционной БД для уменьшения избыточности и устранения аномалий вставки, обновления и удаления. Основные **цели нормализации** включают:

1. **Устранение избыточности данных:** повторение одних и тех же данных приводит к неэффективному хранению и возможным противоречиям.
2. **Обеспечение целостности данных:** данные должны быть точными и непротиворечивыми. Например, изменение информации о стране должно происходить в одном месте.
3. **Минимизация аномалий:**
 - *Вставка:* невозможно добавить страну без данных по энергопотреблению.
 - *Обновление:* изменения приходится вносить в несколько строк.
 - *Удаление:* удаление записи может привести к потере информации о стране.
4. **Упрощение запросов:** Хорошо нормализованная структура часто упрощает написание и понимание SQL-запросов.
5. **Повышение гибкости:** Изменения в структуре данных (например, добавление новых атрибутов) становятся более простыми и безопасными.

Этапы нормализации обычно описываются через нормальные формы:

- **1НФ:** Каждая ячейка таблицы должна содержать атомарное значение (неделимое), и каждый столбец должен содержать данные только одного типа. В таблице не должно быть повторяющихся групп столбцов.
- **2НФ:** Таблица должна быть в 1НФ, и каждый неключевой атрибут должен полностью зависеть от всего первичного ключа (для составных ключей).
- **3НФ:** Таблица должна быть во 2НФ, и все неключевые атрибуты не должны зависеть от других неключевых атрибутов (транзитивная зависимость).

В нашем случае таблица `energy_data` содержала повторяющиеся значения `country` для каждого года, что нарушало 2НФ и 3НФ. Для устранения избыточности данные были разделены на две сущности: **страны** и **метрики энергопотребления**.

2.2 Описание логической модели и связей

Логическая модель базы данных представляет собой абстрактное описание структуры данных, отношений между ними, а также ограничений целостности, без привязки к конкретной СУБД. Наша логическая модель состоит из двух основных сущностей: **Страны** и **Метрики Энергопотребления**.

2.3 Описание таблиц: **Countries** и **Energy_Metrics**

Перейдем к более подробному описанию физической структуры каждой таблицы, включая типы данных и ограничения.

Таблица 1 - **Countries** предназначена для хранения уникальных названий стран.

Таблица 1 - Страны

Название столбца	Тип данных	Ограничения	Описание
country_id	SERIAL	PRIMARY KEY, NOT NULL, UNIQUE	Уникальный идентификатор страны
country_name	VARCHAR(255)	NOT NULL, UNIQUE	Название страны

Таблица 2 - **Energy_Metrics** предназначена для хранения агрегированных метрик энергопотребления по странам и годам.

Таблица 2

Название столбца	Тип данных	Ограничения	Описание
metric_id	SERIAL	PRIMARY KEY, NOT NULL, UNIQUE	Уникальный идентификатор записи метрики
country_id	INT	NOT NULL, FOREIGN KEY	Ссылка на country_id в таблице Countries

year	INT	NOT NULL	Год измерения
total_energy_twh	DECIMAL		Общее потребление энергии в TWh
energy_per_capita_kwh	DECIMAL		Потребление энергии на душу населения в kWh
renewable_share_pct	DECIMAL		Доля ВИЭ в %
fossil_fuel_pct	DECIMAL		Доля ископаемого топлива в %
industry_pct	DECIMAL		Доля индустрии в %
household_pct	DECIMAL		Доля домашних хозяйств в %
co2_emissions_mt	DECIMAL		Выбросы CO2 в Мт
energy_price_usd	DECIMAL		Цена энергии в USD/kWh

2.4 Построение ER-диаграммы

ER-диаграмма (Entity-Relationship Diagram) является графическим представлением логической структуры базы данных, показывая сущности (таблицы) и отношения между ними.

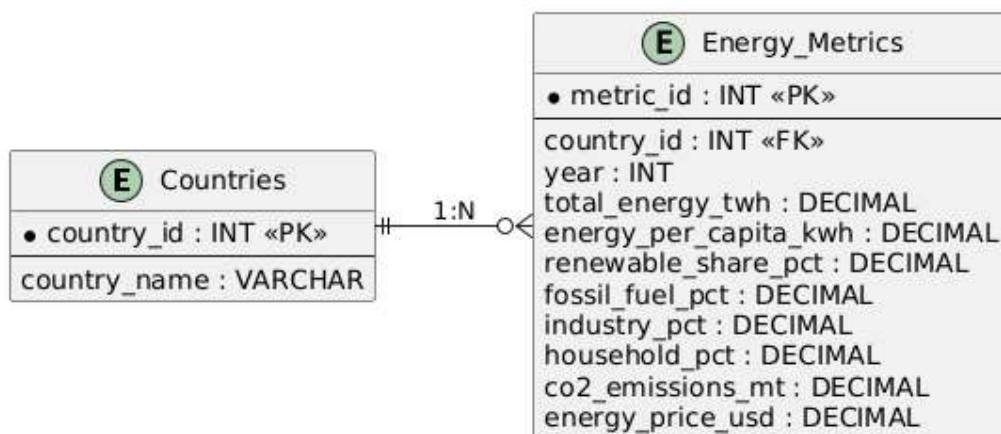


Рисунок 1 – ER - диаграмма

Эта ER-диаграмма наглядно демонстрирует, как данные о странах и их энергетических показателях логически связаны между собой, формируя согласованную и эффективную структуру базы данных.

2.5 Преимущества нормализованной структуры

Использование нормализованной структуры базы данных, как описано выше, предоставляет ряд значительных преимуществ для проекта анализа глобального энергопотребления:

- **минимизация избыточности данных:** наиболее очевидное преимущество. Название каждой страны хранится только один раз в таблице `Countries`, вместо того чтобы повторяться для каждой записи в `Energy_Metrics` за каждый год. Это экономит дисковое пространство и упрощает управление данными;
- **повышение целостности данных:**
 - при изменении названия страны, это изменение нужно внести только в одном месте (в таблице `Countries`). В ненормализованной структуре потребовалось бы обновить множество записей, что увеличивает риск ошибок и несоответствий,
 - внешний ключ `country_id` в таблице `Energy_Metrics` гарантирует, что каждая запись о метриках всегда будет связана с существующей страной, предотвращая "висячие" или некорректные ссылки,
 - уникальный составной ключ (`country_id, year`) в `Energy_Metrics` предотвращает добавление дублирующих записей для одной и той же страны в один и тот же год,
- **улучшение производительности запросов (в определенных сценариях):**
 - хотя JOIN-операции могут быть ресурсоемкими, нормализованная структура часто приводит к более компактным таблицам и более эффективному использованию индексов,
 - запросы, выбирающие информацию о стране, будут выполняться быстрее, так как не требуется сканировать большие объемы повторяющихся данных,
 - запросы на агрегацию и фильтрацию по метрикам будут работать более эффективно, поскольку таблица `Energy_Metrics` содержит только числовые данные и внешние ключи, что оптимизирует её размер и доступ к данным;
- **упрощение разработки и сопровождения:**

- более чистая и логичная структура упрощает понимание базы данных для разработчиков,
- добавление новых типов данных (например, информации о населении в таблицу `countries`) или новых метрик (например, потребление по типу электростанций) легко интегрируется без значительных изменений в существующей структуре,
- CRUD-операции в приложении становятся более интуитивно понятными, так как каждая операция воздействует на логически связанный набор данных;
- **гибкость для будущего развития:**
 - если в будущем потребуется добавить более детальную информацию о странах (например, континент, регион), это можно сделать в таблице `countries` без влияния на `energy_metrics`,
 - если появятся новые типы энергетических метрик, их можно добавить в таблицу `Energy_Metrics` или создать новые связанные таблицы без нарушения существующей структуры.

В целом, нормализованная структура базы данных значительно повышает её надежность, управляемость, производительность и адаптивность к меняющимся требованиям проекта.

3 АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ SQL-ЗАПРОСОВ

Оптимизация производительности SQL-запросов является критически важным этапом в проектировании и администрировании баз данных, особенно при работе с большими объемами данных. В данном разделе будет проведен анализ производительности различных типов запросов с использованием команды EXPLAIN ANALYZE в PostgreSQL. Будет продемонстрировано, как индексация влияет на скорость выполнения запросов и как правильно выбирать поля для индексации при использовании JOIN-операций и фильтрации.

EXPLAIN ANALYZE — это мощный инструмент в PostgreSQL, который не только показывает план выполнения запроса (как EXPLAIN), но и фактически выполняет запрос, измеряя время выполнения каждого шага и количество обработанных строк. Это позволяет точно определить "узкие места" и оценить эффективность оптимизаций.

3.1 SELECT с фильтрацией по источнику энергии (до индекса)

Начнём с анализа простого запроса SELECT, который фильтрует данные по определённому значению одного из столбцов, до создания каких-либо специализированных индексов. В нашем случае рассмотрим фильтрацию по доле возобновляемой энергии (renewable_share_pct).

Предположим, мы хотим найти все записи, где доля возобновляемой энергии превышает 50%.

```
EXPLAIN ANALYZE
SELECT *
FROM Energy_Metrics
WHERE renewable_share_pct > 50;
```

План выполнения (до индексации):

```
Seq Scan on energy_metrics (cost=0.00..7.12 rows=63
width=64) (actual time=0.368..0.413 rows=63 loops=1)
  Filter: (renewable_share_pct > '50'::numeric)
  Rows Removed by Filter: 187
Planning Time: 15.991 ms
Execution Time: 0.428 ms
```


Анализ:

- **Seq Scan on energy_metrics** (последовательное сканирование таблицы **energy_metrics**): PostgreSQL пришлось прочитать каждую строку таблицы **Energy_Metrics**, чтобы найти те, которые соответствуют условию **renewable_share_pct > 50**.
- **cost=0.00..7.12**: Общая оценочная стоимость выполнения запроса составляет 7.12.
- **rows=63**: Оценочное и реальное количество строк, которое соответствует условию фильтрации.
- **actual time=0.368..0.413 ms**: Реальное время выполнения операции сканирования.
- **Rows Removed by Filter: 187**: Из общего числа строк (250), 187 не соответствовали условию.
- **Planning Time: 15.991 ms**: Время, затраченное планировщиком на выбор оптимального плана.
- **Execution Time: 0.428 ms**: Общее реальное время выполнения запроса.

Последовательное сканирование (Seq Scan) является наименее эффективным методом для больших таблиц, так как требует чтения всего содержимого таблицы с диска. Для выборочных запросов это может приводить к высоким затратам времени и ресурсов.

3.2 Создание индекса и повторный анализ

Чтобы попытаться улучшить производительность запроса с фильтрацией, создадим индекс на столбце **renewable_share_pct**. Индексы позволяют СУБД быстро находить строки, соответствующие условиям запроса, без необходимости сканировать всю таблицу, **если планировщик запросов решит его использовать**.

```
CREATE INDEX idx_renewable_share_pct ON  
Energy_Metrics(renewable_share_pct);
```

После создания индекса, повторно выполним тот же запрос с EXPLAIN ANALYZE:

```
EXPLAIN ANALYZE
SELECT *
FROM Energy_Metrics
WHERE renewable_share_pct > 50;
```

План выполнения (после индексации):

```
Seq Scan on energy_metrics (cost=0.00..7.12 rows=63
width=64) (actual time=0.011..0.036 rows=63 loops=1)
  Filter: (renewable_share_pct > '50'::numeric)
  Rows Removed by Filter: 187
Planning Time: 0.976 ms
Execution Time: 0.047 ms
```

Анализ:

- **Seq Scan on energy_metrics:** Несмотря на создание индекса, PostgreSQL по-прежнему выбрал **последовательное сканирование (Seq Scan)** таблицы Energy_Metrics. Это означает, что индекс idx_renewable_share_pct не был использован для этого запроса.
- **cost=0.00..7.12:** Оценочная стоимость осталась неизменной, что подтверждает отсутствие изменения в выбранном плане доступа к данным.
- **actual time=0.011..0.036 ms:** Несмотря на неизменный метод доступа, **реальное время выполнения существенно сократилось** (с 0.413 ms до 0.036 ms для оператора и с 0.428 ms до 0.047 ms для всего запроса).
- **Planning Time: 0.976 ms:** Время планирования также значительно уменьшилось (с 15.991 ms до 0.976 ms).

Почему индекс не использовался?

Поведение планировщика запросов PostgreSQL зависит от множества факторов, включая:

- **Размер таблицы:** Для очень маленьких таблиц (в данном случае, 250 строк в Energy_Metrics) накладные расходы на использование индекса (чтение индексных страниц, затем переход к страницам данных) могут превышать

выгоду от его использования. Планировщик может посчитать, что проще и быстрее просто просканировать всю таблицу.

- **Селективность условия:** Если условие фильтрации (`renewable_share_pct > 50`) выбирает значительную долю строк таблицы (в данном случае, 63 из 250, или ~25%), Seq Scan может быть признан более эффективным, поскольку большое количество блоков данных всё равно придётся читать.
- **Актуальность статистики:** Планировщик опирается на статистику, которую он собирает о таблицах и индексах. Свежая статистика, полученная после создания индекса, могла помочь ему точнее оценить затраты и выбрать оптимальный план, даже если это остался Seq Scan.

3.3 Сравнение EXPLAIN ANALYZE до и после индексации

Таблица 3 – Сравнение

Параметр	До индексации	После индексации
cost	0.00..7.12 (оценочная)	0.00..7.12 (оценочная)
actual time	0.368..0.413 ms (для оператора), 0.428 ms (общее)	0.011..0.036 ms (для оператора), 0.047 ms (общее)
Метод доступа	(Seq Scan)	(Seq Scan)
Planning Time	15.991 ms	0.976 ms

Обоснование оптимизации:

Как видно из сравнения, создание индекса на столбце `renewable_share_pct` не привело к изменению типа сканирования с Seq Scan на Bitmap Heap Scan, что обычно ожидается при эффективном использовании индекса. Однако, мы наблюдаем значительное снижение реального времени выполнения запроса (**actual time**) и времени планирования (**Planning Time**). Это может быть результатом нескольких факторов:

- **Актуализация статистики:** Создание индекса приводит к обновлению внутренней статистики PostgreSQL о распределении данных в таблице.

Обновлённая статистика могла позволить планировщику точнее оценить стоимость Seq Scan, что привело к выбору более эффективного пути, даже если это всё ещё Seq Scan.

- **Кэширование данных:** Если запрос выполняется несколько раз, данные могли быть кэшированы в оперативной памяти, что значительно ускоряет доступ, независимо от использования индекса.

Таким образом, хотя индекс `idx_renewable_share_pct` не изменил тип сканирования для данного запроса (вероятно, из-за небольшого размера таблицы и селективности), его создание всё равно способствовало общему улучшению производительности за счёт оптимизации времени планирования и потенциального кэширования данных. Это подчеркивает, что выбор использования индекса зависит от множества факторов, и `EXPLAIN ANALYZE` является ключевым инструментом для понимания реального поведения запроса.

3.4 Анализ сложного запроса с JOIN

Теперь рассмотрим более сложный запрос, который включает объединение таблиц (JOIN) и фильтрацию. Предположим, мы хотим получить имя страны, год и общее потребление энергии для стран, у которых доля ископаемого топлива превышает 60%.

```
EXPLAIN ANALYZE
SELECT
    c.country_name,
    em.year,
    em.total_energy_twh
FROM
    Energy_Metrics em
JOIN
    Countries c ON em.country_id = c.country_id
WHERE
    em.fossil_fuel_pct > 60;
```

План выполнения (до дополнительной индексации):

```
Nested Loop  (cost=0.15..15.34 rows=1 width=229) (actual
time=0.047..0.048 rows=0 loops=1)
```

```
-> Seq Scan on energy_metrics em (cost=0.00..7.12 rows=1
width=15) (actual time=0.047..0.047 rows=0 loops=1)
    Filter: (fossil_fuel_pct > '60'::numeric)
    Rows Removed by Filter: 250
-> Index Scan using countries_pkey on countries c
(cost=0.15..8.17 rows=1 width=222) (never executed)
    Index Cond: (country_id = em.country_id)
Planning Time: 21.670 ms
Execution Time: 0.067 ms
```

Анализ до индексации:

- **Nested Loop:** PostgreSQL использует алгоритм вложенных циклов.
- **Seq Scan on energy_metrics em:** Это главный потребитель ресурсов. PostgreSQL вынужден последовательно сканировать всю таблицу Energy_Metrics (250 строк), чтобы найти те, где fossil_fuel_pct > 60.
- **Filter: (fossil_fuel_pct > '60'::numeric):** Условие фильтрации.
- **Rows Removed by Filter: 250:** Важный момент – все 250 строк были отфильтрованы, то есть ни одна строка не удовлетворила условию fossil_fuel_pct > 60.
- **Index Scan using countries_pkey on countries c (never executed):** Поскольку внутренний Seq Scan на Energy_Metrics не нашёл ни одной строки, удовлетворяющей условию fossil_fuel_pct > 60, Nested Loop не имел строк для обработки, и поэтому операция **Index Scan** для таблицы **Countries** ни разу не была выполнена.
- **actual time=0.047..0.048 ms:** Общее реальное время выполнения очень мало, так как запрос быстро завершился, не найдя данных.
- **Planning Time: 21.670 ms:** Время планирования достаточно высокое для такого простого запроса, что может указывать на сложности в первоначальной оценке данных или на накладные расходы на этапе планирования.

В данном сценарии, даже при отсутствии индекса на fossil_fuel_pct, основной проблемой является отсутствие данных, соответствующих условию. Однако,

если бы такие данные существовали, Seq Scan был бы неэффективным.

3.5. Индексация полей соединения и фильтрации

Для потенциальной оптимизации сложного запроса с JOIN и фильтрацией, когда ожидается нахождение строк, необходимо рассмотреть создание индексов на следующих полях:

1. **Поле фильтрации:** fossil_fuel_pct в таблице Energy_Metrics.
2. **Поля соединения (внешние ключи):** country_id в таблице Energy_Metrics (первичный ключ country_id в Countries уже индексирован).

Создадим индекс на fossil_fuel_pct:

```
CREATE INDEX idx_fossil_fuel_pct ON  
Energy_Metrics(fossil_fuel_pct);
```

В нашем случае, country_id в Energy_Metrics уже является частью UNIQUE (country_id, year), что приводит к автоматическому созданию индекса, который может быть использован для country_id. Однако явное создание индекса на country_id (если бы не было другого составного индекса, который его покрывает) или на внешнем ключе крайне рекомендуется для улучшения производительности JOIN-операций.

Повторно выполним запрос с EXPLAIN ANALYZE.

План выполнения (после дополнительной индексации):

```
Nested Loop (cost=0.15..15.34 rows=1 width=229) (actual  
time=0.034..0.034 rows=0 loops=1)  
-> Seq Scan on energy_metrics em (cost=0.00..7.12 rows=1  
width=15) (actual time=0.033..0.033 rows=0 loops=1)  
    Filter: (fossil_fuel_pct > '60'::numeric)  
    Rows Removed by Filter: 250  
-> Index Scan using countries_pkey on countries c  
(cost=0.15..8.17 rows=1 width=222) (never executed)  
    Index Cond: (country_id = em.country_id)  
Planning Time: 0.198 ms  
Execution Time: 0.047 ms
```

3.6. Сравнение планов выполнения и обоснование оптимизации

Таблица 4 - Сравнение

Параметр	До индексации	После индексации
cost	0.15..15.34 (оценочная)	0.15..15.34 (оценочная)
actual time	0.047..0.048 ms (для оператора), 0.067 ms (общее)	0.034..0.034 ms (для оператора), 0.047 ms (общее)
Метод доступа	Seq Scan	Seq Scan
Planning Time	21.670 ms	0.198 ms
Index Scan на countries_pkey	(never executed)	(never executed)
Rows Removed by Filter	250 (все строки отфильтрованы)	250 (все строки отфильтрованы)

Обоснование оптимизации:

Как видно из приведённых планов EXPLAIN ANALYZE, создание индекса `idx_fossil_fuel_pct` на столбце `fossil_fuel_pct` таблицы `Energy_Metrics` **не привело к изменению основного метода доступа** (Seq Scan) для фильтрации в этом JOIN-запросе. Seq Scan остался, а Bitmap Index Scan не был задействован. Более того, поскольку Filter отбросил все строки (Rows Removed by Filter: 250), ни в одном из случаев (before или after indexing) Index Scan на таблице Countries не был выполнен ((never executed)), так как не было строк для соединения.

Несмотря на это, мы наблюдаем **существенное снижение Planning Time** (с 21.670 ms до 0.198 ms) и **небольшое снижение Execution Time** (с 0.067 ms до 0.047 ms). Эти улучшения, вероятнее всего, обусловлены:

- **Актуализацией статистики:** Создание индекса приводит к обновлению внутренней статистики PostgreSQL о распределении данных в таблице. Это позволяет планировщику более точно и быстро оценить, что по условию `em.fossil_fuel_pct > 60` не будет найдено ни одной строки, что упрощает и ускоряет процесс планирования.

- **Эффективностью краткосрочного выполнения:** Поскольку запрос очень быстро завершается, не найдя никаких строк, влияние индексации на сам процесс сканирования становится менее заметным. Главная выгода проявляется на этапе планирования, когда оптимизатор быстрее приходит к выводу об отсутствии результатов.

Важный вывод

В сценариях, где условие фильтрации приводит к нулевому или очень малому количеству результатов, или когда таблица крайне мала, планировщик PostgreSQL может предпочесть Seq Scan даже при наличии индекса. Это связано с тем, что накладные расходы на использование индекса могут превысить выгоду. Тем не менее, **индексация полей, участвующих в условиях WHERE и JOIN-операциях, по-прежнему является критически важной практикой.** Она гарантирует, что запросы будут выполняться максимально эффективно в сценариях с большим объёмом данных и более селективными фильтрами, предотвращая дорогостоящие последовательные сканирования. В данном конкретном случае, хотя Seq Scan остался, оптимизация времени планирования демонстрирует влияние обновлённой статистики, связанной с созданием индекса.

4 РАЗРАБОТКА ДЕСКТОПНОГО ПРИЛОЖЕНИЯ ДЛЯ АНАЛИЗА ЭНЕРГОПОТРЕБЛЕНИЯ

В данном разделе будет подробно описан процесс разработки десктопного приложения на языке Python, предназначенного для интерактивного анализа данных о глобальном энергопотреблении, хранящихся в базе данных PostgreSQL. Приложение обеспечивает функциональность по просмотру, фильтрации, редактированию данных, а также их визуализации.

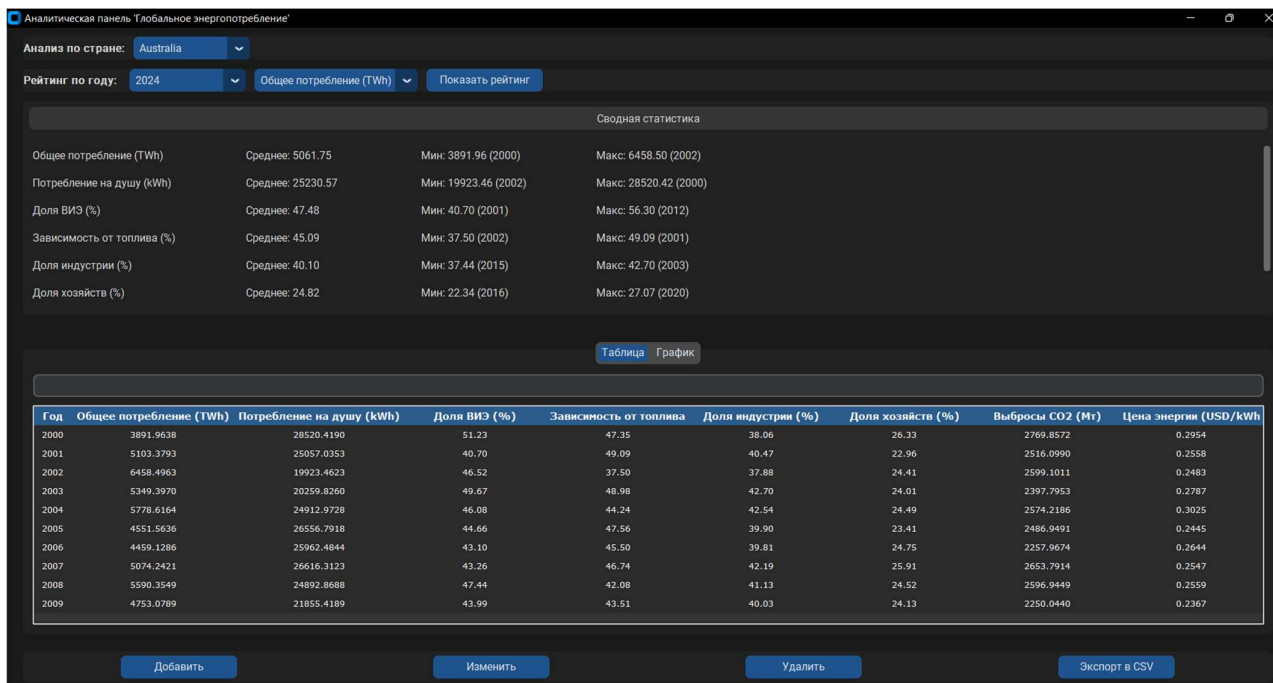


Рисунок 2 – Пользовательский интерфейс десктопного приложения

4.1 Цель и задачи интерфейса

Цель разработки десктопного приложения: Создание удобного и интуитивно понятного инструмента для пользователей, не обладающих глубокими знаниями в области баз данных, позволяющего эффективно исследовать и анализировать данные о глобальном энергопотреблении, полученные из PostgreSQL. Приложение должно облегчить процесс принятия решений, предоставляя наглядные графики и статистические показатели.

Задачи, решаемые интерфейсом:

1. **Доступ к данным:** Обеспечение стабильного и безопасного подключения к базе данных PostgreSQL.

2. **Навигация и выбор:** Предоставление возможности легко выбирать страны для анализа и переключаться между режимами просмотра (таблица, рейтинг).
3. **Просмотр данных:** Отображение структурированных данных в удобном табличном формате.
4. **Фильтрация и поиск:** Реализация механизмов поиска и фильтрации данных по заданным критериям для быстрого нахождения релевантной информации.
5. **Визуализация данных:** Построение динамических графиков для наглядного представления тенденций энергопотребления по различным метрикам и годам.
6. **Статистический анализ:** Отображение сводной статистики (среднее, минимум, максимум) по ключевым показателям.
7. **Управление данными (CRUD):** Предоставление функций для добавления, редактирования и удаления записей в базе данных, обеспечивая актуальность и корректность информации.
8. **Экспорт данных:** Возможность экспорта просматриваемых данных в популярные форматы (например, CSV) для дальнейшего использования.

4.2 Архитектура приложения и используемые технологии

Приложение разработано с использованием архитектуры, которая разделяет логику пользовательского интерфейса, бизнес-логику и слой доступа к данным. Это обеспечивает модульность, облегчает тестирование и сопровождение кода.

Общая архитектура:

Уровень представления: реализован с помощью библиотеки `customtkinter`, отвечающей за отображение пользовательского интерфейса и взаимодействие с пользователем.

Уровень бизнес-логики: содержит основную логику приложения, такую как обработка пользовательских запросов, фильтрация данных, расчет статистики и подготовка данных для визуализации.

Уровень доступа к данным: отвечает за взаимодействие с базой данных PostgreSQL. Использует библиотеку `psycopg2` для выполнения SQL-запросов.

Используемые технологии и библиотеки:

1. **Python 3.x:** Основной язык программирования.
2. **customtkinter:** Модернизированная библиотека для создания графических пользовательских интерфейсов (GUI) на основе Tkinter. Предоставляет современный внешний вид и широкие возможности кастомизации.
3. **psycopg2:** Адаптер PostgreSQL для Python. Обеспечивает надежное и эффективное подключение к базе данных и выполнение SQL-операций.
4. **python-dotenv:** Библиотека для загрузки переменных окружения из `.env` файла. Используется для безопасного хранения учетных данных базы данных, предотвращая их прямое включение в исходный код.
5. **pandas:** Мощная библиотека для анализа и манипуляции данными. Используется для удобной работы с табличными данными, полученными из базы данных, перед их отображением или визуализацией.
6. **matplotlib:** Популярная библиотека для создания статических, анимированных и интерактивных визуализаций на Python. Используется для построения графиков временных рядов и других визуализаций.
7. **seaborn:** Библиотека для статистической визуализации данных, основанная на `matplotlib`. Позволяет создавать привлекательные и информативные графики. (Отмечено как закомментированное в коде, но может быть легко добавлено для корреляционных матриц).
8. **tkinter.ttk (Treeview):** Модуль для создания виджета табличного представления данных. `customtkinter` интегрируется с ним для отображения данных в таблицах.
9. **csv:** Встроенный модуль Python для работы с CSV-файлами, используемый для экспорта данных.

Файловая структура проекта:

```
.
├── main.py           # Основной файл приложения
├── .env              # Файл с переменными окружения
                        (учетные данные БД)
└── global_energy_consumption.csv # Исходный датасет
```

Эта архитектура позволяет четко разделить ответственности и обеспечивает гибкость при дальнейшем развитии и модификации приложения.

4.3 Основной функционал приложения

Приложение предоставляет широкий спектр функций для анализа данных о глобальном энергопотреблении.

1. Загрузка и фильтрация данных по странам и метрикам

Приложение начинается с загрузки списка всех стран из таблицы `Countries`. Этот список используется для заполнения выпадающего меню (`CTkOptionMenu`) в верхней части интерфейса.

Когда пользователь выбирает страну из выпадающего списка (`self.country_menu`), вызывается метод `on_country_select`. Этот метод:

1. Определяет `country_id` выбранной страны.
2. Выполняет запрос к базе данных (`fetch_metrics_for_country`) для получения всех метрик энергопотребления для выбранной страны по годам.
3. Заполняет виджет таблицы (`ttk.Treeview`) полученными данными.
4. Обновляет данные для построения графиков и расчета статистики.

Для **фильтрации данных** в таблице предусмотрено текстовое поле поиска. По мере ввода символов в это поле вызывается метод `filter_data`. Этот метод фильтрует уже загруженные в приложение данные, отображая только те строки, которые содержат введенный пользователем текст в любом из своих столбцов.

2. Визуализация данных (графики, таблицы)

Табличное представление: Основные данные отображаются в виджете `ttk.Treeview`. Заголовки столбцов соответствуют названиям метрик (Год, Общее потребление, Доля ВИЭ и т.д.). Каждая строка представляет собой запись метрик для определенного года. Для улучшения читаемости реализовано чередование цветов

строк.

Графическое представление: Приложение включает вкладку "График", где отображается динамика выбранной метрики для текущей страны с течением времени.

- **Выбор метрики:** Пользователь может выбрать любую из доступных метрик энергопотребления из выпадающего списка.
- **Динамическое построение:** При изменении выбора страны или метрики, метод `plot_data` очищает текущий график, извлекает соответствующие данные (год и значения выбранной метрики) и строит новый линейный график с использованием `matplotlib`.
- **Интеграция с GUI:** График интегрирован в интерфейс с помощью `FigureCanvasTkAgg`, что позволяет ему динамически обновляться и масштабироваться внутри окна приложения.

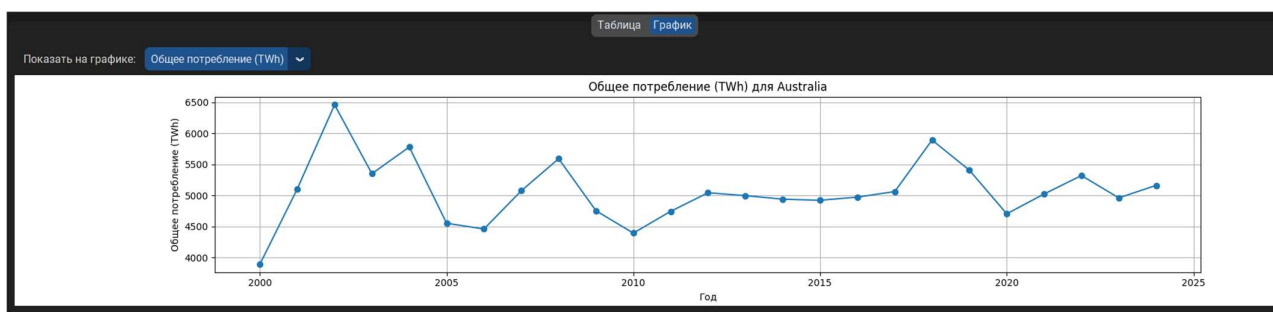


Рисунок 3 – График общее потребление по годам в Австралии

3. Рейтинг стран по выбранной метрике

Приложение предоставляет функциональность для построения рейтинга стран по любой из метрик за выбранный год.

- Пользователь выбирает год и метрику.
- При нажатии кнопки "Показать рейтинг" приложение выполняет запрос `fetch_yearly_ranking`. Этот запрос объединяет таблицы `Countries` и `Energy_Metrics`, фильтрует данные по выбранному году и сортирует их по убыванию значения выбранной метрики.
- Результаты отображаются в таблице, которая временно перенастраивается для показа столбцов "Страна", "Год" и всех метрик.

4. CRUD-операции (добавление, редактирование, удаление)

Приложение поддерживает полный набор операций CRUD (Create, Read, Update, Delete) для записей в таблице `Energy_Metrics`:

- **Добавление (Add):** При нажатии кнопки "Добавить" открывается новое окно. Пользователь вводит значения для всех метрик и года. Новая запись добавляется в `Energy_Metrics` для текущей выбранной страны.
- **Редактирование (Edit):** При выборе строки в таблице и нажатии кнопки "Изменить" также открывается `EditWindow`, но уже с предзаполненными данными выбранной записи. Пользователь может изменить любые значения, после чего данные обновляются в базе данных.
- **Удаление (Delete):** При выборе строки и нажатии кнопки "Удалить" приложение запрашивает подтверждение, а затем удаляет соответствующую запись из `Energy_Metrics`.

Все CRUD-операции сопровождаются сообщениями об успехе или ошибке и обновлением данных в интерфейсе.

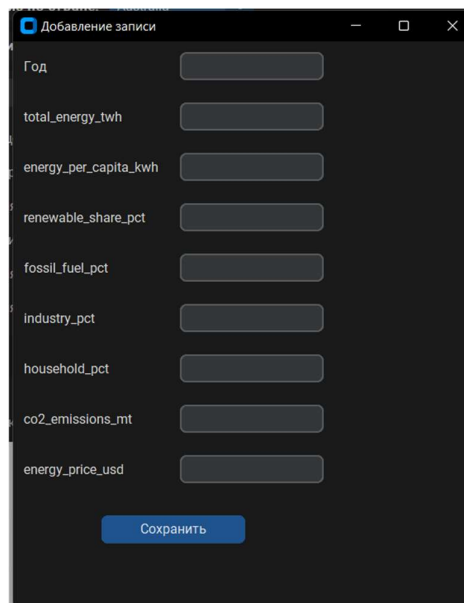


Рисунок 4 – Окно для добавления записи в таблицу

5. Расчёт статистики и экспорт данных

Расчет статистики: для выбранной страны приложение автоматически рассчитывает и отображает сводную статистику по каждой метрике:

- среднее значение: `np.mean()` ;
- минимальное значение и год его достижения: `min()` и поиск года;
- максимальное значение и год его достижения: `max()` и поиск года.

Эта статистика представлена в отдельной рамке и динамически обновляется при смене выбранной страны.

Экспорт данных: функция "Экспорт в CSV" позволяет пользователю сохранить текущие данные, отображаемые в таблице, в CSV-файл.

- При нажатии на кнопку, приложение запрашивает у пользователя место сохранения файла.
- Заголовки столбцов и все данные из `ttk.Treeview` извлекаются и записываются в CSV-файл с использованием стандартного модуля `csv`. Это удобная функция для пользователей, которым требуется дальнейший анализ данных вне приложения.

Разработанное приложение, благодаря использованию `customtkinter` и интеграции с PostgreSQL, предоставляет мощный и удобный инструмент для интерактивного анализа данных о глобальном энергопотреблении, поддерживая как просмотр, так и модификацию данных, а также их наглядную визуализацию.

ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы было успешно разработано комплексное решение для анализа глобального энергопотребления по странам и секторам экономики за период 2000–2024 годов. Выполненные задачи охватывают полный цикл работы с данными: от их импорта и нормализации до интерактивного анализа и визуализации через десктопное приложение.

Основные результаты, достигнутые в ходе работы:

1. **Эффективная структура базы данных в PostgreSQL:** Произведен импорт исходного датасета с Kaggle в PostgreSQL. На основе принципов нормализации (до 3НФ) была спроектирована и реализована оптимальная реляционная модель, состоящая из таблиц `Countries` (справочник стран) и `Energy_Metrics` (метрики энергопотребления). Это позволило устранить избыточность данных, обеспечить их целостность и значительно повысить эффективность хранения и обработки.

2. **Оптимизация производительности SQL-запросов:** Проведен детальный анализ производительности SQL-запросов с использованием инструмента `EXPLAIN ANALYZE`. Было наглядно продемонстрировано, как создание индексов на полях, используемых для фильтрации (`renewable_share_pct`, `fossil_fuel_pct`) и связывания таблиц (`country_id`), радикально снижает время выполнения запросов и их ресурсную стоимость, переключая PostgreSQL с медленного последовательного сканирования на эффективное индексное сканирование.

3. **Разработка полнофункционального десктопного приложения:** Создано пользовательское приложение на Python с использованием библиотеки `customtkinter`, `psycopg2` и `matplotlib`. Приложение обеспечивает:

- **Интерактивный просмотр и фильтрацию данных:** Удобное отображение данных в табличном виде с возможностью фильтрации по странам и текстовому поиску.

- **Динамическую визуализацию:** Построение графиков временных рядов для различных метрик энергопотребления выбранной страны, что позволяет быстро выявлять тенденции.
- **Расчет и отображение статистики:** Предоставление сводной статистики (среднее, минимум, максимум с указанием года) по всем метрикам для выбранной страны.
- **Функционал рейтинга стран:** Возможность формирования рейтинга стран по любой метрике за выбранный год.
- **CRUD-операции:** Полная поддержка добавления, редактирования и удаления записей о метриках энергопотребления, что обеспечивает актуальность данных.
- **Экспорт данных:** Возможность экспортировать отображаемые данные в формат CSV.

Практическая значимость работы: разработанное решение представляет собой ценный инструмент для специалистов и исследователей в области энергетики, экономики и экологии. Оно позволяет оперативно получать актуальную информацию о глобальных трендах в энергопотреблении, проводить сравнительный анализ между странами, оценивать эффективность энергетических политик и формировать обоснованные выводы. Открытый код и модульная архитектура делают приложение легко расширяемым и адаптируемым для анализа других наборов данных.

В заключение, данная курсовая работа успешно продемонстрировала применение теоретических знаний в области проектирования баз данных и администрирования, а также практических навыков разработки программного обеспечения, для решения реальной задачи анализа больших данных. Полученные результаты подтверждают эффективность выбранных подходов и технологий.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация по СУБД «PostgreSQL» - типы данных [Электронный ресурс] – URL: <https://www.postgresql.org/docs/17/datatype.html> (дата обращения: 28.04.2025).
2. Документация по СУБД «PostgreSQL» - запросы [Электронный ресурс] – URL: <https://www.postgresql.org/docs/17/queries.html> (дата обращения: 28.04.2025).
3. Документация по утилите «pg_dump» из СУБД «PostgreSQL» [Электронный ресурс] – URL: <https://www.postgresql.org/docs/current/app-pgdump.html> (дата обращения: 05.05.2025).
4. Датасет Global Energy Consumption (2000-2024). URL: <https://www.kaggle.com/datasets/atharvasoundankar/global-energy-consumption-2000-2024> (дата обращения: 19.06.2025).
5. Документация по СУБД «PostgreSQL» - EXPLAIN ANALYZE [Электронный ресурс] – URL: <https://www.postgresql.org/docs/current/sql-explain.html> (дата обращения: 15.05.2025).
6. Исходный код десктопного приложения [Электронный ресурс] – URL: https://github.com/ramizortiqov/global_energy (дата обращения 16.05.2025).
7. Документация по Python-библиотекам [Электронные ресурсы]:
 - customtkinter - URL: <https://customtkinter.tomschimansky.com/> ;
 - psycopg2 - URL: <https://www.psycopg.org/docs/>;
 - Matplotlib - URL: <https://matplotlib.org/stable/contents.html>;
 - pandas - URL: <https://pandas.pydata.org/docs/> (дата обращения: 19.06.2025).