**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**Федеральное государственное бюджетное образовательное учреждение высшего образования «Южно-Российский государственный политехнический университет (НПИ) имени М.И. Платова»**



Факультет **информационных технологий и управления**

Кафедра **«Программное обеспечение вычислительной техники»**

Направление **09.04.01 –** **Информатика и вычислительная техника**

**ОТЧЕТ**

**по Лабораторной работе №4**

по дисциплине: **Программное и аппаратное обеспечение**

**информационных систем**

**Выполнил студент** 1 **курса, группы** ТИСа-о24 Купрысь Н.А.

Фамилия, имя, отчество

**Принял доцент, кандидат технических наук** Рыбалкин А.Д.

Фамилия, имя, отчество

«\_\_\_\_\_» **\_\_**\_\_\_\_\_\_\_\_\_\_\_\_2024 г. **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Подпись

Новочеркасск, 2024 г

**Лабораторная работа №4**

**«Обеспечение защиты и хранения данных ИС и АС»**

**Цель работы:** Разработать структуру хранилища данных, подключить инструменты хранения данных (базе данных) типа SQL или NoSQL и обеспечить безопасное хранение или передачу простым шифрованием.

**Теоретический материал:** Современные базы данных могут быть разделены на два основных типа: SQL и NoSQL. SQL базы данных, такие как MySQL и PostgreSQL, основаны на реляционной модели и используют язык запросов SQL для управления данными. Они обеспечивают строгую структуру данных и поддерживают транзакции, что делает их идеальными для приложений, требующих надежности и согласованности.

NoSQL базы данных, такие как MongoDB и Cassandra, предлагают более гибкую структуру данных, позволяя хранить данные в формате JSON, графов или ключ-значение. Эти системы лучше справляются с горизонтальным масштабированием и могут обрабатывать большие объемы неструктурированных данных, что делает их подходящими для приложений, где данные могут быстро изменяться.

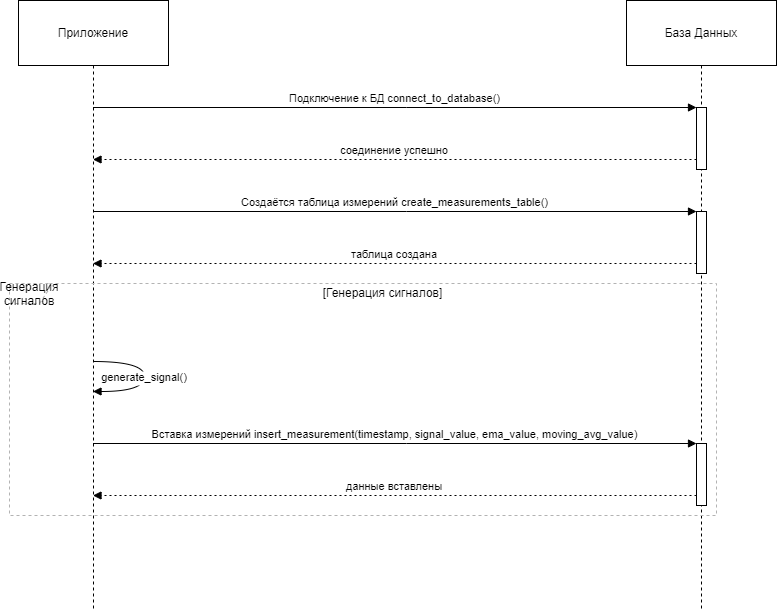
Проектирование структуры баз данных включает в себя создание схемы, описывающей взаимосвязи между сущностями. Важно учитывать нормализацию, чтобы минимизировать дублирование данных и повысить эффективность хранения. CASE-средства помогают визуализировать и проектировать эти структуры, упрощая процесс разработки.

Защита данных в современных системах реализуется с помощью методов шифрования, которые обеспечивают безопасность данных при передаче и хранении. Алгоритмы, такие как AES и RSA, используются для шифрования данных, а также могут применяться расширения, такие как pg\_crypto для PostgreSQL, для защиты информации на уровне базы данных. Протоколы передачи, такие как HTTPS, дополнительно обеспечивают защиту данных при их перемещении между клиентом и сервером.

**Ход работы:**

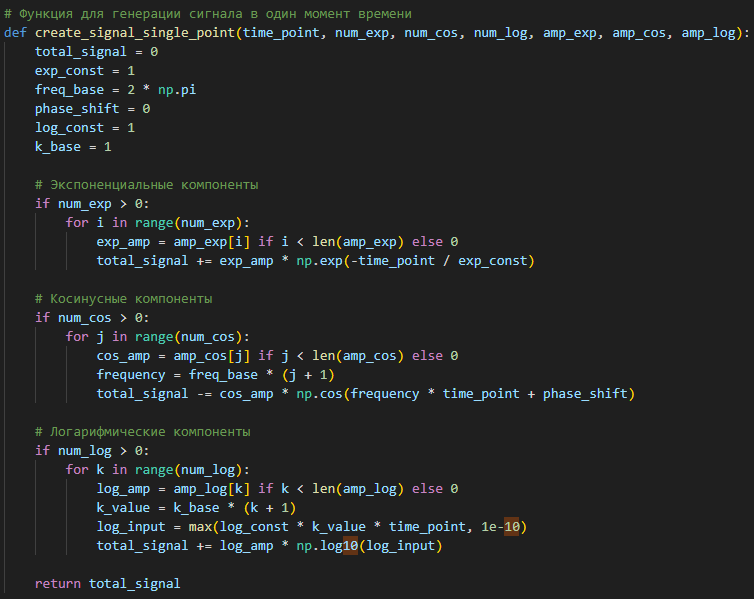
1) Разработаны две схемы: схема классов и схема последовательности, которые описывают взаимодействие компонентов системы SignalApp с базой данных. На схеме классов показано, как SignalApp использует класс Database для работы с базой данных через методы, такие как connect\_to\_database() для установления соединения, create\_measurements\_table() для создания таблицы и insert\_measurement() для записи данных. Схема последовательности иллюстрирует процесс, начинающийся с вызова connect\_to\_database() для установления соединения с базой данных, после чего вызывается create\_measurements\_table() для создания таблицы хранения данных. В процессе генерации сигнала SignalApp регулярно вызывает insert\_measurement() для записи значений сигнала, EMA и скользящего среднего в базу данных, а база данных подтверждает успешную вставку данных. Этот процесс обеспечивает надежное сохранение и обработку данных в реальном времени, что иллюстрируется на рисунках 1 и 2.

*Рисунок 1 – Схема классов*



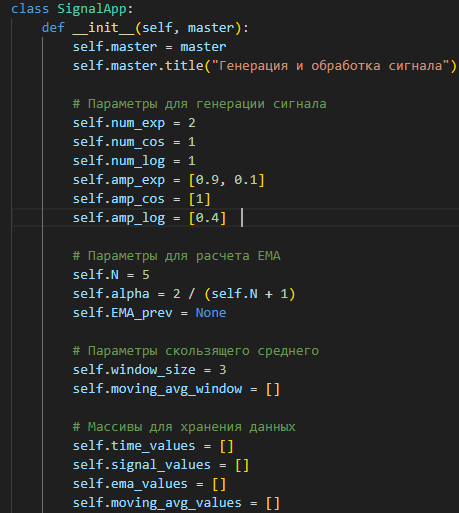
*Рисунок 2 – Схема последовательности с БД*

2) В первой части кода производится импорт ряда ключевых библиотек. NumPy используется для математических вычислений, таких как работа с массивами и функциями для обработки сигнала. Для подключения к внешним источникам данных добавлены библиотеки для работы с API (requests) и базой данных SQL Server (pyodbc). Основная функция в этой части — create\_signal\_single\_point, которая генерирует сигнал на определённой временной точке. Она использует три разных математических компонента для формирования сигнала: экспоненциальные, косинусные и логарифмические функции. Для экспоненциальных компонентов сигнал уменьшается по мере увеличения времени, косинусные компоненты добавляют колебания, а логарифмические компоненты — медленный рост, обеспечивая сложную структуру сигнала. Эта функция сигнала представлена на рисунке 3.



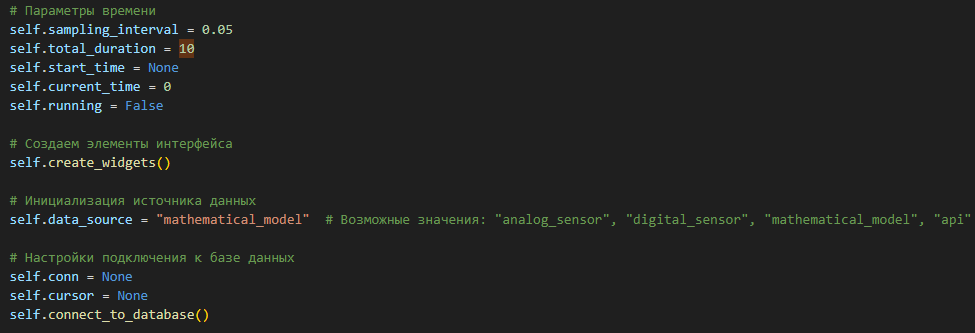
*Рисунок 3 – Импорт библиотек и создание функции генерации сигнала*

3) Во второй части кода создаётся класс SignalApp, который отвечает за создание графического интерфейса для генерации и обработки сигналов. В конструкторе класса инициализируются параметры сигнала, а также амплитуды для каждого из них. Задаются параметры для расчёта экспоненциального скользящего среднего (EMA) и скользящего среднего, а также инициализируются массивы для хранения значений времени, сигнала и вычисленных средних. Эта часть отображена на рисунке 4.



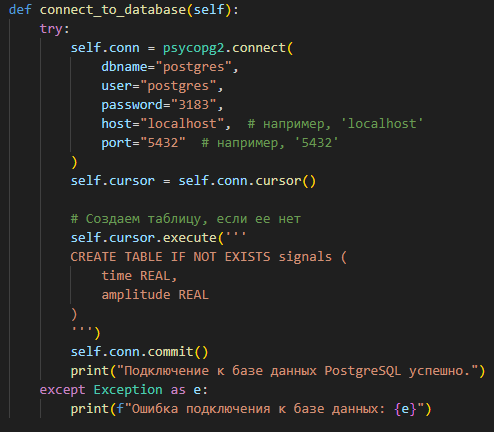
*Рисунок 4 – Создание класса SignalApp и инициализация параметров*

4) В третьей части кода задаются параметры времени, такие как интервал дискретизации и общая продолжительность сигнала. Также инициализируются элементы графического интерфейса через метод create\_widgets. Важно отметить, что здесь происходит подключение к базе данных с помощью метода connect\_to\_database, который устанавливает соединение с локальной базой данных SQL Server. Подключение к базе данных показано на рисунке 5.

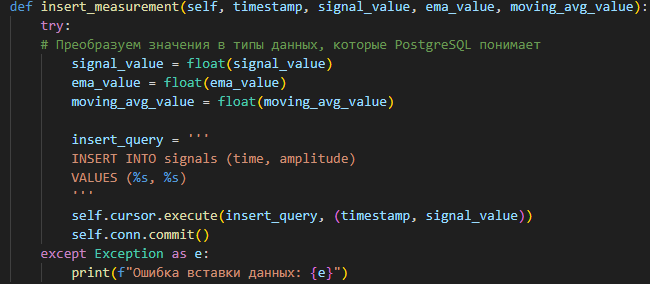


*Рисунок 5 – Инициализация интерфейса и подключение к базе данных*

5) В четвёртой части кода определяется процесс подключения к базе данных и создания таблицы для хранения измерений. Метод create\_measurements\_table проверяет, существует ли таблица с измерениями, и если нет, создаёт её. Таблица использоваться для хранения значений сигнала, EMA и скользящего среднего во время выполнения программы (рисунок 6).

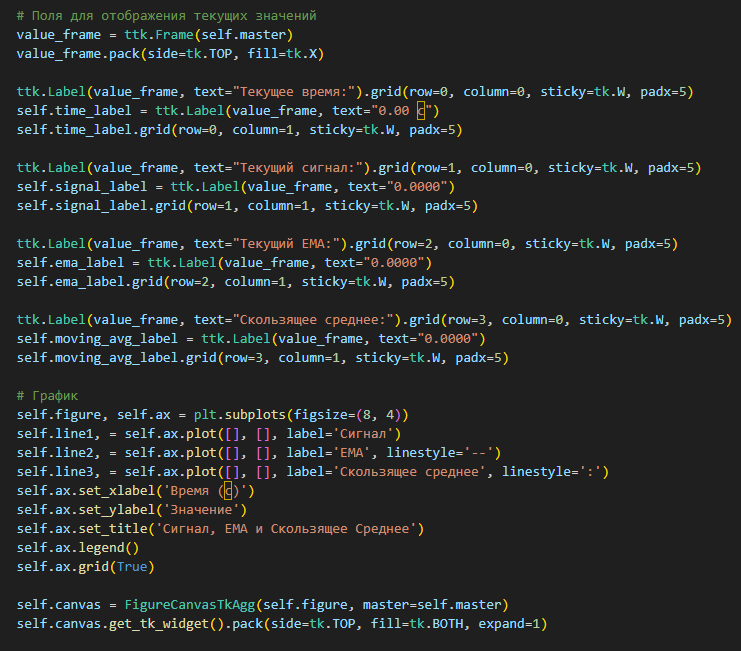


*Рисунок 6 – Подключение и создание таблицы в базе данных*

6) В пятой части представлен метод insert\_measurement, который отвечает за запись данных в базу. Этот метод принимает значения времени, сигнала, EMA и скользящего среднего, после чего выполняет SQL-запрос для вставки этих значений в таблицу БД, что показано на рисунке 7.

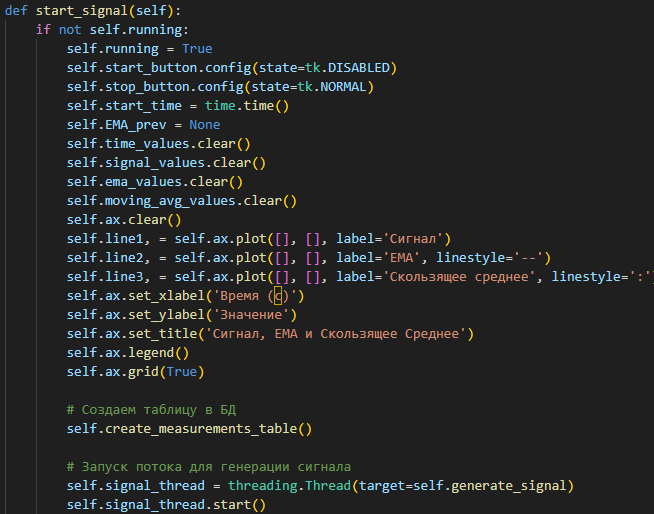
*Рисунок 7 – Вставка измерений в базу данных*

7) В шестой части создаётся интерфейс пользователя. Интерфейс включает в себя метки для отображения текущего времени, значений сигнала, EMA и скользящего среднего. Также создаётся график для визуализации изменений данных в реальном времени с использованием библиотеки Matplotlib. Этот интерфейс показан на рисунке 8.

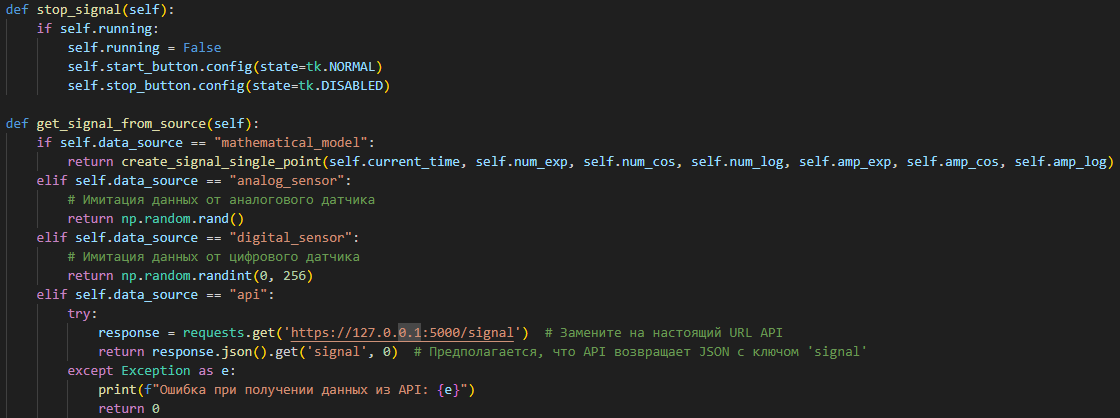


*Рисунок 8 – Создание элементов управления и отображения значений*

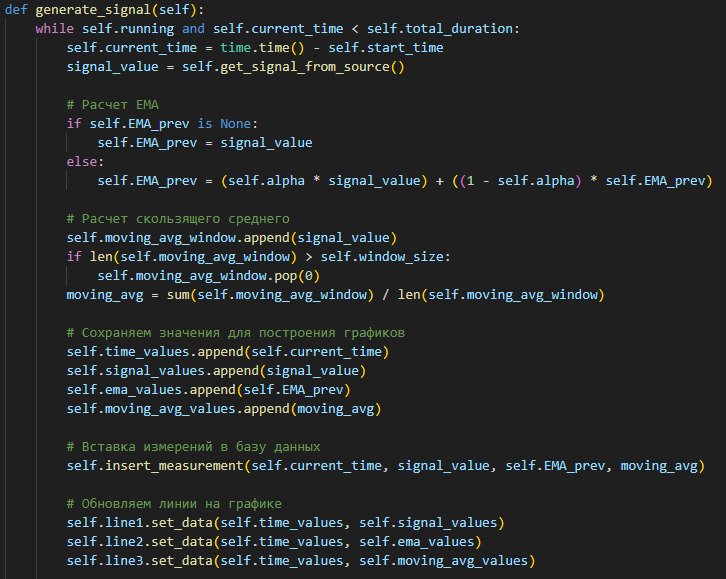
8) В седьмой части описаны методы start\_signal и stop\_signal, которые управляют запуском и остановкой процесса генерации сигнала. Метод start\_signal инициализирует время, очищает массивы данных и запускает поток для генерации сигнала, тогда как stop\_signal завершает этот процесс. Запуск и остановка генерации сигнала показаны на рисунке 9.



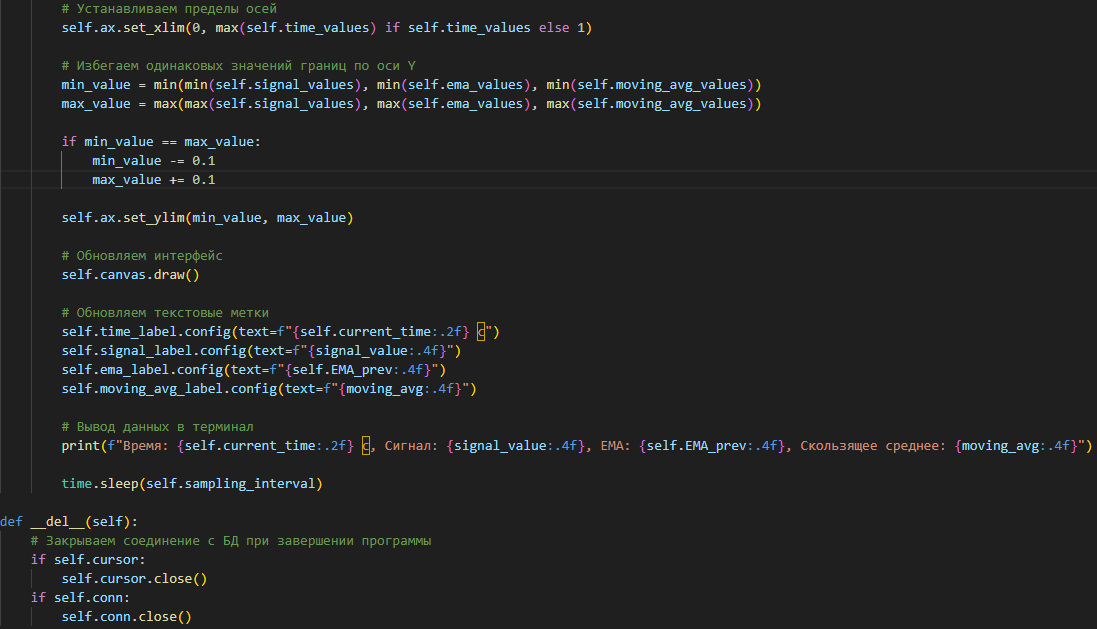
*Рисунок 9 – Управление процессом генерации сигнала*

9) В восьмой части кода метод get\_signal\_from\_source отвечает за получение сигнала из выбранного источника. В зависимости от источника данных метод возвращает соответствующее значение сигнала. Этот процесс проиллюстрирован на рисунке 10.

*Рисунок 10 – Получение данных сигнала из различных источников*

10) В девятой части кода метод generate\_signal реализует непрерывную генерацию сигнала, расчёт EMA и скользящего среднего, а также обновление интерфейса в реальном времени. Внутри метода данные записываются в базу данных, и обновляется график. Важно отметить, что динамические изменения графика и текстовых меток в интерфейсе происходят во время выполнения программы. Этот процесс показан на рисунке 11.

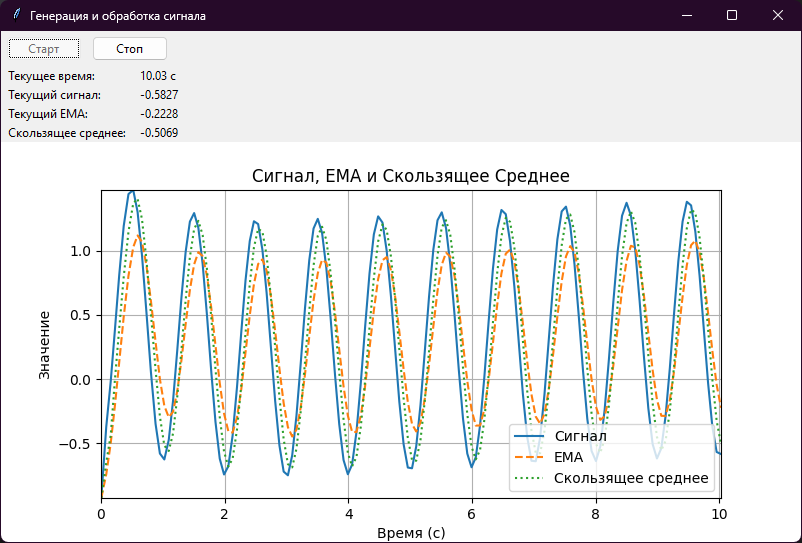
*Рисунок 11 – Генерация сигнала и обновление интерфейса*

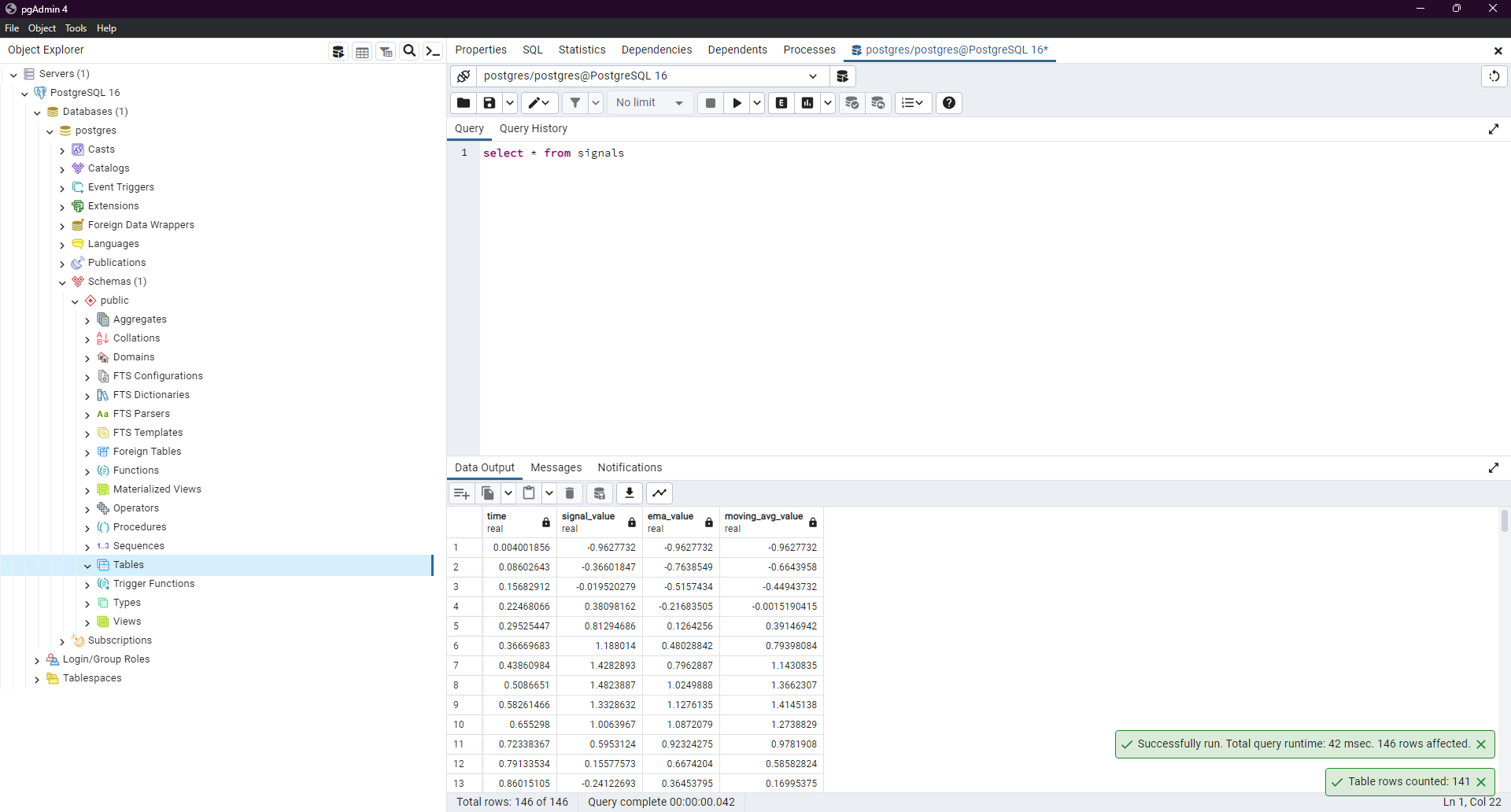
11) В финальной части кода происходит настройка динамических пределов осей для графика, который обновляется в реальном времени. Метод set\_xlim устанавливает горизонтальные пределы оси X, чтобы график охватывал весь временной диапазон, прошедший с начала генерации сигнала. Метод set\_ylim автоматически адаптирует пределы оси Y, чтобы учесть минимальные и максимальные значения для сигнала, EMA и скользящего среднего. Это помогает избежать ситуаций, когда значения по оси Y остаются неизменными и график выглядит неподвижным. Всё это продемонстрировано на рисунке 12.

*Рисунок 12 – Генерация сигнала и обновление интерфейса*

**Результаты работы:**

После нажатия кнопки "Старт" запускается процесс генерации сигнала, который состоит из экспоненциальных, косинусных и логарифмических компонентов, и отображается на графике в реальном времени. На графике по оси X показывается время, а по оси Y — значения сигнала, EMA и скользящего среднего, с соответствующими линиями: сплошной для сигнала, пунктирной для EMA и точечной для скользящего среднего. Одновременно обновляются текстовые метки с текущими значениями сигнала, EMA и скользящего среднего. График и текстовые поля интерфейса обновляются в реальном времени, а результаты выводятся в терминал. Кроме того, данные о времени, сигнале, EMA и скользящем среднем записываются в базу данных для дальнейшего анализа, что продемонстрировано на рисунках 13 и 14.

*Рисунок 13 – Визуализированный сигнал и данные за 10 секунд*

**

*Рисунок 14 – Данные, перенесённые в БД*

**Вывод:** Разработана программа для генерации и визуализации сложных сигналов в реальном времени с расчетом экспоненциального скользящего среднего (EMA) и простого скользящего среднего (SMA). Программа обеспечивает удобный интерфейс для наблюдения и анализа сигналов, обновляющихся в реальном времени, с возможностью записи данных в базу для последующего анализа.