Вкладка 1

# Техническое задание: Структура Базы Данных

## 1. Общая концепция

Система предназначена для сбора, хранения и визуализации экономических индикаторов. База данных спроектирована для поддержки двух основных типов данных: **числовых временных рядов** (например, значения PMI) и **структурированных текстовых релизов** (например, отчёты ISM).

* **СУБД**: SQLite
* **Кодировка**: UTF-8

## 2. Описание таблиц

### Таблица 1: indicators

Справочник, содержащий мета-информацию о каждом индикаторе.

| Поле | Тип данных | Описание и комментарии для разработчика |
| --- | --- | --- |
| id | INTEGER PRIMARY KEY | Уникальный идентификатор, автоинкремент. |
| name | TEXT | Короткое системное имя. **Рекомендация:** lowercase\_with\_underscores (например, us\_ism\_manufacturing\_pmi). |
| full\_name | TEXT | Полное, человекочитаемое название (например, "ISM Manufacturing PMI"). |
| source | TEXT | Основной источник данных (например, "FRED / ismworld.org"). |
| description | TEXT | Краткое описание сути индикатора. |

### Таблица 2: indicator\_values

Основное хранилище для всех числовых данных.

| Поле | Тип данных | Описание и комментарии для разработчика |
| --- | --- | --- |
| id | INTEGER PRIMARY KEY | Уникальный идентификатор, автоинкремент. |
| indicator\_id | INTEGER | Внешний ключ, ссылается на indicators.id. |
| date | TEXT | Дата, к которой относится значение. **Формат:** YYYY-MM-DD. |
| category | TEXT | Необязательная подкатегория. **Примеры:** для кривой доходности — срок (10 Yr), для разрешений — тип (1 unit), для UMCSI — тип релиза (preliminary / final). Для простых рядов оставлять NULL. |
| value | REAL | Числовое значение индикатора. |
| created\_at | TEXT | Дата и время добавления записи. **Формат:** YYYY-MM-DD HH:MM:SS. |

### Таблица 3: indicator\_releases

Хранилище для текстовых и структурированных данных из отчётов.

| Поле | Тип данных | Описание и комментарии для разработчика |
| --- | --- | --- |
| id | INTEGER PRIMARY KEY | Уникальный идентификатор, автоинкремент. |
| indicator\_id | INTEGER | Внешний ключ, ссылается на indicators.id. |
| date | TEXT | Дата публикации релиза. **Формат:** YYYY-MM-DD. |
| category | TEXT | Необязательная подкатегория. **Примеры:** тип релиза UMCSI (preliminary / final), тип отчёта ISM (Manufacturing / Services). |
| release\_data | TEXT | Структурированные данные отчёта. **Важно:** поле хранит текст в формате **JSON**. См. финальную структуру в Приложении 1. |
| source\_url | TEXT | Прямая ссылка на страницу с релизом. |
| created\_at | TEXT | Дата и время добавления записи. **Формат:** YYYY-MM-DD HH:MM:SS. |

### Таблица 4: comments

Хранилище для пользовательских комментариев, привязанных к датам на графиках.

| Поле | Тип данных | Описание и комментарии для разработчика |
| --- | --- | --- |
| id | INTEGER PRIMARY KEY | Уникальный идентификатор, автоинкремент. |
| indicator\_id | INTEGER | Внешний ключ, ссылается на indicators.id. |
| date | TEXT | Дата на графике, к которой привязан комментарий. **Формат:** YYYY-MM-DD. |
| comment\_text | TEXT | Текст комментария. |
| created\_at | TEXT | Дата и время создания комментария. **Формат:** YYYY-MM-DD HH:MM:SS. |

## Приложение 1: Финальная структура JSON для release\_data

JSON

{

"headline\_index": {

"name": "Services PMI",

"value": 53.5,

"gdp\_correlation": "Corresponds to a 2.0% increase in real GDP annually."

},

"summary": {

"demand": "Comments on new orders and backlogs.",

"supply": "Comments on supplier deliveries and inventories."

},

"sub\_indices": {

"business\_activity": { "value": 56.0, "ranking": "growing for the 15th month" },

"new\_orders": { "value": 55.8, "ranking": "growing at a slower rate" }

},

"industry\_summary": {

"growing\_industries\_count": 14,

"contracting\_industries\_count": 4,

"growing\_industries\_list": [

"Accommodation & Food Services"

],

"contracting\_industries\_list": [

"Agriculture, Forestry, Fishing & Hunting"

]

},

"commodities": {

"up\_in\_price": [

"Diesel Fuel",

"Gasoline"

],

"down\_in\_price": [

"Steel Products"

],

"in\_short\_supply": [

"Transformers"

]

},

"quotes": [

{

"industry": "Health Care & Social Assistance",

"comment": "Starting to see some slowing in patient volumes..."

}

]

}

## Приложение 2: Источники данных для сбора

* **Yield Curve (номинальная и реальная)**
  + **Источник**: U.S. Department of the Treasury (home.treasury.gov).
  + **URL (пример)**: https://home.treasury.gov/resource-center/data-chart-center/interest-rates/TextView?type=daily\_treasury\_yield\_curve
  + **Нюансы для сборщика**: Данные представлены в виде HTML-таблицы. Потребуется парсинг веб-страницы для извлечения значений по каждой дате и сроку погашения.
* **Real M2 Money Stock**
  + **Источник**: FRED.
  + **Идентификатор (Series ID)**: M2REAL.
  + **Нюансы для сборщика**: Простой числовой ряд. Загружается напрямую через API FRED.
* **Building Permits**
  + **Источник**: FRED.
  + **Идентификаторы (Series IDs)**: PERMIT1 (1 unit), PERMIT24 (2-4 units), PERMIT5 (5+ units).
  + **Нюансы для сборщика**: Необходимо загрузить три отдельных ряда и сохранить их с соответствующими category.
* **UMCSI (University of Michigan Consumer Sentiment)**
  + **Источник**: Гибридный.
  + **Числовое значение**: FRED, Series ID UMCSENT.
  + **Текст отчёта**: Официальный сайт (https://data.sca.isr.umich.edu/).
  + **Нюансы для сборщика**: Сборщик должен выполнить два действия: забрать число с FRED, затем перейти на сайт UMich и спарсить текстовый блок с аналитикой, сохранив его в indicator\_releases. Необходимо отслеживать предварительные (preliminary) и финальные (final) релизы.
* **ISM Manufacturing & Services PMI**
  + **Источник**: Гибридный.
  + **Числовые значения**: FRED, Series IDs NAPM (Manufacturing) и ISMNS (Services).
  + **Текст отчёта**: Официальный сайт (https.www.ismworld.org/).
  + **Нюансы для сборщика**: Сборщик забирает основное числовое значение с FRED. Затем парсит страницу с пресс-релизом на сайте ISM, извлекая все необходимые компоненты (субиндексы, списки отраслей, цитаты и т.д.) для формирования сложного JSON-объекта согласно Приложению 1.

Конечно. Вот проект документации для вашего DAO.

Вы можете скопировать этот текст и добавить его в конец вашего файла «Техническое задание: Структура Базы Данных» как новый раздел.

## Приложение 3: Слой Доступа к Данным (DAO)

### 1. Общая концепция DAO

Слой доступа к данным (Data Access Object, DAO) реализован в виде Python-класса IndicatorDAO в файле dao.py. Этот класс является **единственным посредником** между прикладными скриптами (сборщиками данных, модулями визуализации) и базой данных economic\_indicators.db.

Основная задача DAO — инкапсулировать (скрыть) всю логику SQL-запросов. Прикладной код не пишет SQL-запросы напрямую, а вызывает простые и понятные Python-методы DAO (например, dao.get\_indicator\_values(...)). Это обеспечивает:

* **Надежность:** Вся логика работы с БД находится в одном месте.
* **Простоту:** Прикладным скриптам не нужно знать о структуре таблиц или синтаксисе SQL.
* **Гибкость:** При смене СУБД потребуется изменить только код внутри DAO.

### **2. Расположение файла БД и синхронизация с облаком**

#### **Расположение файла**

Файл базы данных economic\_indicators.db не хранится внутри папки с проектом. DAO автоматически определяет путь к **домашней директории текущего пользователя** и размещает файл в его папке **«Документы»**.

* **Пример пути в Windows:** C:/Users/ИмяПользователя/Documents/economic\_indicators.db
* **Пример пути в macOS/Linux:** /Users/ИмяПользователя/Documents/economic\_indicators.db

Такой подход гарантирует, что все части приложения на одном компьютере всегда обращаются к одному и тому же файлу.

#### **Механизм синхронизации**

Синхронизация данных между разными устройствами обеспечивается **внешним приложением облачного хранения** (например, Google Drive, Dropbox, OneDrive, и т.д.), а не самим DAO.

1. Пользователь настраивает свой облачный клиент так, чтобы он синхронизировал папку «Документы».
2. Приложение на **Устройстве А** через DAO записывает данные в локальный файл .../Documents/economic\_indicators.db.
3. Облачный клиент автоматически обнаруживает изменения и загружает новую версию файла в облако.
4. Облачный клиент на **Устройстве Б** скачивает обновленный файл из облака в локальную папку «Документы».
5. Приложение на **Устройстве Б**, обращаясь через DAO к своему локальному файлу, получает уже синхронизированные данные.

**Важно:** DAO не знает о существовании облака. Он всегда работает с локальным файлом в предсказуемом месте.

### **3. Описание методов (API Reference)**

#### **\_\_init\_\_(self)**

* **Описание:** Конструктор класса. Автоматически подключается к базе данных при создании экземпляра dao = IndicatorDAO().
* **Параметры:** Нет.
* **Возвращает:** Ничего.

#### **add\_indicator(self, name, full\_name, source, description)**

* **Описание:** Добавляет новый индикатор в справочник indicators. Если индикатор с таким name уже существует, не создает дубликат, а возвращает ID существующей записи.
* **Параметры:**
  + name (str): Короткое системное имя (e.g., us\_ism\_services\_pmi).
  + full\_name (str): Полное название для отображения.
  + source (str): Источник данных.
  + description (str): Описание.
* **Возвращает:** int — ID созданной или найденной записи. None в случае ошибки.

#### **add\_indicator\_value(self, indicator\_id, date, value, category=None)**

* **Описание:** Добавляет одно числовое значение в таблицу indicator\_values.
* **Параметры:**
  + indicator\_id (int): ID индикатора.
  + date (str): Дата в формате 'YYYY-MM-DD'.
  + value (float): Числовое значение.
  + category (str, optional): Необязательная подкатегория.
* **Возвращает:** Ничего.

#### **add\_indicator\_release(self, indicator\_id, date, release\_data, source\_url, category=None)**

* **Описание:** Добавляет структурированный текстовый релиз в indicator\_releases.
* **Параметры:**
  + indicator\_id (int): ID индикатора.
  + date (str): Дата релиза в формате 'YYYY-MM-DD'.
  + release\_data (dict): Python-словарь с данными отчета. Метод сам преобразует его в JSON.
  + source\_url (str): Ссылка на источник.
  + category (str, optional): Необязательная подкатегория.
* **Возвращает:** Ничего.

#### **add\_comment(self, indicator\_id, date, comment\_text)**

* **Описание:** Добавляет пользовательский комментарий в comments.
* **Параметры:**
  + indicator\_id (int): ID индикатора.
  + date (str): Дата, к которой привязан комментарий ('YYYY-MM-DD').
  + comment\_text (str): Текст комментария.
* **Возвращает:** Ничего.

#### **get\_indicator\_values(self, indicator\_id)**

* **Описание:** Извлекает всю историю числовых значений для индикатора.
* **Параметры:**
  + indicator\_id (int): ID индикатора.
* **Возвращает:** pandas.DataFrame с колонками date (тип datetime) и value.

#### **get\_indicator\_release(self, indicator\_id, date)**

* **Описание:** Извлекает один текстовый релиз по ID индикатора и дате.
* **Параметры:**
  + indicator\_id (int): ID индикатора.
  + date (str): Дата релиза в формате 'YYYY-MM-DD'.
* **Возвращает:** Кортеж (dict, str), где первый элемент — Python-словарь с данными отчета, а второй — ссылка на источник. (None, None) если ничего не найдено.

#### **get\_comments(self, indicator\_id)**

* **Описание:** Извлекает все комментарии для индикатора.
* **Параметры:**
  + indicator\_id (int): ID индикатора.
* **Возвращает:** pandas.DataFrame с колонками date (тип datetime) и comment\_text.

#### **close(self)**

* **Описание:** Корректно закрывает соединение с базой данных. Обязательно вызывать в конце работы со скриптом.
* **Параметры:** Нет.
* **Возвращает:** Ничего.

### **4. Пример использования**

Python

# Пример скрипта, использующего DAO

from dao import IndicatorDAO

# 1. Инициализация DAO (автоматическое подключение к БД)

dao = IndicatorDAO()

# 2. Запись данных

# Сначала получаем ID индикатора (создастся, если нет)

pmi\_id = dao.add\_indicator(name='us\_pmi', full\_name='PMI', source='test', description='test')

# Добавляем значение, используя полученный ID

dao.add\_indicator\_value(indicator\_id=pmi\_id, date='2025-08-23', value=55.0)

# 3. Чтение данных

# Получаем все значения для этого индикатора в виде таблицы

pmi\_data = dao.get\_indicator\_values(pmi\_id)

print("Прочитанные данные:")

print(pmi\_data)

# 4. Обязательно закрываем соединение

dao.close()

Нет, вопросов нет. План ясен.

Вот описание вашего текущего рабочего окружения. Вы можете скопировать его и вставить в ваш документ как новый раздел.

## Приложение 4: Описание Рабочего Окружения

Этот раздел описывает состояние и структуру проекта по завершении первого этапа — разработки слоя доступа к данным (DAO). Он служит отправной точкой для последующих этапов, таких как создание сборщиков данных и модулей визуализации.

### **1. Среда Разработки (VS Code)**

Работа ведется в редакторе **Visual Studio Code**. Ключевые компоненты настроенной среды:

* **Интерпретатор Python**: VS Code настроен на использование **изолированного** интерпретатора из виртуального окружения проекта (.venv/bin/python3). Это гарантирует, что зависимости проекта не конфликтуют с системными библиотеками.
* **Основные расширения**:
  + **Python** (от Microsoft): Обеспечивает базовую поддержку языка, отладку, автодополнение и анализ кода.
  + **GitHub Copilot**: Используется как AI-ассистент для генерации и редактирования кода.
  + **SQLite Explorer**: Позволяет просматривать и взаимодействовать с файлом базы данных economic\_indicators.db прямо в интерфейсе редактора.
  + **Black Formatter**: (Рекомендовано) Используется для автоматического форматирования кода по единому стандарту.

### **2. Структура Проектного Каталога**

Проект расположен в папке economic\_indicators. Все команды выполняются из этой папки.

economic\_indicators/

│

├── .venv/

│ └── (содержимое виртуального окружения Python)

│

├── \_\_pycache\_\_/

│ └── (автоматически генерируемый кеш Python)

│

├── dao.py

│ # Основной модуль проекта. Содержит класс IndicatorDAO, который является

│ # единственной точкой доступа к базе данных. Вся SQL-логика инкапсулирована здесь.

│

├── database\_setup.py

│ # Служебный скрипт для первоначального создания структуры базы данных.

│ # Создает файл economic\_indicators.db и все необходимые таблицы в нём.

│

├── test\_dao.py

│ # Скрипт для комплексного тестирования dao.py. Очищает, записывает и

│ # читает данные из всех таблиц, чтобы убедиться в корректной работе DAO.

│

└── requirements.txt

# список всех установленных библиотек и их версий

**Важное замечание о расположении файла Базы Данных:**

Согласно коду в dao.py, сам файл базы данных **economic\_indicators.db** создается и используется не в папке проекта, а в системной папке **«Документы»** (/home/alks/Documents/economic\_indicators.db). Это сделано намеренно, чтобы обеспечить простую синхронизацию файла БД между разными устройствами с помощью облачных сервисов (Google Drive, Dropbox и т.д.).

### **3. Установленные Библиотеки**

Все перечисленные ниже библиотеки установлены в изолированное виртуальное окружение .venv с помощью менеджера пакетов pip3.

* **pandas**: Ключевая библиотека для анализа и манипуляции данными. В DAO она используется для удобного представления данных, прочитанных из базы, в виде таблиц (DataFrame).
* **fredapi**: Клиентская библиотека для простого и удобного доступа к API Федерального резервного банка Сент-Луиса (FRED), который является одним из источников данных для проекта.

**Рекомендация по управлению зависимостями:**

Для того чтобы в будущем можно было легко воссоздать это же окружение на другом компьютере, рекомендуется создать файл requirements.txt. Это можно сделать, выполнив в терминале с активным окружением (.venv) следующую команду:

Bash

pip3 freeze > requirements.txt