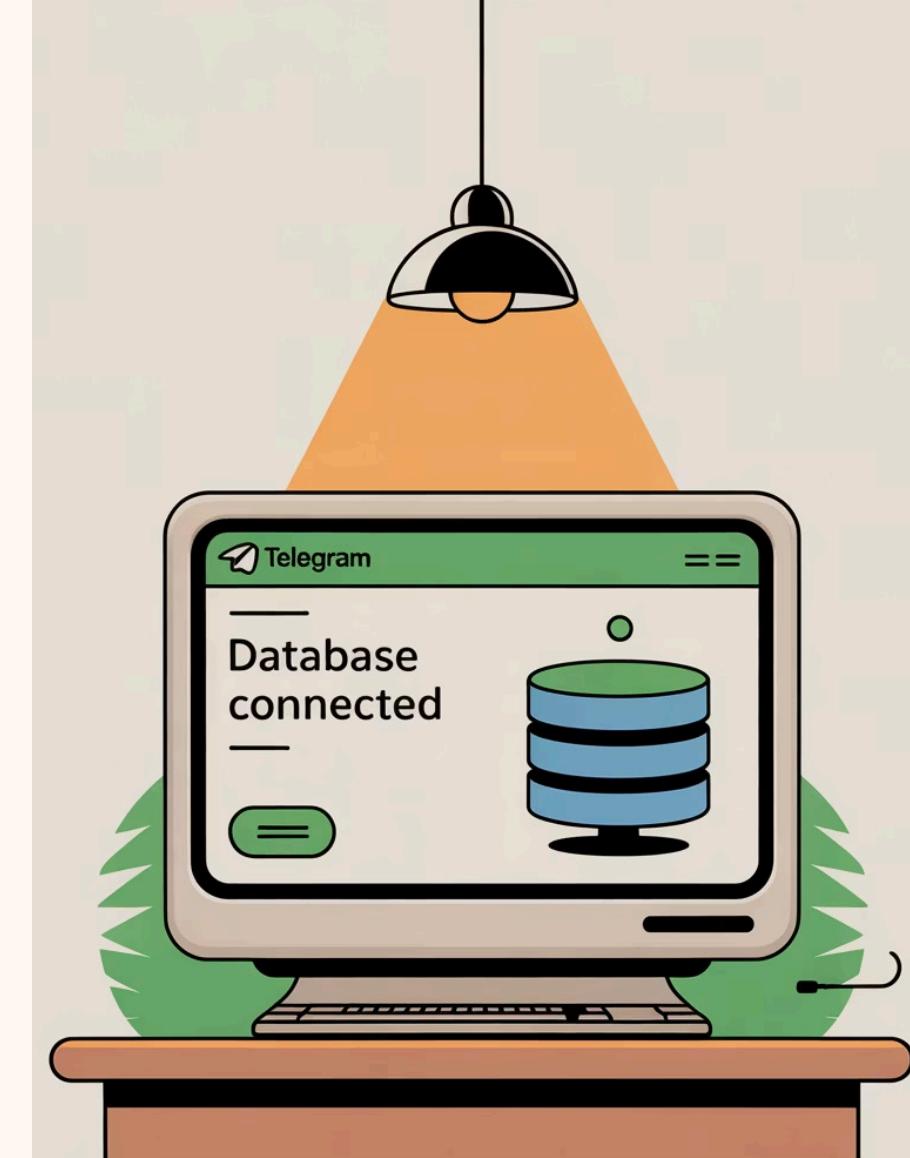


S5 – SQLite в Telegram-боте: заметки и CRUD

Делаем бота «с памятью»: база данных, команды, статистика, экспорт



План занятия

01

Зачем БД боту

Мотивация и проблемы хранения в памяти

02

Схема notes и init_db()

Создание таблицы и инициализация

03

CRUD-команды

/note_add, /note_list, /note_find, /note_edit,
/note_del

04

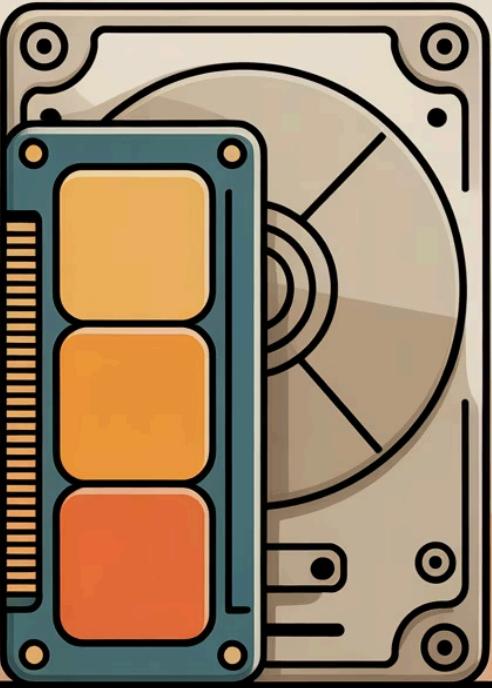
Статистика и визуализация

Анализ данных и ASCII-графики

05

Продвинутые функции

Ограничения, экспорт, надёжность, командная работа



RAM

Disk Storage

Почему без БД неудобно 🤔

Данные в памяти

Списки и словари живут только в ОЗУ

Потеря при перезапуске

Перезапуск = все данные исчезают

SQLite = решение

Файл на диске для долговременного хранения

```
notes = []
```

```
notes.append("купить хлеб")
```

```
# после перезапуска пусто!
```

Что такое SQLite для нас



Встроена в Python

Модуль `sqlite3` включён по умолчанию,
дополнительная установка не нужна



Один файл

Вся база данных хранится в одном файле
`bot.db` в корне проекта



Идеальна для обучения

Простая миграция, лёгкий бэкап, отлично
подходит для локальных проектов



Project

Структура проекта бота 📁

config.py

TOKEN, DB_PATH, настройки логирования

db.py

Все функции для работы с базой данных

main.py

TeleBot команды и обработчики сообщений

.env и .gitignore

Секретные данные и исключения из Git

Безопасность: .env и .gitignore 🔒

Проблема

- TOKEN нельзя хранить в коде
- Секреты попадают в Git
- Компрометация токена

Решение

- TOKEN в файле .env
- .env добавлен в .gitignore
- DB_PATH настраивается

```
from dotenv import load_dotenv  
load_dotenv()  
TOKEN = os.getenv("TOKEN")  
DB_PATH = os.getenv("DB_PATH", "bot.db")
```

Схема таблицы notes



```
CREATE TABLE IF NOT EXISTS notes (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL,
    text TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

- **id** — уникальный номер заметки
- **text** — содержимое заметки
- **user_id** — кто создал заметку
- **created_at** — когда создана



Инициализация БД: init_db() 🚀

```
def init_db():
    schema = """
CREATE TABLE IF NOT EXISTS notes (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL,
    text TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
"""

    with _connect() as conn:
        conn.executescript(schema)
```

Функция создаёт таблицы при первом запуске бота. Если таблицы уже существуют, ничего не происходит благодаря IF NOT EXISTS.

Правильное подключение с PRAGMA

```
def _connect():
    conn = sqlite3.connect(DB_PATH, timeout=5.0)
    conn.row_factory = sqlite3.Row
    conn.execute("PRAGMA foreign_keys = ON")
    conn.execute("PRAGMA journal_mode = WAL")
    conn.execute("PRAGMA busy_timeout = 5000")
    return conn
```

→ **WAL режим** уменьшает блокировки

→ **busy_timeout** ждёт 5 секунд при
блокировке

→ **Row factory** позволяет обращаться к
колонкам по именам

Точка входа: запуск бота 🤖

```
bot = telebot.TeleBot(TOKEN)

# Инициализируем базу данных при старте
db.init_db()

if __name__ == "__main__":
    bot.infinity_polling(skip_pending=True)
```

- ☐ **skip_pending=True** — не обрабатывать старые сообщения, которые накопились пока бот был выключен

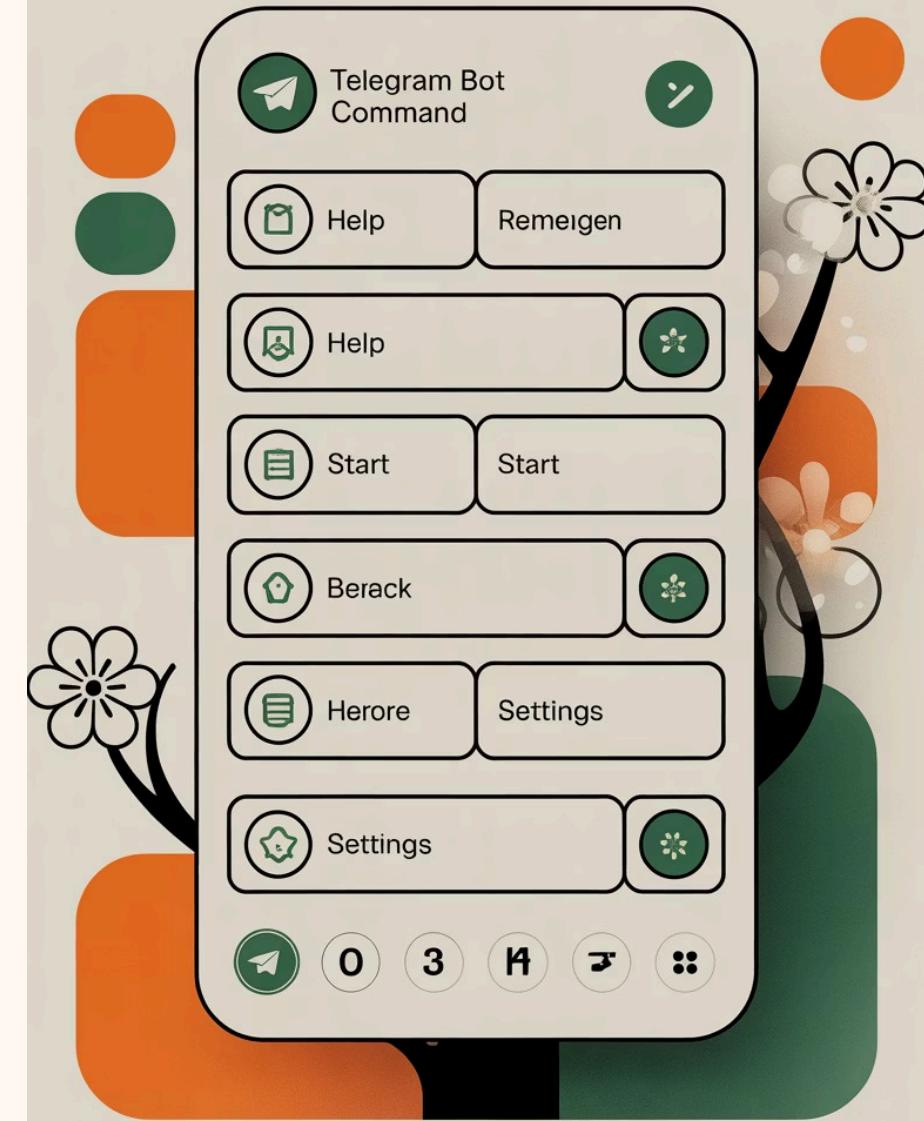
"Connecting.."



Меню команд в Telegram (UX)


```
def setup_bot_commands():
    bot.set_my_commands([
        types.BotCommand("note_add", "Добавить заметку"),
        types.BotCommand("note_list", "Список заметок"),
        types.BotCommand("note_find", "Поиск"),
        types.BotCommand("note_edit", "Правка"),
        types.BotCommand("note_del", "Удалить"),
        types.BotCommand("note_count", "Сколько всего"),
    ])
```

Пользователи увидят команды в меню Telegram, что значительно улучшает удобство использования бота.



CREATE: /note_add

```
def add_note(user_id: int, text: str) -> int:  
    with _connect() as conn:  
        cur = conn.execute(  
            "INSERT INTO notes(user_id, text) VALUES (?, ?)",  
            (user_id, text)  
        )  
        return cur.lastrowid
```

Пример использования

/note_add купить хлеб

↓

"Заметка #1 добавлена"

Что происходит

- Безопасная вставка с параметрами
- lastrowid возвращает ID новой записи
- Автоматическое закрытие соединения



READ: /note_list



```
def list_notes(user_id: int, limit: int = 10):  
    with _connect() as conn:  
        cur = conn.execute(  
            """SELECT id, text, created_at  
            FROM notes  
            WHERE user_id = ?  
            ORDER BY id DESC  
            LIMIT ?""",  
            (user_id, limit)  
        )  
        return cur.fetchall()
```

Показываем последние заметки первыми (ORDER BY id DESC)

Ограничиваем количество для скорости и UX

READ: /note_find

```
SELECT id, text  
FROM notes  
WHERE user_id = ?  
AND text LIKE '%' || ? || '%'  
ORDER BY id DESC  
LIMIT 10;
```

Пример

/note_find хлеб

Найдёт: "купить хлеб", "хлеб и молоко"

Важно

- Безопасная параметризация
- Никаких f-строк в SQL!
- LIKE с % для поиска подстроки

UPDATE:/note_edit



```
def update_note(user_id: int, note_id: int, text: str) -> bool:  
    with _connect() as conn:  
        cur = conn.execute(  
            """  
                UPDATE notes  
                SET text = ?  
                WHERE user_id = ? AND id = ?  
            """,  
            (text, user_id, note_id)  
        )  
        return cur.rowcount > 0
```

rowcount

Показывает, сколько строк было изменено. Если 0 — значит заметка не найдена или не принадлежит пользователю

Безопасность

Проверяем и user_id, и id — пользователь может редактировать только свои заметки



DELETE:/note_del



```
def delete_note(user_id: int, note_id: int) -> bool:  
    with _connect() as conn:  
        cur = conn.execute(  
            "DELETE FROM notes WHERE user_id = ? AND id = ?",
            (user_id, note_id)
        )  
        return cur.rowcount > 0
```

Золотое правило: всегда фильтруем по user_id! Пользователь должен управлять только своими данными.

Быстрый тест CRUD



1

/note_add купить хлеб

Создаём первую заметку

2

/note_list

Видим: "1: купить хлеб"

3

/note_find хлеб

Находим: "1: купить хлеб"

4

/note_edit 1 купить молоко

Изменяем содержимое

5

/note_del 1

Удаляем заметку

Статистика:/note_count



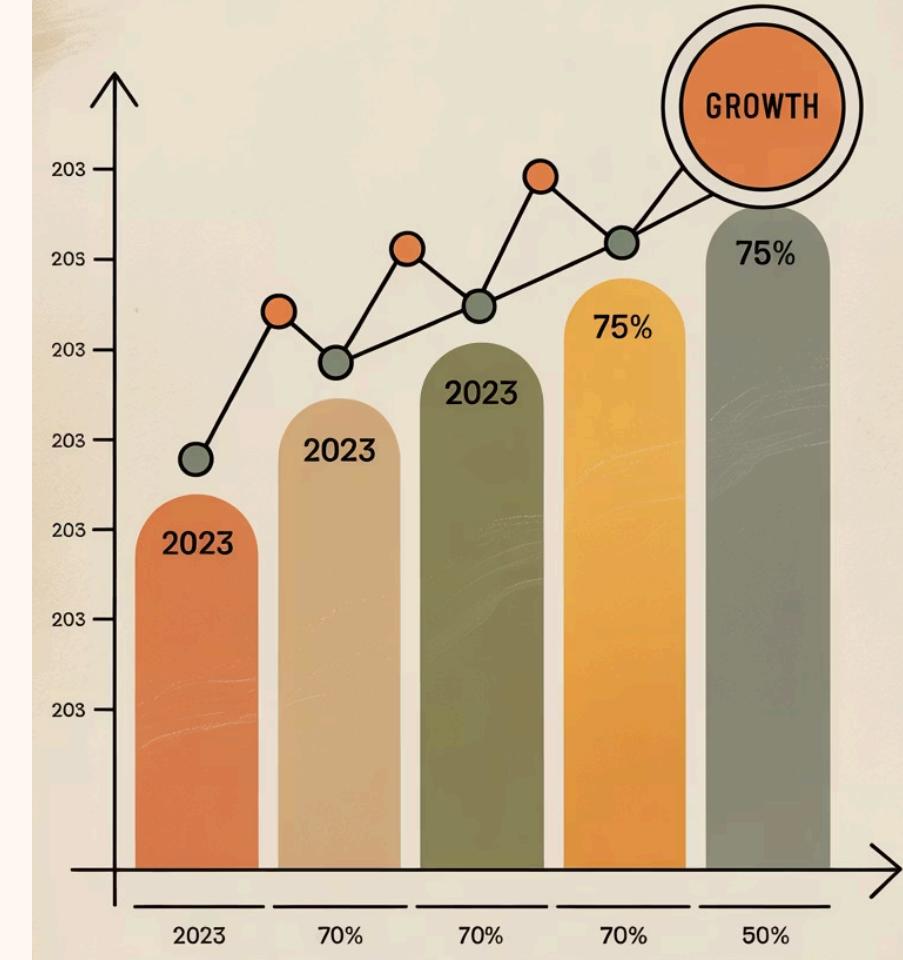
```
SELECT COUNT(*)  
FROM notes  
WHERE user_id = ?;
```

Простой пример

Ответ бота: "У вас 5 заметок"

Польза

- Быстрая оценка активности
- Мотивация к использованию
- Основа для аналитики



ASCII-визуализация



```
SELECT date(created_at) AS d, COUNT(*) AS total  
FROM notes  
WHERE user_id = ?  
GROUP BY date(created_at)  
ORDER BY d DESC  
LIMIT 7;
```

Результат в чате

```
2025-10-02: ... 3  
2025-10-01: .. 2  
2025-09-30: . 1
```

Эффект

Данные превращаются в наглядную картинку прямо в Telegram!

Ограничения: CHECK и UNIQUE



```
text TEXT NOT NULL  
    CHECK(length(text) BETWEEN 1 AND 200),  
    UNIQUE(user_id, text)  
    ON CONFLICT IGNORE
```

CHECK

Текст от 1 до 200 символов

UNIQUE

Нет дубликатов у одного пользователя

ON CONFLICT

Тихо игнорируем дубли



Обработка ошибок ограничений



```
try:  
    add_note(user_id, text)  
    bot.reply_to(message, "Заметка добавлена!")  
except sqlite3.IntegrityError:  
    bot.reply_to(message, "Такая заметка уже есть.")
```

- ❑ Ловим `IntegrityError` и даём пользователю понятное объяснение вместо технической ошибки.

Экспорт:/note_export

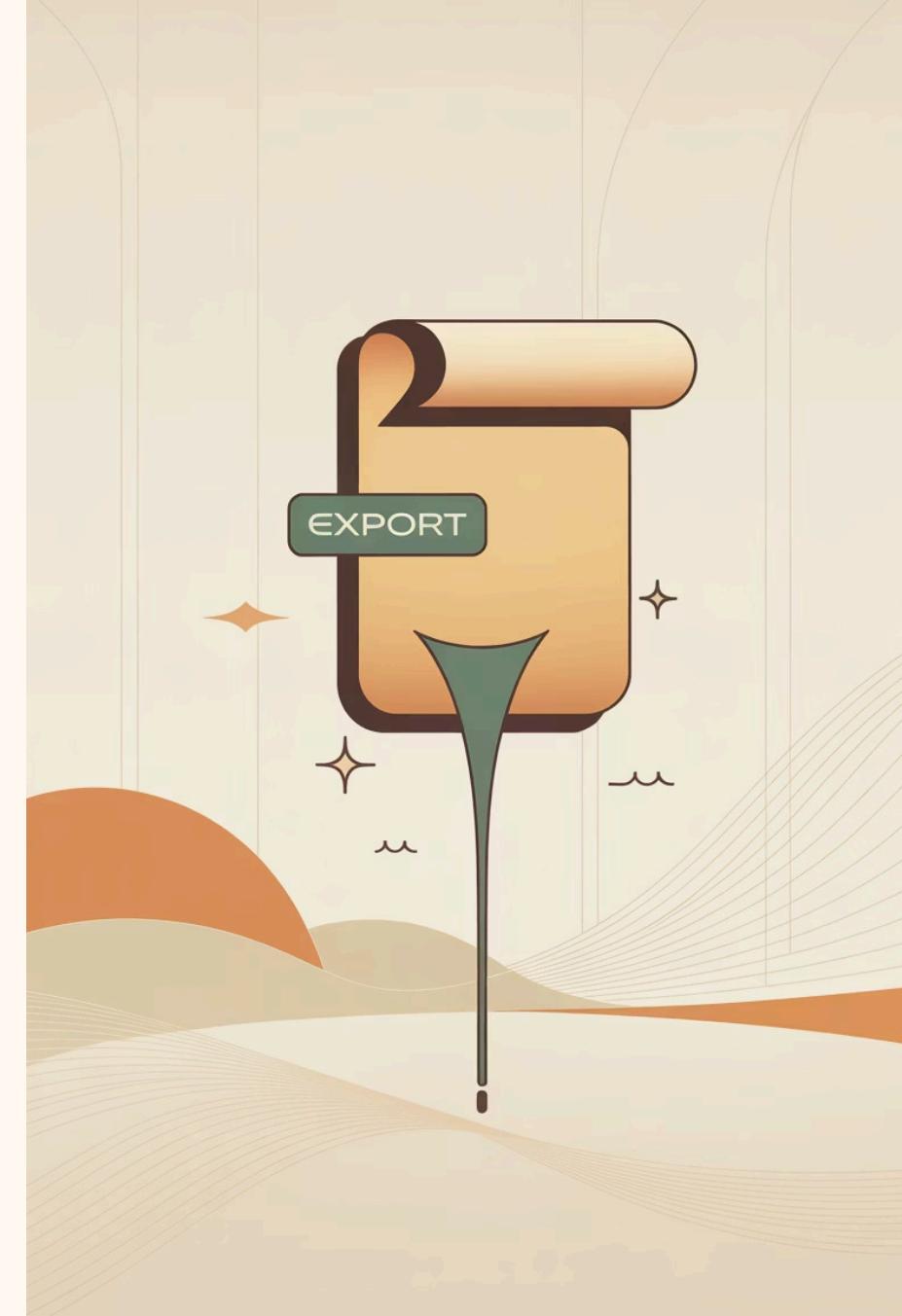


```
rows = db.list_notes(user_id, limit=1000)
fname = f"notes_{user_id}.txt"

with open(fname, "w", encoding="utf-8") as f:
    for r in rows:
        f.write(f"{r['id']}\t{r['text']}\n")

with open(fname, "rb") as f:
    bot.send_document(chat_id, f)
```

Пользователь получает файл со всеми своими заметками. Это создаёт "вау-эффект" — бот возвращает данные в удобном формате!





Опасность

```
DELETE FROM notes;
```

Эта команда сотрёт **ВСЕ** заметки всех пользователей!

Вывод: Всегда уточняем условия: WHERE user_id = ? AND id = ?

CAUTION



Избегаем "database is locked"



1

Контекст-менеджеры

`with _connect()` автоматически закрывает соединения

2

WAL режим

`journal_mode = WAL` позволяет читать во время записи

3

Таймаут ожидания

`busy_timeout = 5000` ждёт 5 секунд перед ошибкой



Логирование для отладки



```
logging.basicConfig  
    level=logging.INFO,  
    format="%(asctime)s [%(levelname)s] %(name)s: %(message)s"  
)  
  
logging.info("Adding note for user %s: %s", user_id, text[:20])
```

Логи показывают поток событий в консоли, помогают быстро найти проблему, когда "бот молчит".

Частые ошибки и решения



"no such table"

Вызвать init_db() или проверить DB_PATH

"database is locked"

Проверить WAL/timeout/with в _connect()

Пустой ответ

Проверить WHERE условия и LIMIT



UX: Удобство команд



Хороший бот подсказывает пользователю, что делать:

- Меню команд через `set_my_commands()`
- Краткие описания каждой команды
- Примеры использования в ответах
- Понятные сообщения об ошибках

Демо: "5 минут до результата"



Командная работа: распределение задач

Команда А

/note_add + /note_list

Создание и просмотр

Команда В

/note_find + /note_edit

Поиск и редактирование

Команда С

/note_del + /note_count

Удаление и статистика

Команда D

/note_export + ASCII-график

Экспорт и визуализация

15-20 минут на реализацию, затем интеграция!



Интеграция кода команд



Объединяем в main.py

Копируем все обработчики команд

Проверяем конфликты

Следим за дублированием имён функций

Тестируем все команды

Быстрый прогон каждой функции



SQL "вручную" в терминале



```
$ sqlite3 bot.db  
  
sqlite> INSERT INTO notes(user_id, text)  
VALUES (1, 'Привет из терминала!');  
  
sqlite> SELECT id, text FROM notes  
WHERE user_id = 1;
```

Полезно для отладки: видим, что бот делает то же самое, что мы можем сделать руками через sqlite3.

Параметризация vs f-строки



✗ Опасно

```
f"INSERT INTO notes VALUES ({user_id}, '{text}')"
```

SQL-инъекции, проблемы с кавычками

✓ Безопасно

```
"INSERT INTO notes VALUES (?, ?)"  
(user_id, text)
```

Автоматическое экранирование

Всегда используйте параметризованные запросы с символами ?

Пагинация (дополнительно)



```
SELECT id, text, created_at  
FROM notes  
WHERE user_id = ?  
ORDER BY id DESC  
LIMIT ? OFFSET ?;
```

Пример

/note_list 2 10

Страница 2, размер 10

Расчёт OFFSET

Страница 1: OFFSET 0

Страница 2: OFFSET 10

Страница 3: OFFSET 20



Поиск без учёта регистра



```
SELECT id, text  
FROM notes  
WHERE user_id = ?  
AND text LIKE '%' || ? || '%' COLLATE NOCASE  
ORDER BY id DESC;
```

COLLATE NOCASE делает поиск нечувствительным к регистру. Теперь "хлеб" найдёт и "Хлеб", и "ХЛЕБ".

Мини-бэкап SQLite



```
import sqlite3

def backup_database():
    with sqlite3.connect(DB_PATH) as src, \
        sqlite3.connect("backup.db") as dst:
        src.backup(dst)

    print("Бэкап создан: backup.db")
```

Мгновенный локальный бэкап всей базы данных в один файл. Идеально для сохранения состояния перед экспериментами.



Пример JOIN (бонус)

```
SELECT t.id, t.title, c.name AS category  
FROM tasks t  
LEFT JOIN categories c ON c.id = t.category_id  
WHERE t.user_id = ?  
ORDER BY t.id DESC  
LIMIT 20;
```

Мостик к следующему модулю "трекер задач". JOIN позволяет получать данные из связанных таблиц одним запросом.

Мини-квиз (1/2)

Вопрос 1

Что возвращает lastrowid?

Вопрос 2

Зачем проверять rowcount после
UPDATE/DELETE?

Вопрос 3

Чем опасен `DELETE FROM notes;`?

Ответы квиза (1/2)

1

lastrowid

ID последней вставленной записи



rowcount

Показывает количество изменённых строк



DELETE без WHERE

Удалит ВСЕ записи из таблицы!

Мини-квиз (2/2)

Вопрос 4

Как ограничить количество результатов поиска?

Вопрос 5

Какие настройки делают SQLite быстрее и стабильнее?

Ответы квиза (2/2)



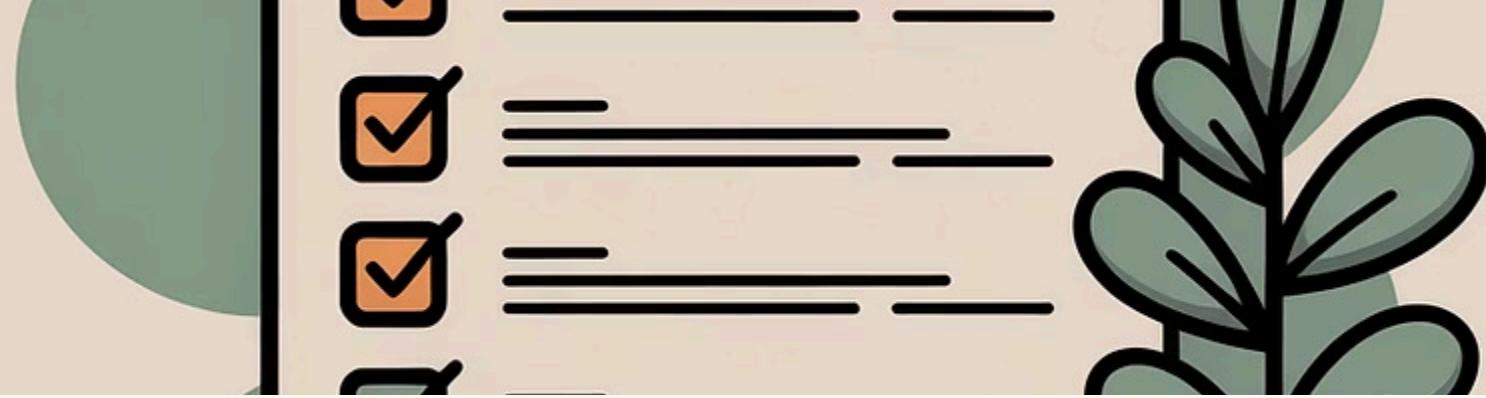
Ограничение выборки

LIMIT и комбинация LIMIT + OFFSET для пагинации



Оптимизация SQLite

WAL режим, busy_timeout, контекст-менеджеры with_connect()



Домашнее задание (обязательное)



1 Реализовать полный CRUD

/note_add, /note_list, /note_find, /note_edit, /note_del, /note_count

2 Загрузить на GitHub

В репозиторий курса, не забыть .gitignore для .env

Важно: Токен НЕ коммитим! Только .env в .gitignore

Домашнее задание (дополнительно) ★

1

Экспорт в файл

/note_export создаёт notes_*.txt и отправляет пользователю

2

Лимит заметок

Максимум 50 заметок на пользователя
(CHECK ограничение)

3

ASCII-визуализация

Гистограмма активности за неделю

Чек-лист готовности



База данных

init_db() вызывается, таблицы создаются



Команды работают

Все CRUD операции отвечают корректно



Безопасность

.env в .gitignore, токен не в коде



Документация

Краткий README с примерами команд

Анализ логов при работе



```
2025-01-15 10:30:15 [INFO] telebot: Received message: /note_add
2025-01-15 10:30:15 [INFO] db: Adding note for user 12345: "купить хлеб"
2025-01-15 10:30:15 [INFO] db: Note added with ID: 1
2025-01-15 10:30:15 [INFO] telebot: Sent reply: "Заметка #1 добавлена"
```

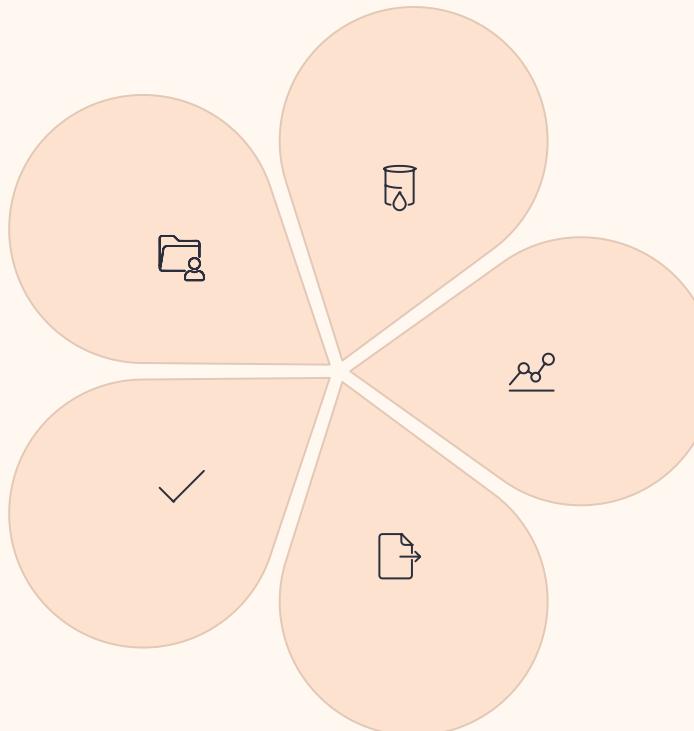
Логи показывают полный путь от получения команды до ответа пользователю. Это ключ к быстрому устранению проблем.

Наши достижения



Постоянное хранение
Бот умеет сохранять данные

Готовы к трекеру
Переходим к таблице tasks



CRUD операции

Создание, чтение, обновление, удаление

Статистика

Подсчёт и визуализация данных

Экспорт

Выгрузка данных в файлы

Обратная связь

Что было понятно?

Поделитесь успешными моментами

Что повторить?

Какие темы требуют закрепления

Нужна помощь с Git?

Поддержка в процессе загрузки на GitHub

Благодарность и что дальше 🚀

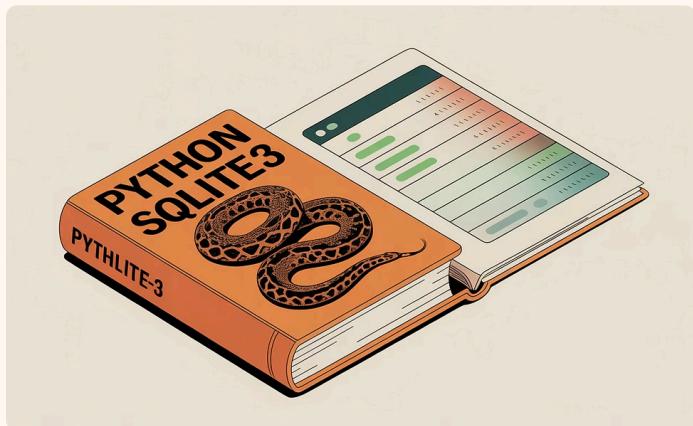
Спасибо за активную работу!

Следующий шаг в нашем путешествии — **"Задачи: таблица tasks, статусы, фильтры"**. Мы расширим наш бот до полноценного трекера задач с приоритетами и категориями.

- Репозиторий курса для загрузки домашних заданий
- Чат группы для вопросов и взаимопомощи
- Материалы семинара в общем доступе

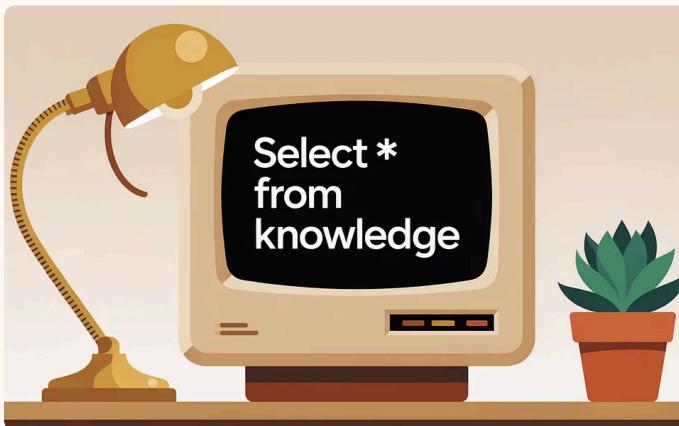


Ресурсы для самостоятельной работы



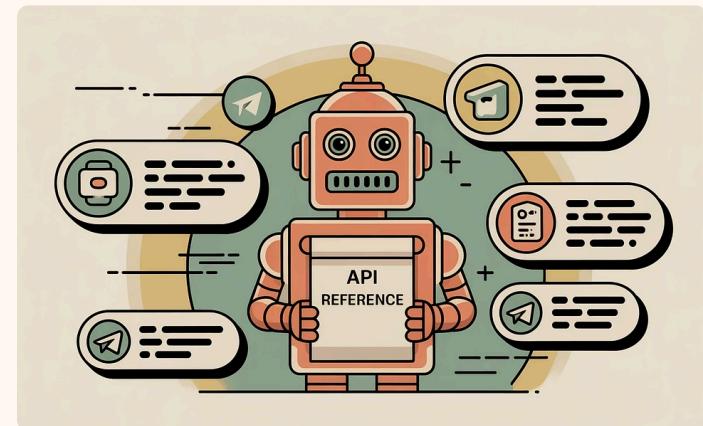
Документация sqlite3

Официальная документация Python для работы с SQLite



SQL туториалы

Дополнительные материалы для изучения SQL запросов



Telegram Bot API

Полный справочник возможностей Telegram ботов



До встречи!

Продолжайте экспериментировать с SQLite и Telegram ботами. Следующий семинар будет ещё интереснее!