

- 1. spanish-voice-trainer
- 2. Project Summary:
 - 2.1. 🛠️ The Core Technical Stack
 - 2.2. 🔄 How the System Works
- 3. Create Basic Structure as suggested
 - 3.1. Commands
- 4. use uv
- 5. Install Your Dependency Stack
- 6. checking the packages are installed
- 7. Running Scripts with uv Moving Forward
- 8. this is the basic
- 9. Developed sqllite schema
 - 9.1. The Structure inside database/connection.py
 - 9.2. Triggering It inside main.py
 - 9.3. Why This Placement is Ideal
 - 9.3.1. Idempotent Execution: Using CREATE TABLE IF NOT EXISTS means this code runs beautifully every time you start your app. If the database is already there, SQLite silently skips creation and proceeds to boot without overwriting your hard-earned Aula Internacional data.
 - 9.3.2. Crash Prevention: By placing it at the absolute top of main(), you ensure that no other component (like a UI field trying to load your textbook list) can execute queries against a database that hasn't finished setting up its columns yet.
 - 9.3.3. Clean Decoupling: Your root main.py handles the when (on boot), while database/connection.py holds the how (the specific table schemas).
- 10. It work 100%
- 11. App to read sqlite
- 12. brew
 - 12.1. Standard Formula (Default: No Flag)
 - 12.2. Cask Extension (--cask)
 - 12.3. Why This Is Useful
- 13. Beekeeper Studio
- 14. Does edge-tts always connect to the Microsoft Cloud ?
 - 14.1. How edge-tts Works (The Cloud Pipeline)
 - 14.2. Why This is Highly Advantageous for Phase 1
 - 14.3. Future Proofing: Going Fully Offline Down the Track
- Steps of app creation
- Gemini Reposne
- So we fix the bulk upload file
- Fix up of data
- Gemini initial response
- Gemini Response
- SQLite3
 - how to backup the sqlite3 database
 - how ro alter a table within the data base
 - to examine the PRAGMA table
 - 1. The Cleanest Output: .schema

- [sqlite3 schema](#)
- [sqlite3 fullschema](#)
- [Pro-Tip: Running it Interactively](#)
- [how to add a new table via command line](#)

1. spanish-voice-trainer

2. Project Summary:

Custom Spanish Voice TrainerA high-performance, completely private desktop application built on macOS to accelerate Spanish language training through automated flashcard creation and intelligent pronunciation analysis.The application utilizes a local-first architecture to ensure complete data privacy, storing all configurations, historical student analytics, and multimedia binary files strictly on the user's local drive.

2.1. The Core Technical Stack

- Environment & Package Management: uv (Rust-based Python package manager) for ultra-fast, isolated virtual environments and dependency locking.
- Database & Persistence: SQLite to map phrase metadata, local media file paths, and chronological user practice scores without external database infrastructure.
- Audio & Signal Processing: sounddevice for hands-free, voice-activated microphone capture; librosa and fastdtw (Dynamic Time Warping) to extract phoneme features (MFCCs) and score user pronunciation accuracy against reference files.
- Asset Generation: edge-tts to stream high-quality, neural text-to-speech Spanish audio clips, and Pillow (PIL) to auto-render widescreen flashcard JPEGs matching the phrases.
- User Interface: Developed in two phases—starting as a clean, text-based Command Line Interface (CLI) before migrating to a dual-mode desktop GUI built with PyQt6.

2.2. How the System Works

The application operates across two distinct, integrated operational frameworks managed via a unified QStackedWidget interface:

1. Content Creator Mode The user inputs a Spanish phrase and its English translation. The system automatically triggers the asset generator to output a custom high-quality flashcard image and a native-sounding neural audio file. The localized text strings and file paths are instantly committed to the SQLite database.
2. Student Training ModeThe system pulls cards from the database, displays the visual flashcard image, and plays the target Spanish pronunciation. The student speaks into their MacBook microphone. The system detects when the student begins and stops talking via a voice-activation threshold, records the sample, and runs a Dynamic Time Warping alignment algorithm to provide an objective pronunciation match score (e.g., 87% accuracy).
3.  Long-Term Capability: Custom Video Compilations Because all image assets share standard HD video dimensions (1280 X 720) and all audio samples are tracked deterministically in the database, the core engine can be commanded to interface with ffmpeg-python. It can seamlessly

compile entire batches of database assets into standalone, continuous .mp4 video lessons complete with timed visual pauses and silent audio gaps, providing an additional passive learning medium for language immersion.

3. Create Basic Structure as suggested

```
spanish-voice-trainer/
├── .gitignore
├── pyproject.toml
├── uv.lock
├── README.md
├── main.py                # Entry point for the application
├── database/              # Storage & Data Layer
│   ├── __init__.py
│   └── connection.py      # SQLite schema initialization and CRUD
├── queries
│   ├── core/              # Pure Python Business Logic (The Engine)
│   │   ├── __init__.py
│   │   └── audio_engine.py # Recording (sounddevice) and scoring
│   └── (librosa/DTW)
│       └── asset_generator.py # TTS generation (edge-tts) and flashcard
│           creation (Pillow)
├── ui/                    # Presentation Layer (Views)
│   ├── __init__.py
│   ├── cli/
│   │   └── interface.py    # Simple command-line menus for Phase 1
│   └── gui/
│       ├── interface.py    # Main QMainWindow shell for Phase 2
│       ├── creator_mode.py # QWidget for content creation panel
│       └── trainer_mode.py # QWidget for student practice panel + QThread
├── workers
│   └── media/              # Local storage for physical binary files
│       └── .gitkeep        # Keeps directory alive in Forgejo repo
```

3.1. Commands

```
stephenlohning@Scotty 139_spanish-voice-trainer % mkdir doc
stephenlohning@Scotty 139_spanish-voice-trainer % mkdir doc/images
stephenlohning@Scotty 139_spanish-voice-trainer % touch .gitignore
stephenlohning@Scotty 139_spanish-voice-trainer % git branch -M main
stephenlohning@Scotty 139_spanish-voice-trainer % git remote -v
origin  https://dev.oxnee.com/stephen/139_spanish-voice-trainer.git (fetch)
origin  https://dev.oxnee.com/stephen/139_spanish-voice-trainer.git (push)
stephenlohning@Scotty 139_spanish-voice-trainer % git config --global
push.followTags true
stephenlohning@Scotty 139_spanish-voice-trainer % touch main.py
stephenlohning@Scotty 139_spanish-voice-trainer % mkdir database
stephenlohning@Scotty 139_spanish-voice-trainer % touch
```

```

database/__init__.py
stephenlohning@Scotty 139_spanish-voice-trainer % touch
database/connection.py
stephenlohning@Scotty 139_spanish-voice-trainer % mkdir core
stephenlohning@Scotty 139_spanish-voice-trainer % touch core/__init__.py
stephenlohning@Scotty 139_spanish-voice-trainer % touch
core/audio_engine.py
stephenlohning@Scotty 139_spanish-voice-trainer % touch
core/asset_generation.py
stephenlohning@Scotty 139_spanish-voice-trainer % mkdir ui
stephenlohning@Scotty 139_spanish-voice-trainer % touch ui/__init__.py
stephenlohning@Scotty 139_spanish-voice-trainer % mkdir ui/cli
stephenlohning@Scotty 139_spanish-voice-trainer % touch ui/cli/interface.py
stephenlohning@Scotty 139_spanish-voice-trainer % mkdir ui/gui
stephenlohning@Scotty 139_spanish-voice-trainer % touch ui/gui/interface.py
stephenlohning@Scotty 139_spanish-voice-trainer % touch
ui/gui/create_mode.py
stephenlohning@Scotty 139_spanish-voice-trainer % touch
ui/gui/trainer_mode.py
stephenlohning@Scotty 139_spanish-voice-trainer % mkdir media
stephenlohning@Scotty 139_spanish-voice-trainer % touch media/.gitkeep

```

4. use uv

```

stephenlohning@Scotty 139_spanish-voice-trainer % uv venv
Using CPython 3.13.5
Creating virtual environment at: .venv
Activate with: source .venv/bin/activate
stephenlohning@Scotty 139_spanish-voice-trainer %

```

5. Install Your Dependency Stack

Run the uv add commands to populate your environment lockfile:

```
uv add sounddevice numpy scipy librosa fastdtw pillow edge-tts pyqt6
```

6. checking the packages are installed

```

(139_spanish-voice-trainer) stephenlohning@Scotty 139_spanish-voice-trainer
% uv pip list
Package              Version
-----
aiohappyeyeballs     2.6.2

```

aiohttp	3.14.0
aiosignal	1.4.0
attrs	26.1.0
audioop-lts	0.2.2
audioread	3.1.0
certifi	2026.5.20
cffi	2.0.0
charset-normalizer	3.4.7
decorator	5.3.1
edge-tts	7.2.8
fastdtw	0.3.4
frozenlist	1.8.0
idna	3.18
joblib	1.5.3
lazy-loader	0.5
librosa	0.11.0
llvmlite	0.47.0
msgpack	1.1.2
multidict	6.7.1
narwhals	2.22.1
numba	0.65.1
numpy	2.4.6
packaging	26.2
pillow	12.2.0
platformdirs	4.10.0
pooch	1.9.0
propcache	0.5.2
pycparser	3.0
pyqt6	6.11.0
pyqt6-qt6	6.11.1
pyqt6-sip	13.11.1
requests	2.34.2
scikit-learn	1.9.0
scipy	1.17.1
sounddevice	0.5.5
soundfile	0.14.0
soxr	1.1.0
standard-aifc	3.13.0
standard-chunk	3.13.0
standard-sunau	3.13.0
tabulate	0.10.0
threadpoolctl	3.6.0
typing-extensions	4.15.0
urllib3	2.7.0
yaml	1.24.2

7. Running Scripts with uv Moving Forward

One of the cleanest features of uv is that you don't necessarily have to remember to run `source .venv/bin/activate` every single time you open a new terminal window to write code.

Instead, you can prefix your execution scripts using `uv run`. It automatically detects your local `.venv`, verifies that all dependencies in your lockfile are fully up to date, and runs the script seamlessly:

```
# Example: To execute your main script once you write it later today
uv run main.py
```

8. this is the basic

9. Developed sqllite schema

This was easy I just discussed what I thought we should have and gemini suggested the following

9.1. The Structure inside database/connection.py

In this file, you define your schema and wrap the connection/creation logic inside a function called `init_db()`.

```
# database/connection.py
import sqlite3
import os

DB_NAME = "spanish_trainer.db"

def get_connection():
    """Returns a standard connection object to the SQLite database."""
    return sqlite3.connect(DB_NAME)

def init_db():
    """
    Initializes the SQLite database tables if they do not exist.
    This safely runs on every boot without wiping your existing data.
    """
    print(f"🔍 Checking database status for '{DB_NAME}'...")

    # The SQL schema we designed for your glossary, cross-references, and
    tracks
    schema = """
    CREATE TABLE IF NOT EXISTS phrases (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        text TEXT NOT NULL,
        language TEXT NOT NULL,
        textbook TEXT DEFAULT NULL,
        unit INTEGER DEFAULT NULL,
        source_context TEXT DEFAULT NULL,
        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );
```

```

CREATE TABLE IF NOT EXISTS translations (
    source_phrase_id INTEGER,
    target_phrase_id INTEGER,
    PRIMARY KEY (source_phrase_id, target_phrase_id),
    FOREIGN KEY (source_phrase_id) REFERENCES phrases(id) ON DELETE
CASCADE,
    FOREIGN KEY (target_phrase_id) REFERENCES phrases(id) ON DELETE
CASCADE
);

CREATE TABLE IF NOT EXISTS audio_tracks (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    phrase_id INTEGER NOT NULL,
    voice_gender TEXT NOT NULL,
    voice_name TEXT NOT NULL,
    file_path TEXT NOT NULL,
    is_reference INTEGER DEFAULT 1,
    FOREIGN KEY (phrase_id) REFERENCES phrases(id) ON DELETE CASCADE
);
"""

conn = get_connection()
try:
    cursor = conn.cursor()
    # executescript allows running multiple CREATE TABLE statements at
once
    cursor.executescript(schema)
    conn.commit()
    print("✅ Database tables verified and initialized successfully.")
except sqlite3.Error as e:
    print(f"❌ Database initialization failed: {e}")
finally:
    conn.close()

```

9.2. Triggering It inside main.py

Now, your root startup script handles execution order perfectly. Before launching the UI, it imports and fires `init_db()`. If the `.db` file doesn't exist, SQLite handles creating it seamlessly on the spot.

```

# main.py
import sys
from database.connection import init_db

def main():
    print("🚀 Booting Castilian Voice Trainer...")

    # Step 1: Ensure infrastructure is present before doing anything else
    init_db()

    # Step 2: Initialize Core Engines (AudioEngine, AssetGenerator)
    # Step 3: Launch your PyQt6 Graphical User Interface

```

```

print("🚀 Launching GUI...")

# (PyQt6 window exec loops will go here)

if __name__ == "__main__":
    main()

```

9.3. Why This Placement is Ideal

9.3.1. Idempotent Execution: Using CREATE TABLE IF NOT EXISTS means this code runs beautifully every time you start your app. If the database is already there, SQLite silently skips creation and proceeds to boot without overwriting your hard-earned Aula Internacional data.

9.3.2. Crash Prevention: By placing it at the absolute top of main(), you ensure that no other component (like a UI field trying to load your textbook list) can execute queries against a database that hasn't finished setting up its columns yet.

9.3.3. Clean Decoupling: Your root main.py handles the when (on boot), while database/connection.py holds the how (the specific table schemas).

10. It work 100%

```

It Looks like it worked
stephenlohning@Scotty 139_spanish-voice-trainer % source
/Users/stephenlohning/Documents/01_Projects/139_spanish-voice-train
er/.venv/bin/activate
(139_spanish-voice-trainer) stephenlohning@Scotty 139_spanish-voice-trainer
% uv run main.py
🚀 Booting Castilian Voice Trainer...
📁 Checking database status for 'spanish_trainer.db'...
✅ Database tables verified and initialized successfully.
🚀 Launching GUI...

```

```

Create database in root
(139_spanish-voice-trainer) stephenlohning@Scotty 139_spanish-voice-trainer
% ls -la *.db
-rw-r--r--  1 stephenlohning  staff   24576 Jun 12 21:04 spanish_trainer.db
(139_spanish-voice-trainer) stephenlohning@Scotty 139_spanish-voice-trainer
%

```

11. App to read sqlite

Gemini gave a few suggestions, I picked the second one "Bee Keeper Studio"

I used brew to install it

```
### Top Modern/Aesthetic Option: Beekeeper Studio (Community Edition)
If you want something that matches modern macOS design language (with
gorgeous dark mode formatting, clean autocomplete tabs, and high-DPI
scaling), Beekeeper is phenomenal. It is fast, private, and highly visual.
* **How to install via Homebrew:**
  ```bash
brew install --cask beekeeper-studio
```

I did not know what `--cask` was doing so I asked, and I got a bit of info about brew the installation manager.

## 12. brew

---

In Homebrew, the `--cask` flag tells the package manager that you want to install a macOS desktop application that comes with a Graphical User Interface (GUI), rather than a command-line tool.

Here is the breakdown of how Homebrew treats these two styles of software:

### 12.1. Standard Formula (Default: No Flag)

When you type `brew install`, Homebrew downloads, compiles, or unpacks command-line utilities or developer libraries. These programs do not have an icon in your Applications folder and run entirely inside your terminal.

Examples: `uv`, `git`, `sqlite3`, `ffmpeg`

### 12.2. Cask Extension (`--cask`)

When you type `brew install --cask`, Homebrew downloads a pre-compiled macOS binary directly from the developer (usually packaged as a `.dmg`, `.pkg`, or `.app` file). It automatically runs the installer, bypasses the drag-and-drop step, and places the application straight into your `/Applications` folder.

Examples: `beekeeper-studio`, `visual-studio-code`, `db-browser-for-sqlite`

### 12.3. Why This Is Useful

Instead of opening Safari, searching for Beekeeper Studio, downloading a disk image, opening it, dragging the icon to your Applications folder, and cleaning up the installer file, Homebrew does all of that for you behind the scenes in a single terminal line.

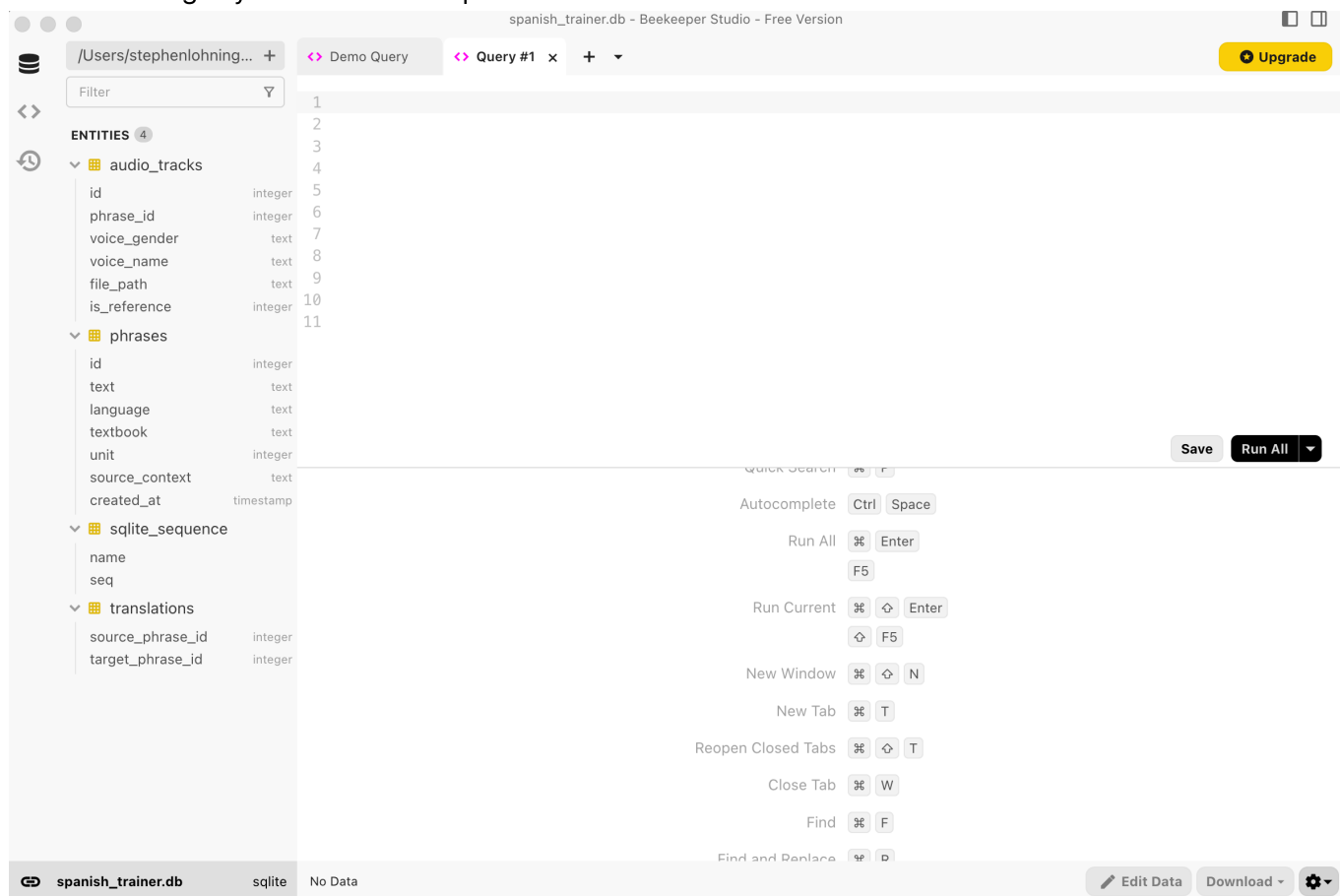
Furthermore, whenever you run your system updates down the road using `brew upgrade`, Homebrew will automatically update your desktop apps right alongside your command-line tools!

## 13. Beekeeper Studio

---

Has an introduction YouTube video

The main thing is you double click upon the data base.db we created.



## 14. Does edge-tts always connect to the Microsoft Cloud ?

The short answer is yes.

edge-tts is a Python library that wraps around the communication protocol used by Microsoft Edge's built-in "Read Aloud" feature. Because it doesn't run neural voice models locally on your MacBook, it must make an active network connection to the Microsoft cloud servers every time you request audio synthesis.

Here is a closer look at what is happening under the hood when you call `await communicate.save()` on Scotty:

### 14.1. How edge-tts Works (The Cloud Pipeline)

Instead of keeping massive gigabyte-sized neural networks running in your local system memory, edge-tts leverages Microsoft's cloud infrastructure using secure WebSockets.

**The Request:** Your Python script establishes an asymmetrical WebSocket connection to a Microsoft Azure speech endpoint.

**The Handshake:** It passes your text string along with configuration meta-data (like requesting the specific es-ES-AlvaroNeural Castilian voice).

The Synthesis: Microsoft's remote cloud supercomputers synthesize the text into human-like phonemes in real time.

The Stream: The server streams raw audio binary packets back down to edge-tts, which chunks them cleanly together and writes them to disk as your target .mp3 file.

Because this relies entirely on a remote API endpoint, if Scotty is completely offline or disconnected from Wi-Fi, the edge-tts script will throw a connection error and fail to generate audio.

## 14.2. Why This is Highly Advantageous for Phase 1

While running things offline is fantastic for complete privacy, using a cloud-backed pipeline for your audio assets right now offers three massive benefits for your project development:

Zero Hardware Strain: Deep-learning voice synthesis is computationally heavy. Running a true high-fidelity audio model locally requires significant CPU/GPU processing. edge-tts offloads 100% of that heavy lifting to the cloud, meaning your script runs instantly in seconds without draining your MacBook's battery or spinning up the fans.

Instant, Production-Ready Audio: You don't have to download, compile, or tune massive multi-gigabyte voice weights (like tortoise-tts or Bark) inside your Python environment. You get pristine, studio-quality Castilian speech with two lines of code.

## 14.3. Future Proofing: Going Fully Offline Down the Track

Because you prefer data privacy and local-first software engineering, it's worth noting that your current architecture is completely future-proofed for offline migration when you reach an advanced phase of development.

Because we explicitly decoupled your code—separating the asset creation logic into `core/asset_generator.py` and the application logic into `main.py`—you can swap out the backend audio engine later without changing a single line of your GUI code.

If you want to make your voice trainer run 100% locally in the future without hitting the cloud, you can replace the internal logic of `AssetGenerator` with an offline framework like:

Piper TTS: A fast, local, neural text-to-speech system optimized to run beautifully on local hardware architectures.

Kokoro: A highly lightweight, open-source local voice model that produces incredibly realistic human inflections.

For now, leveraging Microsoft's cloud via edge-tts gives you native-speaker accuracy with zero setup friction, letting you focus entirely on mastering your Aula Internacional curriculum!

I suppose this OK in phase 1 , I may want to migrate Kokoro next

## Steps of app creation

---

```
(139_spanish-voice-trainer) stephenlohning@Scotty 139_spanish-voice-trainer
% git tag -n
list basic framework created
v1.0 Tested text speak with 2 files added to the root directory
v2.0 added db creation, to project, also loaded installed Bee
Keeper Studio on Scotty
v3.0 phase manager added main modified to test this,
bulk_importer drafted not tested
v4.0 bulk importer did not work using docling, so we going to
export the markdown into a text file as an intermediate step
v5.0 modified to export the markdown file, to be used to modify
bulk importer
```

v5.0 docling gave us

```
alphabetical GLOSSARY
```

años60 m	1960s	U8_7A	bañ osmpl
bath	U1_1A		
-----	-----	-----	-----
-----	-----	-----	
aparecer (zc)	toappear	U3_6A	bar m
bar	U1_LEX		
apasionado/a	passionate	U5_15C	barato/a
cheap	U4_6C		
apellido m	surname	U1_4A	barba f
beard	U5_9A		
aprender	tolearn	U2_2C	Barcelona
Barcelona	U3_8A		

## Gemini Reposne

Col 1 (ES)	Col 2 (EN)	Col 3 (Unit)	Col 4 (ES)	Col 5 (EN)	Col 6
(Unit)					

## So we fix the bulk upload file

```
git tag -a v6.0 -m "working version of bulk importer, but it leaves grammar
info"
```

## Fix up of data

---

We should probably remove the grammar categorisation which is done. 5 años60 m es Aula Internacional Plus 1 87 U87A 2026-06-13 03:00:53 The m 9 aparecer (zc) es Aula Internacional Plus 1 36 U36A 2026-06-13 03:00:53 The (zc) 13 apasionado/a es Aula Internacional Plus 1 515 U515C 2026-06-13 03:00:53 The /a maybe has be duplicated into two words

## Gemini initial response

---

Was was to make a module to clean up the data, I said that we should be keeping the data

## Gemini Response

---

You raise an exceptional database design point. You are absolutely right: knowing whether a word is a masculine noun, a feminine noun, or an irregular verb is vital linguistic data that we shouldn't just throw away.

The issue isn't that the information is bad, but rather that we were storing it directly inside the text field itself. Keeping it there causes two major problems:

Audio Synthesis Ruin: As mentioned, edge-tts will literally read aloud "años sesenta eme" or "aparecer zeta ce".

Search and Match Failure: If you query your database for the word "aparecer", a direct lookup will fail because the database thinks the word is literally "aparecer (zc)".

The Relational Solution: Keep the Info, Just Move the Slot The cleanest engineering approach is to strip the grammar tags out of the conversational text field, but save them into dedicated metadata columns on the exact same row.

Let's modify your SQLite phrases schema to add columns for word\_type (noun, verb, adjective, phrase) and grammar\_note (m, f, zc, ie). This keeps your data rich and informative, keeps your text search crisp, and lets us pass clean strings to the voice engine.

Here is the plan to gracefully adjust your schema and run an updated, data-preserving migration:

This requires adding another column to our database

## SQLite3

---

### how to backup the sqlite3 database

```
sqlite3 database/trainer.db ".backup 'database/trainer_backup_$(date +%Y%m%d_%H%M%S).db'"
```

## how to alter a table within the data base

```
sqlite3 spanish_trainer.db "ALTER TABLE translations ADD COLUMN tags TEXT
DEFAULT ' ';"
```

## to examine the PRAGMA table

```
sqlite3 spanish_trainer.db "PRAGMA table_info(translations);"
```

## 1. The Cleanest Output: .schema

The most precise command is the dot-command .schema. You can pass it directly into the CLI call:

### sqlite3 schema

```
sqlite3 spanish_trainer.db ".schema"
```

Why this is exactly what you need:

It prints out the exact SQL CREATE TABLE statements used to build your entire database structure.

It automatically includes any constraints (like PRIMARY KEY, FOREIGN KEY, or UNIQUE).

It shows all other structural items, such as your indexes and triggers, giving you a complete blueprint of the file.

2. The Comprehensive Alternative: .fullschema If you ever start implementing complex custom database views or virtual tables down the road, you can use:

### sqlite3 fullschema

```
sqlite3 spanish_trainer.db ".fullschema"
```

This does everything .schema does, but it also appends statistical metadata and structural configuration variables used to optimize the query planner.

## Pro-Tip: Running it Interactively

If you are already inside an active SQLite session using your terminal, you don't need to specify the database name or use quotation marks. You can just type the dot-command at the prompt:

SQL sqlite> .schema If you want a more compact table checklist just to see what names are present before inspecting their structures, you can use:

SQL sqlite> .tables

how to add a new table via command line

```
sqlite3 spanish_trainer.db "CREATE TABLE IF NOT EXISTS settings (key TEXT
PRIMARY KEY, value TEXT);"
```