**Essence of Each Step: Building Your Materials & Guitar Website**

**The Big Analogy**

* **Your Website = A Small Library + Music Room.**
  + **Materials Data = Books on shelves (with info cards).**
  + **Guitar Clips = MP4 tracks on a jukebox.**
  + **Database = Library’s master catalog (tells you what books and tracks exist, their details, lets you update the list).**

**Step 1: Set Up Your Toolbox**

**Purpose:** Install tools so you can *build and run* the site locally on your PC.

* **Python:** Like your toolbox for building anything.
* **Flask:** The “library manager”—a simple Python program that hands out books (materials data) and lets people play music (guitar clips).
* **SQLite:** Your “library card index”—a file on your disk that acts as your *mini-database*. It holds all your materials and clip records.

**RDMS Concepts Here:**

* Database = A structured digital “notebook” (SQLite file).
* Table = A specific list (materials or guitar clips).
* Row = One entry (one material or one clip).
* Column = Info about that entry (name, file path, description).

**Pro Tip:**

* Use **SQLite** now for zero hassle. Upgrade to PostgreSQL later if you ever need a big, multi-user database.

**Step 2: Sketch Your Database**

**Purpose:** Decide what info you want to store.  
*Analogy: Designing the index cards for your books and tracks.*

* **Materials Table:** Like an index card per book: ID, name, description, property1, etc.
* **Guitar Clips Table:** Card per music track: ID, title, filepath, description.

**RDMS Concepts Here:**

* **Schema:** The blueprint for what each card looks like (table definition).
* **Primary Key:** Unique ID number for each card (so you don’t confuse two entries).

**Pro Tip:**

* Keep the schema simple at first! You can always add more columns later.

**Step 3: Build the App’s Skeleton (Flask)**

**Purpose:** Make a basic website that lets you see your data.  
*Analogy: Putting up the “library walls” and “jukebox”, even before books/music are in place.*

* **Flask Routes**: Different “rooms” in your library.
  + /materials — Shows all materials
  + /add-material — Lets you add a new material
  + /guitar — Shows all your guitar clips (with play buttons for mp4)
* **HTML Templates:** Simple web pages (like decorated walls).

**RDMS Concepts Here:**

* **CRUD** (Create, Read, Update, Delete):
  + View all entries (Read)
  + Add a new entry (Create)
  + Edit an entry (Update) – optional at first
  + Delete an entry (Delete) – optional at first

**Pro Tips:**

* Start with just listing and adding (Read + Create).
* Use Flask’s built-in development server; it restarts automatically.
* Use Bootstrap for instant style (just paste one line in your HTML).

**Step 4: Connect to Your Database**

**Purpose:** Link the “walls” (web pages) to the “card catalog” (database).  
*Analogy: When someone clicks “See all books,” your app pulls out the index cards and shows the info on the wall.*

* Use Python’s sqlite3 library to read/write database entries.
* On /materials, run a SQL query to fetch all materials and show them.
* On /add-material, insert a new row when the user submits the form.
* Store file paths for mp4s in the database, not the files themselves.

**RDMS Concepts Here:**

* **SQL queries:** The language you use to “search the cards” (SELECT), “add a card” (INSERT), etc.
* **Foreign Key:** Not needed yet, but later you could link materials to clips, etc.

**Pro Tip:**

* Don’t store mp4s **inside** the database—just store their file paths, and keep actual files in a /static folder.

**Step 5: Add Guitar Clip Upload/Playback**

**Purpose:**  
Let users upload mp4 files and play them in the browser.

*Analogy: Adding the jukebox and a slot to add new tracks.*

* Use a simple Flask form for file upload.
* Store files in static/guitar\_clips/, and their paths in the DB.
* Use the HTML <video> tag to let people play clips right on the site.

**RDMS Concepts Here:**

* More practice with **CRUD**—now with media.

**Pro Tips:**

* Limit uploads to .mp4.
* Auto-generate filenames to avoid conflicts.
* You can always delete and re-upload from the admin side.

**Step 6: Deploy to the Web**

**Purpose:** Make your library public!

*Analogy: Opening your library’s front door to the world (with a doorman—your web host).*

* Free hosts: [Render.com](https://render.com/), [PythonAnywhere](https://pythonanywhere.com/)
* Sign up, connect your GitHub repo, and let the host run your Flask app.
* Upload your 5 mp4s via your admin interface, or copy them into the host’s file manager.

**RDMS Concepts Here:**

* **Production environment:** Running your app “for real,” not just on your PC.
* **Backups:** Export your database file regularly (it’s just one .db file in SQLite).

**Pro Tips:**

* Keep your secret keys/configs out of the code (use .env files).
* Always test locally before pushing to the web.
* Document your repo with a simple README.md.

**Summary Table: Steps, Outputs, RDMS Concepts**

| **Step** | **Output on Website** | **RDMS Idea in Action** | **Pro Dev Tips** |
| --- | --- | --- | --- |
| 1 | Local server, blank pages | Create DB & tables | Use SQLite for ease |
| 2 | Data structure ready | Schema, PKs | Sketch on paper first |
| 3 | Pages for materials/guitar | CRUD basics | Use templates |
| 4 | Dynamic listing, add forms | SQL queries | Bootstrap for style |
| 5 | Upload, play mp4s in browser | CRUD w/media paths | Store paths only |
| 6 | Live site on the internet | Production, backup | Test before deploy |