Implementing a **Retrieval-Augmented Generation (RAG) pipeline** for interacting with semi-structured PDF data involves several key steps. Let me guide you through the implementation and provide insights into handling the given example. Here's how you can proceed:

---

1. Data Ingestion

Steps:

1. **Extract Text from PDF:**
   - Use libraries like PyPDF2, pdfminer.six, or PyMuPDF to extract text.
   - For structured information, OCR tools like Tesseract may help if the PDF contains scanned images.
2. **Segment Data:**
   - Divide extracted text into chunks (e.g., paragraphs, tables, or sections).
   - Logical segmentation can be achieved by using regular expressions or headings to delimit sections.
3. **Generate Embeddings:**

    - Use a pre-trained model like OpenAI’s text-embedding-ada-002 or Sentence-BERT to create vector embeddings for each chunk.

4. **Store in a Vector Database:**
    - Store embeddings and metadata (e.g., page number, section) in a vector database such as Pinecone, Weaviate, or FAISS for efficient similarity-based retrieval.

Code:

```
from PyPDF2 import PdfReader

from sentence_transformers import SentenceTransformer

import faiss


# Extract text from PDF

reader = PdfReader("example.pdf")

text = ""

for page in reader.pages:

    text += page.extract_text()
```

```
# Split text into chunks
chunks = text.split("\n\n")  # Split by double newline as a simple heuristic

# Generate embeddings
model = SentenceTransformer('all-MiniLM-L6-v2')  # Replace with your embedding model
embeddings = model.encode(chunks)

# Store in vector database (e.g., FAISS)
dimension = embeddings.shape[1]
index = faiss.IndexFlatL2(dimension)
index.add(embeddings)
```

---

2. Query Handling

Steps:

1. **Embed the User Query:**
   - Convert the user's question into an embedding using the same model.
2. **Similarity Search:**

   - Retrieve the most relevant chunks from the vector database using cosine similarity or Euclidean distance.
3. **Pass to LLM:**
   - Combine retrieved chunks with a prompt and feed them into an LLM like GPT to generate responses.

Code:

```
# Embed the user query
query = "What is the unemployment information based on degree type on page 2?"
query_embedding = model.encode([query])

# Retrieve relevant chunks
D, I = index.search(query_embedding, k=3)  # Retrieve top 3 matches
relevant_chunks = [chunks[i] for i in I[0]]
```

---

3. Handling Comparison Queries

Steps:

1. Identify key terms in the query that suggest a comparison (e.g., "compare", "difference", "contrast").
2. Extract corresponding chunks from multiple files.
3. Perform structured aggregation of extracted data (e.g., organizing into a table).

Code for Comparison:

```
comparison_query = "Compare unemployment rates for bachelor’s and master’s degrees."

query_embedding = model.encode([comparison_query])

D, I = index.search(query_embedding, k=5)


# Extract chunks and aggregate

comparison_chunks = [chunks[i] for i in I[0]]

# Use structured aggregation (e.g., parsing numerical data) for better comparison.
```

---

4. Response Generation

Steps:

1. Pass the retrieved data and user query as a prompt to the LLM.
2. Ensure factuality by explicitly including retrieved data in the generated response.

Example Prompt for LLM:

"Based on the retrieved information, provide a comparison of unemployment rates for bachelor's and master's degrees as presented on page 2. Format your response in a tabular format."

Handling the Example Data:

Page 2 - Unemployment by Degree Type:

- Extract relevant text or table containing unemployment rates.
- Ensure chunks containing "unemployment" and "degree" are included in retrieval.

Page 6 - Tabular Data:

- Parse tabular data using libraries like camelot or tabula for precise extraction.
- Store the extracted table data in a structured format (e.g., CSV or DataFrame).

Code for Table Extraction:

```
import camelot

tables = camelot.read_pdf("example.pdf", pages="6")
tables[0].to_csv("page6_table.csv")
```

Tools and Libraries:

- **PDF Extraction:** PyPDF2, pdfminer.six, camelot, tabula.
- **Embeddings:** SentenceTransformers, OpenAI’s Embedding API.
- **Vector Database:** FAISS, Pinecone, Weaviate.
- **LLM Integration:** OpenAI API, Hugging Face Transformers.