# PDF Processing & Query System - Explanation

# PDF Processing & Query System - Full Explanation

## Overview
This system extracts tables from PDFs, stores them in DuckDB & Neo4j, and retrieves information using a ColPali model for AI-powered search.

## Step 1: Import Required Libraries
```python
import pdfplumber  # PDF table extraction
import duckdb  # In-memory database for structured data
import torch  # Machine Learning computations
import pandas as pd  # Handling tabular data
from transformers import AutoProcessor  # Query/Image processing
from neo4j import GraphDatabase  # Neo4j connection
from colpali_engine.models.paligemma_colbert_architecture import ColPali  # AI retrieval
from colpali_engine.utils.image_from_page_utils import load_from_dataset  # Image extraction
from torch.utils.data import DataLoader  # Batch processing
from tqdm import tqdm  # Progress bar
from PIL import Image  # Image handling
```

## Step 2: Extract Tables from PDF & Store in Databases

### PDFProcessor Class
Extracts tables and saves them in DuckDB & Neo4j.

```python
class PDFProcessor:
    def __init__(self, pdf_path):
        self.pdf_path = pdf_path
        self.db = duckdb.connect(":memory:")  # In-memory DuckDB
        self.graph_driver = GraphDatabase.driver("bolt://localhost:7687", auth=("neo4j", "newpassword"))
```

### Extract Tables from PDF

```python
    def extract_tables(self):
        tables = []
        with pdfplumber.open(self.pdf_path) as pdf:
            for page in pdf.pages:
                extracted_tables = page.extract_tables()
                for table in extracted_tables:
                    if table:
                        tables.append(table)
        return tables
```

### Store Tables in DuckDB

```python
    def store_in_db(self, tables):
        for i, table in enumerate(tables):
            if table and table[0]:
                df = pd.DataFrame(table[1:], columns=table[0])
                self.db.execute(f"CREATE TABLE table_{i} AS SELECT * FROM df")
```

## Step 3: Store Data in Neo4j

```python
    def store_in_graphdb(self, tables):
        with self.graph_driver.session() as session:
            for table in tables:
                if not table or not table[0]: continue
                headers = table[0]
                for row in table[1:]:
                    properties = {col: val for col, val in zip(headers, row) if col}
                    if properties:
                        session.run("CREATE (n:TableEntry $props)", props=properties)
```

## Step 4: Semantic Retrieval using AI Model

### ColPali Model for Retrieval

```python
class SemanticRetrieval:
    def __init__(self):
        model_name = "akshayballal/colpali-merged"
        device = "cuda" if torch.cuda.is_available() else "cpu"
        self.model = ColPali.from_pretrained(model_name, torch_dtype=torch.bfloat16, device_map=device).eval()
        self.processor = AutoProcessor.from_pretrained(model_name)
```

## Step 5: Process User Queries

### QueryProcessor Class

```python
class QueryProcessor:
    def __init__(self):
        self.semantic_retrieval = SemanticRetrieval()
        self.graph_driver = GraphDatabase.driver("bolt://localhost:7687", auth=("neo4j", "password"))
        self.db = duckdb.connect(":memory:")
```

### Process User Query

```python
    def process_query(self, query):
        if "qualification" in query.lower() or "composition" in query.lower():
            return self.graphdb_lookup(query)
        else:
            return self.semantic_retrieval.retrieve_info(query)
```

## Step 6: Run the Pipeline

```python
if __name__ == "__main__":
    pdf_processor = PDFProcessor("sample.pdf")
    pdf_processor.process_pdf()

    query_processor = QueryProcessor()
    user_query = "What is the tensile strength of Beta Annealed Ti64 Plates?"
    result = query_processor.process_query(user_query)

    print("Answer:", result)
```