Data Warehouse & Business Intelligence Project Report.

Collab Notebook Link: **DWH.ipynb**

Assignment Requirements

- Connect to at least two academic data sources (e.g., Semantic Scholar and Google Scholar).
- Write Python functions to fetch research papers using APIs.
- Design a data warehouse schema inside **Supabase** containing tables such as `papers`, `authors`, `paper_authors`, and `ingest_log`.
- Handle errors, API limits, and data quality issues.
- Implement deduplication logic so that the same paper appearing in both sources is stored only once.
- Log each ingestion with topic, counts, and timestamp.
- Submit a well-documented Jupyter Notebook and Report.

Proposed Solution and Architecture

The implemented solution was a modular, step-by-step ETL pipeline built in **Python (Jupyter Notebook)**. It follows the three classic stages of data warehousing:The overall flow is visualized as:

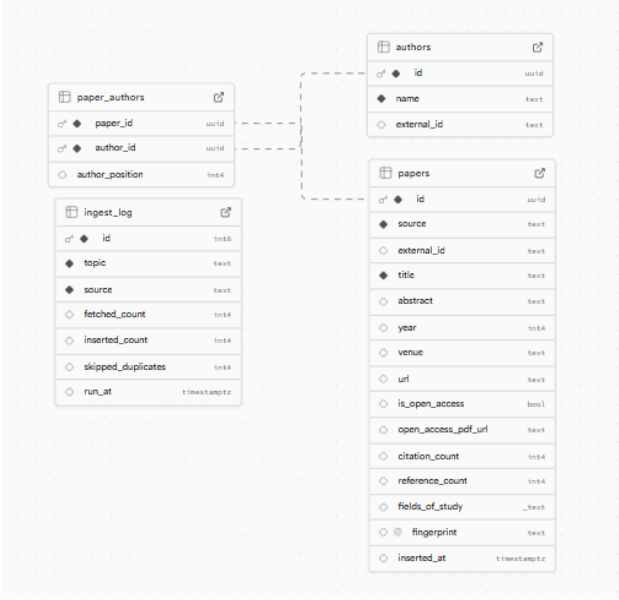
```
Semantic Scholar + Google Scholar
                ↓
        Fetch & Parse JSON
                ↓
  Normalize (title, authors, year)
                ↓
     Generate Fingerprints
                ↓
          Deduplicate
                ↓
         Insert → Supabase
                ↓
          Log Summary
```

Detailed Step-by-Step Implementation

Step 1 — Connecting to Supabase

A Supabase project was created and configured with the URL and Anon Key.
Tables were defined using SQL DDL:

The Python client was tested with a simple `.select("*")` query to confirm connection.



Step 2 — Fetching from Semantic Scholar

A function `fetch_all_from_semantic_scholar(api_key, topic, max_pages, per_page)` used the Semantic Scholar REST API.

Step 3 — Fetching from Google Scholar via SerpApi

Because Google Scholar has no public API, the **SerpApi** REST endpoint was used.
The function `fetch_from_google_scholar_serpapi(topic, api_key, limit)` fetched 10 papers per page until the limit was reached.
It extracted titles, snippets, authors, links, and citation counts.
This ensured compliance with API rate limits and avoided blocking errors (`429 Too Many Requests`).

Step 4 — Data Merging

Instead of inserting data separately (as in earlier versions), both lists were concatenated:

```
all_papers = semantic_papers + google_papers
```

This produced one combined dataset representing all research results for the given topic.

Step 5 — Data Cleaning and Normalization

- Converted all titles to lowercase.
- Removed punctuation and extra spaces.
- Replaced missing values with empty strings.
- Ensured consistent author list format (e.g., `['A. Smith', 'B. Jones']`).

Step 6 — Fingerprint Deduplication

A custom function `make_fingerprint()` was defined:

```
def make_fingerprint(title, first_author, year):
```

By combining normalized title, first author, and year, each record became uniquely identifiable. The helper `deduplicate_papers()` removed any repeated fingerprints, ensuring one unique record per paper across both sources.

Step 7 — Loading into Supabase

The function `populate_all_tables(topic, cleaned_papers, batch_size=50)` performed batch insertions:

1. Inserted into `papers`.
2. Inserted unique authors into `authors`.
3. Created links in `paper_authors`.
4. Logged counts in `ingest_log`.
5. All database operations used Supabase's Python client for transactional safety.