# Conversational Financial Query Resolver (POC)

## Executive Summary

This document outlines a proof of concept for an AI-driven financial analysis system that processes natural language queries about financial data, performs calculations, and visualizes the results. The system leverages multiple specialized AI agents to analyze financial metrics, generate SQL queries, perform calculations, and create data visualizations.

**Key Observations from the Provided Data**

This PoC demonstrates a system designed to process, structure, and analyze financial balance sheet data spanning multiple years. The input format is an annual balance sheet report from April to April, segmented hierarchically into key financial sections.

The goal is to enable **efficient querying**, **automated analysis**, and **dynamic visualization** of financial data by converting it from Excel to a structured database format and integrating it with intelligent agents for end-to-end processing.

## Data Format and Sections

The provided document presents several financial sections organized as follows:

**Revenue Sheet Sections**:

* **Income**: Revenue, rebates, capitalized work, other income
* **Direct Expenses**: COGS, raw materials, freight, labor, utilities
* **Indirect Expenses**: Salaries, admin, legal, travel
* **Profit**: EBIT, EBITDA, Gross Profit, PAT, PBT
* **Depreciation**: Asset amortization
* **Finance Costs**: Interest, leasing interest

**Balance Sheet Sections**:

* **Sources of Fund**: Equity, retained earnings
* **Application of Fund**: Inventory, cash, fixed assets, current assets
* **Free Cash**: Cash flow, CapEx, working capital, taxes
* **Other Financials**: Net debt, ratios, working capital.

### Implementation Approach

Data Ingestion and Preprocessing

**Input Format**: Excel workbook with the sections above.

**Conversion**: Each sheet is converted to CSV format.

**Normalization**: Data is restructured into a columnar format:

Year | Month | Section | Description | Value

**e.g : 2022 | January | Section: Indirect Expenses | Indirect Cost = 2883.02**

### The data is hierarchical, but I have simplified it by breaking it down into sections to make the calculations less complex. However, we can use a relational database and incorporate the hierarchical structure by designing a relevant schema, allowing us to manipulate the data accordingly.

### Approaches

### Initial Approach: Embedding-Based Solution

I initially tried to create embeddings in text chunks, as I have worked with unstructured data in the past. However, in this case, the data is structured, so it was not working efficiently, and the data was not being fetched properly as expected.

Initially, the data was approached using **embeddings** created from text chunks, which was a method I had successfully applied in previous projects involving **unstructured data**. The idea was to use these embeddings to process the data and perform efficient queries. However, given that the data in this case is **highly structured**, this approach proved to be inefficient. The embeddings did not align well with the structured nature of the data, and the query results were inconsistent and inaccurate

**Revised Approach: Relational Database**

### After further research and consideration, I concluded that a relational database would be the most effective solution for handling this structured data. The relational model provides the necessary flexibility and efficiency for querying large datasets, particularly when there is a need to work with multiple sections of the data, each with interdependencies.

**Here’s how the revised approach was implemented:**

1. **Data Conversion**: The original data, stored in an Excel workbook, was converted into **CSV format** to facilitate easier processing and loading.
2. **Database Setup**: The CSV files were uploaded to **Google Drive** and mounted to the **Colab runtime** for real-time access.
3. **Schema Creation**: A schema was dynamically generated based on the column names from the CSV files. This schema ensures that the structure of the data is consistent and supports future data retrieval and manipulation.
4. **Data Ingestion**: The CSV data was then ingested into the **relational database**, ensuring that each entry was properly mapped to its respective columns and section, enabling efficient querying based on user inputs.

**Decisions**

* **Data Format Conversion:** The original Excel data was converted into CSV format to facilitate processing and easy loading into a relational database.
* **Database Choice:** A SQLite relational database was chosen for its lightweight and flexible nature, making it ideal for storing financial data.
* **Schema Design:** A dynamic schema was generated based on the structure of the CSV files, allowing the system to efficiently query the data based on sections and years.
* **Agent Architecture:** A multi-agent approach was implemented with agents like Query Analyzer, CFO Agent, Chart Agent, and Quality Checker, ensuring efficient processing, calculations, and validation.

This relational database setup significantly improved data retrieval accuracy, performance, and scalability. It now provides a foundation for building more complex queries and analyses as part of the overall system architecture.

**Data Processing Flow**

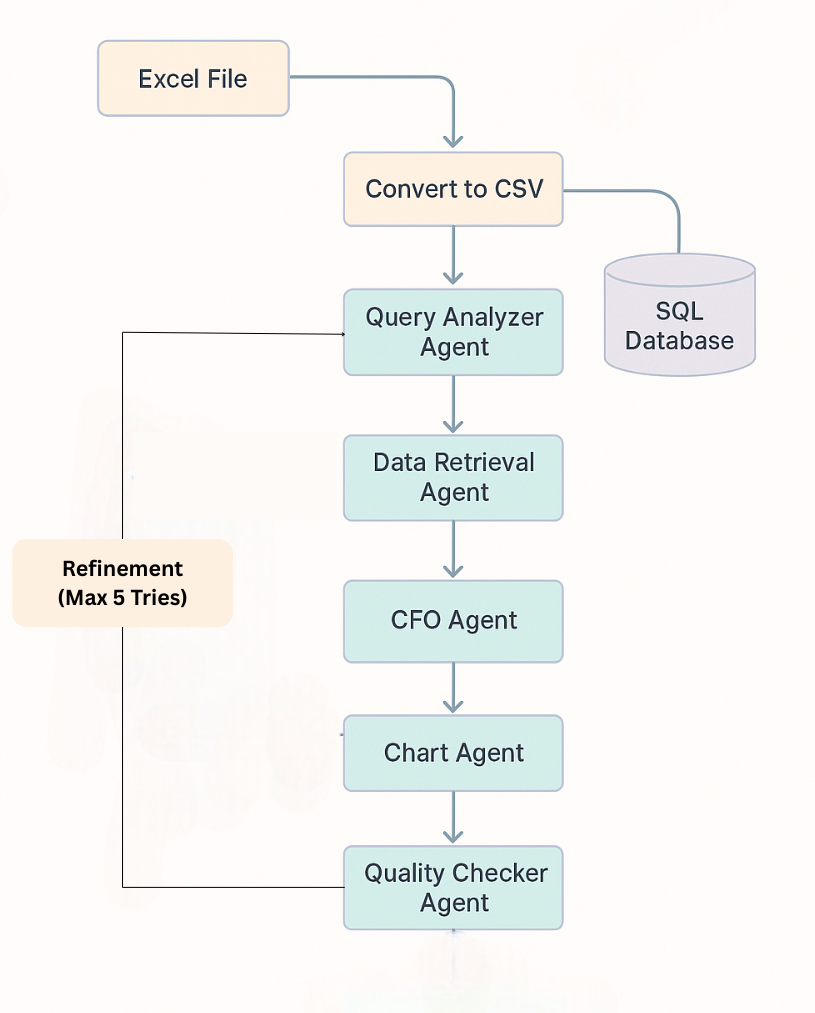
1. **Data Mounting**: The data is mounted to the runtime's temporary memory in Colab. And created the schema based on column names and whole data from csv file updated in the sql DB
2. **User Query**: The user provides a query.
3. **Query Analyzer Agent**: This agent summarizes the user's query, checks which data is present in the schema, identifies which formulas are needed to calculate the answer, and checks whether the required data is present in the schema. It then generates a query for data retrieval, based on the year and section.
4. **Data Retrieval**: The query fetches the data from DB based on the .query generated
5. **CFO Agent**: The CFO agent takes the retrieved data and applies the necessary formulas to calculate insights and analysis based on the user's query .
6. **Chart Agent**: Based on the CFO agent's response, the chart agent will decide which chart type is most appropriate for better explanation (such as pie charts, line charts, bar graphs, or tables).
7. **Quality Checker Agent**: This agent checks whether the response from the agents aligns with the user's query. If not, the process starts over with a refined prompt. This can be done up to three times for better accuracy.

## Data Source

The system connects to a SQLite database that contains financial data with the following structure:

* row\_id: Unique identifier for each record
* year: The fiscal year (2022-2025)
* month: The month (1-12)
* section: The financial statement section (e.g., Income, Direct Expenses)
* description: Specific description of the item (e.g., Revenue, COGS)
* value: The monetary value
* source\_file: The source document

**Flow Chart**



## 

## System Workflow

1. **User Query Entry**: The user enters a natural language question about financial data
2. **Query Analysis**: The Query Analyzer interprets the question and generates an SQL query
3. **Data Retrieval**: The system fetches relevant financial data from the database
4. **Financial Calculations**: The CFO Analyst performs calculations and analysis
5. **Visualization Generation**: The Chart Strategist creates specifications for multiple charts
6. **Quality Verification**: The Quality Analyst ensures all steps were completed successfully
7. **Refinement Loop**: If any issues are detected, the system repeats with feedback
8. **Chart Rendering**: Visualizations are generated using matplotlib

## Visualizations

The system generates multiple chart types depending on the query:

* **Line charts**: For trends over time
* **Bar charts**: For comparing discrete values
* **Pie charts**: For composition analysis
* **Stacked bar charts**: For comparing composition across categories

Each chart includes proper titling, axis labels, and annotations for clear data presentation.

## Technical Components

1. **Dependencies**:
   * pandas: Data manipulation
   * sqlite3: Database connectivity
   * crewai: Multi-agent orchestration
   * matplotlib/numpy: Data visualization
   * re/json: Processing output data

## Limitations and Future Enhancements

**Current Limitations**:

* Limited to predefined financial sections and metrics
* Visualizations restricted to basic chart types
* In-memory database may not scale for large datasets
* **Hierarchical Data:** The hierarchical structure of financial data is simplified for the PoC, preventing complexity in its analysis

**Future Enhancements**:

* Support for more complex financial queries and metrics
* Integration with external financial data sources
* Interactive dashboards with drill-down capabilities
* Scheduling and automated financial reporting
* Natural language explanation of results