DWBI Problem Statement: SQL Generation with LLM

Problem Statement

You are part of the Data Warehouse & Business Intelligence (DWBI) team at a retail company. The business team frequently asks ad-hoc questions across two data sources:

1. Sales Data Warehouse (sales_dw)
2. Marketing Data Warehouse (marketing_dw)

Both sources are in Table.

The challenge is to use a Large Language Model (LLM) to generate SQL queries automatically. You will provide the schema information of both data sources, feed 20 business questions, and capture the SQL queries generated. The output should be stored in a CSV with columns:

- question_id
- question (as given)
- target_source (sales_dw or marketing_dw)
- sql (the query generated)
- assumptions (if any made by the LLM)
- confidence (0.0–1.0 scale)

Data Source 1: Sales Data Warehouse (sales_dw)

Table: sales

Column	Type	Description
sale_id	INT	Unique identifier for each sale
product_id	INT	Foreign key → products.product_id
region	VARCHAR	Sales region
sale_date	DATE	Date of transaction
sales_amount	DECIMAL	Revenue from transaction

quantity	INT	Number of units sold

Table: products

Column	Type	Description
product_id	INT	Unique product ID
product_name	VARCHAR	Name of the product
category	VARCHAR	Product category
subcategory	VARCHAR	Subcategory
brand	VARCHAR	Product brand

Data Source 2: Marketing Data Warehouse (marketing_dw)

Table: campaigns

Column	Type	Description
campaign_id	INT	Unique campaign ID
channel	VARCHAR	Marketing channel
start_date	DATE	Campaign start date
end_date	DATE	Campaign end date
budget	DECIMAL	Campaign budget

Table: impressions

Column	Type	Description
campaign_id	INT	Foreign key → campaigns.campaign_id
day	DATE	Date of impressions
impressions	INT	Number of impressions shown
clicks	INT	Number of clicks received

Business Questions

1. What are the top 5 products by sales amount in the last 90 days?

2. Show the month-over-month sales growth by region for the past 6 months.

3. Which categories contributed the most to total revenue in the last year?

4. Find the average order value (AOV) per region in the current quarter.

5. Identify the top 3 brands with highest quantity sold in the last 30 days.

6. Which subcategory had the sharpest decline in sales compared to the previous quarter?

7. What is the percentage contribution of each region to total sales this year?

8. Show the trend of sales_amount vs quantity sold for Electronics products.

9. Find the product with the highest sales per unit (sales_amount ÷ quantity) in the last 60 days.

10. List the top 10 customers by revenue (if customer table exists).

11. Which channel had the highest total impressions in the last quarter?

12. Calculate the average click-through rate (CTR) per channel last month.

13. Which campaign delivered the lowest cost per click (CPC) in the last 6 months?

14. Find the total budget spent per channel in the last year.

15. Identify the top 3 campaigns by impressions during their active periods.

16. What is the daily average impressions vs clicks trend for Social Media campaigns?

17. Which channel shows the highest conversion ratio (clicks ÷ impressions) overall?

18. List campaigns that ran for more than 60 days and their total spend.

19. Compare campaign budgets vs actual clicks to highlight underperforming campaigns.

20. Find the month with the highest total impressions across all campaigns.

Here is a detailed solution that can be easily understood by anyone, from the project's problem statement to its core components and final output.

💡 **Solution & Methodology**

The solution is an **SQL SCRIBE - AUTONOMOUS SQL GENERATION TOOL**. It is built on a SQLGenerationPipeline class in Python that orchestrates the entire workflow from start to finish. This pipeline uses a Large Language Model (LLM) as its core engine, but the true value lies in the structured prompting and validation logic that guides the LLM to produce reliable and transparent results.

The process is broken down into the following stages:

- **1. Data & Schema Ingestion:** The pipeline first loads the schema definitions for the Sales and Marketing data warehouses from JSON files (sales_dw.json and marketing_dw.json). It also loads the list of 20 business questions from a CSV file (questions.csv).
- **2. LLM Prompting:** For each business question, the system constructs a highly-detailed prompt that includes:
    - The complete schemas of both data warehouses.
    - A clear set of instructions for the LLM to think step-by-step, validate the existence of tables and columns in a single source, and use only documented relationships for joins.
    - A requirement for the LLM to provide its reasoning (assumptions) and a self-rated confidence score from 0.0 to 1.0.
- **3. Automated Validation & Post-Processing:** The LLM's raw output is not used directly. The pipeline includes logic to post-process the generated SQL, correcting for common syntax variations (e.g., converting TOP to LIMIT) to ensure ANSI compliance.
- **4. Structured Output & Reporting:** The final result for each question is a structured JSON object. This data is then used to generate a comprehensive CSV file with columns for question_id, question, target_source, sql, assumptions, and confidence. The system also generates an optional Markdown report summarizing key performance metrics and providing examples of the LLM's reasoning.

📊 Project Output & Results

The output of the pipeline is a complete set of SQL queries, along with a detailed record of the LLM's performance.

- **For successful queries:** The output includes the generated SQL query, the target data source (e.g., sales_dw), and a high confidence score.

- **For impossible queries (like Question #10):** The AI correctly identifies that the required customer table does not exist in either schema. It provides a detailed assumption for why it cannot generate the query and assigns a low confidence score, demonstrating its ability to be transparent and avoid "hallucinating" an answer.

This project serves as a practical example of using LLMs in a production-like environment, where robustness, validation, and explainability are just as important as the model's core generation capabilities.

📝 What the Script Does (app.py)

The app.py script is the core of the project. It handles the entire end-to-end pipeline:

1. **Data Loading:** It automatically reads the data warehouse schemas (sales_dw.json, marketing_dw.json) and the list of questions (questions.csv).
2. **LLM Interaction:** For each question, it sends a carefully crafted prompt to the LLM. This prompt includes the schema information and clear instructions on how to generate and validate the SQL.
3. **Validation:** The script specifically tells the LLM to only use tables and columns that exist within a **single** provided schema. If a table doesn't exist, the LLM is instructed to explain why it cannot generate the query and return a low confidence score.
4. **Output & Reporting:** After processing all questions, the script compiles the results (including the generated SQL, the LLM's reasoning, and a confidence score) into a structured CSV file and a summary report.

📋 Requirements & Setup

To run this project, you need to set up the following:

- **Python:** Install Python 3.11 or a later version on your system.
- **API Key:** Obtain an API key from Groq to access their LLM services.[1]
- **File Structure:** Organize your files into the following directory structure:
- your_project_folder/
- ├── app.py
- ├── requirements.txt
- └── data/

- ├── sales_dw.json
- ├── marketing_dw.json
- └── questions.csv
- Libraries: The project requires a few Python libraries, which can be installed from the requirements.txt file using a single command:

```
pip install -r requirements.txt
```

🧠 The Role of the LLM & Groq

- **LLM (Large Language Model):** An LLM is an AI model that understands and generates human language.[2] In this project, the LLM is the "brain." Its job is to comprehend the natural language question, understand the provided database schemas, and generate the correct SQL syntax.[3]
- **Groq:** Groq is the **inference engine** that provides the LLM.[4] Instead of being a traditional cloud provider, Groq uses specialized hardware called Language Processing Units (LPUs) to run LLMs at incredibly high speeds.[5] This makes the query generation process near-instantaneous and highly efficient, which is crucial for real-world applications.

✅ How it Solves the Problem

This project is not just a demo; it's a practical solution to a real-world business problem.

- **Democratizes Data Access:** It allows non-technical users to get the data they need by simply asking a question in plain English, eliminating the need to learn complex SQL.
- **Ensures Accuracy:** By incorporating a validation and reasoning step directly into the LLM's prompt, the system ensures that the generated SQL is both syntactically correct and semantically valid against the provided schemas.
- **Handles Edge Cases:** The project is specifically designed to handle challenging scenarios, such as when a question references a table that doesn't exist. Instead of failing or "hallucinating" an answer, it transparently explains why the query cannot be generated, which builds user trust.
- **Provides Transparency:** The confidence score and detailed reasoning from the LLM give users a clear understanding of why a query was generated (or not), allowing them to review and verify the results before use.