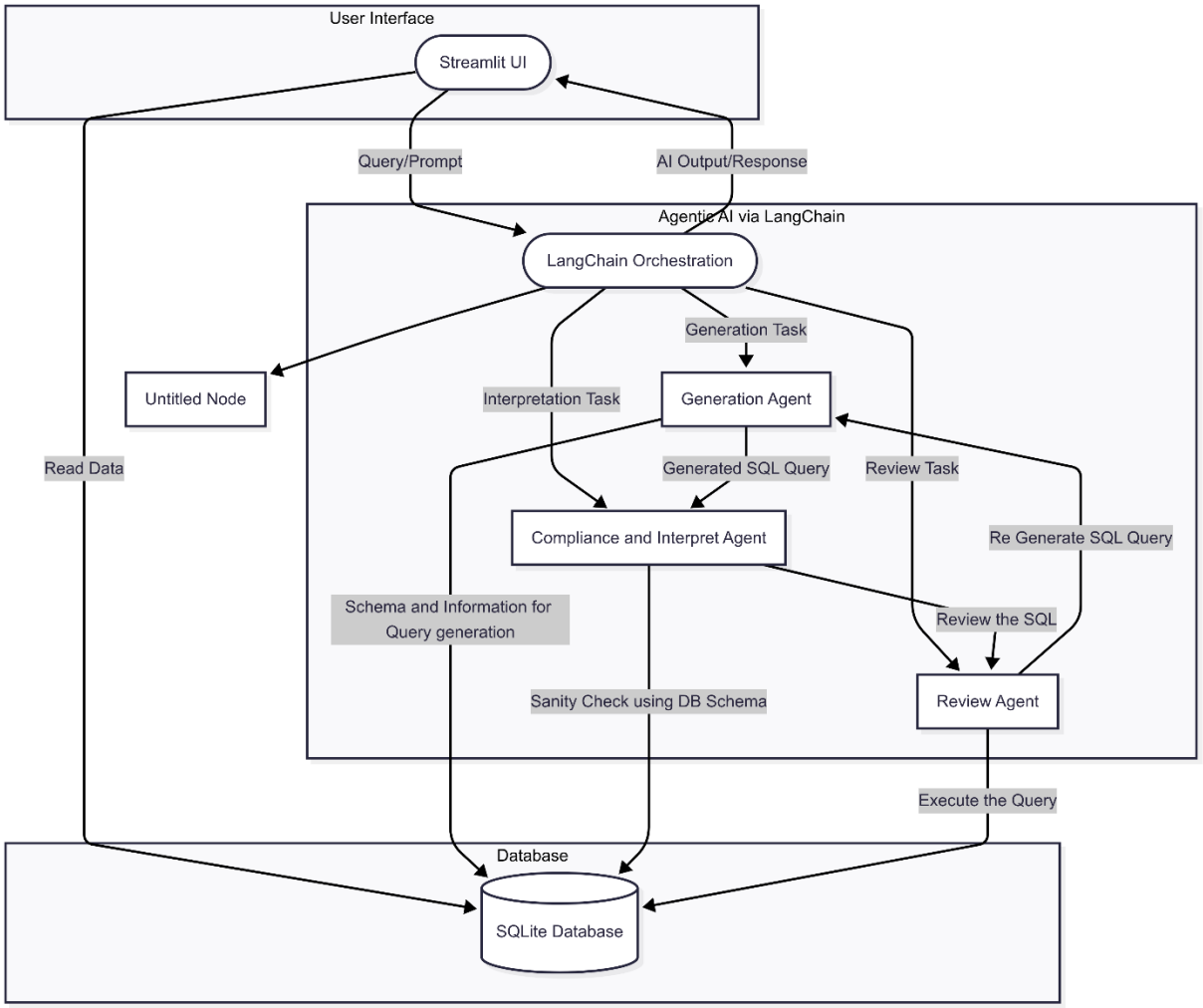
DocNexus AI Submission - AI Engineer Role

Name – Anshul Ranjan **Email - anshulpranjan@gmail.com**

Component	Tool/Library
Frontend UI	Streamlit – Can be switched to Next JS or Angular
Backend Database	SQLite (lightweight + introspectable) ---- Can be any DB based on requirements
LLM Orchestration	LangChain agents + Python (Can be changed to Crew AI (Not free))
Language Model	OpenAI GPT-4 / Gemini 2.5 Flash (Can be any model)
Query Safety	Custom compliance agent

GitHub Link - https://github.com/AnshulRanjan2004/SQLRx-DocNexusAI

I implemented a Retrieval-Augmented Generation (RAG) architecture to dynamically ingest schema information either from structured text files or directly via database introspection to enhance the accuracy and context-awareness of SQL generation.



Discarded Approaches

Attempt 1: Next.js + Vanna AI + Streamlit (https://vanna.ai/)

Initially explored using Vanna AI with both a Next.js frontend and a Streamlit interface. However, the model produced unreliable results with frequent hallucinations and offered limited control over query generation. Due to time constraints and integration challenges with Next.js, this approach was ultimately discontinued.

Reference: See the vanna-ai/ folder for this iteration.

Attempt 2: CrewAI + Streamlit

Tested CrewAI with a modular agent setup inside Streamlit. While conceptually strong, the free version lacked performance, and the paid version was cost-prohibitive for this prototype. CrewAI could be a better fit for production-scale systems, but LangChain offered a more practical path at this stage.

Reference: See the config/ folder for this experiment.

My Approach:

I chose LangChain for building the agentic workflow due to its flexibility, modularity, and scalability while being entirely free to use. For the language model, I integrated Gemini Flash during development for cost efficiency, with the architecture designed to be model-agnostic. Replacing it with a more powerful commercial model (e.g., GPT-4, Claude, or Gemini Pro) can significantly enhance performance.

To demonstrate direct agent interaction with the database, I used SQLite, ideal for lightweight local development. The system supports RAG (Retrieval-Augmented Generation) to dynamically retrieve schema information from either text files or direct introspection. The setup is database-agnostic—supporting both SQL and NoSQL backends. With proper metadata indexing, this approach can scale efficiently to large datasets while maintaining fast query performance.

The system can also support auto-execution of generated SQL queries, capture both the results or error messages, and feeding them back to the agent for iterative refinement. This feedback loop can be repeated for 2–3 iterations to automatically correct errors and improve query accuracy and relevance.

📈 Key Features Implemented

✅ Query Logging & Accuracy Tracking: Metrics are logged and visualized on the main dashboard, including full query history.

✅ UI-Based Execution: Queries can be executed directly from the frontend, making the tool interactive and intuitive.

✅ Result Interpretation: The UI provides a detailed analysis of the query to ensure results are understandable and actionable for commercial teams.

🔁 Regeneration Support: Users can regenerate queries if the initial result isn't satisfactory.

📝 Notes & Assumptions

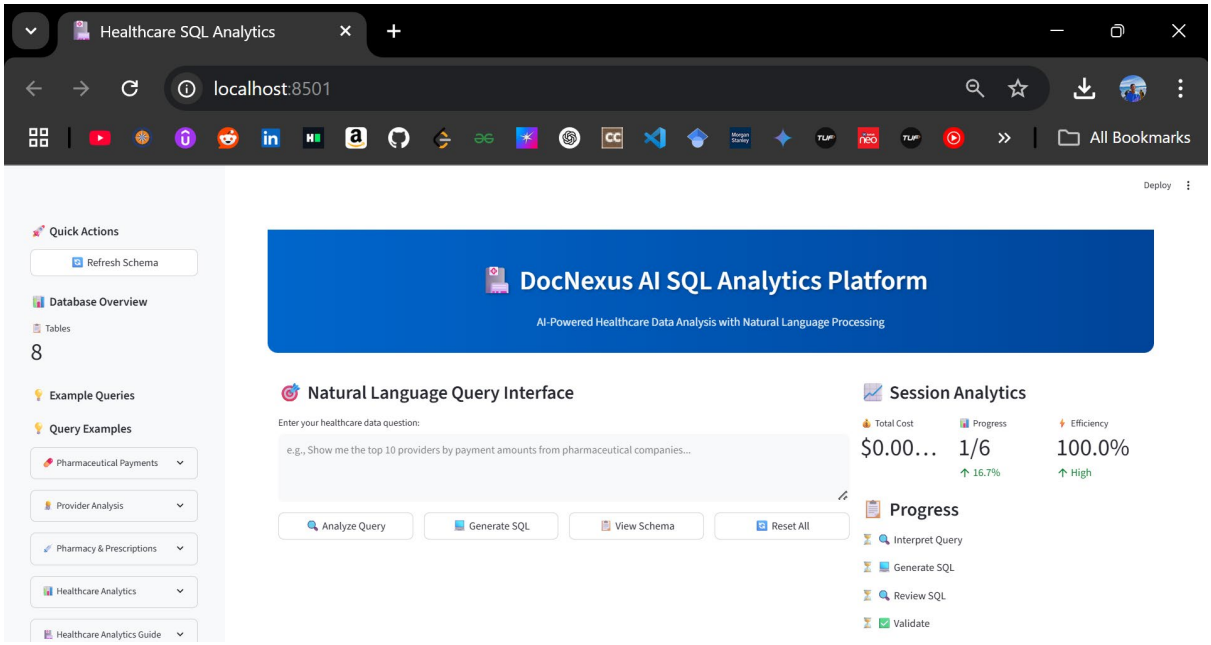
Domain context (e.g., ICD-10, CPT, HCP, ASC) is embedded into all agent prompts for accurate interpretation.

Schema notes are interpreted dynamically at runtime, enhancing adaptability.

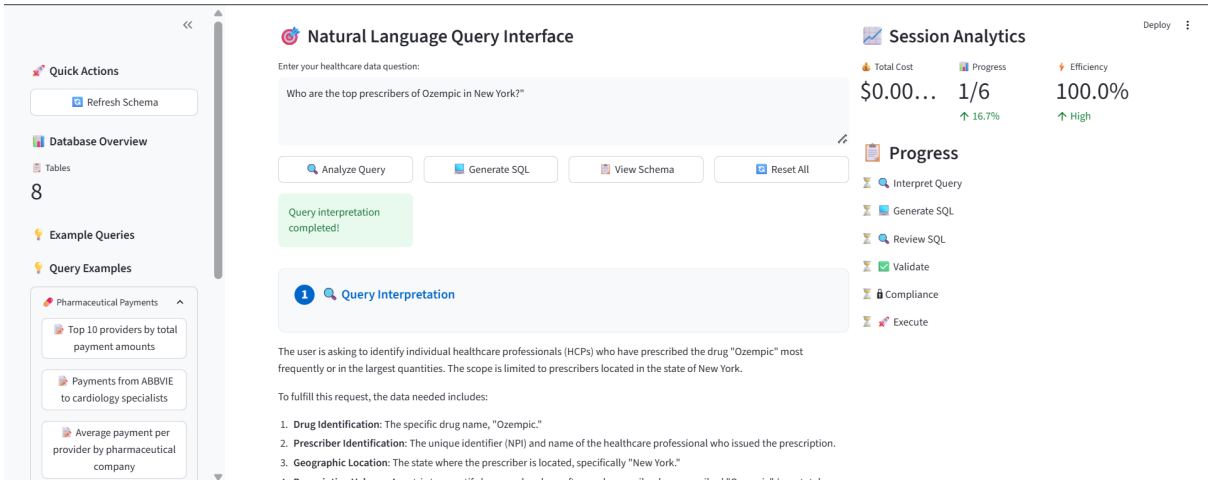
Optimized for local prototyping using SQLite but can be extended to cloud-scale solutions like Snowflake or PostgreSQL.

The compliance agent uses simplified rules to detect potential PHI/PII violations for prototyping purposes.

Output Screenshot



Home Screen



Analysing the User's Input

Output Generation

Deploy

Natural Language Query Interface

Enter your healthcare data question:

Who are the top prescribers of Ozempic in New York?"

Analyze Query | Generate SQL | View Schema | Reset All

SQL generated successfully!

1 Query Interpretation

The user is asking to identify individual healthcare professionals (HCPs) who have prescribed the drug "Ozempic" most frequently or in the largest quantities. The scope is limited to prescribers located in the state of New York.

To fulfill this request, the data needed includes:

1. **Drug Identification**: The specific drug name, "Ozempic."
2. **Prescriber Identification**: The unique identifier (NPI) and name of the healthcare professional who issued the prescription.

Session Analytics

Total Cost	Progress	Efficiency
$0.001344	2/6	86.6%
↑ $0.001344	↑ 33.3%	↑ High

Progress

- Interpret Query
- Generate SQL
- Review SQL
- Validate
- Compliance
- Execute

1. **Drug Identification**: The specific drug name, "Ozempic."
2. **Prescriber Identification**: The unique identifier (NPI) and name of the healthcare professional who issued the prescription.
3. **Geographic Location**: The state where the prescriber is located, specifically "New York."
4. **Prescription Volume**: A metric to quantify how much or how often each prescriber has prescribed "Ozempic" (e.g., total dispensed quantity or total number of claims/prescriptions).

Relevant Tables and Columns:

- `fct_pharmacy_clear_claim_allstatus_cluster_brand` (**Pharmacy Claims**): This table is the primary source for this query.
  - `NDC_PREFERRED_BRAND_NM`: This column will be used to filter for "Ozempic" prescriptions.
  - `PRESCRIBER_NPI_NBR`: This column provides the unique National Provider Identifier for each prescriber.
  - `PRESCRIBER_NPI_NM`: This column provides the name of the prescriber.
  - `PRESCRIBER_NPI_STATE_CD`: This column will be used to filter prescribers located in "New York" (assuming 'NY' is the state code).
  - `DISPENSED_QUANTITY_VAL`: This column represents the quantity of the drug dispensed, which can be summed to determine the "top" prescribers by volume. Alternatively, a count of `RX_CLAIM_NBR` could be used to determine the top prescribers by the number of prescriptions.

The user is looking for a ranked list of prescribers (their names and NPIs) based on their prescription activity for Ozempic within New York.

2 Generated SQL Query

```sqlite
SELECT
  PRESCRIBER_NPI_NM,
  SUM(DISPENSED_QUANTITY_VAL) AS TotalDispensedQuantity
FROM fct_pharmacy_clear_claim_allstatus_cluster_brand
WHERE
  NDC_PREFERRED_BRAND_NM = 'OZEMPIC' AND PRESCRIBER_NPI_STATE_CD = 'NY'
GROUP BY
  PRESCRIBER_NPI_NM
ORDER BY
  TotalDispensedQuantity DESC
LIMIT 10;
```

Review & Validate | Regenerate | Clear Results

3 Reviewed & Optimized SQL

```sqlite
SELECT
  PRESCRIBER_NPI_NM,
  SUM(DISPENSED_QUANTITY_VAL) AS TotalDispensedQuantity
FROM fct_pharmacy_clear_claim_allstatus_cluster_brand
WHERE
  NDC_PREFERRED_BRAND_NM = 'OZEMPIC' AND PRESCRIBER_NPI_STATE_CD = 'NY'
GROUP BY
  PRESCRIBER_NPI_NM
ORDER BY
  TotalDispensedQuantity DESC
LIMIT 10;
```

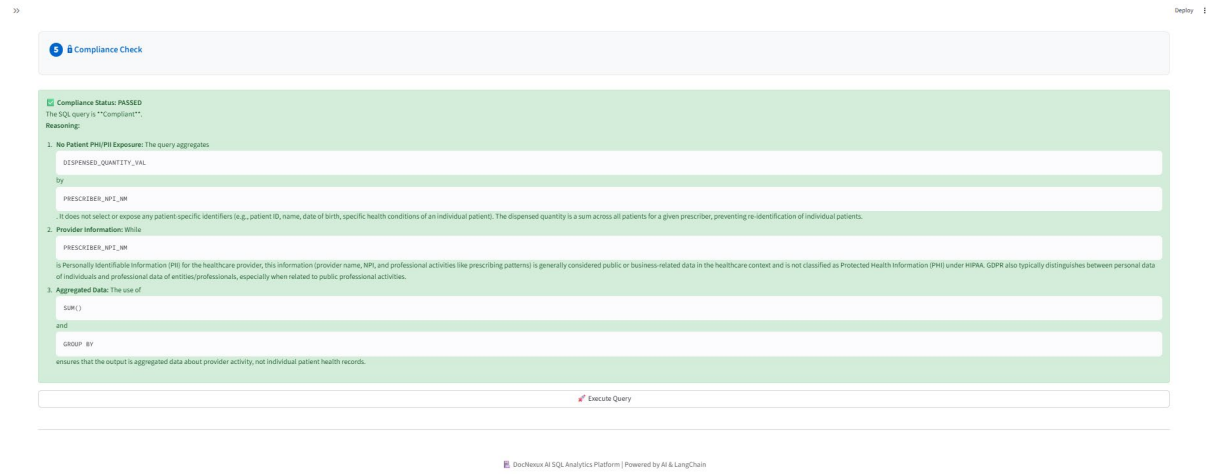
4 Validation Report

Validation Report: SQL Query for Healthcare Database

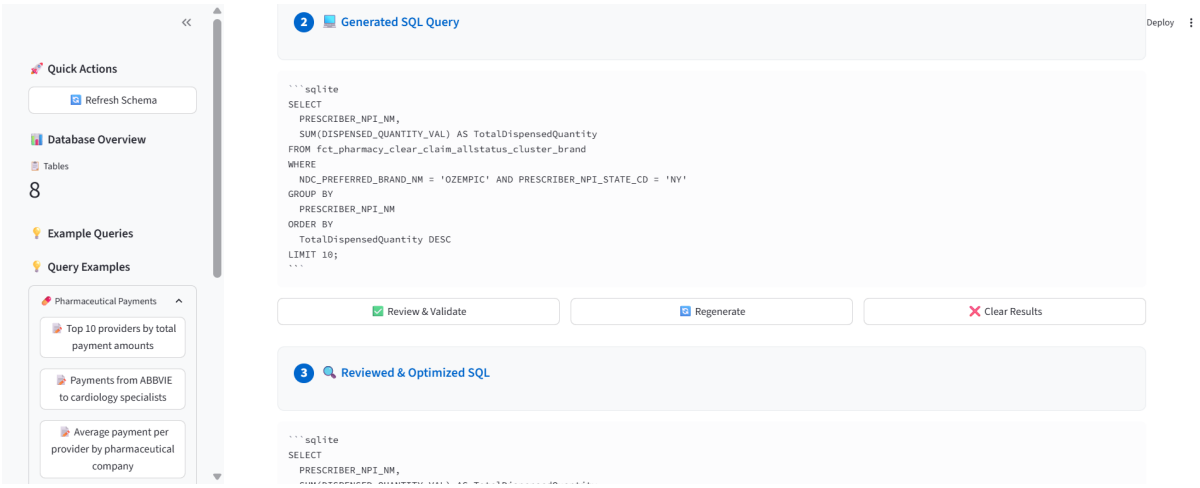
Query Validity: Valid

Detailed Analysis:

1. **Correct Table and Column Names:**
   - **Table:** `fct_pharmacy_clear_claim_allstatus_cluster_brand` is correctly identified from the schema as "Table 5: Pharmacy Claims file".
   - **Columns:**
     - `PRESCRIBER_NPI_NM`: Exists in `fct_pharmacy_clear_claim_allstatus_cluster_brand` (String type). Correctly used in `SELECT` and `GROUP BY`.
     - `DISPENSED_QUANTITY_VAL`: Exists in `fct_pharmacy_clear_claim_allstatus_cluster_brand` (Decimal(38, 9) type). Correctly used with `SUM()` aggregation.
     - `NDC_PREFERRED_BRAND_NM`: Exists in `fct_pharmacy_clear_claim_allstatus_cluster_brand` (String type). Correctly used in `WHERE` clause for string comparison.
     - `PRESCRIBER_NPI_STATE_CD`: Exists in `fct_pharmacy_clear_claim_allstatus_cluster_brand` (String type). Correctly used in `WHERE` clause for string comparison.
   - All table and column names are accurate and match the provided schema.
2. **Proper Joins and Relationships:**
   - The query operates on a single table (`fct_pharmacy_clear_claim_allstatus_cluster_brand`) and does not involve any joins. Therefore, this check is not applicable.
3. **Healthcare-Specific Data Handling:**

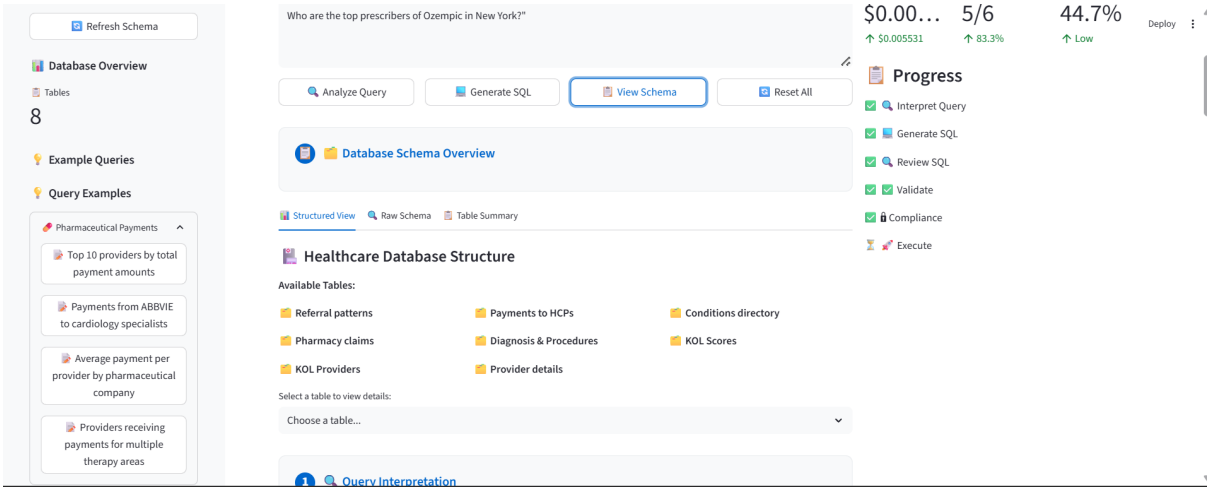


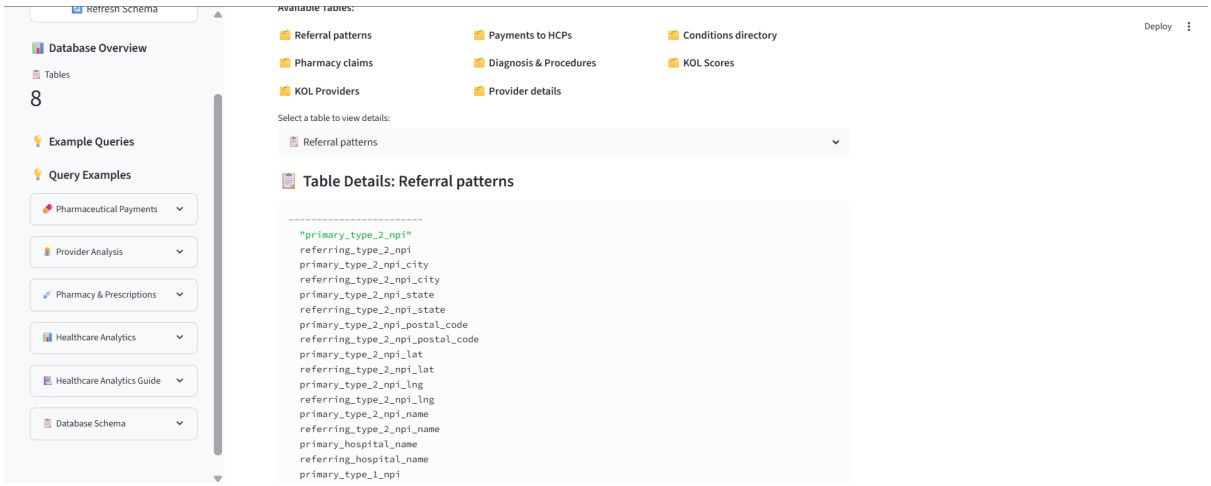
Compliance Checks and Execute the Generated Query



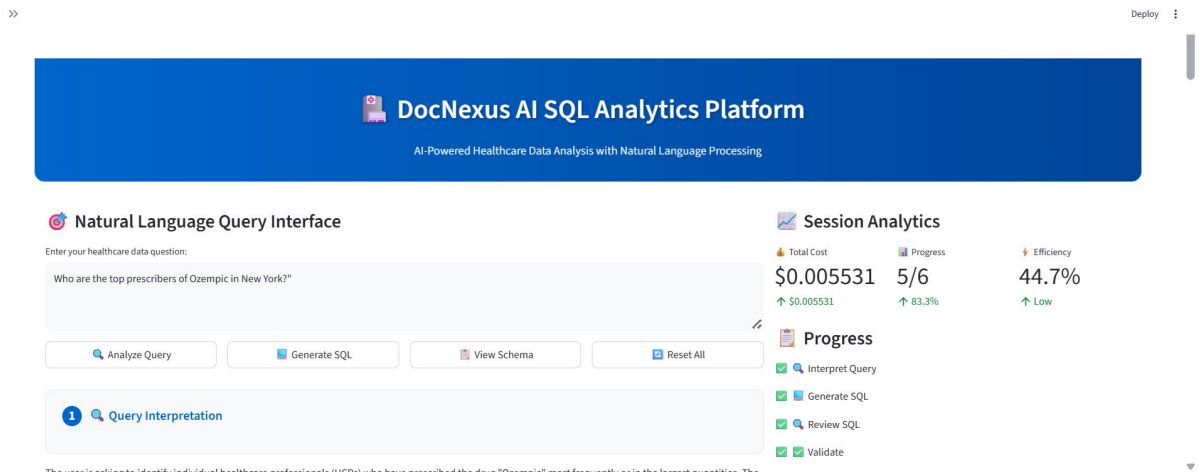
Regenerate is not satisfied

Ability to view the schema from the database directly





Query Logging and Accuracy Logging



More Validation for best accuracy