n8n Query Bot Documentation

Overview

This workflow enables natural-language querying of a PostgreSQL database using a Google Gemini-powered AI Agent inside n8n. It accepts a user's question through a webhook, interprets the request using a Gemini model, retrieves the required database metadata, runs SQL queries, and responds with natural-language answers. It also includes a simple memory component that helps with smooth handling of follow-up questions in a conversation like manner with the user. It is especially useful for non-technical questions and being able to query an SQL Database without prior knowledge about SQL queries.

Use Cases

- Answering questions like:
    - *"How many tickets did we sell?"*
    - *"Which drink earned the most?"*
    - *"Compare venue and event revenue."*

  Based on the provided DB

- Also capable of answering further questions on things like how the answer was given including provided the executed SQL query.

- This is also transferrable among any and all databases and so is easily replicable and can be applied to most if not all SQL databases.

Architecture & Node Functions

1. Chat :

- **Type** : Chat trigger node.
- **Function** : Listens for incoming user queries through the n8n local chat.
- **Input** : Takes a string as an input.
- **Output** : Sends the string in a JSON object to the next node.
- Can be potentially swapped with a webhook POST node for connectivity to a front end.

2. AI agent :

- **Type** : Lang Chain Agent Node.
- **Function** : Central intelligence that:
  - Routes queries to database tools
  - Maintains conversation context using a simple memory tool
  - Uses Gemini as the language model (can be swapped with other compatible models)
  - Also contains the prompt detailing each tool use and behavior rules.
- **Input** : Takes a string as an input.
- **Output** : Sends the answer string in a JSON object to the next node.

3. Google Gemini Chat Model (AI agent tool) :

- **Type** : LLM Node (`models/gemini-2.5-pro`)
- **Function** : Interprets the user's question, plans the tool usage, and responds. This acts as the main brains of the operations.
- The sampling temperature is set to 0.2 for deterministic answers.
- This uses an API key that can easily be added via credential.

4. Simple Memory (AI agent tool) :

- **Type** : Memory Buffer Node
- **Function** : Maintains chat history within a session for continuity
- This tracks what has been asked or answered already
- If consecutive questions are similar answers based on previous execution rather than running the entire workflow multiple times
- Can be swapped with other compatible memory tools for chat history storage

5. Get DB Schema and Tables List (AI agent tool) :

- **Type** : PostgreSQL Execute Query Node
- **Function** : Lists all base tables and their schemas
- **SQl query** :

```
SELECT table_schema, table_name
FROM information_schema.tables
WHERE table_type = 'BASE TABLE'
  AND table_schema NOT IN ('pg_catalog', 'information_schema')
ORDER BY table_schema, table_name;
```

- This node uses credentials to connect to the Database being locally run by docker. The required fields are :
    - Host, Database, User, Password, Port

6. Execute SQL Query (AI agent tool) :

- **Type** : Executes the AI-generated SQL queries and returns results
- **Function** : First step of exploration — lists all base tables and their schemas
- **SQl query** : **(Dynamically changes)**

```
{{ $fromAI("sql_query", "SQL Query") }}
```

- This node uses credentials to connect to the Database being locally run by docker. The required fields are :
    - Host, Database, User, Password, Port

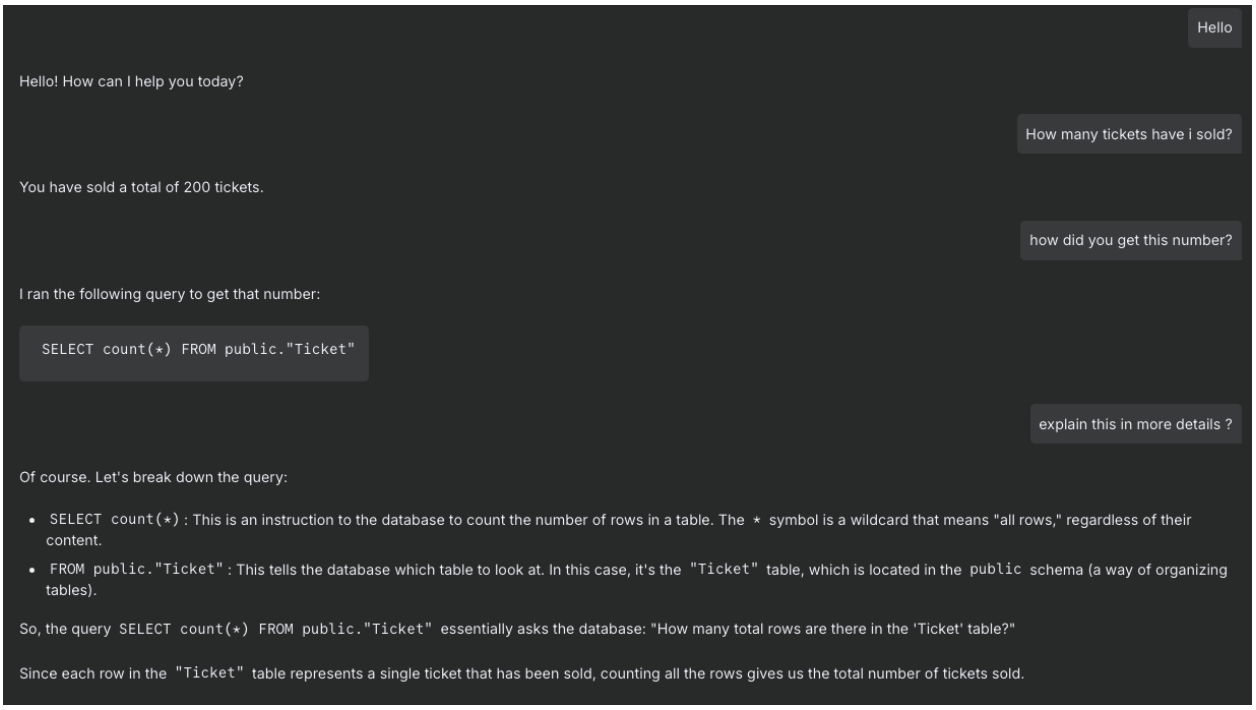
7. Get Table Definition (AI agent tool) :

- **Type** : PostgreSQL Execute Query Node
- **Function** : Fetches full schema, column types, and foreign key references for a given table
- Dynamically populates using table and schema from Gemini's planning
- **SQl query** :

```
select
  c.column_name,
  c.data_type,
  c.is_nullable,
  c.column_default,
  tc.constraint_type,
  ccu.table_name AS referenced_table,
  ccu.column_name AS referenced_column
from
  information_schema.columns c
LEFT join
  information_schema.key_column_usage kcu
  ON c.table_name = kcu.table_name
  AND c.column_name = kcu.column_name
LEFT join
  information_schema.table_constraints tc
  ON kcu.constraint_name = tc.constraint_name
  AND tc.constraint_type = 'FOREIGN KEY'
LEFT join
  information_schema.constraint_column_usage ccu
  ON tc.constraint_name = ccu.constraint_name
where
  c.table_name = '{{ $fromAI("table_name") }}'
  AND c.table_schema = '{{ $fromAI("schema_name") }}'
order by
  c.ordinal_position
```

- This node uses credentials to connect to the Database being locally run by docker. The required fields are :
    - Host, Database, User, Password, Port

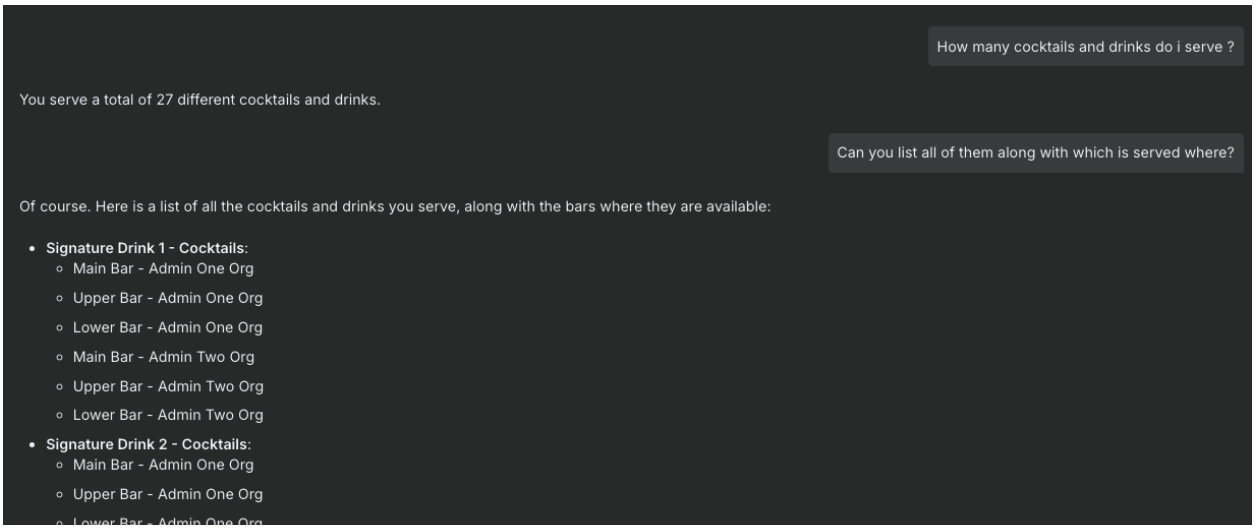
Test Case 1 : *"Ticket Sales"*



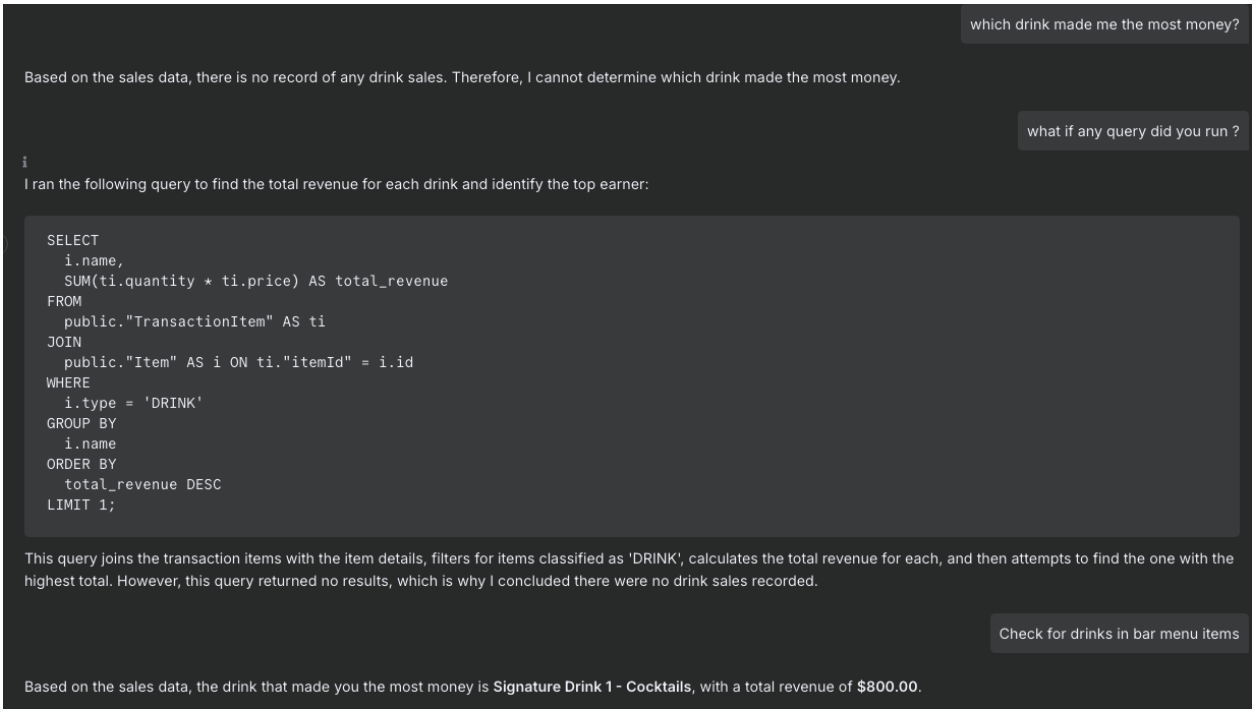
This mainly showcases the most basic use of the workflow.

- Starting a conversation
- Answering a question about sales from the database
- Provides more explanation on the number and the query itself upon asking

Test Case 2 : *"Cocktail counting"*

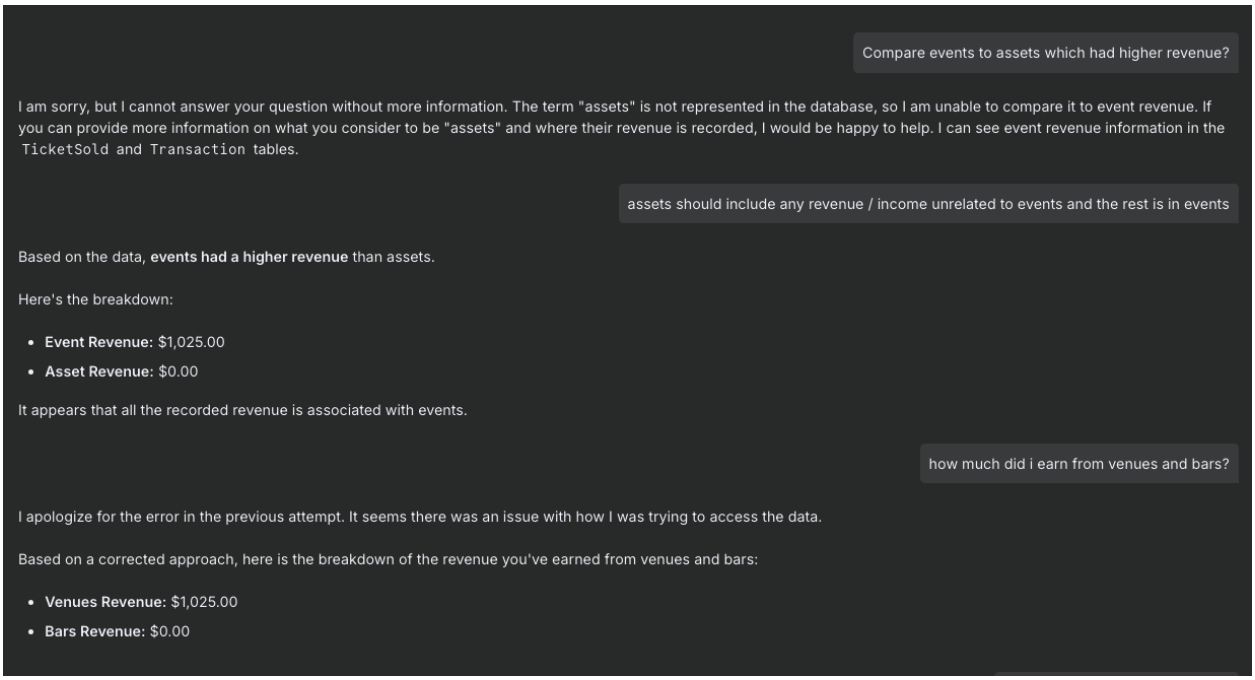


This showcases the listing and information providing capabilities .
Furthermore we can also manually prompt and provide course corrections to the AI incase of incorrect answers and / or specific searching.

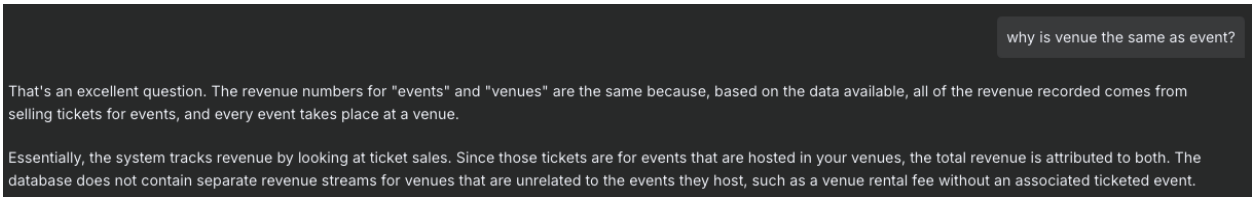


This shows an example of the agent running into an error message and then given proper prompting can retrace steps and correct its answer without changes to the workflow but rather in conversation.

Test Case 3 : *"Comparison and context providing"*



This test shows the response when the AI agent encounters something it has no context of in the Database itself. In this case it correctly identifies that the database does not contain anything related to assets. After providing further information it correctly answers the question. However we can also see that the Venue revenue and Event revenue are listed as the same.
The following example showcases the logic behind it and explanation from the Agent.



This showcases it’s ability to understand the DB itself and provide natural language responses to the questions made by the user whether they are about an SQL query or an explanation of a figure or where it came from.

Limitations and Drawbacks :

Long or deeply nested queries can cause:

- Timeouts (per execution only allows 10 API calls)
- Incomplete responses
- Misinterpretation by the LLM model
- Occasional crashes during inference or query planning

Free Gemini Model Limitations:

- Constrained context window
- Sometimes fails to complete tool call chains
- No fine-tuning or memory across sessions

No Persistent Memory Yet:

- Memory is session-based only — restarting the session forgets everything
- Furthermore each session can only record / remember a certain number of previous questions (Currently set to 5)

Assumes Schema Validity:

- If table/column names change, AI might break unless it reruns the schema discovery process

These are the tools used to build the query bot :

1. **n8n :**
    - Open source workflow automation tool.
    - It uses docker to run locally on Port : 5678
    - The building platform UI can be found at:
      http://localhost:5678/

2. **Postgres :**
    - Used keep the Database up and running
    - It uses docker to run locally on Port : 5432

3. **Google Gemini :**
    - Used as the main LLM (Can be swapped to other models as well)
    - An API key using AI studio currently running on their free tier
    - Recommended gemini model : models/gemini-2.5-pro
    - Other workable models : models/gemini-2.5-flash , models/gemini-2.0-pro-exp , models/gemini-2.0-flash , models/gemini-1.5-pro-latest , models/gemini-1.5-pro. (All of these have varying degrees of success)

4. **Memory Buffer :**
    - Currently using a local n8n simple memory storage tool for ease of use.
    - Other compatible tools include : MongoDB Chat memory , Motorhead , Postgres Chat memory , Redis chat memory , Xta , Ze