🤖 Snowflake QA Analytics Bot

This project demonstrates how to build and deploy a **Question–Answer (QA) analytics bot** using **Snowflake Cortex**, **Snowpark**, and **Streamlit**.

Users can ask questions in **natural language**, and the system answers them by running **SQL queries on Snowflake data**.

The application is:

- Publicly accessible
- Secure
- Read-only
- Does not expose Snowflake credentials

📌 What I Built

I built a **web-based QA bot** that:

- Accepts questions in English
- Uses **Snowflake Cortex** to convert questions into SQL
- Executes SQL queries on Snowflake data
- Displays results in a Streamlit UI
- Is deployed publicly using **Render**

🏗️ Architecture Overview

```
User
 ↓
Streamlit UI (Browser)
 ↓
Python App (Render)
 ↓
Snowflake Cortex (NLP → SQL)
 ↓
Execution SQL View
 ↓
Snowflake Tables
 ↓
Answer shown to user
```

🗄️ Snowflake Data Used

Database

```
DEMO_DB
```

Schema

```
PUBLIC
```

Tables

CUSTOMERS

- CUSTOMER_ID
- CUSTOMER_NAME
- CITY

SALES

- ORDER_ID
- CUSTOMER_ID
- AMOUNT
- ORDER_DATE

The relationship between tables:

```
SALES.CUSTOMER_ID = CUSTOMERS.CUSTOMER_ID
```

This allows analytics such as:

- Total sales per customer
- Orders by customer
- Sales trends

⚙️ Warehouse Used

Warehouse

```
DEMO_WH
```

This warehouse:

- Executes all queries
- Is cost-efficient
- Automatically starts and stops

- Is suitable for demo and learning workloads

🧠 Semantic View (Understanding Layer)

A **Semantic View (YAML)** was created to describe:

- Business meaning of columns
- Relationships between tables
- Common analytical questions

Important:

- Semantic Views **do NOT store data**
- Semantic Views **cannot be queried directly**
- They are used only to help **Cortex understand the data**

🧱 Execution SQL View (Why & How)

Why this view was needed

Snowflake does not allow:

```
SELECT * FROM SEMANTIC_VIEW
```

So a **normal SQL VIEW** was created for execution.

View Created thru semantic view yaml file.

```
CREATE OR REPLACE VIEW DEMO_DB.PUBLIC.SHIVANSHI_EXEC_VIEW AS
SELECT
    c.CUSTOMER_ID,
    c.CUSTOMER_NAME,
    s.ORDER_ID,
    s.ORDER_DATE,
    s.AMOUNT
FROM DEMO_DB.PUBLIC.SALES s
INNER JOIN DEMO_DB.PUBLIC.CUSTOMERS c
    ON s.CUSTOMER_ID = c.CUSTOMER_ID;
```

This view:

- Does NOT duplicate data
- Is safe for large datasets

- Provides a single execution surface for AI-generated SQL

🔐 Role & Security Setup

A **read-only role** was created following least privilege security.

Role Name

```
SPCS_AGENT_ROLE
```

Permissions Granted

- Read access to tables and views
- Usage access to warehouse
- Access to Snowflake Cortex
- ❌ No write or admin permissions

This role is assigned to the Snowflake user used by the app.

🖥️ Streamlit Application (`app.py`)

The Streamlit app:

- Displays a UI
- Takes user questions
- Enforces a minimum of 4 words
- Disables Snowflake query caching
- Uses Cortex to generate SQL
- Executes SQL on the execution view
- Displays results in a table

📄 Full `app.py` Code

```
import streamlit as st
from snowflake.snowpark import Session
import os

# --------------------------------------------------
# Page Config
# --------------------------------------------------
st.set_page_config(
    page_title="QA Bot",
    page_icon="🤖",
    layout="centered"
)

# --------------------------------------------------
# Header
# --------------------------------------------------
st.markdown(
    """
    <h1 style="text-align:center;">🤖 QA Bot</h1>
    <p style="text-align:center; color: gray;">
        QA bot powered by Snowflake Cortex & Semantic View
    </p>
    """,
    unsafe_allow_html=True
)

st.divider()

# --------------------------------------------------
# Example Questions
# --------------------------------------------------
st.markdown("### 💡 Try an example question")

example_questions = [
    "List all customers and their orders",
    "Which customer has the highest total sales",
    "Show total sales by city",
]

cols = st.columns(len(example_questions))
for i, q in enumerate(example_questions):
    if cols[i].button(q):
        st.session_state["question"] = q

st.divider()

# --------------------------------------------------
# Question Input
# --------------------------------------------------
question = st.text_input(
    "Ask a question about Sales or Customers",
    key="question",
    placeholder="e.g. Which customer has the highest total sales"
)

# --------------------------------------------------
# Ask Button
# --------------------------------------------------
if st.button("Ask 🤔", use_container_width=True):

    # Rule: Minimum 4 words
    if len(question.strip().split()) < 4:
        st.error("❌ Invalid Length")
        st.stop()

    with st.spinner("Thinking..."):

        # Snowflake connection
        session = Session.builder.configs({
            "account": os.getenv("SNOWFLAKE_ACCOUNT"),
            "user": os.getenv("SNOWFLAKE_USER"),
            "password": os.getenv("SNOWFLAKE_PASSWORD"),
            "role": os.getenv("SNOWFLAKE_ROLE"),
            "warehouse": "DEMO_WH",
            "database": "DEMO_DB",
            "schema": "PUBLIC"
        }).create()

        # Disable caching
        session.sql("ALTER SESSION SET USE_CACHED_RESULT = FALSE").collect()

        # Cortex prompt
        prompt = f"""
You are a Snowflake Cortex Analyst.

You understand the business using this semantic model:
- Customers place sales orders
- SALES.CUSTOMER_ID joins to CUSTOMERS.CUSTOMER_ID
- Measures: AMOUNT
- Time dimension: ORDER_DATE

Generate ONE valid Snowflake SELECT query.
Do not explain.
Do not include markdown.
Do not include comments.

Execute queries ONLY on this view:
DEMO_DB.PUBLIC.SHIVANSHI_EXEC_VIEW

Question:
{question}
"""

        cortex_sql = f"""
        SELECT SNOWFLAKE.CORTEX.COMPLETE(
            'snowflake-arctic',
            $$ {prompt} $$
        ) AS SQL_QUERY
        """

        generated_sql = session.sql(cortex_sql).collect()[0]["SQL_QUERY"]

        if not generated_sql.strip().lower().startswith("select"):
            st.error("⚠️ Unable to generate a valid query")
            st.stop()

        df = session.sql(generated_sql).to_pandas()
        session.close()

    st.success("✅ Answer generated")

    with st.expander("🔍 View generated SQL"):
        st.code(generated_sql, language="sql")

    st.markdown("### 📊 Result")
    st.dataframe(df, use_container_width=True)
```

🔐 Credential Management

Snowflake credentials are stored as **environment variables** (in Render):

- SNOWFLAKE_ACCOUNT
- SNOWFLAKE_USER
- SNOWFLAKE_PASSWORD
- SNOWFLAKE_ROLE

They are never stored in code and never visible to users.

🚀 Deployment

- Code pushed to GitHub
- Streamlit app deployed on Render
- Environment variables configured in Render
- Public URL generated

Users can access the app without logging into Snowflake.