# Welcome

AMERICAN EQUITY™
INVESTMENT LIFE INSURANCE COMPANY®

## Team Introduction

Prajakta Wankhede

Tom Swenson

Mercy Parimala

# Agenda

Iowa State University

Business Analytics Capstone

American Equity

T

## Business Context

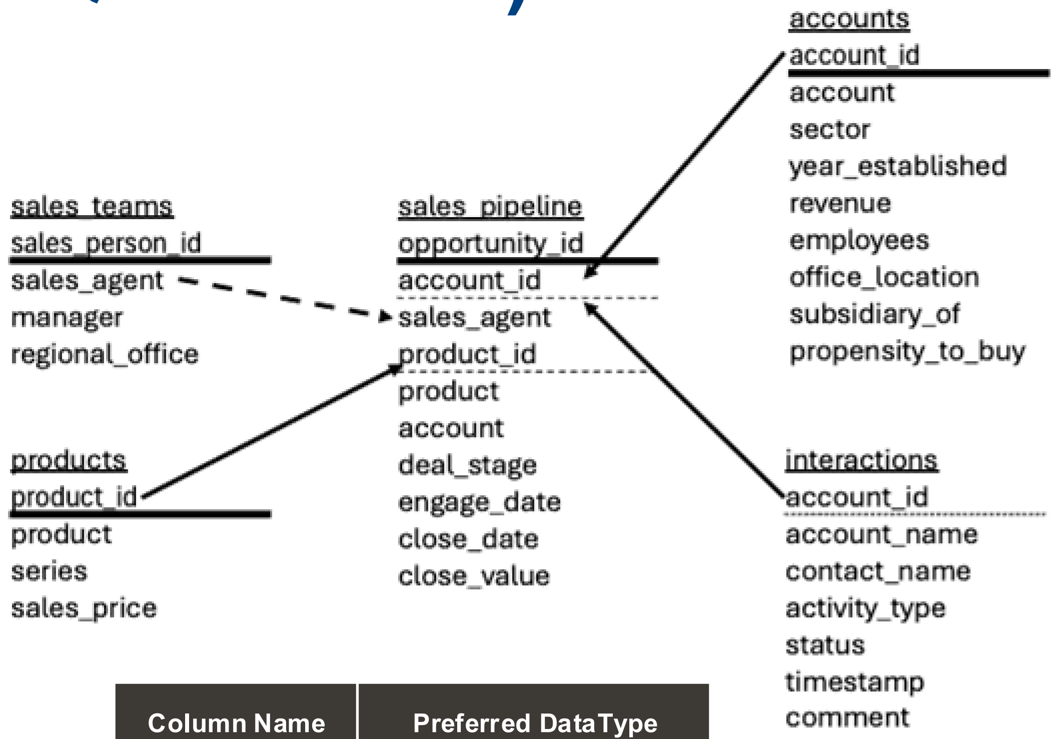
American Equity operates a relationship-centered financial services model where advisors rely on CRM data to manage accounts, track pipeline activity, and prepare for client interactions.
This data spans structured tables and unstructured notes, making it difficult and time-consuming for sales teams to locate the information they need.

The core challenge is that key insights are scattered across multiple systems, and traditional reporting tools require training and predefined queries. As a result, advisors spend significant time searching instead of engaging with clients, slowing follow-up, preparation, and decision-making.

This project addresses that problem by building a Retrieval-Augmented Generation (RAG) chatbot using American Equity's simulated CRM dataset. The goal is to give advisors fast, natural-language access to account history, interactions, and sales activity. The prototype blends SQL retrieval for structured data with semantic search for text fields, supporting real sales workflows while remaining simple, secure, and reproducible for future internal use.

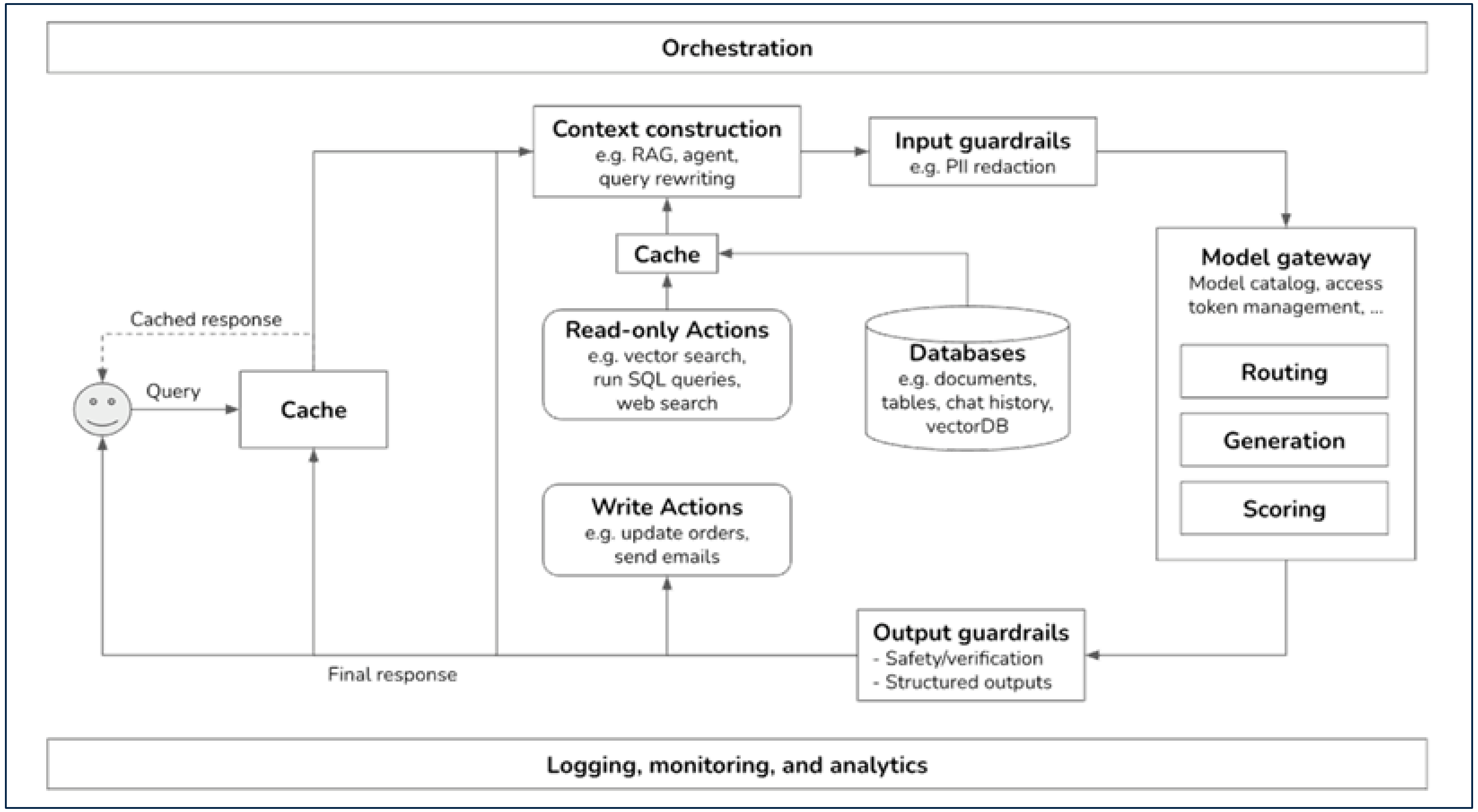
# DDL – Data Definition Language (SQL Schema)

- Establishes a clear and authoritative schema that ensures all SQL queries are accurate, consistent, and aligned with the underlying data model and types.
- Provides the structural foundation needed to build a reliable JSON catalog, ensuring each query's name, path, and description correctly reference real tables and relationships.
- Reduces redesign and debugging later by creating a stable framework that both the SQL layer and LLM routing logic can depend on.



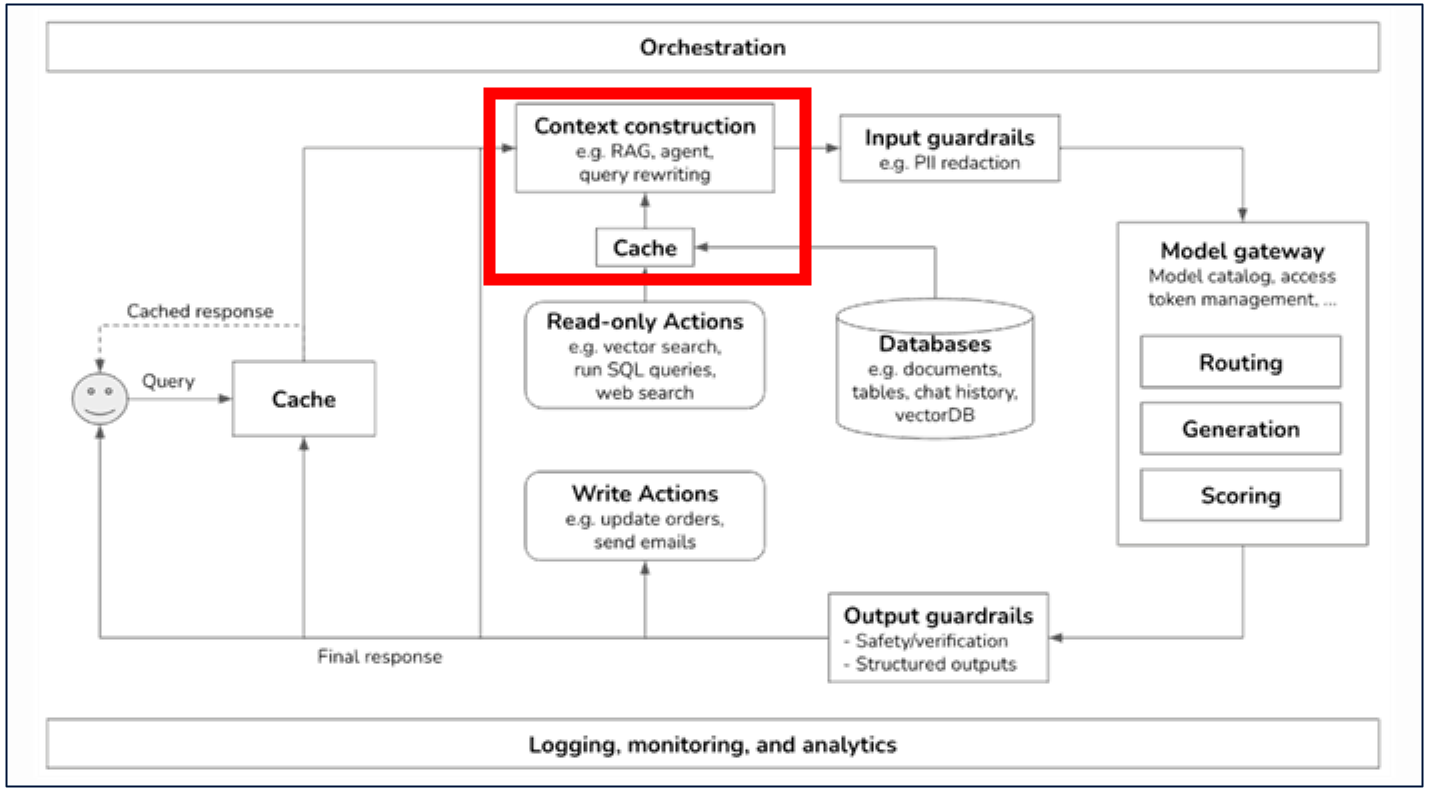
| Column Name | Preferred DataType |
|---|---|
| opportunity_id | TEXT |
| account_id | INT |
| sales_agent | TEXT |
| product_id | INT |
| product | TEXT |
| account | TEXT |
| deal_stage | TEXT |
| engage_date | DATE |
| close_date | DATE |
| close_value | INT |

## Generic Generative AI Platform



# RAG – Retrieval Augmented Generation

- CSV Files
- SQL Schema
- Data Type Transformation
- Missing Values & General EDA
- User Input
- User Input (summarized to 25 words)
- Predefined Prioritized SQL Queries
- JSON List of Queries
- Previous Questions



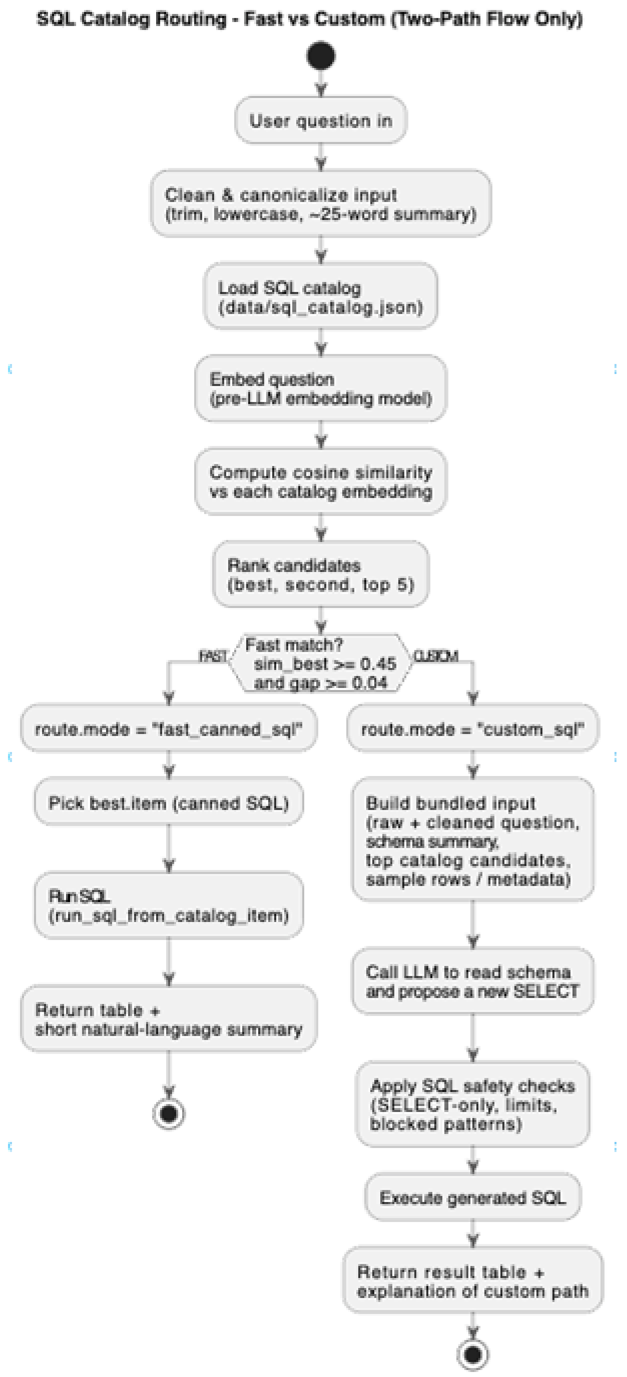
## Model Design & Selection – SQL v LLM

- We route most questions to predefined or parameterized SQL templates to ensure fast, accurate, and low-cost answers
- When a question falls outside those patterns, the LLM safely generates SQL within approved tables and guardrails
- This gives users flexibility while keeping financial data delivery governed, auditable, and efficient

| Factor | SQL Querying | | | LLM Generated Retrieval |
|---|---|---|---|---|
| **Cost to Run** | Very low once infrastructure exists | ✓ | ✗ | Higher per-request consumption of model credits |
| **Speed of Response** | Instant lookup on indexed tables | ✓ | ✗ | Slower due to reasoning and token generation |
| **Accuracy of Data** | Exact, deterministic, traceable values | ✓ | ✗ | May infer or hallucinate missing data |
| **Audit and Compliance Risk** | High transparency and easy logging | ✓ | ✗ | Harder to trace logic, may produce unverifiable output |
| **Scalability for Many Users** | Efficient under heavy load | ✓ | ✗ | Scales but cost increases as usage grows |
| **Best Fit Use Case** | Factual values, metrics, financial tables, agent lookups, reporting | ✓ | ✗ | Natural language explanation, fuzzy intent interpretation, summaries, recommendations |

# Generative AI Process Map

- The process map became our blueprint for building the routing engine
- It clarified the split between Fast SQL and Custom SQL paths
- It aligned business, data, and engineering teams on how decisions occur
- It surfaced risk and control points early rather than after coding
- It created a scalable model we can iterate, audit, and teach


SQL Catalog Routing - Fast vs Custom (Two-Path Flow Only)
User question in
Clean & canonicalize input (trim, lowercase, ~25-word summary)
Load SQL catalog (data/sql_catalog.json)
Embed question (pre-LLM embedding model)
Compute cosine similarity vs each catalog embedding
Rank candidates (best, second, top 5)
Fast match? sim_best >= 0.45 and gap >= 0.04
FAST
CUSTOM
route.mode = "fast_canned_sql"
Pick best.item (canned SQL)
Run SQL (run_sql_from_catalog_item)
Return table + short natural-language summary
route.mode = "custom_sql"
Build bundled input (raw + cleaned question, schema summary, top catalog candidates, sample rows / metadata)
Call LLM to read schema and propose a new SELECT
Apply SQL safety checks (SELECT-only, limits, blocked patterns)
Execute generated SQL
Return result table + explanation of custom path

# Prompt Engineering & SQL Queries

## Speed and consistency for the most common questions

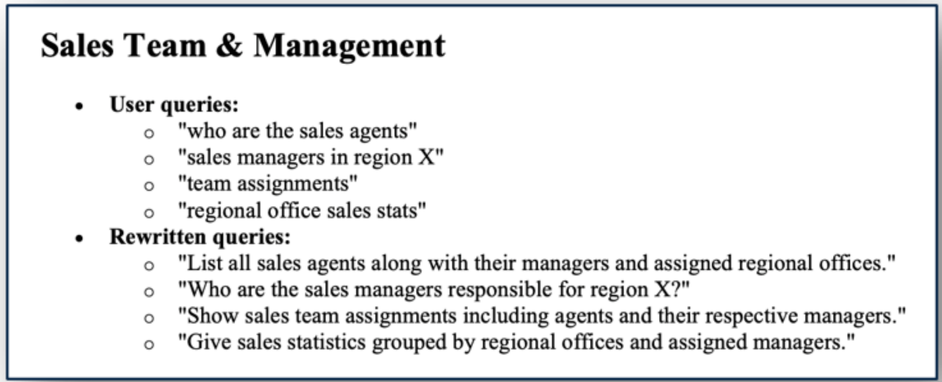
You instantly answer the top 70–80 percent of agent queries because the heavy-hitters are already mapped. There is no interpretation lag, no query building delay, and the results are consistent every time.

## Lower cost, lower risk, and better control

Pre-approved SQL reduces hallucination risk, supports compliance, and avoids extra LLM processing cost. You know exactly what data is being retrieved, how it is filtered, and how it will be presented.

## Better product adoption and agent experience

Agents learn the system faster because the answers match their natural questions. They see reliable, trusted output around the topics they care about most: revenue, pipeline, customers, teams, deals, and product stats.



### Sales Team & Management

- **User queries:**
  - "who are the sales agents"
  - "sales managers in region X"
  - "team assignments"
  - "regional office sales stats"
- **Rewritten queries:**
  - "List all sales agents along with their managers and assigned regional offices."
  - "Who are the sales managers responsible for region X?"
  - "Show sales team assignments including agents and their respective managers."
  - "Give sales statistics grouped by regional offices and assigned managers."

# Cosine Similarity & Embeddings

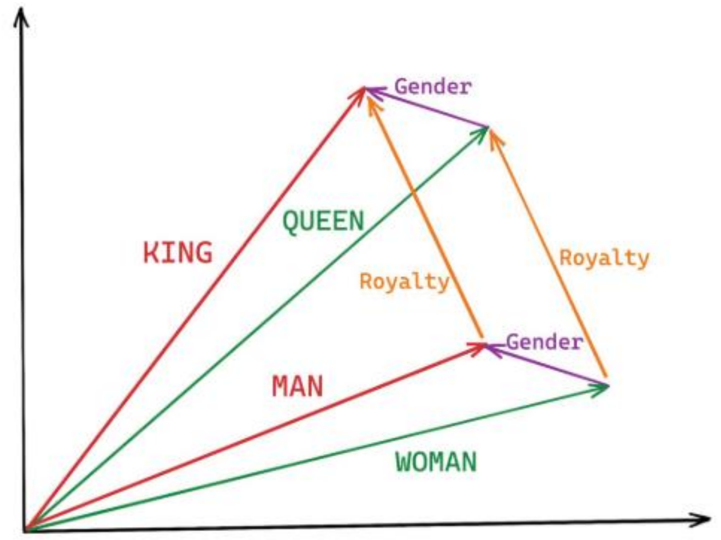
### Embeddings turn words and sentences into numbers
Each term or phrase becomes a vector so the computer can compare meaning, not spelling.

### Cosine similarity measures how close two meanings are
It checks the angle between vectors rather than distance, showing how aligned two ideas are.

### Similar ideas point in the same direction
“King” and “Queen” sit close because they share the concept of royalty, while “king” and “man” share gender meaning

### This lets systems find the best match even if text does not match exactly
Users can say “top customers” and the system finds “highest revenue accounts” as the closest intent

### It is the backbone of routing in our assistant
We embed user questions, compare them to known queries, and pick the closest meaning for fast SQL retrieval



T

## JSON for SQL Queries

- The JSON file is our catalog of canned queries, each stored with a name, path, and description
- The description becomes the semantic meaning of the query
- We embed the description so the system can measure similarity between user questions and catalog queries
- When a user asks something, we embed their question and find the closest description
- The catalog becomes the backbone of routing - cheap, fast, and governed

```
{  name:        "product_sales_value",
  path:        "SQL/product_sales_value.sql",
  description: "Show total sales value and won deal counts
grouped by product."
}
```

## JSON for Sort and Choose

- The JSON file is our catalog of canned queries, each stored with a name, path, and description
- The description becomes the semantic meaning of the query
- We embed the description so the system can measure similarity between user questions and catalog queries
- When a user asks something, we embed their question and find the closest description
- The catalog becomes the backbone of routing
  - cheap, fast, and governed

```
{  name:        "product_sales_value",

  path:        "SQL/product_sales_value.sql",

  description: "Show total sales value and won deal counts
grouped by product."

}
```

## Learnings – Technical, Business, & Risks

**Technical Learnings**

- Shift to SQL-first for accuracy & control
- Added question-history handling
- SQL guardrails reduced hallucinations
- JSON + embeddings improved routing

**Business & Risks**

- Bundled Input = richer answers but higher cost & oversight

## Future Roadmap – App Extension

- Expand to a full App Experience
- High-Speed Query Intelligence
- Expanded Business Coverage
- Visual Insights & Dashboards

## Conclusion

- Delivered a functional Generative AI Sales Assistant prototype
- Established a governed SQL-first + LLM architecture for accurate insights
- Enabled smart question routing using embeddings and similarity matching
- Created a scalable foundation for future expansion and enterprise adoption

## Thank you!

We’ve appreciated the opportunity to learn and help!

## DEMO

## App Launch Instructions

In terminal...

Prompt 1:

```
cd desktop/chatbot
```

Prompt 2:

```
python3 -m streamlit run app.py
```