Conversational AI over SQL Data

A Production-Ready PoC using Semantic Layer + AST Approach

Author:	Varun - Senior Data Engineer
Organization:	Datazen Consulting & Analytics LLP
Date:	February 03, 2026
Version:	1.0.0
Approach:	No LLM-Generated SQL

Key Highlights

✓ LLMs for Intent Understanding - Extract metrics, dimensions, filters

✓ Semantic Layer - Business logic, metadata catalog, validation

✓ AST-based Query Builder - Deterministic, safe SQL generation

✓ Type-Safe - Structured intermediate representations

✗ NO LLM-Generated SQL - Eliminates hallucination, injection risks

Table of Contents

1. Executive Summary

This document presents a **production-ready approach** for building Conversational AI systems over SQL/OLAP databases **without using LLMs to generate SQL queries**.

Architecture Flow

User Question

↓

LLM Intent Recognition (JSON Output)

↓

Semantic Layer Validation

↓

AST Query Builder

↓

SQL Execution

↓

LLM Response Formatting

2. Why NOT Use LLMs for SQL Generation?

Critical Problems with LLM-Generated SQL

Issue	Impact	Example
SQL Injection Risk	Security vulnerability	'; DROP TABLE users; --
Schema Hallucination	Wrong column/table names	SELECT non_existent_column
Join Logic Errors	Incorrect results	Missing/wrong foreign keys
Performance Issues	Unoptimized queries	Missing indexes, cartesian joins
Business Logic Bypass	Wrong calculations	Ignoring metric definitions
Non-Deterministic	Same question → different SQL	Inconsistent results

Our Approach: Structured Pipeline

■ **BAD:** LLM directly generates SQL

```
sql = llm.generate(f"Convert to SQL: {user_query}") # Dangerous!
```

■ **GOOD:** Structured semantic approach

```
intent = llm.extract_intent(user_query) # Returns JSON
validated_intent = semantic_layer.validate(intent)
ast = query_builder.build_ast(validated_intent)
sql = ast.to_sql() # Deterministic, safe
```

3. Solution Architecture

High-Level Components

1. USER INTERACTION LAYER

Natural language questions from web/mobile/chat interfaces

2. INTENT RECOGNITION (LLM)

Extract structured JSON with metrics, dimensions, filters, time ranges

3. SEMANTIC LAYER

Validate intent, resolve synonyms, apply business rules, determine joins

4. QUERY BUILDER (AST)

Build Abstract Syntax Tree and generate optimized SQL

5. QUERY EXECUTION

Execute against OLAP database and return structured results

6. RESPONSE GENERATION (LLM)

Format results into natural language with insights

4. Core Components

4.1 Data Models & Schemas

Defines the foundational data structures including Dimension, Metric, Relationship, and SemanticModel classes. These classes represent the semantic layer's metadata catalog.

4.2 Intent Schema (LLM Output)

Pydantic models that define the structured output from LLM intent recognition. Includes Filter, TimeRange, SortBy, and IntentObject classes with built-in validation.

4.3 Retail Sales Semantic Model

A complete implementation of the semantic model for retail sales analytics. Includes dimensions (date, product, store, customer), metrics (revenue, profit, quantity), and their relationships.

4.4 Semantic Layer Validator

Validates user intent against the semantic model, resolves synonyms to canonical names, converts relative time expressions to absolute dates, and determines required table joins.

4.5 AST Query Builder

Builds an Abstract Syntax Tree from validated intent and generates deterministic SQL queries. Handles SELECT, JOIN, WHERE, GROUP BY, ORDER BY, and LIMIT clauses systematically.

4.6 Intent Recognition

Integrates with Claude API to extract structured intent from natural language. Uses temperature=0 for deterministic extraction and comprehensive system prompts.

4.7 Main Orchestrator

The ConversationalSQL class that ties all components together. Manages the complete pipeline from question to response with conversation history.

5. Implementation Guide

Retail Sales OLAP Schema (Star Schema)

The implementation uses a standard star schema with one fact table (fact_sales) and four dimension tables (dim_date, dim_product, dim_store, dim_customer).

Sample Database Statistics

Table	Rows	Description
dim_date	730	2 years of date dimension data
dim_product	500	Products across 5 categories
dim_store	60	Stores across 4 regions
dim_customer	1,000	Customer segments and types
fact_sales	100,000	Transaction fact records

Installation & Setup

1. Install Dependencies

```
pip install anthropic pydantic faker
```

2. Set API Key

```
export ANTHROPIC_API_KEY='your-api-key-here'
```

3. Generate Sample Data

```
python generate_sample_data.py
```

4. Run Demo

```
python demo.py
```

6. Testing & Validation

The implementation includes comprehensive unit and integration tests to ensure reliability:

Unit Tests

• Metric and dimension synonym resolution

• Validation error handling for invalid inputs

• Relative time range calculations

• Filter operator validation

Integration Tests

• End-to-end SQL generation from intent

• Complex multi-table join handling

• WHERE clause generation with multiple filters

• ORDER BY and LIMIT clause correctness

Performance Tests

• Query execution time benchmarks

• Large result set handling

• Concurrent request processing

• Cache effectiveness metrics

7. Production Deployment

Deployment Architecture

Frontend Layer: Web UI (React/Vue), Mobile App (React Native), Teams/Slack Bot

API Gateway: Authentication (JWT/OAuth), Rate Limiting, Request Routing

Conversational SQL Engine: Intent Recognition, Semantic Validation, Query Building, Response Generation

Data Layer: Redshift/Snowflake (OLAP), Connection Pooling, Query Result Caching (Redis)

Security Considerations

• API key authentication for all endpoints

• Rate limiting to prevent abuse (60 requests/minute per IP)

• SQL injection prevention through parameterized queries

• Row-level security based on user permissions

• Audit logging of all queries and results

• Encrypted connections (HTTPS/TLS)

• Regular security audits and penetration testing

8. Alternative: BFSI Implementation

The same architecture can be applied to Banking, Financial Services, and Insurance (BFSI) domains. Example: Credit Card Transaction Analytics

BFSI Semantic Model Highlights

Dimensions:

transaction_date, card_type, card_tier, merchant_category, customer_segment, age_group, country, city

Metrics:

transaction_amount, transaction_count, avg_transaction_value, approval_rate, fraud_rate

Example Queries:

What was total credit card spend last month? **|** Show me fraud rate by merchant category **|** Compare Platinum vs Gold card usage

9. Conclusion

Key Advantages of This Approach

✓ **Security:** No SQL injection risks through parameterized queries and validation

✓ **Accuracy:** No schema hallucinations - all queries use known, validated schema

✓ **Performance:** Optimized, deterministic queries with proper indexing strategies

✓ **Maintainability:** Business logic centralized in semantic layer, easy to update

✓ **Scalability:** Caching opportunities, query optimization, horizontal scaling

✓ **Auditability:** Complete query logging, lineage tracking, compliance-ready

Production Checklist

■ Implement comprehensive logging and monitoring

■ Add query result caching with Redis

■ Set up alerts for query failures and performance issues

■ Implement proper authentication and authorization

■ Add rate limiting to prevent abuse

■ Create data governance and access policies

■ Set up CI/CD pipeline for automated deployments

■ Write comprehensive test coverage (>80%)

■ Document semantic model changes and versioning

■ Train users and create user documentation

Next Steps

6. Enhance semantic model with calculated fields and derived metrics

5. Integrate with visualization libraries (Plotly, D3.js) for charts

4. Add multi-tenant support with organization-level isolation

3. Implement advanced analytics (time series forecasting, anomaly detection)

2. Create voice interface with speech-to-text integration

1. Build mobile applications (iOS/Android) for on-the-go analytics

Document Information	
Created by:	Varun - Senior Data Engineer
Organization:	Datazen Consulting & Analytics LLP
Experience:	15+ years in Cloud Data Engineering
Specialization:	Generative AI, AWS Bedrock, LangChain, LangGraph
Date:	February 03, 2026
License:	MIT License
Repository:	Available on GitHub