AI Log - Group 6: Client Services

Case: Client Services **Group Number:** 6 **Date:** February 13, 2026

Open Access Links

This project utilized Antigravity. The full prompt history and chatbot interaction logs are provided as an attached Markdown (.md) file to ensure full transparency.

Tools Used

1. **Antigravity (Google DeepMind)** - Primary coding agent for complex data engineering, SQL generation, and Python scripting.
2. **Gemini 1.5 Pro/Flash** - Underlying LLMs used for reasoning, code generation, and data pattern recognition.

Purpose of Use

AI was leveraged as a **Senior Data Engineer / Architect** to transform raw, unstructured CSV data into a production-grade relational database for analysis.

Task Category	Specific Application
Data Engineering	Handling 64+ CSV files (14GB total) with inconsistent schemas and headers.
Database Design	Designing a 15-table relational schema (SQLite) with optimized indexing.
Code Generation	Writing robust Python scripts to import 188 million rows with error handling.
Data Validation	Creating automated validation reports to ensure 100% Referential Integrity.
Documentation	Auto-generating comprehensive technical documentation and Mermaid diagrams.

Prompt Examples

1. Initial Data Strategy

> **Prompt:** "*I have a folder with 14GB of CSV files for the Hobart database. The headers are inconsistent across years (2025 vs 2026). How should I structure this into a SQL database? Analyze the files and propose a schema.*"

AI Response: Analyzed file structures, identified the "Double ID" issue (duplicate IDs across time periods), and proposed a schema with `original_id` + `load_period` composite keys.

2. Handling Complex Logic (The "Double ID" Problem)

> **Prompt:** "*The SR_ID in the `history` tables doesn't match the IDs in the main `SR` table because the IDs reset each year. Create a mapping logic in Python to link them correctly.*"

AI Response: Wrote a Python script using dictionary mapping `(original_id, period) -> new_db_id` to ensure Foreign Key integrity across 50 million+ history records.

3. Strategic Decision Making

> **Prompt:** "*I found a folder `client_queries` with 133M rows. Is it important? What do I gain vs. what can I do without it?*"

AI Response: Analyzed the specific data (Customer-Ticket mapping), explained the trade-off (Operational Analysis vs. Customer Segmentation), and advised on the value for the business case.

4. Validation & Safety

> **Prompt:** "*Do one final rigorous check. Are all CSVs now in the database? Is the structure correct? Be rigorous because I will delete the CSVs after this.*"

AI Response: Performed a 6-step validation check (Row counts, Schema, FK orphans, Integrity check) and generated a safety report confirming 100% match before deletion.

Outputs and Validation

1. The Artifacts

- **SQLite Database (`hobart.db`):** 14 GB, 188 Million rows, 100% normalized.
- **Python Scripts:** 4 robust ETL scripts handling batch processing and memory management.
- **Validation Reports:** Markdown reports confirming 0 orphaned records across 8 foreign key relationships.

2. Validation Process

We employed a "Trust but Verify" approach:

1. **AI Generation:** AI wrote the import scripts.
2. ** Automated Testing:** AI wrote SQL queries to count rows in CSVs vs. DB to ensure no data loss.
3. **Human Review:** We reviewed the "Orphaned Record" counts (all zero) and random sample data before approving the final deletion of raw files.

Reflection (Critical Analysis)

Where AI Performed Better

1. **Pattern Recognition at Scale:** AI instantly spotted that `ACTIVITY_ID` and `COMMUNICATION_ID` columns were mixed up in the 2025 vs 2026 CSV headers. A human analysis of 50+ files would have likely missed this subtle inconsistency, leading to data corruption.
2. **Writing Boilerplate SQL/Python:** Generating 5 tables with 50+ columns, correct data types, and Foreign Key constraints took seconds. Manually writing `CREATE TABLE` statements for 188 columns would have been tedious and error-prone.
3. **Complex Logic Implementation:** The logic to map `original_id` + `period` to a new `surrogate_key` across 50M rows was complex. AI wrote an optimized Python script using in-memory dictionary mapping that ran efficiently (2.3M rows/min).

Where WE (Humans) Outperformed AI

1. **Strategic Business Value:** When AI identified the 133M row `client_queries` table, it provided the *facts*, but the *decision* to include it was human. We evaluated if "Customer Segmentation" was relevant to our specific business case hypothesis, overruling the initial plan to potentially skip it.
2. **Risk Management:** AI was ready to delete files once the script finished. We (Humans) insisted on a "*Final Pre-Deletion Safety Check*" with specific criteria. The

judgment call to "stop and verify" before destructive actions is a human responsibility.

3. **Contextual Interpretation:** AI saw "data." We saw "Business Processes." We interpreted that `SR_CONTACT` wasn't just a table, but represented the "Agent Workload." We directed the AI to focus on *operational metrics* (time-to-resolve) rather than just technical metrics.

Conclusion

This project demonstrated a powerful **Human-in-the-Loop** workflow. AI acted as the "Hands" and "Technical Architect"—handling the massive scale of data engineering that would be impossible manually. The Human team acted as the "Head" and "Product Manager"—defining the *value* of the data, setting safety boundaries, and interpreting the results for business insights.