UNIVERSITY OF WESTERN AUSTRALIA

Project Report

Explainable Natural Language Query Interface for Rational Databases Using a Multi-Agent System

Group: 9

Student Name	**Student ID**
1. Cedrus Dang	24190901
2. Franco Meng	23370209
3. Laine Mulvay	22708032
4. Nirma Rajapaksha Senadherage	24268122
5. Aswathy Mini Sasikumar	24331072
6. RuiZhe Wang	24260749

CITS 5553 – Data Science Capstone Project SEM-2 2025

Client: Dr. Sirui Li

School of Information Technology, Murdoch University

Introduction

Relational databases remain central to enterprise data management yet querying them still requires understanding of the database schema and SQL expertise. Recent research on text-to-SQL models have enabled natural language database querying, but many solutions lack transparency, limiting trust and adoption in enterprise settings.

Our project aims to build an **explainable natural language interface** that lets users explore relational databases without writing SQL, while maintaining transparency through step-by-step reasoning and visual schema exploration. We use the **Spider v1 dataset**, a large, complex, cross-domain benchmark originally annotated by 11 Yale students, contains over 200 databases, each averaging 28 columns and 9 foreign keys, amounting to more than 5,600 columns in total for SQL generation (Yu et al., 2018). Although we originally planned to focus on easy-to-medium queries, the strong performance of our design allowed us to include all difficulty levels in testing.

Our approach automatically extracts all database schemas (tables, columns, foreign keys) from user-uploaded SQLite files, converts them into database – table – column pairs to apply a 3 multi agents system pipeline:

Select the most relevant database → Identify candidate tables → Generate and execute the final SQL query

Each agent uses retrieval-augmented generation (RAG) to produce result and ground its reasoning, the system returns both the SQL output and clear explanations with an interactive schema graph. The application is implemented with a Python/Django backend, Next.js frontend, and deployed in Docker for cross-system compatibility.

Methods & Result

Given the complexity of the task: to construct 1 SQL query from over 5,600 columns across 200 databases based on one natural language query. We have leveraged recent advances in agentic AI, using the LangChain framework, to decompose the complex problem into subtasks handled by 3 specialized agents, each agent is a gpt5-mini LLM, orchestrated sequentially to improve the overall performance of the pipeline, and to provide detailed, step-by-step explanations to the user.

Agent 1:

We experimented extensively with ways to prepare and structure database schemas so they could be efficiently processed by the LLM, along with prompt engineering. The first agent: selecting a single database from more than 200 candidates, proved the most challenging.

Our final method comprises three key components:

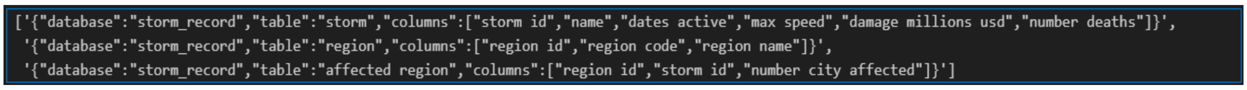
1. **Balanced schema information**
   We aimed to provide enough detail for accurate reasoning without introducing noise. **Only database names, table names, and column names** were included, while foreign keys, data types, and other metadata were omitted to reduce ambiguity.

2. **Robust data structure and format**
   Rather than using the Spider-provided schema files, we designed a process to extract schema information directly from the uploaded SQLite databases, reflecting real-world conditions where clean, up-to-date schema files are usually not available. After testing multiple formats, a row-level structure proved most effective.

Table 1 The Format Example of the original JSON schema provided in Spider dataset.

'db_id':	'storm_record',
'table_names':	['storm', 'region', 'affected region'],
'table_names_original':	['storm', 'region', 'affected_region']}
'column_names':	[[-1, '*'], [0, 'storm id'], [0, 'name'], [0, 'dates active'], [0, 'max speed'], [0, 'damage millions usd'], [0, 'number deaths'], [1, 'region id'], [1, 'region code'], [1, 'region name'], [2, 'region id'], [2, 'storm id'], [2, 'number city affected']],
'column_names_original':	[[-1, '*'], [0, 'Storm_ID'], [0, 'Name'], [0, 'Dates_active'], [0, 'Max_speed'], [0, 'Damage_millions_USD'], [0, 'Number_Deaths'], [1, 'Region_id'], [1, 'Region_code'], [1, 'Region_name'], [2, 'Region_id'], [2, 'Storm_ID'], [2, 'Number_city_affected']],
'column_types':	['text', 'number', 'text', 'text', 'number', 'number', 'number', 'number', 'text', 'text', 'number', 'number', 'number'],
'primary_keys':	[1, 7, 10],
'foreign_keys':	[[11, 1], [10, 7]],

Table 2 The Example of reformatted row level schema:



```
['{"database":"storm_record","table":"storm","columns":["storm id","name","dates active","max speed","damage millions usd","number deaths"]}',
 '{"database":"storm_record","table":"region","columns":["region id","region code","region name"]}',
 '{"database":"storm_record","table":"affected region","columns":["region id","storm id","number city affected"]}']
```

3. **Retrieval-Augmented Generation (RAG)**
   Instead of feeding all 200 schemas to the language model, we firstly embedded both the user's natural language query and the full prepared schema representations, performed similarity search (L2 distance) to select the top ***K*** most relevant schemas (default *K*=5, user-adjustable). These top ***K*** candidates were then provided to the first agent, enabling accurate database selection while keeping input compact and relevant.

Agent 2:

Once the target database was identified by Agent 1, Agent 2 retrieved its complete schema, since the first agent had only seen limited table-level information during the top-*K* selection step. Agent 2 then combined **the full schema** with the user's query to recommend the most relevant tables.

Agent 3:

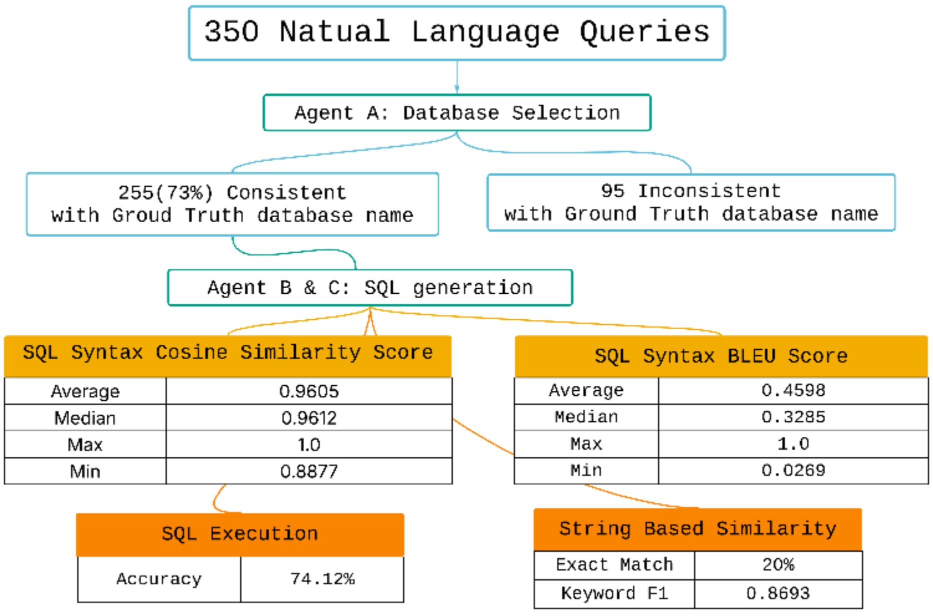
Agent 3 functioned as the final consolidation stage. It retrieved the complete schema of the selected database - **now augmented with foreign key** in JSON format to capture the accurate relational structure for SQL JOIN statement. Leveraging this enriched schema context, Agent 3 review the recommended tables from Agent 2 again, then synthesized the final executable SQL query.

Table 3 The Format Example of the full schema presented to Agent 3

'storm':	{'columns': ['Storm_ID', 'Name', 'Dates_active', 'Max_speed', 'Damage_millions_USD', 'Number_Deaths'], 'primary_key': ['Storm_ID'], 'foreign_keys': []},
'region':	{'columns': ['Region_id', 'Region_code', 'Region_name'], 'primary_key': ['Region_id'], 'foreign_keys': []},
'affected_region':	{'columns': ['Region_id', 'Storm_ID', 'Number_city_affected'], 'primary_key': ['Region_id', 'Storm_ID'], 'foreign_keys': [{'from_column': 'Storm_ID', 'ref_table': 'storm', 'ref_column': 'Storm_ID'}, {'from_column': 'Region_id', 'ref_table': 'region', 'ref_column': 'Region_id'}]}}

Testings & Results:

From the Spider dataset's ground-truth file, we randomly sampled 350 question - SQL pairs. **Only the question texts** were provided to Agent A for database selection, cases where Agent A identified the correct database were then passed to Agents B and C.


350 Natual Language Queries
Agent A: Database Selection
255(73%) Consistent with Groud Truth database name
95 Inconsistent with Ground Truth database name
Agent B & C: SQL generation

SQL Syntax Cosine Similarity Score	
Average	0.9605
Median	0.9612
Max	1.0
Min	0.8877

SQL Syntax BLEU Score	
Average	0.4598
Median	0.3285
Max	1.0
Min	0.0269

SQL Execution	
Accuracy	74.12%

String Based Similarity	
Exact Match	20%
Keyword F1	0.8693

Figure 1Testing Result

Table 4 Explanation of Evaluation Metrics

Cosine similarity Score: quantifies how semantically similar the embeddings of the predicted and ground-truth SQL queries are. Although this method was novel, it was proposed by our client Sirui, as well as inspired by graph-based/structural metrics (e.g. Zhan et al. 2025), where graph embedding was used for similarity comparison.
BLEU Score (Bilingual Evaluation Understudy): measures how closely the predicted SQL query matches the ground-truth SQL in terms of token sequence and structure.
SQL Execution: The accuracy is evaluated by comparing the execution outputs of the predicted and ground-truth queries, where identical results indicate correctness. Row order is ignored, while column order must align with the ground truth.
String Based Similarity: - **Exact Match:** evaluates whether two SQL queries are identical character by character after normalization, which includes removing extra spaces, converting to lowercase, and omitting semicolons. The score is typically low, as minor variations such as table aliases can lead to mismatches even when the queries are syntactically equivalent.The limitations of both EXE (Execution Accuracy) and ESM (Exact Set Matching) thoroughly discussed by Ascoli, Kandikonda, and Choi (2025). - **Keyword F1:** evaluates the overlap of SQL keywords between the predicted and ground-truth queries. It is computed by extracting and comparing the sets of SQL keywords from both queries to derive precision, recall, and their harmonic mean (F1). This metric reflects how accurately the key structural components of the SQL query are captured by the prediction. Please refer Appendix table 1 for classification of keywords.

Web Application

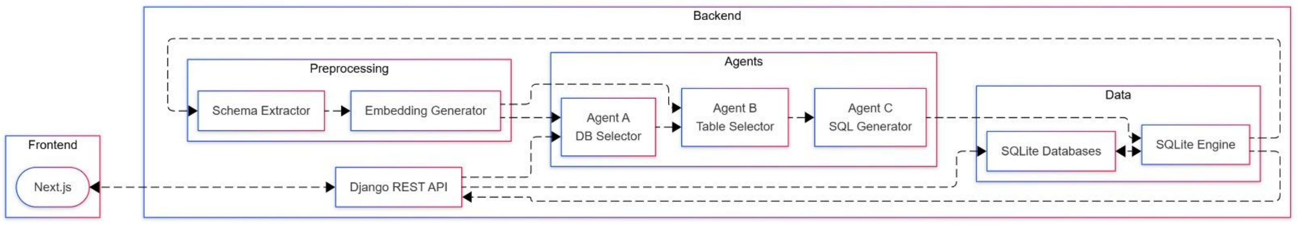


Figure 2 Web Application System Design

The figure shows the architecture of the application. The frontend, built with Next.js, sends user input to the backend through a Django REST API. This can be SQLite databases to the Data Layer or the prompt of the query to the Agents.

The backend includes three main parts:

- Preprocessing - Schema Extractor and Embedding Generator prepare database metadata and vector embeddings.
- Agents - Agent A (DB Selector), Agent B (Table Selector), and Agent C (SQL Generator) cooperate to interpret natural-language queries.
- Data Layer - SQLite Databases and SQLite Engine execute the generated SQL and return results to the frontend.

The app also includes other modules like graphical visualization of database schema in the backend and the MySQL-like query result table in the frontend, to enhance user experience. User can also adjust top-*K* of Agent A and select the level of explainability in the frontend.

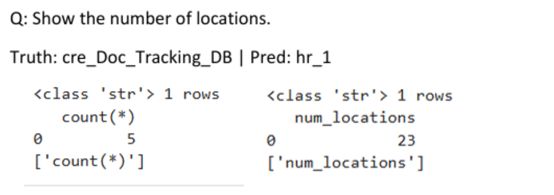
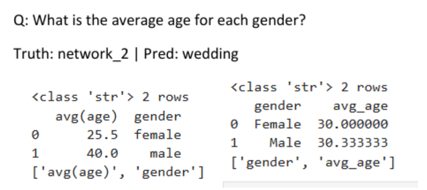
Discussion

The system demonstrates strong functional performance, achieving 74% execution accuracy, which would further improve if column order constraints were relaxed. The high cosine similarity and Keyword F1 scores indicate the model's effectiveness in generating semantically and structurally sound SQL queries. In contrast, the relatively low BLEU and Exact Match scores highlight the inherent flexibility of SQL syntax, where multiple valid formulations can produce the same execution result.

After extensive experimentation and analysis, we further identified the following:

1. **Why agent A recommends different database?**
   Agent A's predicted database name differed from the ground truth in **95 cases** (see Figure 1). Manual review shows the primary causes are

   i) **Query ambiguity**
   Broad/generic questions can be correctly answered by more than one database.

 Q: Show the number of locations. Truth: cre_Doc_Tracking_DB | Pred: hr_1	 Q: What is the average age for each gender? Truth: network_2 | Pred: wedding
hr_1 also contains a table called Location, able to perfectly answer the question	Wedding Database contains a table called 'People' recording gender and age for each Person_ID

```
<class 'str'> 1 rows
   count(*)
0         5
['count(*)']
```

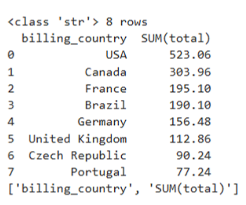
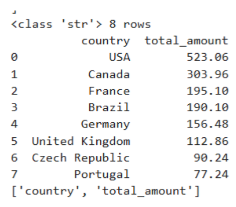
```
<class 'str'> 1 rows
   num_locations
0             23
['num_locations']
```

```
<class 'str'> 2 rows
   avg(age)  gender
0      25.5  female
1      40.0    male
['avg(age)', 'gender']
```

```
<class 'str'> 2 rows
   gender    avg_age
0  Female  30.000000
1    Male  30.333333
['gender', 'avg_age']
```

Table 5

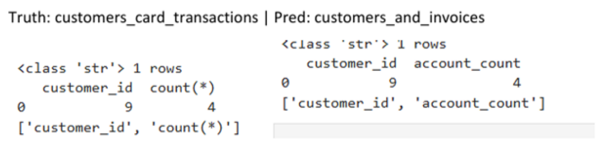
   ii) **Different database names sometimes contain the same tables/data, leading to multiple valid selections.**
   Our agents successfully surface duplication/overlap that would be impractical for a human to track. Example below showing different database name predictions, but SQL execution results are the same.

Question: What are the names of the top 8 countries by total invoice size and what are those sizes? **Truth: store_1 | Pred: chinook_1**  
Both *chinook_1* and *Store_1* databases contains **identical** tables and data.

```
<class 'str'> 8 rows
  billing_country  SUM(total)
0             USA      523.06
1          Canada      303.96
2          France      195.10
3          Brazil      190.10
4         Germany      156.48
5  United Kingdom      112.86
6  Czech Republic       90.24
7        Portugal       77.24
['billing_country', 'SUM(total)']
```

```
<class 'str'> 8 rows
          country  total_amount
0             USA        523.06
1          Canada        303.96
2          France        195.10
3          Brazil        190.10
4         Germany        156.48
5  United Kingdom        112.86
6  Czech Republic         90.24
7        Portugal         77.24
['country', 'total_amount']
```

Question: What is the customer id of the customer with the most accounts, and how many accounts does this person have?
Truth: customers_card_transactions | Pred: customers_and_invoices



Truth: customers_card_transactions | Pred: customers_and_invoices

```
<class 'str'> 1 rows
   customer_id  count(*)
0            9         4
['customer_id', 'count(*)']
```

```
<class 'str'> 1 rows
   customer_id  account_count
0            9              4
['customer_id', 'account_count']
```

Both *customers_card_transaction* and *customers_and_invoices* contain **identical** tables and data.

Table 6

iii) Context limitation in Agent A:

Agent A receives only top-*K* schema snippets from similarity search (per-table, not a full-database view). This partial context can nudge it toward a 'possible' database instead of the ground truth.

Question : What are the names and years of the movies that has the top 3 highest rating star?
Truth: movie_1 | Prediction: imdb

Comment: Agent A predicted imdb, based on the given information of :
{"database":"imdb","table":"movie","columns":["mid","title","release year","title aka","budget"]}
Which is a plausible prediction.
It is interesting to see when IMDB passed into agent B to recommend tables, Agent B correctly identified the possible issue with this database:
Selected Tables: ['movie', 'cast', 'actor']
Agent Reason: *The movie table contains title and release year (required to return names and years). The cast table links movies (msid) to actors (aid) so you can identify which actor is the star of a movie. The actor table provides actor names.* ***However, the provided schema contains no rating column or ratings table, so there is no data to determine the top-3 highest-rated stars; to answer the query you need a ratings table/column (e.g., movie_rating or actor_rating) or clarification of where 'rating star' is stored***

Table 7

2. Why divergence between Similarity / BLEU, and Keywords F1 / Exact Match?

Overall, the embedding-based similarity score tracks how close two SQL strings are, but we observe large gaps versus BLEU, which is n-gram precision and order-sensitive. Similar in Keywords F1 and Exact Match metrics.
Manual inspection of low-scoring cases highlights 5 main reasons:

i) SQL flexibility

Many semantically equivalent SQL rewrites change token order (e.g., join order, correlated vs. uncorrelated subqueries). BLEU and Exact Match penalize valid rewrites, while similarity is less sensitive.

"query"	"What are the names of documents that do not have any sections?",
"sql_truth"	"SELECT document_name FROM documents WHERE document_code NOT IN (SELECT document_code FROM document_sections)",
"sql_pred"	"SELECT d.document_name FROM Documents d LEFT JOIN Document_Sections ds ON d.document_code = ds.document_code WHERE ds.section_id IS NULL",
"similarity Score"	0.88
"BlEU Score"	0.07
Comment	Ground truth using NOT IN subquery where the prediction using the rows where can not been joined to Document_Sections.

Table 8

ii) Alias choice

Table aliases in SQL are arbitrary. The ground truth tends to use ‘T1, T2’ as normal practise, while predictions often introduce abbreviation of table names. This inflates token mismatch for BLEU/Exact Match without affecting execution. Ascoli, Kandikonda, and Choi (2025) discussed the Enhanced Tree Matching (ETM) method to negate the alias issue. Where the normalisation process with ensure to recognise structure and alias do not affect the query meaning.

iii) Dialect differences & minor errors

This is another major limitation of the current system, where the predicted SQL sometimes use dialect-specific features (e.g., PostgreSQL ‘ILIKE’, MySQL ‘SUBSTRING_INDEX’) that don’t exist in SQLite, where a replacement of LIKE LOWER('%…%') is needed. These differences depress BLEU and can break execution despite near-identical intent.

iv) Term ambiguity

LLMs may adopt plausible but different interpretations (e.g., “host” as host city vs. ground truth’s human host), yielding structurally sound but lexically divergent SQL.

"query"	What are the hosts of competitions whose theme is not "Aliens"?
"sql_truth"	SELECT Hosts FROM farm_competition WHERE Theme != 'Aliens'
"sql_pred"	SELECT DISTINCT c.official_name FROM farm_competition fc JOIN city c ON fc.host_city_id = c.city_id WHERE fc.theme <> 'Aliens'
"Similarity Score"	0.909
"BLEU Score"	0.039
Comment	The LLM interpreted “host” as the host city, which aligns with common usage in sports and competitions, whereas the ground truth labeled “host” as the human host.

Table 9

i) Robust Coding

LLM can generate more robust coding practise than human, below is an example where the ground truth will produce an arbitrary result when there are more than 1 equally highest evaluation.

"query"	What is the country of the airport with the highest elevation?
"sql_truth"	SELECT country FROM airports ORDER BY elevation DESC LIMIT 1
"sql_pred"	SELECT country FROM airports WHERE elevation = (SELECT MAX(elevation) FROM airports)
"similarity Score"	0.946
"BLEU Score"	0.162
Comment	The LLM can produce more robust coding practise.

Table 10

Conclusion

Compared with traditional text-to-SQL systems - which often struggle on unseen questions and untrained domains. Our LLM-driven multi-agent architecture scales to hundreds of databases while providing transparent reasoning: why a database was selected, why specific tables were recommended, how the schema was mapped, and why the final SQL was produced. This will materially improve operator confidence for real-world deployment.

We provide below considerations for further enhancement to address above identified issues:

1. **An SQL Dialect / Syntax Adapter**
   Address the dialect issues noted in Discussion **2.iii** by adding an agent that analyses the final query, ensures compatibility with the target DBMS, to auto-translates constructs (e.g., ILIKE → LOWER(col) LIKE ...)

2. **A Data Type & Sample Reviewer:**
   Incorporate column types and small value samples to refine predicates and encodings (e.g., gender stored as Female/Male/Other, F/M/O, or 0/1/2). This agent would be able to update the SQL syntax to sure the successful filtering of the data.

3. **SQL Canonicalization:**
   To handle major limitation mentioned in **1.ii**, Finegan-Dollak et al. (2018) proposed SQL Canonicalization to enable consistent evaluation. Including order fields alphabetically, standardize table alias, and standardize capitalization and spaces.

4. **More robust evaluation metrics:**
   As shown in **Figure 1**, the BLEU score, being inherently based on n-gram overlap, tends to penalize the syntactic flexibility of SQL queries. Chen et al. (2019) introduce SQL-BLEU and Canonical-BLEU, which are variants of BLEU tailored for comparing generated and reference SQL queries by focusing on SQL keywords or abstracting identifiers.

5. **Parallel Candidates for Ambiguous Queries:**
   To handle cases in discussion **1.i** where multiple databases can answer a generic question, run top *K* candidates in parallel, returning side-by-side rationales to allow user to encourage user asking more specific questions.

6. **Review / Re-ranker with Feedback Loop:**
   To address above issue **1.iii**, once the Agent 2 detected the potential issue where a column is missing to answer the question, this agent should drop the candidate and requesting the next best database from Agent A. This prevents non-executable SQL with ‘hallucinated’ columns.

7. **Temperature Settings:**
   The current application uses a fixed LLM temperature of 0 for consistent results. Allowing users to adjust this value, or provide feedback on database selection, could improve flexibility and user satisfaction.

Reference

Ascoli, B., Kandikonda, Y. S. R., & Choi, J. D. (2025). *ETM: Modern insights into perspective on Text-to-SQL evaluation in the age of large language models* (arXiv preprint arXiv:2407.07313). https://doi.org/10.48550/arXiv.2407.07313

Chen, F., Hwang, S.-w., Choo, J., Ha, J.-W., & Kim, S. (2019). *NL2pSQL: Generating Pseudo-SQL Queries from Under-Specified Natural Language Questions*. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (pp. 2603–2613). Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1262

Finegan-Dollak, C., Kummerfeld, J. K., Zhang, L., Ramanathan, K., Sadasivam, S., Zhang, R., & Radev, D. (2018). *Improving Text-to-SQL evaluation methodology*. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics* (pp. 351–360). Association for Computational Linguistics. https://aclanthology.org/P18-1033

Yu, T., Zhang, R., Yasunaga, M., Tan, Y. C., Lin, X., Li, S., Er, H., Li, I., Pang, D., Chen, T., Ji, Y., Dixit, S., & Liang, P. (2018). *Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task* [Preprint]. arXiv. https://arxiv.org/abs/1809.08887

Appendix A : SQL Keywords

```
# Basic clauses
'select', 'from', 'where', 'group', 'by', 'having', 'limit',
# Join keywords
 'join', 'inner', 'left', 'right', 'full', 'outer', 'on', 'as',
# Set operations
 'union', 'intersect', 'except',
 # Logical operators
'and', 'or', 'not',
 # Other operators
'in', 'like', 'exists', 'between',
# Sorting
 'asc', 'desc',
 # Special
 'distinct',
# Aggregation functions
'count', 'sum', 'avg', 'max', 'min'
```

Appendix B: Contribution Table

Member name	**Member name**	**Tasks for the project**	**Skills contributed to the team & project**
Cedrus Dang	24190901	Application, schema extraction pipeline development and proposal researching.	Backend–Frontend, Design in web application and agentic-AI system development, documenting, and research.
Franco Meng	23370209	Team Lead, Schema representation, Agents Design, Evaluation Setup/Prompt improvement, Client liaison/updates, Report writing	Schema transformations, Embedding, similarity search, LangChain framework design, RAG implementation, prompt engineering. Evaluation.
Laine Mulvay	22708032	meeting minutes and task planning, repo setup, helped develop and finalise schema extraction scripts, helped work on all agent development in notebooks, built out CLI agents with inbuilt testing based in agent notebooks, explainable interface, auto upload all spider data, containerised project.	Project management, meeting documentation, technical including frontend, notebook and script development.
Rajapaksha Senadherage	24268122	Agent development, embedding-based query schema alignment testing,	Database and Table Selector agent prototype development, Ground truth and predicted SQL similarity

Nirma Dilrukshi		SQL evaluation, similarity metric validation, documentation	evaluation, analytical reasoning for inconsistent DB selections, documentation in proposal
Aswathy Mini Sasikumar	24331072	Collaborated on literature review and dataset analysis, contributed to proposal drafting, assisted in agent design, and conducted manual and performance evaluations aligned with client goals.	Research and analytical skills, System design using OpenAI embeddings, LangChain RAG concepts, and Prompt engineering, Evaluation and problem-solving to ensure performance.
RuiZhe Wang	24260749	Participated in building the front end and back end. Designed the system architecture diagram. Improved proposals and documentation content.	UI flow design, API contract refinement, tests, and integration bug-fixing for the final release. Shipped the first full-stack implementation and setup that the team iterated on. Figma redesign for readability/usability; early build wasn’t adopted as-is but informed the final system.