SwellDB: GenAI-Native Query Processing via On-the-Fly Table Generation

Victor Giannakouris
Supervised by: Immanuel Trummer
Cornell University
Ithaca, NY, USA
vg292@cornell.edu

ABSTRACT

We present SwellDB, a data system that enables analytical query processing over tables generated dynamically with Large Language Models (LLMs). Typically, traditional database systems function under the closed-world assumption, allowing query execution over already existing tables. In contrast, SwellDB dynamically generates the required tables based on input SQL queries and user-defined table definitions. Powered by an LLM, SwellDB integrates external data sources, including diverse datasets, database systems, and search engines to extract and blend relevant information into a structured table. Given an input query and schema, SwellDB intelligently selects data sources, synthesizes structured tables according to the specified schema, and makes them queriable through the backend execution engine. SwellDB addresses data generation and integration challenges in several domains such as web search, bioinformatics, and finance.

VLDB Workshop Reference Format:
Victor Giannakouris
Supervised by: Immanuel Trummer. SwellDB: GenAI-Native Query Processing via On-the-Fly Table Generation. VLDB 2025 Workshop: PhD Workshop.

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/SwellDB/SwellDB.

1 INTRODUCTION

We present SwellDB [3], a data system that follows a new architecture that *generates tables at runtime*. Unlike traditional systems which rely on previously stored tables, SwellDB synthesizes tables on the fly based on input queries, utilizing Large Language Models (LLMs) to generate structured data according to user-defined prompts and schemas. By efficiently integrating data from multiple sources, including LLMs, file formats, databases, and raw text such as web search results, SwellDB bridges the gap between *structured query execution and unstructured data retrieval*.

Closed-world Assumption. Traditional database systems operate under the closed-world assumption, where queries are executed on pre-existing, structured tables. Structured and queryable data offers several key advantages. First, database systems enable users to leverage SQL, a widely adopted and standardized language over the past decades, for executing analytical queries and retrieving precise results. Second, structured data facilitates seamless integration with external services, such as mobile applications and websites, through a wide range of connectors and APIs. However, while querying structured data is highly desirable, the vast majority of data resides in external semi-structured or unstructured sources. Consequently, end-users are often burdened with developing custom Extract-Transform-Load (ETL) workflows to create structured tables, enabling them to query this data using SQL.

Large Language Models. LLMs offer a compelling solution to the challenges of information integration and table generation. Trained on vast amounts of data embedded within their weights, LLMs excel at data generation and augmentation tasks. For example, given input datasets and a search engine, an LLM can determine how to utilize these data sources to generate a table with specific content and schema. LLMs can efficiently answer questions such as which code or SQL query to execute to retrieve a portion of the requested data from the input data sources, or which search queries to issue in order to retrieve data not available in the LLM or input sources. Additionally, LLMs can serve as preprocessors, transforming unstructured data (e.g., web-retrieved information) into structured formats that are queryable with SQL.

Taking these considerations into account, we pose the following question:

Given a set of input data sources, how can we leverage Large Language Models to dynamically generate tables of any content based on an input query?

We address this question by introducing a data system architecture that enables query execution over dynamically generated tables, according to the input query. Our system, SwellDB, leverages Large Language Models to tackle the table generation problem as follows. Given a logical table definition $L = (P, A)$, where P a natural language prompt that describes the table content and $A = \{(a_1, d_1), (a_2, d_2), ..., (a_n, d_n)\}$ the schema, where a_i the attribute i and d_i each description, SwellDB interacts with the LLM to compute a table generation plan. This plan specifies *which input data sources* can be used to retrieve portions of the data and how *to utilize the input data sources* in order to effectively synthesize a table that complies with the input query. SwellDB is based on the following insight. An LLM can be leveraged as a data processing operator to *transform unstructured data into a structured format*, ensuring that the individual extracted chunks of information can be seamlessly integrated into a unified table. As a result, SwellDB

Proceedings of the VLDB Endowment. ISSN 2150-8097.

enables a paradigm shift from the *closed-world to the open-world assumption* in query execution. Instead of restricting queries to pre-existing tables, SwellDB dynamically generates tables on the fly, incorporating any relevant information based on the input query.

2 OVERVIEW

Before presenting the technical details, we illustrate a representative use case that SwellDB addresses. Assume that we have a dataset in some local or remote storage that contains genetic mutations, consisting of the *sample_id* and *mutation* columns. The end-user needs to retrieve all genetic mutations that are related to cancer. Since the only available information is the mutation codes, this task would consist of three steps: (1) extract all mutation codes from the dataset, (2) find the associated disease(s) of each mutation and (3) keep the ones that are related to cancer and filter-out the rest. Figure 1 visualizes this pipeline in SwellDB. During step 1, the *mutation* column is extracted by issuing an SQL query (this assumes that this dataset is loaded in some SQL query engine). In step 2, for each of the mutations SwellDB issues search queries to retrieve the associated protein and disease from the web. The results are fed into an LLM, which extracts a table that contains the *protein* and *disease* columns. Finally, in step 3 the resulting table is synthesised by combining the intermediate results to form a single table, and by keeping only the rows related to cancer (green colored).

2.1 Architecture

Figure 2 illustrates SwellDB's architecture, which consists of the following components:

Catalog. The catalog keeps track of the registered tables to the system and their metadata. For each table, the catalog stores its description as a natural language prompt, and a set of attributes with their descriptions.

Table Generator. The Table Generator is the backbone of SwellDB, consisting of the *LLM API* and the *Table Planner*. The LLM API, implemented using LangChain[1], integrates seamlessly with widely used LLMs such as DeepSeek[2], OpenAI's GPT[3] and Ollama[4]. The Table Planner constructs a generation plan for each table definition. This plan outlines the steps to create the table, interacting with the LLM as needed. For instance, the planner may query the LLM to determine whether a specific column can be generated from input datasets or requires direct LLM computation. The final table is constructed by executing the plan using the LLM for individual operations and the query engine for integration.

Query Engine. The query engine supports SwellDB in two key ways. First, it acts as the primary interface for executing SQL queries. Second, it integrates intermediate results of the table generation pipeline into a unified table. Intermediate results are represented as PyArrow Tables, and the query engine uses SQL operations, such as joins, to blend these results into the final table. SwellDB's query engine uses Apache DataFusion[5], an embedded, in-memory query engine. However, we provide a more abstract query engine API for easy integration with more DataFrame-compatible query engines, such as DuckDB [6] and Spark SQL [7].

Data Source API. The Data Source API allows developers to add custom drivers for integrating external data sources with SwellDB. It supports two types of data sources.

Unstructured Data Sources. These ingest diverse formats, such as text or HTML/XML, and rely on the LLM to transform data into structured PyArrow Tables for integration.

Structured Data Sources. These require predefined schemas (e.g., database systems, CSV, or Parquet). Structured sources are more efficient, as the planner can directly generate SQL queries for data extraction, bypassing additional transformation steps by the LLM. A structured data source is loaded in SwellDB's backend query engine such that it can be queried using SQL.

2.2 Table Generation

The table generation process is the cornerstone of SwellDB. The table generation is split into smaller operations by the table planner. Each plan operator is responsible for creating a subset of the table columns using the LLM and the input data sources. The results of all operators are PyArrow Tables that share one or more common keys, referred to as *base columns*. In the final step, the individual outputs are joined using those columns, defined as a parameter named *base_columns*.

Planning. Once a table is defined, the table planner analyzes the table definition (prompt and attributes) along with the connected input data sources (if any). The output of this step is a table generation plan that describes *which* and *how* the input operators and data sources will be used to synthesize the final table. Algorithm 1 represents a simplified version of SwellDB's planner. The input to the planner is a logical table definition $L = (P, A)$ where P the prompt and $A = \{(a_1, d_1), (a_2, d_2), .., (a_n, d_n)\}$ the set of attributes and their descriptions, and a list of operators O (e.g. *LLMTable*, *DatasetTable* or *SearchTable*). The output table plan will consist of one or more of these operators. The planner will iterate through all the given operators. For each operator $op \in O$, the planner will generate a prompt to ask the LLM which columns in $L.A$ can be generated by operator op. The resulting columns are stored in the *op_cols* variable. Finally, an instance of op is created with the current *root* as its child, and *op_cols* as its assigned columns. Once we reach the last operator, we assign it all the remaining columns.

Materialization. This step executes the table generation plan obtained by the previous step, in order to create the final *table materialization*, which contains the actual rows. In the final step all operators are materialized as PyArrow Tables [8], and they are integrated using the *join* method on the provided *base_column* keys.

2.3 DataFrame Interface

SwellDB provides a Python interface that is inspired by DataFrames. We use Apache Arrow as our object representation format for easy integration with most frameworks of the same ecosystem. While

[1] https://www.langchain.com/
[2] https://www.deepseek.com/
[3] https://openai.com/
[4] https://ollama.com/
[5] https://datafusion.apache.org/
[6] https://duckdb.org/
[7] https://spark.apache.org/sql/
[8] https://arrow.apache.org/docs/python/generated/pyarrow.Table.html

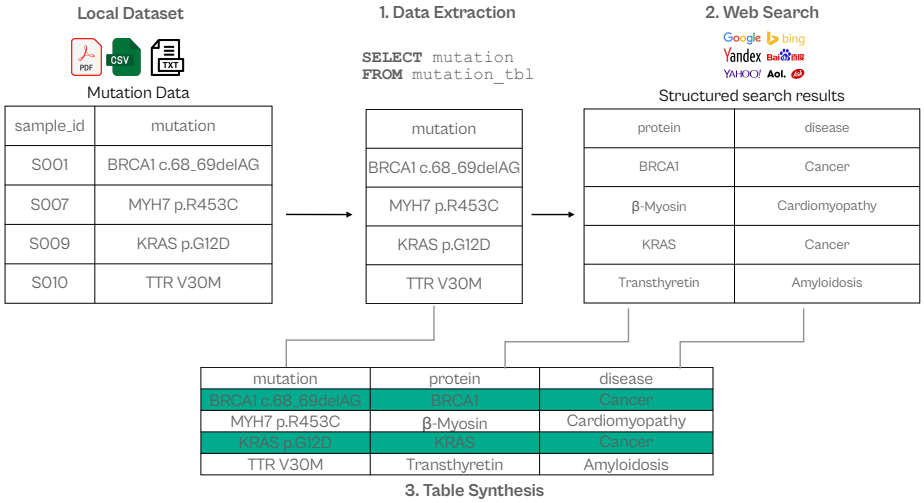

Figure 1: Table Generation Example

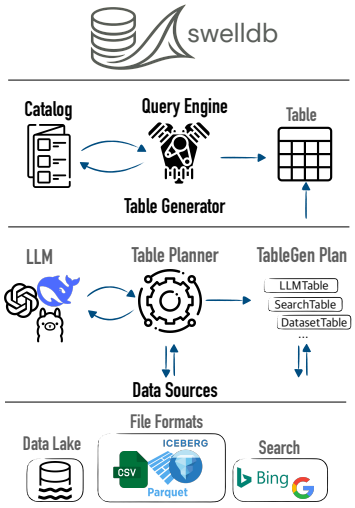

Figure 2: SwellDB Architecture

Algorithm 1: SwellDB's Planner

```
1  Function PlanTable(T, O):
       /* L: The logical table definition                    */
       /* O: The input operators                             */
2      root = None
3      while !O.isEmpty() do
4          op = O.pop()
           /* Generate a prompt that asks the LLM which columns
              op can handle                                  */
5          cols_prompt = op.get_prompt(L)
6          if !O.isEmpty() then
7              op_cols = llm(cols_prompt)
8              L_A = L_A − op_cols
9          else
               /* If op is the last operator, assign it all the
                  remaining columns                          */
10             op_cols = L_A
           /* Create an instance of the operator             */
11         root = op.init(cols = op_cols, child = root)
12     return root
```

all of the intermediate results as well as the final table are represented as PyArrow Tables, the integration with other DataFrame-based frameworks is straightforward since the *Table* class provides a *to_pandas*() method which converts the table to a Pandas DataFrame. As a result, SwellDB tables can be directly integrated with any such framework. Figure 1 depicts an example in which SwellDB is used to generate a table that contains all of the U.S. states on the fly, using less than 10 lines of code.

3 RELATED WORK

Large Language Models (LLMs) have revolutionized data management, opening new research directions for integrating LLMs into database systems. Several works have explored how LLMs can be efficiently incorporated into query execution, with a predominant focus on integrating declarative languages with LLM invocations [7] for hybrid querying, as seen in systems like SWAN [11], ThalamusDB [5], BlendSQL [4], Palimpzest [6], and LOTUS [10]. These systems optimize declarative querying by supporting tasks such as data augmentation (e.g., generating new columns) and enabling semantic operations (e.g., joins and filters with predicates defined in natural language). However, they operate on pre-existing structured datasets. Galois [8] introduces an alternative paradigm where LLMs function as a storage layer for SQL execution. While this eliminates

```python
from swelldb import SwellDB, OpenAILLM

swelldb: SwellDB = SwellDB(OpenAILLM(model="gpt-4o"))

table_builder = swelldb.table_builder()
table_builder.set_content("US States")
table_builder.set_schema("state_name str, region str")

tbl = table_builder.build()

# Explore the table generation plan
tbl.explain()

# Table plan
# LLMTable[schema=['state_name', 'region']

# Create the table
table = tbl.materialize()

# Output (top 5 rows)
    state_name     region
0      Alabama      South
1       Alaska       West
2      Arizona       West
3     Arkansas      South
4   California       West
```

Figure 3: Table generation code snippet

reliance on external databases, it remains constrained by the LLM's internal knowledge and lacks integration with external structured or unstructured data sources. Beyond-the-database querying has also been explored in prior crowdsourced systems like CrowdDB [2] and DECO [9], though these approaches depend on user-provided data rather than autonomous LLM-based extraction. Evaporate [1] further advances LLM-powered data retrieval by generating extraction scripts for heterogeneous data lakes. SwellDB distinguishes itself by introducing a dynamic table generation model that constructs relational structures on the fly, tailored to input SQL queries. Unlike systems that enhance existing tables, SwellDB synthesizes tables by intelligently selecting and blending data from multiple sources, including LLMs, structured databases, and web search results. This approach bridges the gap between structured query execution and unstructured data retrieval.

4 CONCLUSIONS AND FUTURE WORK

We presented SwellDB, the data system that generates tables at runtime. SwellDB introduces a paradigm shift from the closed-world to the open-world assumption, enabling SQL-based database systems to issue analytical queries over any data format, or requested table. Compared to previous approaches that attempt to integrate LLM invocations into a query engine, SwellDB is GenerativeAI-native, built from the ground up to fully leverage the latest advancements of LLMs. SwellDB is still at a very early stage, yet providing very promising results and addressing real-world data integration problems. Our next steps include the following.

Optimizer — Similarly to traditional data systems, SwellDB uses a table generation plan which consists of a pipeline of physical operators that have to be executed in a specific order to generate different parts of the requested logical table. This happens by splitting the initial logical table into smaller ones with disjoint column sets, and assigning each one to the proper physical operator. Since the intermediate results are merged using a join operator, it is clear that the way in which we split the logical table, as well as the order in which we execute these steps will have a significant effect on the performance, as well as the monetary cost of the table generation process. We are working towards a table generation planner to minimize the cost of this process.

Multi-LLM Table Generation — SwellDB performs several LLM invocations in different table generation steps, starting from the planning and optimization steps, to the critical path of its executor. We aim to make this component more modular and pick different LLMs for each task. For instance, simpler tasks could be performed using lightweight models that require fewer reasoning capabilities, while bigger models can be used for more complex generation tasks. This will allow SwellDB to minimize the table generation costs without compromising the output quality.

Data Source — We envision SwellDB as a data source for multiple state-of-the-art database systems that support pluggable storage, such as Postgres, MariaDB, Presto, or Spark SQL. We aim to develop an API that will allow the easy integration with such database systems in order to enable them to dynamically generate tables.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Award No. 2239326.

REFERENCES

[1] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language models enable simple systems for generating structured views of heterogeneous data lakes. *arXiv preprint arXiv:2304.09433* (2023).

[2] Amber Feng, Michael Franklin, Donald Kossmann, Tim Kraska, Samuel R Madden, Sukriti Ramesh, Andrew Wang, and Reynold Xin. 2011. Crowddb: Query processing with the vldb crowd. (2011).

[3] Victor Giannakouris and Immanuel Trummer. 2025. SwellDB: Dynamic Query-Driven Table Generation with Large Language Models. In *Companion of the 2025 International Conference on Management of Data*. 95–98.

[4] Parker Glenn, Parag Pravin Dakle, Liang Wang, and Preethi Raghavan. 2024. BlendSQL: A Scalable Dialect for Unifying Hybrid Question Answering in Relational Algebra. *arXiv preprint arXiv:2402.17882* (2024).

[5] Saehan Jo and Immanuel Trummer. 2024. Thalamusdb: Approximate query processing on multi-modal data. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–26.

[6] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, et al. [n.d.]. Palimpzest: Optimizing AI-Powered Analytics with Declarative Query Processing. ([n. d.]).

[7] Shu Liu, Asim Biswal, Audrey Cheng, Xiangxi Mo, Shiyi Cao, Joseph E Gonzalez, Ion Stoica, and Matei Zaharia. 2024. Optimizing llm queries in relational workloads. *arXiv preprint arXiv:2403.05821* (2024).

[8] Paolo Papotti. 2024. Large Language Models as Storage for SQL Querying. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 5657–5658.

[9] Aditya Ganesh Parameswaran, Hyunjung Park, Hector Garcia-Molina, Neoklis Polyzotis, and Jennifer Widom. 2012. Deco: declarative crowdsourcing. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. 1203–1212.

[10] Liana Patel, Siddharth Jha, Carlos Guestrin, and Matei Zaharia. 2024. Lotus: Enabling semantic queries with llms over tables of unstructured and structured data. *arXiv preprint arXiv:2407.11418* (2024).

[11] Fuheng Zhao, Divyakant Agrawal, and Amr El Abbadi. 2024. Hybrid Querying Over Relational Databases and Large Language Models. *arXiv preprint arXiv:2408.00884* (2024).