Graphing the Stars: Graph Queries in SiriusDB

Atharv Kirtikar
aakirtikar@cs.wisc.edu
University of Wisconsin-Madison

Elliott Weinshenker
eweinshenker@cs.wisc.edu
University of Wisconsin-Madison

Yayen Lin
lin383@wisc.edu
University of Wisconsin-Madison

Synopsis

SiriusDB has demonstrated strong performance in analytical workloads through GPU acceleration, yet it currently lacks support for spatial and graph operations that are becoming more critical for modern data-intensive applications. This project aims to extend Sirius with graph capabilities for native GPU execution. Rather than reimplementing spatial functionality from scratch, we will leverage DuckDB community extensions for query parsing, type handling, and logical planning. Our approach maintains Sirius as a composable execution engine, either integrating existing GPU-native spatial libraries or developing custom GPU kernels for critical operations.

Keywords

SiriusDB, OpenGIS, GQL Operators, OpenGIS, ST_Spatial, ST_GeomFromText, DuckDB

1 Introduction

SiriusDB has demonstrated exceptional performance in analytical workloads by leveraging GPU acceleration for high-throughput query processing and vectorized computation. However, many emerging data-intensive applications require understanding relationships between entities—an area more naturally expressed and computed through graph models. This proposal introduces native graph query capabilities within SiriusDB, integrating them directly into its existing GPU execution framework.

By implementing graph primitive operators to support algorithms such as shortest distance, we aim to extend these primitives toward more complex graph algorithms including shortest path and connected components. This approach unifies relational and graph processing under a single, GPU-optimized engine, eliminating the need to fall back to host databases like DuckDB for graph queries. It will further exploit GPU parallelism for graph traversal, enabling SiriusDB to provide standardized support for both relational and graph analytics.

As a drop-in accelerator, SiriusDB offers GPU residency for a wide range of analytical and relational query operations. Its integration with Substrait facilitates translation between logical query representations and physical GPU execution, leveraging CPU-based query parsing and plan generation. Currently, SiriusDB supports DuckDB and Doris as host databases—these act as fallback execution engines when GPU features are unavailable[3]. While this design reduces complexity, it introduces friction in data movement and coordination between systems. Additionally, distributed query execution relies on the host database's coordinator, which can become a performance bottleneck.

2 Methodology

Our goal is to extend SiriusDB's functionality with OpenGIS-compliant operators implemented natively on the GPU. Specifically, we aim to support the `ST_Distance` [1] and `ST_GeomFromText` [2] functions, currently implemented in the DuckDB ecosystem through the DuckPGQ and DuckDB Spatial extensions. These community-driven projects have already established the groundwork for integrating the GEOMETRY type and parsing graph-based data representations. To maintain SiriusDB's role as a composable execution engine, we will leverage their existing work to generate Substrait logical query plans. Where operators are unsupported, we will evaluate whether the required primitives can be mapped to existing GPU-native operators or, where necessary, develop custom kernels to extend SiriusDB's graph processing capabilities.

3 Timeline

Table 1 provides an overview of our project development timeline.

Table 1: Project Timeline

Phase	Date	Goal
1	Weeks 1–3	System Setup and Preparation
2	Weeks 4–7	Implementation and Integration
3	Weeks 8–10	Evaluation and Documentation

3.1 Phase 1: System Setup and Preparation

During this phase (Oct 17–Nov 2), we will explore the Sirius and DuckDB codebases, focusing on the operators used by DuckDB and how those translate to supported operators in Sirius. We will test several graph workloads with the aforementioned DuckDB extensions and configure access to GPU resources for Sirius.

3.2 Phase 2: Implementation and Integration

During this phase (Nov 8–Dec 5), we will port the core spatial functions `ST_Distance`, `ST_GeomFromText` to libcudf geometry primitives. We will integrate these spatial functions into the Sirius operator pipeline and test end-to-end execution (DuckDB spatial query → Substrait plan → Sirius GPU execution). Additionally, we will examine DuckPGQ's Substrait plan output for simple graph queries (e.g., shortest path) and identify specific operators or patterns that Sirius cannot currently handle.

3.3 Phase 3: Evaluation and Documentation

During this phase (Dec 6–Dec 14), we will conduct performance evaluation, analyze our findings, and complete final documentation. We will assess the feasibility of full graph query GPU execution, identify future work and remaining technical challenges, and write a comprehensive project report with methodology, results, and analysis.

References

[1] DuckDB Labs. 2025. DuckPGQ Extension. https://duckdb.org/community_extensions/extensions/duckpgq. Accessed: 2025-10-17.

[2] Max Gabrielsson. 2023. Spatial Extension Rework: Migrate to GDAL. https://duckdb.org/2023/04/28/spatial. Accessed: 2025-10-17.

[3] Bobbi Yogatama, Yifei Yang, Kevin Kristensen, Devesh Sarda, Abigale Kim, Adrian Cockcroft, Yu Teng, Joshua Patterson, Gregory Kimball, Wes McKinney, Weiwei Gong, and Xiangyao Yu. 2026. Rethinking Analytical Processing in the GPU Era. In *Proceedings of the 16th Conference on Innovative Data Systems Research* (Chaminade, USA) *(CIDR '26)*. https://arxiv.org/abs/2508.04701