Project Ideas

CS 764, Fall 2025

This document contains a list of potential project topics. You are encouraged to explore your own ideas that interest you, which can be related to your own research.

1. Contributed to Sirius (https://github.com/sirius-db/sirius), an open-source GPU database project. [website][paper]

- Run more benchmarks on Sirius: JOB, TPC-DS; report and fix bugs
- **Help Sirius become a DuckDB community extension**
- Support reading Parquet (with and without compression)
- Support reading data from S3
- Support more operators (e.g., window functions, intersection, union, etc.)
- Support variable length data types (e.g., array, list, struct, map)
- Optimize string, top-K, **regex (e.g., fallback to DuckDB)** performance
- **Support vector data type in Sirius**
- Support compression for integer, floating point, and string
- **Run graph queries on Sirius (e.g., shortest path)**

2. Extend and optimize predicate transfer [PT paper][distr. PT][robust PT]

- Enhance PT performance for cyclic queries
- Evaluate PT in latest DuckDB version
- Evaluate PT for more workloads
- Dynamically tune Bloom filter size
- More advanced pruning techniques
- Study PT's effect for reducing intermediate data size
- Study PT with workloads that do not fit in memory

3. Scalable Distributed Snapshot Isolation. **Snapshot Isolation** (SI) is a widely adopted isolation level. It is challenging to implement SI in a distributed environment (DSI). Existing solutions rely on a centralized coordinator for snapshot assignment and conflict resolution, creating a central bottleneck. In this work, we aim to propose a novel DSI protocol that eliminates the need for centralized coordination. You will work with a PhD student in the DB group if you pick this project. Please contact the instructor if you are interested.

- Survey Existing Approaches: Select a state-of-the-art DSI protocol as a baseline.
- Design new DSI: Finish a design to remove centralized coordination and still guarantee SI
- A sketch approach: We can allow transactions to execute optimistically using local snapshots in each node and validate consistency collectively across all participating nodes during the commit phase (2PC)
- Implementation: Choose a base system (a distributed database) and implement both the baseline and proposed protocol.
- Evaluation: Compare performance using OLTP benchmarks (e.g., YCSB, TPC-C).
- Fallback Plan: If no good design is proposed, the project can fall back to re-implement existing solutions and evaluate their performance.

4. Optimize key-value store for storage disaggregation. Traditional key-value store rewrites the full value for each update operation, even if only a small portion of the value is modified. This leads to IO amplification, which is especially problematic in a storage disaggregation architecture. In this work, we propose the Key-Delta-Value (KDV) store, that stores deltas (data modifications) for updates from the compute layer. The storage layer can reconstruct the record by merging deltas. You will work with a PhD student in the DB group if you pick this project. Please contact the instructor if you are interested.

- Survey Existing Approaches for storing deltas for updates.
- Design KDV: Design read/write path for KDV store (e.g., how to reconstruct the full value/how to calculate delta). Design an approach to reduce read overhead
- 3. Implementation: Implement the KDV store
- 4. Evaluation: Compare read/write performance with the KV store using benchmarks like YCSB.

5. Genomic database. Modern genomic and biomedical workflows are gaining importance but the existing analytics tools fall behind on performance, scalability, usability, etc. Our long-term goal is to modernize genomic data analytics by leveraging decades of research and development in relational data analytics. Specifically, we have the following three sub-projects. You will work with a student in the DB group if you pick this project. Please contact the instructor if you are interested.

Characterizing genomic workloads

- Conduct an analysis on the kind of workloads that are prevalent in tertiary genomic analysis.
- Understand if current RDBMs are good/bad at genomic operators and what they would need to be better.
- **Output:** A report. Experiments run on existing database systems will be useful, e.g., output of explain, analyze on Postgres. This can also lead to a potential benchmark workload/paper since none exists in this field as of now.

Making Postgres OLAP friendly

- Tuning Postgres for read-heavy or even read-only workloads, potentially by reducing MVCC limitations.
- Identify existing "knobs" in Postgres that may facilitate this. One example is decreasing the frequency of WAL (we can probably get rid of it entirely) and checkpointing which helps with bulk loading.
- Change source code. Each of these small ideas might require a project:
    - Getting rid of visibility checks in query execution.
    - Getting rid of tuple headers required for MVCC.
    - Getting rid of snapshot acquisition, lock contention code, clog etc.
    - I/O optimizations such as larger page size. Huge pages etc.

Genomic analytics on DuckDB

- Support reading common genomic data formats in DuckDB
- Support several genomic operators in DuckDB
- Experiment with data compression techniques for genomic data
- Investigate potential index structures that can accelerate these queries