Information Systems, Analysis and Design
Class Projects (2025-26, Fall Semester)

1. Comparison between Vector DataBases

In this project, you will be asked to compare the performance of various aspects of two popular and open-source vector database systems. This entails the following aspects that must be tackled by you:

1) **Installation and setup of two specific Vector DBs:** Using local or okeanos-based resources you are asked to successfully install and setup the two systems. This also means that if any (or both) of the DBs got a cluster edition (distributed mode) that this will be also available for testing.
2) **Data generation (or discovery of real data) and loading to the two DBs:** Using either a specific data generator, online data or artificially creating data, you should identify and load a significant amount of vectors into the databases. Ideally, loaded data should be: a) big (or as big as possible), not able to fit in main memory, b) have varying dimensionality (vector size and type - integer, float) and c) the same data loaded in both databases. Data loading is a process that should be monitored, namely: the time it takes to load a small, medium and the final amount of data, as well as the storage space it takes in each of the databases (i.e., how efficient data compression or possible indexing is).
3) **Query generation to measure performance:** A set of *similarity queries* (common to both DBs) must be compiled in order to test the performance of the vector DBs. Queries should target similar similarity metric algorithms (if implemented on both DBs, e.g., Euclidean distance (L2), Inner product, Cosine similarity, etc.) with and without filters (https://github.com/qdrant/ann-filtering-benchmark-datasets)
4) **Measurement of relevant performance metrics for direct comparison:** Client process(es) should pose the queries of the previous step and measure the DB performance (query latency, throughput, CPU load if possible, etc). Teams should be careful and compare meaningful statistics in this important step.

Teams undertaking this project can take advantage of existing benchmarking code either in principle or in whole. I suggest looking at the qdrant Benchmark Suite (https://github.com/qdrant/vector-db-benchmark), which for many of the DBs contains code for data generation, loading, queries and measurement.
Besides the aforementioned project aspects, you are free to improvise in order to best demonstrate the relative strengths and weaknesses of each system. By the end of your project, you should have a pretty good idea of what a modern vector DB is, its data and query processing model and convey their strong/weak points.
This project has 3 different DB combinations:

Vec1: qdrant, milvus

Vec2: milvus, weaviate

Vec3: chroma, pgvector

Also possible: Marqo, pinecone

2. Comparison between text-to-SQL methods

With the advent of Large Language models (LLMs), the task of translating natural language into SQL (text-to-SQL) has gained significant attention in the Database community. LLMs such as Qwen, Tiny Llama, GPT, etc., have shown superior performance in converting text into SQL queries. However, for complex queries the performance tends to drop. As part of this work, you are asked to compare how LLMs perform when generating both simple and complex SQL queries from a text for a Relational Database Management System (RDBMS). Additionally, you are required to develop and scale code to create an agent that integrates LLMs with an RDBMS and compare the performance across different types of queries. This entails the following aspects that must be tackled by you:

1) Installation and setup of two specific systems: Using local or cloud resources you are asked to successfully install and setup the LMM environment and the RDBMS environment.
2) Data Generation: Using existing online datasets for text-to-SQL and creating your own with more complex SQL queries, you should identify the data that will be used for testing the system.
3) Code for measuring performance: Using (python) scripts you must implement a small framework that acts as an agent for natural language input, generated SQL queries, and execution on the RDBMS. The results produced by the system should then be used to compare the performance of different LLMs in generating queries. Performance must be evaluated in terms of accuracy (correctness of generated queries) and computational efficiency (time and resources used). Each team is encouraged to go beyond the minimum requirements by designing additional experiments, such as introducing multiple levels of query complexity. In this step, teams should ensure that comparisons are based on meaningful and statistically valid results.

Teams undertaking this project must use at least two RDBMSs. Similarly, each team should evaluate at least two LLM models, such as Qwen, Tiny-Llama,GPT. Teams are free to explore additional models beyond these examples.Suggested datasets are included in the following link: https://github.com/jkkummerfeld/text2sql-data. Each team must perform experiments using at least three datasets, but they are encouraged to include more if possible from different repositories. Besides the aforementioned project aspects, you are free to improvise.

Note: We recommend running (some of) the experiments on Google Colab, Kaggle, or any other notebook platform that provides free GPU resources. Once the work is completed, we encourage teams to back up and save their results. This project has the following different combinations:

LLMSQL1: Qwen, Tiny-Llamma, MariaDB, PostgreSQL

LLMSQL2: GPT, Tiny-Llamma, SQLite, PostgreSQL

LLMSQL3: GPT, Qwen, MySQL, MariaDB

3. Distributed Execution of SQL Queries

Trino https://trino.io/ is a distributed SQL query engine designed to query large data sets distributed over one or more heterogeneous data sources. In this project, you will be asked to benchmark their performance over different types of queries, data locations and underlying storage technologies. In more detail, the following aspects must be tackled by you:

1) **Installation and setup of three specific stores:** Using local or okeanos-based resources you are asked to successfully install and setup three open-source storage systems or Databases and Trino. These will be used as the source of data for distributed execution of SQL queries.
2) **Data generation and loading:** Using a specific data generator, you should identify and load a significant amount of tuples into the three databases. Loaded data should be: a) big (or as big as possible), not able to fit in main memory, in the order of several tens of GB, b) Different tables of the data should be stored in different stores, according to a strategy that you will devise. This strategy will entail different ways of distributing tables so that you will be able to measure how Trino behaves.
3) **Query generation to measure performance:** A set of queries must be selected in order to test the performance of Trino. Queries should be diverse enough to include both simple queries (e.g., simple selects) and complex ones (range queries and multiple joins and aggregations) and cover some or all the input tables.
4) **Measurement of performance:** You should then pose the queries of the previous step and measure the performance (query latency, optimizer plan) over different cluster resources. This means that each of the queries of the previous step should be posed over i) a different number of workers, ii) a different data distribution plan. Our goal is to identify how the distributed execution engine and optimization works under a varying amount of data, data distributed in different engines and with queries of different difficulty.

Teams undertaking this project should take advantage of existing benchmarking code, namely the TPC-DS benchmark (https://www.tpc.org/tpcds/), which contains code for data generation and relevant queries.
Besides the aforementioned project aspects, you are free to improvise in order to best understand the performance of different queries. By the end of your project, you should have a pretty good idea of how query processing in Trino works and

propose better data distribution strategies. This project has 3 different combinations (using Trino for each of the following):

SQL1: MariaDB, Cassandra, Ignite

SQL2: PostgreSQL, MongoDB, ElasticSearch

4. Evaluation of Graph Embedding Methods on Real-World Datasets

Graph embedding methods map entire graphs into fixed-length vector representations that capture structural and semantic information. With such representations we can do tasks like graph classification, clustering, and visualization. Modern whole-graph embedding algorithms include both unsupervised approaches (e.g., Graph2Vec, NetLSD, FGSD) and supervised deep learning models (e.g., Graph Isomorphism Networks – GIN, InfoGraph). This project plans to benchmark several representative embedding techniques on real-world graph datasets, analyzing their relative performance, scalability, and robustness. In more detail, the following aspects must be tackled by you:

1) Implement or integrate existing open-source implementations using Python libraries such as KarateClub (for Graph2Vec, NetLSD, FGSD and FeatherGraph) and Pytorch Geometric (for GIN and InfoGraph). Ensure all methods produce embeddings of comparable dimensionality for fair evaluation.
2) Use standard benchmark graph classification datasets from the TUDataset or Open Graph Benchmark (OGB) collections:
    - MUTAG
    - ENZYMES
    - IMDB-MULTI
    - REDDIT-MULTI-12K
    - NCI1
3) Evaluate performance on four different tasks:
    a) Classification
        i) For the unsupervised methods, use embeddings as input features to train a simple classifier (SVM, MLP, etc.) for each dataset.
        ii) Report classification accuracy, F1-score, AUC on test data.
        iii) Record training time, generation time, memory use for each method.
        iv) Vary embedding dimensions to analyze accuracy vs. compute cost.
    b) Clustering
        i) k-means and/or spectral clustering to the embeddings.
        ii) Report ARI, and qualitative t-SNE/UMAP visualizations.
        iii) Which embeddings yield the clearest cluster separation?

c) Stability Analysis
   i) introduce random perturbations (add/remove % of edges, shuffle node attributes).
   ii) recompute embeddings. How much do they change w.r.t. the original?
   iii) report embedding stability score, change in classification accuracy.

Graph embedding methods:

https://karateclub.readthedocs.io/en/latest/modules/root.html#whole-graph-embedding
https://github.com/benedekrozemberczki/graph2vec
https://github.com/xgfs/NetLSD
https://github.com/sunfanyunn/InfoGraph
https://pytorch-geometric.readthedocs.io/en/2.5.0/generated/torch_geometric.nn.models.GIN.html

Datasets:

https://chrsmrrs.github.io/datasets/docs/datasets/

This project has 3 different combinations:

EMB1: Graph2Vec, NetLSD, GIN over MUTAG - ENZYMES - IMDB-MULTI

EMB2: NetLSD, FGSD, InfoGraph over REDDIT-MULTI-12K - NCI1 - MUTAG

EMB3: Graph2Vec, FGSD, GIN over ENZYMES- IMDB-MULTI - REDDIT-MULTI-12K

Small datasets - MUTAG, ENZYMES, IMDB-MULTI
Large datasets - REDDIT-MULTI-12K, NCI1

For the GNNs (GIN, InfoGraph), a GPU is needed for training. Using Collab's free tier GPU (T4) is enough. Expect some minutes' time for the large datasets for training. (i.e., GIN model for 80 epochs takes around 5 mins)

Very helpful collab templates:
https://pytorch-geometric.readthedocs.io/en/latest/get_started/colabs.html