University of Zurich UZH

**Department of Informatics**

University of Zürich
Department of Informatics
Binzmühlestr. 14
CH-8050 Zürich
Phone. +41 44 635 43 11
Fax +41 44 635 68 09
www.ifi.uzh.ch/dbtg

UZH, Dept. of Informatics, Binzmühlestr. 14, CH-8050 Zürich

**Prof. Dr. Michael Böhlen**
Professor
Phone +41 44 635 43 33
Fax +41 44 635 68 09
boehlen@ifi.uzh.ch

Zürich, 28. November 2025

**Master Project (15 ECTS)**
**Database Technology**

## Topic: Integrating Vector Similarity Join Computations into MonetDB

A machine learning pipeline includes three core phases: preprocessing, learning, and inference. Preprocessing requires a sequence of data cleaning steps to extract, combine and transform data of interest from different database and/or tables. Data normalization, tokenization and embedding are next to fit the data to the learning algorithm [3]. Learning includes iterative numerical computations to update weights. For different tasks the concrete computations vary. But the core is to make data storage and access pattern fit the computations and minimize the overhead of data copy/movement/transformation. Inference leverages weights to make decision. For example, the learned weight vectors are used to find the top-k most similar weight vectors for a recommendation/translation task. Inference usually needs to be accelerated with an index and handle concurrent queries. Numerous studies [1] have demonstrated that database systems with efficient data processing and indexing techniques offer significant benefits for inference.

The goal of this Master Project is to extend MonetDB with relational operations that support vector search, implement index-based inference of the solution, and evaluate the performance and applicability.

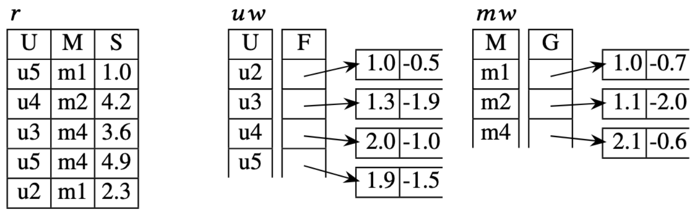
*Example:*

As a running example we use a database with a relation $r$ with schema *Rating(User, Movie, Score)*.

The training process uses stochastic gradient descent to factorize $r$ into relations $uw$ and $mw$.

In a word, for every user and movie shown in attributes U and M in relation r, the training algorithm learns vector representations for these users and movies that capture the preferences expressed by attribute S in relation r.

For our illustrations we use $d = 2$.



| U | M | S |
|---|---|---|
| u5 | m1 | 1.0 |
| u4 | m2 | 4.2 |
| u3 | m4 | 3.6 |
| u5 | m4 | 4.9 |
| u2 | m1 | 2.3 |

Our goal is to implement vector similarity searches over relations $uw$ and $mw$ to e.g. query the most similar users and movies according to the weights of the vectors. Example queries with possible SQL formulations are the following:

Similarity search: e.g. for user u2, return the top 5 movies for which the similarity (measured by the dot product) between their vector representations is largest.

```
SELECT M
FROM uw, mw
WHERE U=u2
ORDER BY dot(F,G) DESC
LIMIT 5;
```

Similarity join: return all user/movie pairs where the similarity (measured by the dot product) between their vector representations is larger than 3.

```
SELECT U,M
FROM uw, mw
WHERE dot(F,G) > 3;
```

For example, dot product and cosine similarity between vector $u2 = [1.0, -0.5]$ and $m4 = [2.1, -0.6]$ are $2.4$ and $0.982872$. For a nested join implementation, all vectors in $F$ will dot product (or compute cosine similarity) with vectors in $G$ in pairs. And it will return the satisfied users and movies. The computation can benefit from data locality and parallelism.

Nested loop join is not efficient if the the number of users and movies grow. Existing approaches for faster similarity join leverage indexes. Or we can trade reasonable precision and recall for efficiency with approximate similarity join, which are selection-based or a further data compression of vectors [1, 2].

The tasks of the Master Project are described below. At the end a carefully worked out report that describes the solutions must be handed in.

**Task 1**: Implementation of SIMILARITY JOIN (nested loop join)

- familiarize yourself with the architecture of MonetDB and the deployment of the system.
- extend SQL syntax and symbol tree (sql_parser.y) so that the system recognize the new syntax for similarity queries
- extend the relevant part of relation tree (rel_select.c) and statement list (rel_bin.c) to context the similarity queries to executable functions
- extend the execution backend (batcalc.c, gdk_calc.c, bat_extension.c) to implement the nested loop similarity queries
- evaluate the performance of your implementation

**Task 2** Evaluating and improving nested loop join with data compression techniques

- before the similarity queries, perform data compression (e.g. PCA, quantization) on all vector
- first we perform the similarity search over the d-reduced vectors to get a small number candidates, then consider the original vectors of these candidates to get the result.
- Evaluate the performance and the precision for different hyperparameters

**Task 3** Evaluating multi-threading potentials of techniques (optional)

- utilize $pthread$ for multi-threading to make the system fully leverage modern hardware in the process of data compression and query process
- Evaluate the performance and the communication overhead for different hyperparameters.

## References

[1] Sun Ji, Li Guoliang, Pan James, Wang Jiang, Xie Yongqing, Liu Ruicheng, and Nie Wen. Gaussdb-vector: A large-scale persistent real-time vector database for llm applications. *PVLDB, 18(12): 4951 - 4963*, 2025.

[2] Xie Jiadong, Xu Yu Jeffrey, and Liu Yingfan. Fast approximate similarity join in vector databases. *ACM Manag. Data 3, 3 (SIGMOD), Article 158*, 2025.

[3] Tim Kraska, Ameet Talwalkar, John C Duchi, Rean Griffith, Michael J Franklin, and Michael I Jordan. Mlbase: A distributed machine-learning system. In *Cidr*, volume 1, pages 2–1, 2013.

**Supervisor:** Xinyu Zhu (`xinyu.zhu@uzh.ch`)

**Start date:** December 1, 2025

**End date:** May 31, 2026

University of Zurich
Department of Informatics

Prof. Dr. Michael Böhlen
Professor