The Power of SQL Lambda Functions

Maximilian E. Schüle
schuele@in.tum.de

Dimitri Vorona
vorona@in.tum.de

Linnea Passing
passing@in.tum.de

Harald Lang
harald.lang@in.tum.de

Alfons Kemper
kemper@in.tum.de

Stephan Günnemann
guennemann@in.tum.de

Thomas Neumann
neumann@in.tum.de

Technical University of Munich

ABSTRACT

This work demonstrates a wide range of applications that use lambda expressions in SQL. Such injected code snippets form a useful technique required by data mining algorithms to overcome the inflexibility of the SQL language, as the language is limited to predefined aggregations only. Following the ’move computation to the data’ paradigm, we extend SQL lambda functions—also known from common programming languages—for machine-learning tasks.

As machine-learning relies mostly on gradient descent and tensor data types, we use lambda expressions for clustering and graph-mining algorithms as well as to formulate loss functions and label data. To underline the flexibility gained in SQL, this work demonstrates a main memory database system with integrated lambda expressions accessible through table functions in SQL. By reusing SQL and performing data mining and machine-learning tasks faster than can dedicated tools, this demonstration aims at convincing data scientists of the capabilities of database systems for computational tasks.

1 INTRODUCTION

Database systems are commonly used for the initial analysis of data, whereas data analysis happens in specialised systems, covering the expensive Export, Transform, Load (ETL) process. In consequence, database systems are often underused as a storage system only, ignoring the advantages coming along with SQL as the declarative query language and the benefits of database systems as providing index structures for fast data retrieval.

Actually, data mining blends well with SQL: Using the same data types, we need only minor adjustments as iterations in SQL or injected code snippets. MADlib [3] proposes a data mining library for extending database systems such as PostgreSQL and Greenplum. The extension uses algorithms that are programmed in Python and can be used as table functions in SQL. Modern main memory database systems try to integrate data mining algorithms in the code generation phase. EmptyHeaded [1] generates and combines code out of special algorithms and relational algebra, whereas HyPer [5] provides specialised operators for data mining tasks. The latter provides flexibility for data mining by so-called SQL lambda functions, which allow for modifications to existing database operators by injecting user-defined code.

Although modern database systems provide data mining algorithms, machine-learning functionalities are still not covered.

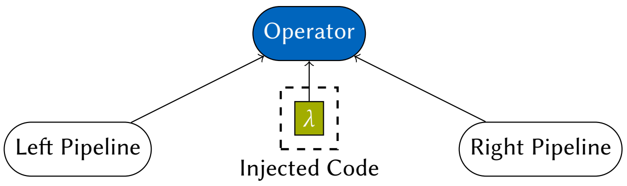


Figure 1: Lambda expression for code injection into an existing operator: The expression can be used inside unary operators to define functions on different tuples of the same relation and inside binary operators to combine tuples of different relations.

As machine-learning tasks often rely on tensors and gradient descent, providing an additional tensor data type and flexible loss functions for gradient descent contribute to in-database machine-learning.

Let us shift the boundary between database systems and the specialised tools to save ETL costs and to enjoy the benefits of database systems a bit more throughout the process. We argue that lambda functions can be adapted for machine-learning tasks and extend SQL to a powerful query language for machine-learning. This results in a computational engine, which merits the full benefits of database systems and can be addressed in SQL.

This work demonstrates the first database system to incorporate lambda expressions for minimisation problems. An extended HyPer is presented in detail. The main memory database system developed at the chair of database systems at TU Munich is already familiar with flexible clustering algorithms. It is now extended by flexible PageRank, gradient descent and labelling algorithms for machine-learning. A gradient descent operator and the integrated tensor data type allow supervised machine-learning tasks. As lambda expressions are part of the database system’s core, the query optimiser implicitly performs optimisations, such as predicate push-down, and reduces unnecessary overhead.

The remainder of this paper is structured as follows. First, the paper recaps lambda expressions and shows how they extend database systems to a uniform tool for solving machine-learning tasks. In the evaluation section, we discuss the measurement of the performance of database systems with lambda functions in comparison to dedicated tools to show the competitiveness. Our demonstration scenario allows users to interact with the database system and to create self-defined lambda expressions, for example, to apply gradient descent, to label data or to measure distances in clustering algorithms. The impact of computational database systems—replacing dedicated machine-learning tools—is summarised in the section on benefits.

2 LAMBDA FUNCTIONS

Lambda functions originate from the lambda calculus invented in 1936 by Alonzo Church [2] and later adapted for programming languages to provide anonymous functions. For use in SQL, the concept of anonymous expressions is adapted to 'inject user-defined code' [4] into analytic operators. Lambda expressions allow a user-friendly way for data scientists—who are already familiar with SQL and lambda functions from other programming languages—to customise algorithms without any modifications of the operators in the database system's core. Another advantage of lambda functions is the implicit deduction of input and output data types from the input tables' attributes without requiring further specification.

Lambda expressions in HyPer have been used in clustering algorithms as distance metrics and in graph-mining algorithms, such as PageRank to specify the edges. As lambda functions allow 'variation points' [6] in inflexible data mining algorithms, their usage can be transferred to machine-learning algorithms to specify loss functions. Lambda expressions are composed as follows:

$$\lambda(< name1 >, < name2 >, \ldots)(< arithmeticexpression >).$$

In the lambda function's header, the arguments name the relations whose table attributes are used in the arithmetic expression itself. The number of expected arguments is hard-coded in the operators as well as in the relations referenced by the arguments. More precisely, in unary operators, all arguments reference one input pipeline, whereas, in binary operators, the references depend on a certain application. Internally, lambda expressions are treated as arithmetic expressions in HyPer like those in the projection operator. Therefore, all known functions supported by the referenced database types can be used inside the expression as long as the expression results in a single value. As HyPer compiles SQL queries, lambda expressions are precompiled to LLVM code and evaluated at runtime.

To demonstrate the use of lambda functions for data mining and machine-learning, we selected two tasks out of each domain where lambda expressions broaden the application area of database operators.

Clustering k-Means as a clustering algorithm assigns points to a predefined number of k centres, which are iteratively adjusted until the summed distance to each centre is minimised. Hereby, the distance function is given as a lambda expression to specify Manhattan (L^1) or Euclidean (L^2) distance.

Graph-mining PageRank is a graph-mining algorithm for gathering the importance of nodes by the number of incoming edges per node weighted by the score of the source node.

Optimisation Machine-learning algorithms rely on optimisation algorithms such as gradient descent. Given a model function parametrised by some weights, one aims to minimise the weights to obtain minimal loss in order to predict the targets/labels of the data and a loss function that measures the deviation from the true labels and their predictions. To allow user-defined loss functions, we adapt the concept of lambda expressions to work as a mathematical function for minimisation on the training set's attributes.

Labelling To label test data, an operator adds the result of a parametrised function to the input relation. Lambda expressions define the loss function to be evaluated on the test data set.

The algorithms for clustering and graph-mining are hard-coded as materialising pipeline breakers in the database system's core, as various iterations are needed to compute the clusters or the PageRank values. The operator for labelling—as part of the pipeline—simply evaluates the lambda expression and adds a new attribute to the input relation. The operator for gradient descent uses a self-developed framework for automatic differentiation based on *placeholders* for the input data and *variables* for the weights and can be designed as either materialising or non-materialising pipeline breaker. All operators define the usage of the lambda functions according to the number of input pipelines. As k-Means and PageRank only need one input pipeline, lambda expressions define the functions between the tuples of one input relation. The lambda expression for the distance metrics in k-Means takes two tuples S, T of the same relation $R\{[x, y]\}$ as argument (L^2 distance, for example):

$$\lambda(S,T)((S.x - T.x)^2 + (S.y - T.y)^2).$$

PageRank—contrary to the other operators—needs **two** lambda expressions to specify the edges: one for the source node and the second one for the destination node of the input relation $R\{[src, dst]\}$:

$$\lambda(R)(R.src), \lambda(R)(R.dst).$$

For gradient descent and labelling, we need data for the placeholders and weights for the variables. Therefore, the lambda expression takes one tuple from each of the two input relations $R\{[a, b]\}$ (initial weights) and $S\{[x, y]\}$ (training data) to formulate the loss function:

$$\lambda(R,S)(R.a * S.x + R.b - S.y)^2.$$

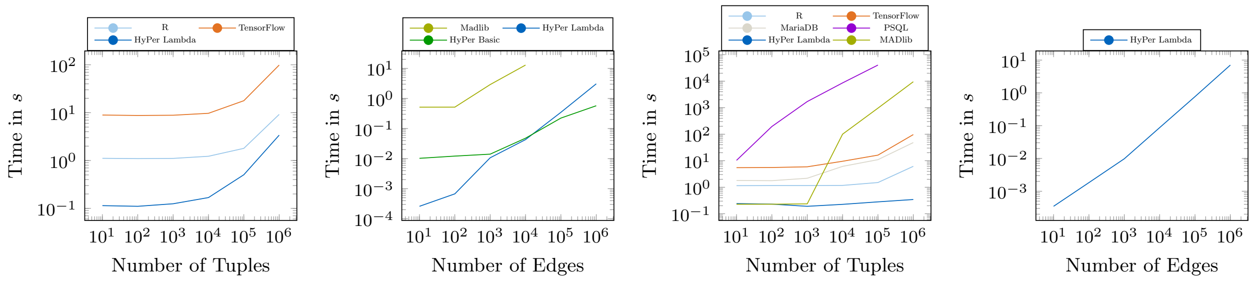
As lambda expressions are part of the database system, they are taken into account by the database's query optimiser. When the data is stored column-wise, only the relevant data for the lambda expression is loaded.

3 EVALUATION

The evaluation section presents the runtimes of the lambda-based operators. The experiments were run on an Intel Xeon E5-2660 v2 (20 cores of 2.20 GHz) server with 256 GB main DDR4 RAM running on Ubuntu 18.04 LTS. The test data was one-month excerpt of the Chicago taxi rides dataset[1] ($> 1.7 * 10^6$ tuples), on which we performed clustering with k-Means and logistic regression by gradient descent, and the LDBC data set with scale factor 10 for graph-mining with PageRank.

Using k-Means, we clustered the taxi dataset geographically by the trip's destination expecting 10 clusters. By logistic regression, we predicted the payment type depending on the distance of a trip and the ages of the customers. We reused the predicted weights to label the data. We take the *person-knows-person*-relationship from the LDBC benchmark to calculate the PageRank values of each person. Each test varied the input sizes to compare the runtime of our extended operators with the lambda functions in HyPer to the runtime of PostgreSQL 9.6.8 with its MADlib v1.13 extension

[1] https://data.cityofchicago.org/Transportation/Taxi-Trips/wrvz-psew



(a) k-Means (2 dims, 10 clusters). **(b) PageRank (100 iter., no damping).** **(c) Gradient Descent.** **(d) Labeling.**

Figure 2: Runtimes of the operators using lambda functions and of the competitors varying the number of input tuples.

(database system with a plugin), TensorFlow 1.3.0 without GPU support (dedicated machine-learning tool) and R 3.4.2 (statistical tool). The tests were run five times and the average runtimes were taken.

As expected, all the operators scaled linearly in the size of the input dataset and no overhead caused by lambda expressions in SQL was measurable. Figure 2a shows k-Means in comparison to R and TensorFlow (both use predefined library functions). Our database operator with the included lambda function outperformed all dedicated tools by at least a factor of three.

The in-database PageRank function provided by the MADlib library scaled linearly as well but was still four times slower than the integrated operator (see Figure 2b). The lambda expressions did not slow down the runtime of the database system as both operators performed at quite the same time.

The currently introduced gradient descent operator (see Figure 2c) reduced the overhead caused by data extraction and transfer as the data could be processed directly. The results were performance gains of at least five times in comparison to R, the fastest dedicated tool. The hard-coded gradient descent function in PostgreSQL and the equivalent procedure in MariaDB ran out of scope and the MADlib extension for logistic regression performed as fast as our lambda-based operator only for small input relations. Figure 2d shows the labelling of the attributes that performed linearly in the size of the input tuples.

In summary, the lambda expressions broadened the application area of the database systems without performance losses and eliminated the need for dedicated tools.

4 DEMONSTRATION

The demonstration convinces by the simplicity of using SQL for machine-learning: the central part is an extended web interface for HyPer with SQL for the input queries and a tabular output representation (see Figure 3). To facilitate the use of database systems for data scientists, data visualisation—comparable to those of business intelligence tools—enriches the output by simple dragging and dropping the table's attributes on the axes of different sorts of diagrams (pie/bar/line charts). As an additional feature, a box for specifying lambda functions with their input weights allows the plotting of lambda expressions as mathematical functions inside the diagrams.

The database system is fed with an excerpt of the Chicago taxi dataset and the current OpenFlights dataset[2]. The web interface lets users choose between predefined example queries with sample lambda expressions or lets them create their own queries. Our predefined examples cover supervised machine-learning tasks based on gradient descent (see Listing 3), labelling and data mining tasks such as clustering (see Listing 1) and PageRank (see Listing 2).

In our demonstration scenario, the user is encouraged to perform machine-learning tasks, such as the predefined regression to predict, for example, the number of tips given or the kind of payment in dependency of the length of a trip. In addition, the users can specify any kind of loss functions to perform predictions or other methods for the type of gradient descent.

To show the performance of the database systems, two charts—one for the runtime and one for the operator tree—provide informations about optimisations. This feature helps the user to understand how operator reordering and predicate push-down of integrated lambda expressions increase the performance.

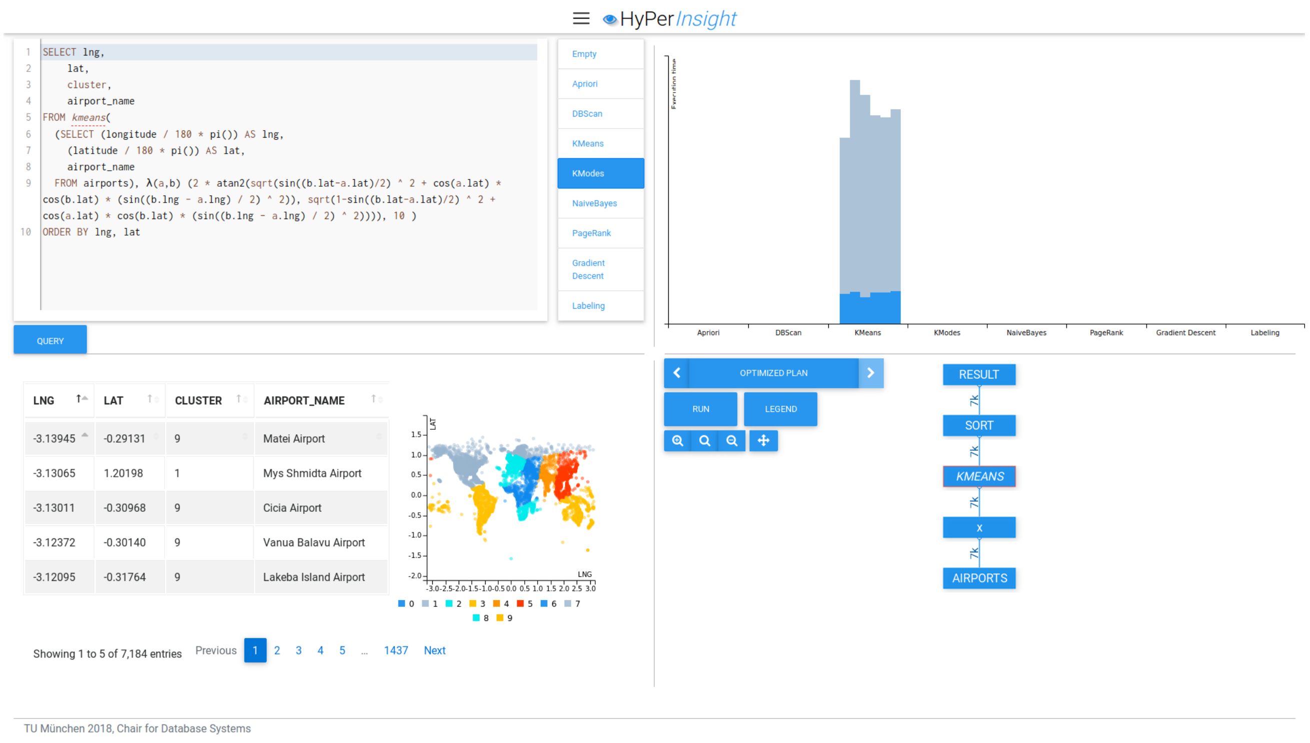
5 CONCLUSION

In this demonstration, we presented SQL lambda functions for in-database machine-learning and data mining with an interactive web interface devoted to its use by data scientists.

We proposed extending already known lambda expressions for use in machine-learning, especially to specify loss functions for gradient descent. By this extension, we used database systems with SQL as a universal computational engine, eliminated the need for dedicated machine-learning tools and reduced the time needed for data communication. To tackle the acceptance of SQL with lambda functions, we created a web interface that interactively combines SQL with data visualisation and provides informations about the optimised query plans.

This work aimed at adding lambda functions in standardised SQL to allow changing the database system as an underlying machine-learning tool. Lambda functions are essential for providing a higher-order machine-learning language to be used by data scientists. For that purpose, a declarative language is needed to further increase the acceptance of database systems and should be designed to be compiled to SQL or an executable to call the application interfaces of current dedicated tools.

[2]https://openflights.org/data.html



```
SELECT lng,
    lat,
    cluster,
    airport_name
FROM kmeans(
  (SELECT (longitude / 180 * pi()) AS lng,
    (latitude / 180 * pi()) AS lat,
    airport_name
  FROM airports), λ(a,b) (2 * atan2(sqrt(sin((b.lat-a.lat)/2) ^ 2 + cos(a.lat) *
cos(b.lat) * (sin((b.lng - a.lng) / 2) ^ 2)), sqrt(1-sin((b.lat-a.lat)/2) ^ 2 +
cos(a.lat) * cos(b.lat) * (sin((b.lng - a.lng) / 2) ^ 2)))), 10 )
ORDER BY lng, lat
```

LNG	LAT	CLUSTER	AIRPORT_NAME
-3.13945	-0.29131	9	Matei Airport
-3.13065	1.20198	1	Mys Shmidta Airport
-3.13011	-0.30968	9	Cicia Airport
-3.12372	-0.30140	9	Vanua Balavu Airport
-3.12095	-0.31764	9	Lakeba Island Airport

Showing 1 to 5 of 7,184 entries Previous 1 2 3 4 5 ... 1437 Next

Figure 3: We adapted our HyPerInsight web interface for demonstrating lambda functions. On the left side, we see the SQL interface with an exemplary lambda expression as a distance metric (top) with tabular output and visualisation (bottom). The right side shows the runtimes of the different algorithms (top) and the operator optimisations (bottom).

```
CREATE TABLE data(x FLOAT, y INTEGER);
CREATE TABLE center(x FLOAT, y INTEGER);
INSERT INTO ...

SELECT * FROM kmeans(
  (SELECT x,y FROM data),
  (SELECT x,y FROM center),
  -- the distance function
  λ(a, b) (a.x-b.x)^2+(a.y-b.y)^2,
  3 -- max. number of iterations
);
```

Listing 1: k-Means.

```
CREATE TABLE edges (a BIGINT, b BIGINT);
INSERT INTO ...

SELECT * FROM pagerank(
  (SELECT * FROM edges),
  λ(src) src.a, -- source
  λ(dst) dst.b, -- destination
  0.85,       -- damping factor
  0.00001,    -- threshold
  100         -- iterations
);
```

Listing 2: PageRank.

```
CREATE TABLE data (x FLOAT, y FLOAT);
CREATE TABLE weights(a FLOAT, b FLOAT);
INSERT INTO ...

SELECT * FROM gradientdescent(
  -- the loss function
  λ(d, w) (w.a*d.x+w.b-d.y)^2,
  -- training data set
  (SELECT x,y FROM data d),
  -- initial weights
  (SELECT a,b FROM weights w),
  0.05, -- learning rate
  100 -- max. number of iteration
);
```

Listing 3: Gradient descent.

Figure 4: Examples of using lambda functions in SQL.

ACKNOWLEDGEMENTS

This work is part of the TUM Living Lab Connected Mobility (TUM LLCM) project and has been funded by the Bavarian Ministry of Economic Affairs, Energy and Technology (StMWi) through the Centre Digitisation.Bavaria, an initiative of the Bavarian State Government. Linnea Passing and Dimitri Vorona have been sponsored in part by the German Federal Ministry of Education and Research (BMBF), grant TUM: 01IS12057. This research was supported by the German Research Foundation (DFG), grant NE 1677/1-1 and has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 725286).

REFERENCES

[1] C. R. Aberger, A. Lamb, K. Olukotun, and C. Ré. Mind the gap: Bridging multi-domain query workloads with emptyheaded. *PVLDB*, 10(12):1849–1852, 2017.
[2] A. Church. An unsolvable problem of elementary number theory. *American journal of mathematics*, 58(2):345–363, 1936.
[3] J. M. Hellerstein, C. Ré, F. Schoppmann, D. Z. Wang, E. Fratkin, A. Gorajek, K. S. Ng, C. Welton, X. Feng, K. Li, and A. Kumar. The madlib analytics library or MAD skills, the SQL. *PVLDB*, 5(12):1700–1711, 2012.
[4] N. Hubig, L. Passing, M. E. Schüle, D. Vorona, A. Kemper, and T. Neumann. Hyperinsight: Data exploration deep inside hyper. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, CIKM 2017, Singapore, November 06 - 10, 2017*, pages 2467–2470, 2017.
[5] A. Kemper and T. Neumann. Hyper: A hybrid oltp&olap main memory database system based on virtual memory snapshots. In *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, pages 195–206, 2011.
[6] L. Passing, M. Then, N. Hubig, H. Lang, M. Schreier, S. Günnemann, A. Kemper, and T. Neumann. SQL- and operator-centric data analytics in relational main-memory databases. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017.*, pages 84–95, 2017.