# HyPerInsight: Data Exploration Deep Inside HyPer

Nina Hubig*
hubig@in.tum.de

Linnea Passing
passing@in.tum.de

Maximilian E. Schüle
schuele@in.tum.de

Dimitri Vorona
vorona@in.tum.de

Alfons Kemper
kemper@in.tum.de

Thomas Neumann
neumann@in.tum.de

Technical University of Munich

## ABSTRACT

Nowadays we are drowning in data of various varieties. For all these mixed types and categories of data there exist even more different analysis approaches, often done in single hand-written solutions. We propose to extend HyPer, a main memory database system to a uniform data agent platform following the "one system fits all" approach for solving a wide variety of data analysis problems. We achieve this by applying a flexible operator concept to a set of various important data exploration algorithms. With that, HyPer solves analytical questions using clustering, classification, association rule mining and graph mining besides standard HTAP (Hybrid Transaction and Analytical Processing) workloads on the same database state. It enables to approach the full variety and volume of HTAP extended for data exploration (HTAPx), and only needs knowledge of already introduced SQL extensions that are automatically optimized by the database's standard optimizer. In this demo we will focus on the benefits and flexibility we create by using the SQL extensions for several well-known mining workloads. In our interactive webinterface for this project named *HyPerInsight* we demonstrate how HyPer outperforms the best open source competitor Apache Spark in common use cases in social media, geo-data, recommender systems and several other.

## CCS CONCEPTS

• **Information systems** **Query operators**; **Data mining**; *Structured Query Language*;

## KEYWORDS

HyPer, Database operators, Query processing, SQL, Apriori, k-Means, DBscan

**ACM Reference format:**
Nina Hubig, Linnea Passing, Maximilian E. Schüle, Dimitri Vorona, Alfons Kemper, and Thomas Neumann. 2017. HyPerInsight: Data Exploration Deep Inside HyPer. In *Proceedings of CIKM'17 , Singapore, Singapore, November 6–10, 2017,* 4 pages.
https://doi.org/10.1145/3132847.3133167

*First four authors in alphabetical order; they contributed equally to this work.


*CIKM'17 , November 6–10, 2017, Singapore, Singapore*

ACM ISBN 978-1-4503-4918-5/17/11…$15.00
https://doi.org/10.1145/3132847.3133167

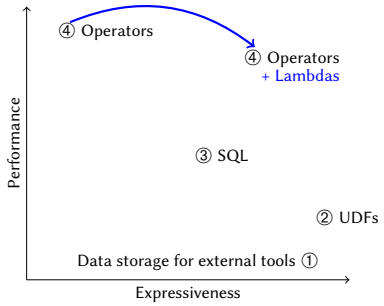


**Figure 1: Overview of approaches to data exploration using relational database systems. Our system supports the novel *layer 4*, where data mining is integrated directly into the database core, thus leading to higher performance. To maintain expressiveness, high-order functions (*lambdas*) can directly be passed as parameters to the database operators.**

## 1 INTRODUCTION

The emergence of huge data volumes is followed by a wide variety of data exploration methods and systems for mining specific data sets. We are extending HyPer from a *Hybrid Transaction and Analytical Processing (HTAP)* system that covers transactional and analytical workloads to *HTAP extended for data exploration (HTAPx)*. *HTAPx* includes data exploration algorithms and queries that process the whole dataset (or extensive subsets), and therefore are computation-intensive and long-running. Typical fields of application are machine learning, data mining, graph analytics, and text mining. Most algorithms boil down to a model-application approach, i.e., a two phase process where a model is created and stored first, and then applied to the model data in a second step. Standalone software systems like Spark [11] and Tupleware [3] which support these kind of algorithms use DBMSs as data source. However, a DBMS as a data storage only does not fully exploit its potential, and is hence impeding performance, expressiveness, and usability.

Classical RDBMSs provide an efficient and update-friendly data management layer and many more useful features to store big data reliably, such as user rights management and recovery procedures. Database systems like SAP HANA [5] and HyPer [6] are already designed to efficiently handle different HTAP workloads in a single system. In contrast to dedicated analytical systems, database systems store data only once and avoid ETL cycles (extraction,

transformation, and loading of data). Therefore, analytical and exploration queries are based on the latest transactionally consistent database state.

HyPer's compilation framework [8] makes it extra useful for integrating computation-intensive *HTAPx* queries. For example, costly data transfers are omitted. Also, embedding data exploration in query evaluation plans leads to better optimization potentials. Furthermore, HyPer's MVCC (Multi-Version Concurrency Control) supports long-running complex queries without interference with mission-critical transactional processing [9]. In general, integrating data exploration in HTAP systems avoids ETL costs, stale data, as well as assembling and administrating complex system environments, and therefore facilitates ad-hoc data exploration. The resulting system, covering both data management and data exploration, simplifies IT architectures. Today's in-memory and parallelization features of database systems, plus amenities like the restore functionality, are further arguments in favor of the "one system fits all" approach. For integrating *HTAPx* algorithms into HyPer we have already discussed our four layer model in [10]. These layers are structured hierarchically depending on their level of integration:

- ***Layer (1)*** DBMSs as data storage with external analytics algorithms—the nowadays most commonly used, but least integrated approach.
- ***Layer (2)*** User-defined functions (UDFs)—code snippets in high-level languages executed by the DBMS.
- ***Layer (3)*** SQL queries—including recursive common table expressions (CTE) and our novel iteration construct.
- ***Layer (4)*** Integration as physical operators—the deepest integration that unleashes highest performance.

All these approaches have certain trade-offs in flexibility and performance as depicted in Figure 1. We proposed implementing multiple approaches to cover the diverse needs of different user groups and application domains in [10]. However, in this demo we emphasize on the benefits of Layer *(4)*: the deep integration of data exploration tasks at the "operator-level". This novel approach of Layer *(4)* combines the highest performance with high flexibility, but has the drawback to be implemented by the database architects only, while Layer *(2)* and Layer *(3)* provide environments in which expert users can implement their own algorithms. To increase flexibility within *(4)*, we propose user-defined code snippets as parameters to our operators. These called *lambda functions*, pre-selecting the data-set and distance metrics and many other, are even able to customize the semantics of a given analytical algorithm in an application-specific way.

The purpose of the demonstration is to experience the efficiency of operator-centric analytics deeply integrated into a Main Memory Database (MMDB) without the need to learn to use specialized tools. We demonstrate a visual web interface that allows users to examine and evaluate the query plan of analytical and exploratory SQL queries. In our system it is possible to combine different mining approaches in a query and flexibly compare distance metrics on several suitable data sets from social media, geo-data and relational data. Besides visualizing the analytical result, our user interface will display runtime, memory footprints, aggregates, the visualization of the query plan which is logical and physical optimized, and further visual interpretations of the analytics result.

## 2 OPERATORS IN HYPER

In contrast to other database systems, the MMDB system HyPer integrates additionally to OLAP and OLTP workloads important data exploration functionality directly into the core of the database system by implementing special highly-tuned *operators* [10]. HyPer provides some additional advantages that further add to the integration depth: Indexes can be used to efficiently select and load input data. NUMA-aware parallel loading and distribution of input data is conducted before our specialized operators are called. HyPer's just-in-time compilation is particularly beneficial for computationally intensive exploration tasks. Furthermore, all computations are performed in main memory, hence the overhead for swapping and buffer management is avoided. Finally, the push-based pipelining execution model makes it simple to efficiently integrate new (sub-) algorithms as operators, and also takes care of parallel execution and other optimizations. Because the internal structures of database systems are fairly different, such operators have to be specifically designed and implemented for each system [10]. In this demo we will show four valuable data exploration approaches and provide at least one implementation for each one of them:

- ***Clustering*** For clustering approaches we implement the model-based clustering algorithms k-Means and k-Modes as well as the density-based algorithm DBscan [4, 10].
- ***Classification*** For classification and prediction of numerical values we implement the standard naive Bayes algorithm [7].
- ***Association rule mining*** For finding frequent itemsets in large data we implement the apriori algorithm [1].
- ***Graph mining*** graph mining is an own area, similar to mining relational data, and usually focuses on community detection and link prediction. We picked PageRank [2] as a representative.

However, without modification they are not flexible, i.e. they are not even applicable in the context of similar but slightly different algorithms. Consider the k-Medians algorithm. It is a variant of k-Means that uses the L1-norm (Manhattan distance) rather than the L2-norm (Euclidean distance) as distance metric. While this distance metric differs between the variants, their implementations have predominant parts of code in common. Even though this common code could be shared, different distance metrics would make different variants of our algorithmic operators necessary.

Instead, when designing data exploration operators, we identified and aimed to exploit such similarities. Our goal was to have one operator for a whole class of algorithms with variation points that could be specified by the user [10]. We use *lambda expressions* in SQL queries to inject user-defined code into variation points of analytics operators. Lambda expressions are anonymous SQL functions that can be specified inside the query. For syntactic convenience, the *lambda expressions*' input and output data types are automatically inferred by the database. Also, for all variation points we provide default lambdas. Thus, non-expert users can easily fall back to basic algorithms. With lambda-enabled analytics operators we strive not only to keep implementation and maintenance costs low, but especially to offer a wide variety of algorithm variants required by data scientists. Also, because *lambda functions* are formulated in SQL, they benefit from existing relational optimizations.

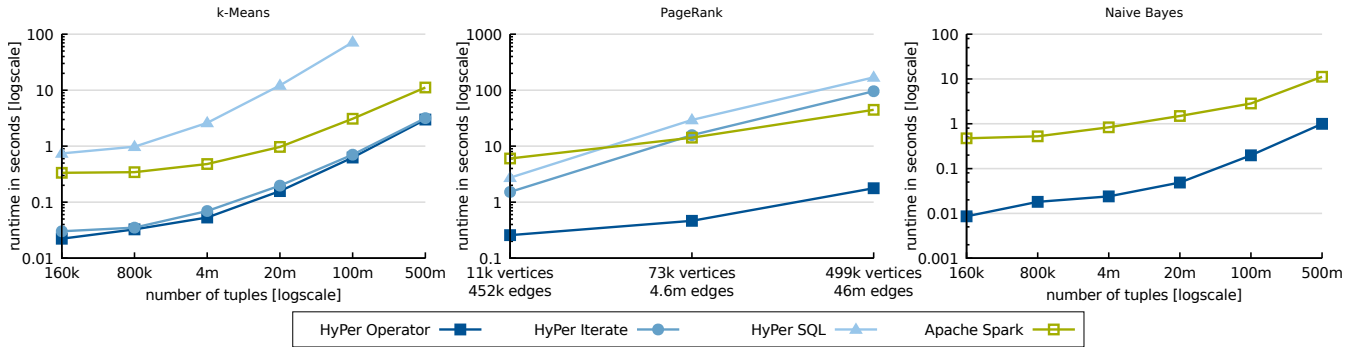


**Figure 2: k-Means experiment: varying the number of tuples $N$; 10 dimensions, 5 clusters. PageRank experiment: using the LDBC SNB dataset, damping factor 0.85, and 45 iterations. Naive Bayes experiment: varying the number of tuples $N$.**

## 3 EVALUATION

All experiments are carried out on a 4-socket Intel Xeon E7-4870 v2 (15×2.3 GHz per socket) server with 1 TB main memory, running Ubuntu Linux 15.4 using kernel version 4.2.

We evaluate our physical operators, denoted as *HyPer Operator*, SQL queries with our iterate operator, denoted as *HyPer Iterate*, and a pure SQL implementation using recursive CTEs, denoted as *HyPer SQL*, against the best open source competitor: Apache Spark. *Apache Spark 1.5.0* with MLlib as a representative of the "big data exploration" platforms. We refer the interested reader to the experimental evaluation of other contender systems for other categories like standalone systems or database extensions in [10]. We evaluate both systems—Spark and HyPer—on three common analytics queries: k-Means, PageRank and naive Bayes. To ensure a fair comparison, both systems have to implement the same variant of all algorithms. For fairness reasons we had to disable some optimizations implemented in Apache Spark MLlib for k-Means: First, the MLlib implementation computes lower bounds for distances using norms, hence reducing the number of distance computations. Second, distance computation uses previously computed norms instead of computing the Euclidean distance (if the error introduced by this method is not too big).

As expected, Apache Spark shows very competitive runtimes in our benchmarking. Spark was especially built for these kinds of algorithms. Still, it is up to two orders of magnitude slower than the HyPer Operator approach, as shown in Figure 2. HyPer's one-system-fits-all approach comes with some overhead of database-specific features that are not present in dedicated analytical systems like Apache Spark. Therefore, it is important that these features do not cause overhead when they are not used. For instance, isolation of parallel transactions should not take a significant amount of time when only one analytical query is running. Some database-specific overhead, stemming e.g. from memory management and user rights management, cannot be avoided. Nevertheless, HyPer shows far better runtimes than dedicated systems, while also avoiding data copying and stale data.

To put it in a nutshell, the experiments match the expected order concerning runtimes: the deeper the integration of the data exploration, the faster the system. Our results also support our idea of one database system being sufficient for multiple workloads. While this has been shown for combining HTAP workloads before [5, 6], our contribution was to integrate one more workload, data exploration, while keeping performance and usability on a high level [10].

## 4 DEMONSTRATION

Our web interface *HyPerInsight* demonstrates the scalability of data exploration inside of HyPer on several large, but memory sufficient data sets. The user interaction concept of *HyPerInsight* is designed to minimize the requirement of users' expertise with the explored data sets. It supports users during query formulation and encourages an iterative approach. Figure 3 shows *HyPerInsight* visualizing a demo query for the deeply integrated k-Means algorithms with explicitly given lambda function. The lambda can be given in two ways: explicitly and implicitly. Implicitly given, it always corresponds to the euclidean distance metric. Explicitly all other metrics are also possible besides pre-selecting specific tuples for the k-Means algorithm. The $\lambda(a, b)$ in the query given as example in *HyPerInsight* refers to the euclidean distance as well, which is the standard for k-Means. On the upper left side, several changeable queries are predefined and can be run on both systems HyPer and Spark in parallel. The lower left side shows the resulting table and a visualization of the given data set with the resulting cluster ids found by k-Means. On the upper right side, *HyPerInsight* gives the visual interpretation of CPU runtime, memory footprint and other system-aware measurements of both competitor systems. In the lower right is the query plan that visualizes in several steps how the query plan is optimized during execution. In the demonstration we will run a *HyPerInsight* instance on a demonstration laptop with 5 graph snapshots of data loaded from Facebook, one snapshot from Amazon item buckets and several snapshots of data from imdB. We will motivate our demonstration by solving several problems in social network data. Examples are: Grouping (clustering) and labeling different contacts or friends attributes in a social network. Classifying specific groups for predicting which friends might know other interesting people. Additionally, we pave way to demonstrate that HyPer is capable of performing these complex data mining techniques on live data, which is constantly updated in the background. Thereby, we enable data experts to work on the most recent version of data, which allows more precise analytics.

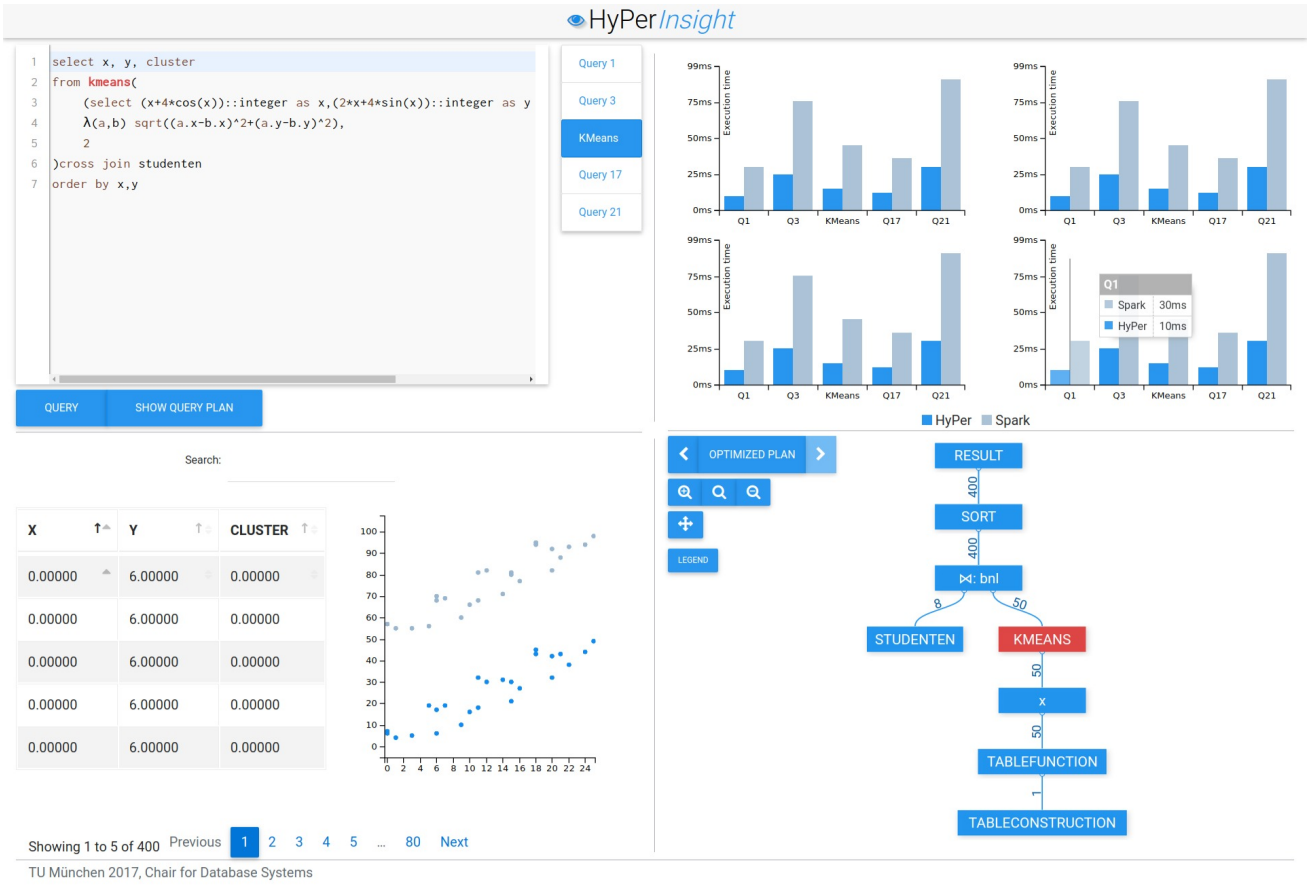


**Figure 3: The webinterface of HyPerInsight. Upper Left: Interactive query selection. Upper Right: Runtime evaluation. Lower Left: Resulting table and graphical interpretation. Lower Right: Query plan optimizations.**

## 5 TAKE-AWAY MESSAGE

In this paper, we presented *HyPerInsight*, a solution to visually show and interpret the deep integration of data exploration task in the main memory database system (MMDB) HyPer. The deep integration of analytical operators into a high-performance MMDB like HyPer includes automatically optimized SQL queries on various kinds of algorithms in the data science segment. We will show that it is possible to build meaningful visualizations on various types of data, using a general purpose database system combining sensible state-of-the-art hand-written solutions for compatibility, performance and many other reasons.

## 6 ACKNOWLEDGEMENTS

This research was supported by the German Research Foundation (DFG), grant NE 1677/1-1. It is part of the TUM Living Lab Connected Mobility (TUM LLCM) project and has been funded by the Bavarian Ministry of Economic Affairs and Media, Energy and Technology (StMWi) through the Center 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.

## REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In *Proc. VLDB Endow.*, Vol. 1215. Morgan Kaufmann, 487–499.
[2] Sergey Brin and Lawrence Page. 1998. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Computer Networks* 30, 1-7 (1998), 107–117.
[3] Andrew Crotty, Alex Galakatos, and Tim Kraska. 2014. Tupleware: Distributed Machine Learning on Small Clusters. *IEEE Data Eng. Bull.* 37, 3 (2014), 63–76.
[4] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Kdd*. AAAI Press, 226–231.
[5] Franz Färber, Norman May, Wolfgang Lehner, Philipp Große, Ingo Müller, Hannes Rauhe, and Jonathan Dees. 2012. The SAP HANA Database – An Architecture Overview. *IEEE Data Eng. Bull.* 35, 1 (2012), 28–33.
[6] Alfons Kemper and Thomas Neumann. 2011. HyPer: A Hybrid OLTP & OLAP Main Memory Database System Based on Virtual Memory Snapshots. In *ICDE 2011*. IEEE Computer Society, 195–206.
[7] Kevin P. Murphy. 2006. Naive Bayes Classifiers. (2006).
[8] Thomas Neumann. 2011. Efficiently Compiling Efficient Query Plans for Modern Hardware. *Proc. VLDB Endow.* 4, 9 (2011), 539–550.
[9] Thomas Neumann, Tobias Mühlbauer, and Alfons Kemper. 2015. Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. In *Proc. SIGMOD 2015*. ACM, 677–689.
[10] Linnea Passing, Manuel Then, Nina Hubig, Michael Schreier, Stephan Günnemann, Alfons Kemper, and Thomas Neumann. 2017. SQL- and Operator-centric Data Analytics in Relational Main-Memory Databases. In *EDBT 2017*. OpenProceedings.org, 84–95.
[11] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *Proc. HotCloud 2010*. USENIX Association, 95.