CS 5513
Advanced Database Management Systems
Spring 2022

Impact of Delete and Update Queries on DBEst++ approximate query processing engine

Shyam Sundar Murali Krishnan
shyamkrishnan@ou.edu

Ramya Sai Vuyyuru
ramya.vuyyuru@ou.edu

ABSTRACT

The introduction of learning algorithms has revealed numerous opportunities for improving the functionality and performance of Data Systems. Approximate Query Processing (AQP) is one such area where machine learning (ML) models have been used to improve query execution efficiency and accuracy, outperforming traditional sampling-based approaches. We introduce an DBEst++ -Up (DBEst++ Update), a novel AQP engine that is extension to the existing previous effort DBEst++. The primary design goal of DBEst++ -Up is to use ML models for the Update and Delete queries. The key feature of DBEst++ -Up is a combination of word embedding models and neural networks tasked with regression-based predictions for density estimation and aggregation-attribute values. We included the functionalities of the delete and update queries by feeding the engine with series of delete and update queries.

1. INTRODUCTION

Nowadays, there is a lot of interest in using Machine Learning models to enhance the functionality of database systems. Machine learning models have recently been adapted for processing various aggregate queries rather than using (samples of) data. It has been proved that Machine Learning has been in help in providing effective results in tuning the parameter for the database [8]. One of the functionalities that the database looks for is the approximate query processing. The initial efforts focused on using regression-based techniques to predict and/or explain approximate query answers. Learned AQP engines have also emerged, promising to improve both accuracy and efficiency by processing aggregate queries holistically. DBEst trains Kernel Density Estimators (KDEs) and Regression Models (RMs) over column sets for query processing [4]. Despite these advancements and the improvements, they brought in terms of accuracy and efficiency, much more work has done in terms of efficiency and accuracy, as well as, more importantly, related memory overheads in the DBEst++ model. This engine used Mixture Density Network (MDN) for predicting the answer to the queries [5]. But the prediction is done only with insertion and selection queries for the analysis of performance and accuracy. With these improvements also there is much more work to be done by taking many other query structures and analyzed, especially the update and delete queries. In this work, the implementation of the new functionalities by extending DBEst++ model to implement the delete and update queries called as DBEst++-Up. The remaining of the paper is organized as follows: Section 2 overviews the related work, Sections 3 overviews on the architecture of DBEst++-Up, Section 4 overviews the datasets that are used for analyzing queries, section 5 overviews the metrics used, Section 6 overviews the results, Section 7 overviews conclusions and future work.

2. RELATED WORK

The tuning of queries and other parameters in a database management system has been a recent topic of research. It has been proved that Machine Learning has been in help in providing effective results in tuning the parameter for the database [8]. One of the functionalities that the database looks for is the approximate query processing. The approximate query processing is the way of predicting the results of the query even before the query is executed in the database. In recent years many such query processing engines have been developed. Some of the popular engines are DBEst [4] and DeepDB [3].

In DBEst the workflow is that queries are sent to the DBEst engine, and the samples are done and later they are used to regression model and the density estimator. Sampling is the first process to be done over to group the data. scikit-learn packages (Grid Search CV) are used to tune the models by using cross-validation. However, the choice of an appropriate regression model is complex because different models work better for different data regions. For the implementation the models used are XGBoost, and GBoost. First, each model is trained separately. Subsequently, the accuracy performance of each of these models is evaluated, using random queries over the independent attribute's domain. Also kernel density estimation is chosen as it can be performed in any number of dimensions which allows the DBEst support multivariate query processing.

Both these engines use Machine Learning algorithms to predict the answer to the queries given. In the case of DBEst the machine learning models like kernel density estimator and regression models have been used. In the case of DeepDB the Relational sum product networks has been used. Despite the success of these models in terms of accuracy and efficiency, there was scope of more improvements that can be done in terms of accuracy, response time and memory overheads.

Then improvements were made to the work of DBEst model called as DBEst++ [5] which uses Mixture Density Network (MDN) for predicting the answer to the queries. This network is mainly used in this article due to its large success in several real-world applications [2] and especially in bigger products like the apple [1]. There are two types of MDN models to be used in DBEst++. The MDN-regressor is used for regression tasks and the MDN-density is used for density estimation. The structures of the networks are the same for MDN-regressor and MDN-density. MDNs are simple and straightforward -one of the main reasons it was selected. Combining a deep neural network and a mixture of distributions creates a MDN model.

Machine Learning methods can only operate on numerical or continuous data. Binary, ordinal encoding methods are usually used for converting the variables into the required numerical or continuous form. These encoding methods map categorical values into arrays of 0 and 1s, and since the output is numerical, they can be used in all machine learning methods. So, a conversion method called the skip gram model is used in this article due to its huge success with words and its combinations [6]. Skip Gram model is used to transform group attribute values and other categorical attributes into a real valued vector representation [6]. As categorical attributes have no meaningful distance between successive values learning a relationship between a categorical independent variable and a dependent one is very difficult. Using word embedding introduces such a meaningful distance between different independent categorical-attribute values becomes easier and more accurate. With all these methods into the workflow of DBEst++ it was shown from this article that DBEst++ was much better model in terms of accuracy, response time and memory overheads. The data used in this article is the TPC-DS data set [7] and the flight data set.

3. SYSTEM ARCHITECTURE

In the existing DBEst++ model [5], the functionalities of Select, Drop, Aggregate (Count, Sum, Avg) and show table are implemented. Queries having the above functionalities only are parsed by the

DBEst++ engine through the Query Parser. In the new implementation, DELETE and UPDATE components are introduced into the Query Parser which was not done in the previous work so that this new addition in the DBEst++ engine now will help perform the delete and update queries as well. Also, in the Figure 1 the components that are circled are the new components that are added in this paper.

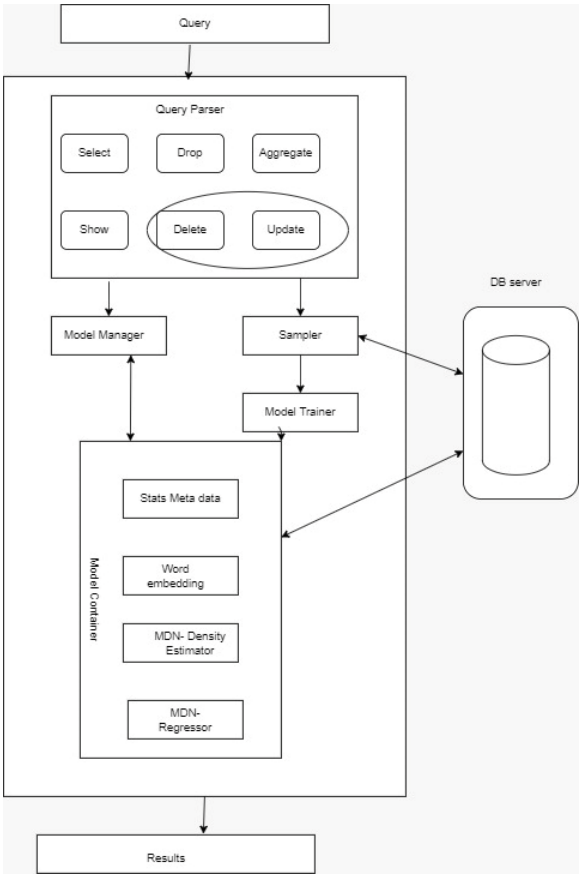

Figure 1: DBEst++-Up System architecture

The **Query Parser** that parses the incoming query and check if the incoming query is the select queries or show queries or aggregate queries or update queries or any creation queries. In this part the select, create, show, and aggregate functions have already been completed in the previous work. Whereas the delete and the update components has not been done in the previous work which are added in this work. Thus, query parser now parses the incoming query having the delete and update queries.

For the **delete component** in Query parser, program searches if the SQL type is “delete” in the parser. Here SQL type is an attribute which tokenizes the query and finds the keyword in the query like delete in this case. And from the parser it gets the table name of the model which is created already else it creates an error message if model didn’t exist earlier. If there is a range in the delete query, the encoding methods skip gram model maps categorical range values into arrays of 0 and 1s, and since the output is numerical and a real valued vector, which is used in machine learning methods MDN Density Estimator and Regressor to predict the values. For query without WHERE range clause, it just maps the categorical range values into arrays and predict. This component has been built in the DBEst++-Up.

For the **update component,** which is similar execution to update, the program searches if the SQL type is “update” in the parser. Here SQL type is an attribute which tokenizes the query and finds the keyword in the query like update in this case. And from the parser it gets the table name of the model which is created already else it creates an error message if model didn’t exist earlier. If there is a range in the update query, the encoding methods skip gram model maps categorical range values into arrays of 0 and 1s, and since the output is numerical and a real valued vector, which is used in machine learning methods MDN Density Estimator and Regressor to predict the values. For query without WHERE range clause, it just maps the categorical range values into arrays and predict. This component has been built in the DBEst++-Up. The only difference between the update and delete is that update has a "set" keyword along with it, but this set keyword

has already been encountered in the previous work. Thus, the query parser in the model already accepts set keyword.

The **Model Manager** selects the appropriate model from the Model Container which is in this case will be word embedding or mixture density network which will use on the query and if the query is the select statement it will also select the data from the range of variable values specified. These representative values are used in the model to approximately evaluate the results of the query.

The **Sampler** creates samples of tables, based on which Machine learning models will be built. The sampler takes the random set of tuples from the database server but for the sake of experimentation the database is in form of csv files. The sampler is designed in such a way that every time it selects a tuple in random it checks if the tuple is already in the sampler. In the case of update queries once it gets updated into the file. Now when the sampler samples this updated tuple it will be checked if it is already there. But since this is the updated tuple, it will be considered as a new tuple and takes into as a sample. In case of delete queries in our work only the queries in the database are deleted. But it does not effectively delete

The **Model Trainer** module trains word Embedding and Mixture density network (MDN) models upon the drawn samples from the sampler. This was completed in the previous work as well.

The **Model Container** maintains the metadata and the models which will be used upon the query. The models used for DBEst++ are Word Embedding and Mixture Density network which again divided into two classes MDN- Density estimator and MDN - Regressor. This was completed in the previous work as well.

The **word embedding** model used in this architecture is the Skip Gram model. There were lots of word embedding models like the PCA (Principal Component Analysis) model which was popularly used earlier as it was much easier and understandable compared to the deep learning models [10] which even though good for word embeddings [9]. But it was shown that Skip gram model was shown to be more useful for word embeddings [6]. Skip Gram model is used to transform attribute values and other categorical attributes into a real valued vector representation. As categorical attributes have no meaningful distance between successive values learning a relationship between a categorical independent variable and a dependent one is very difficult. Using word embedding introduces such a meaningful distance between different independent categorical-attribute values becomes easier and more accurate.
For example, let us consider the salary information of staffs in a university. Let us consider Tom and Alan are given almost the same salaries let us in this case consider 80-83k per year. Therefore, the embedding vectors for Tom and Alan will be similar.

To use this approach, the training data must be prepared by using natural language processing methods. In this case, the data is in form of rows in the table. When preparing the training data the dataset is created in such a way that pairs of categorical attributes values come together in a row of the table. These pairs are given to the Skip Gram model which tries to find similar vector representations. To create the training pairs, the attributes that are involved in the query is used. If the involved attributes are not categorical, it is then discretized first. Furthermore, it is possible that a distinct value exists in two different attributes, so to avoid pushing wrong information to the Skip Gram model, a prefix for each distinct value

in the attributes is added. For example, in value Tom of the attribute name it will be prefixed as "name Tom".

For example, we have a query where the staff information must be grouped by with respect to their department in which they are working on. For density estimation in the **MDN-density estimator** the input features that are sent are the word embedded data or vectorized data of the information that are selected to display. The label or the result expected would be the categorical attributes given after the WHERE command. The density estimator aims in finding the distribution of the categorical attributes across the grouping done in this example mentioned.

For regression estimation in the **MDN - regressor** will take the inputs features as the word embedded data or vectorized data of the information that are selected to display, and the categorical attributes given after the WHERE command. The label will be the resultant answer to the given query. The regressor aims at finding the average resultant values for the group using the input features.

Since all these components are set once the DBEst++-Up environment is setup, and the environment successfully supports the Delete and Update queries.

4. DATASET

The data used for the analysis of the queries are the Flight data and TPC-DS. TPC-DS [7] is a database benchmark used to measure the performance of complex decision support databases. The TPC-DC dataset contains the information of the retail stores, about the logistics of the sales of items which include item sold date, sold item prices, profits. The dataset includes numerous schemas that only vary in the amount of data. Numerous schemas include by giving different values to the layers of the MDN Regressor and MDN Density Estimator. Another data set includes Flight data of having carrier information, origin state, destination state, total fly time, distance travelled.

5. METRICS USED

The relative error is used to find the error in the predicted function so that the machine learning algorithm can learn accordingly and give out good results. The relative error is found by identifying the difference between the predicted answer from the model and the actual answer expected. This can be in the form of root mean squared error.

$$\text{rmse} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - x_i)^2}$$

where n is the total number of data, y is the actual label and x is the predicted label where label is the query output.

The response time is found out by taking the difference between the starting time in which the query goes into the model and the ending time when the result comes out of the model.

6. RESULTS

Now to run the application first must initialize the executor file and the following parameters are initialized. The parameters initialized here are the same set of parameters initialized in the previous works. Generally, the split parameter must be initialized, and it can be any punctuation like a comma or a dashed line. Also based on the dataset used the table header must be initialized because when the attributes are split it must go to the corresponding header in a dictionary.

In case of flight dataset, the number of MDN layer nodes for regressor is 10 and for the number of MDN layer nodes for density function is 15. The number of jobs is set to 1. The number of hidden layers is 1 and number of epochs is 20. The number of gaussians for regressor is set to 8 and number of gaussians for density is set to 10. In case of TPC- DS the encoder is set to embedding and number of epochs is set to 20.

Now we must first give in the query to create the table. To do that we must pass in the query for creating the table followed by the path where the table in present in form of csv file and the method to be uniform followed by the size of the file. Now the model will be created. After that we must pass in the query required which will return the predicted results.

The outputs show the model created and the prediction of the query answer for select, update, and delete queries for both flight dataset and the TPC-DS data set.

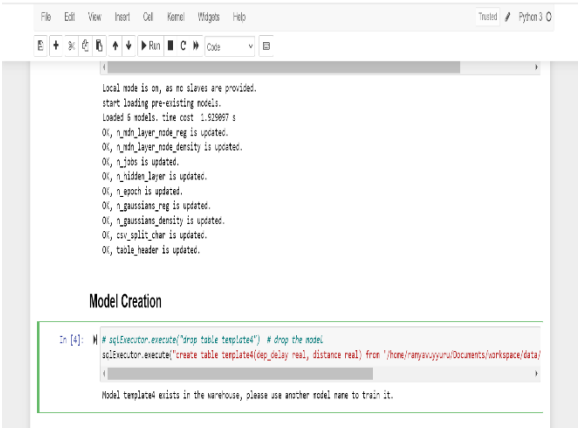

Figure 2: Creating model for select in flight.

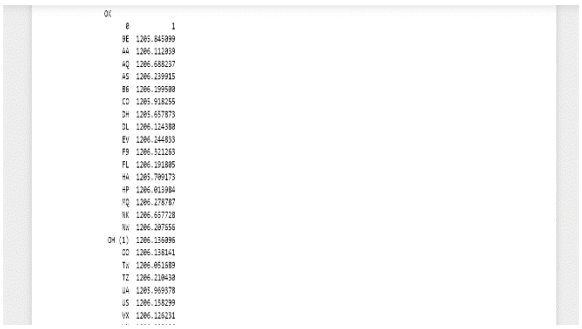

Figure 2: Prediction for select in flight data.

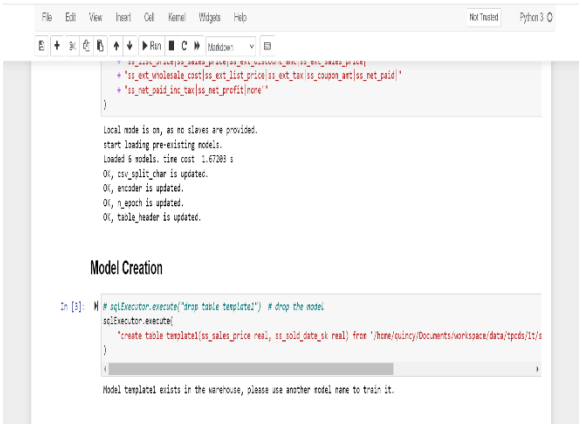

Figure 3: Creating model for select in TPC-DS.

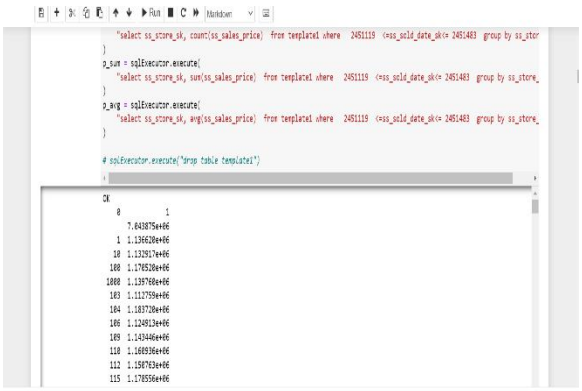

Figure 4: Prediction for select in TPC-DS.

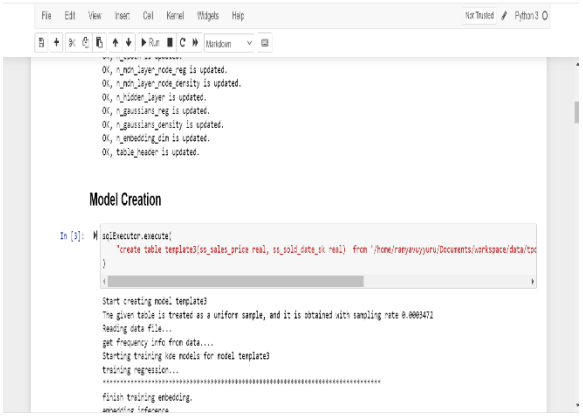

Figure 5: Creating model for update in TPC-DS.

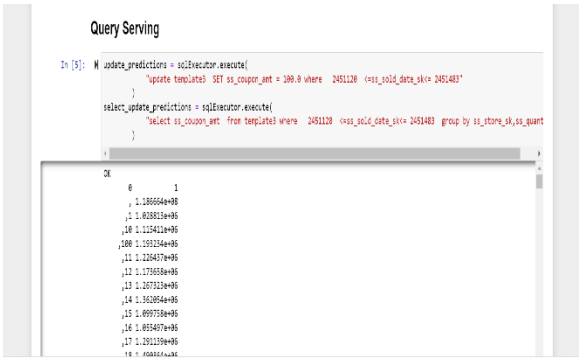

Figure 6: Prediction for update in TPC-DS

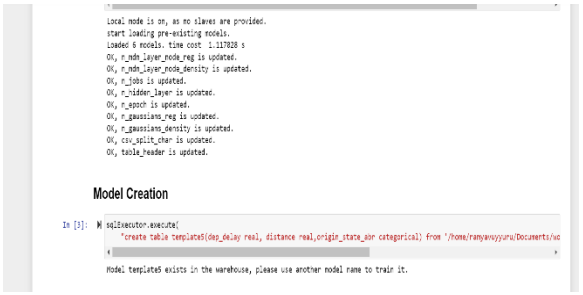

Figure 7: Creating model for update in flight data.

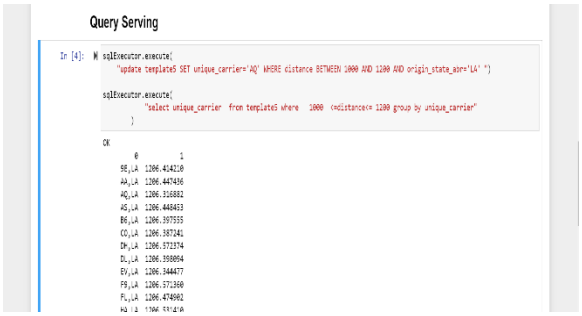

Figure 8: Prediction for update in Flight Data

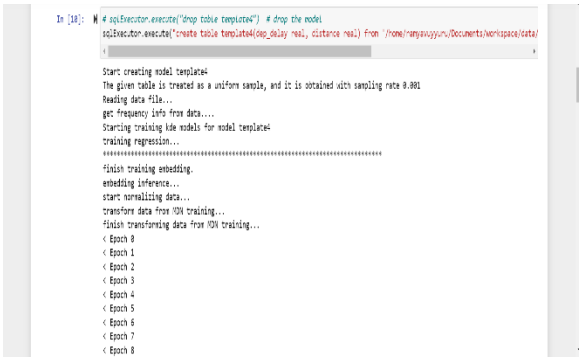

Figure 9: Creating model for delete in flight data.

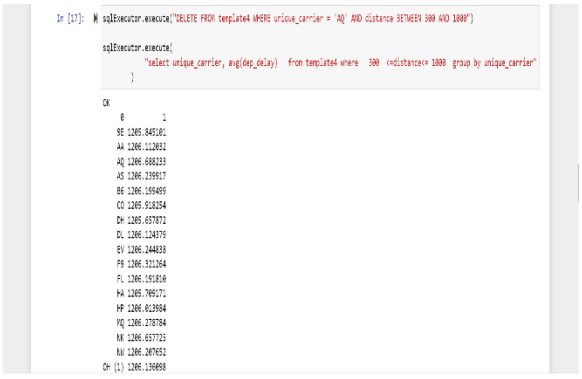

Figure 10: Prediction for delete in Flight Data

Figure 11: Creating model for delete in TPC-DS

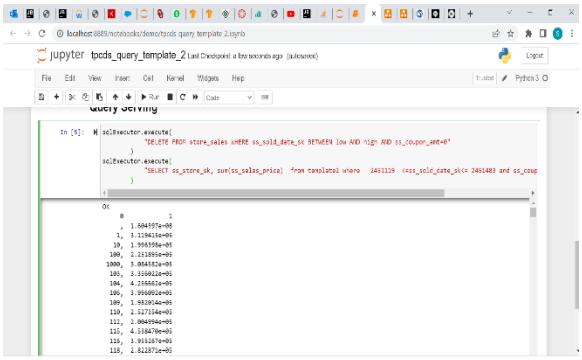

Figure 12: Prediction for delete in TPC-DS

7.CONCLUSIONS AND FUTURE WORK

In this paper, we have implemented an DBEst++ -Up (DBEst++ Update), a novel AQP engine that is extension to the existing previous effort (DBEst++) which supports the delete and update queries. In the previous engine the queries only support for the select and create queries. Now we have extended that engine to make it support for the delete and update queries. Also implemented multiple simple queries on the engine to analyze the performance and accuracy of the results over the large datasets. Currently the engine supports the creation, updating, selection, deletion from data with the simple queries which can be further developed to complex queries in the future work.

The scope of future work which can include the implementation of the complex queries which includes Join, Multiple Joins etc., Also we can implement the engine further to implement the nested queries. As currently it is analyzed over the simple queries, in future the analysis can be extended to the complex and nested queries. As the complex and nested queries take more time for execution on the large datasets, the engine can be modified and implemented such that it accepts the queries and how adversely the queries impact the model and/or their accuracy and training times.

REFERNCES:

[1] Tim Capes, Paul Coles, Alistair Conkie, Ladan Golipour, Abie Hadjitarkhani, Qiong Hu, Nancy Huddleston, Melvyn Hunt, Jiangchuan Li, Matthias Neeracher, et al. Siri on-device deep learning-guided unit selection text-to-speech system. In Interspeech, pages 4011–4015, 2017.

[2] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. Nature, 538(7626):471–476, 2016.

[3] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. Deepdb: learn from data, not from queries! Proceedings of the VLDB Endowment, 13(7):992–1005, 2020.

[4] Qingzhi Ma and Peter Triantafillou. Dbest: Revisiting approximate query processing engines with machine learning models. In Proceedings of the 2019

International Conference on Management of Data, pages 1553–1570, 2019.

[5] Qingzhi Ma, Ali Mohammadi Shanghooshabad, Mehrdad Almasi, Meghdad Kurmanji, and Peter Triantafillou. Learned approximate query processing: Make it light, accurate and fast. In CIDR, 2021.

[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. Advances in neural in- formation processing systems, 26, 2013.

[7] Raghunath Othayoth Nambiar and Meikel Poess. The making of tpc-ds. In VLDB, volume 6, pages 1049–1058, 2006.

[8] Dana Van Aken, Andrew Pavlo, Geoffrey J Gordon, and Bohan Zhang. Automatic database management system tuning through large-scale machine learning. In Proceedings of the 2017 ACM international conference on management of data, pages 1009–1024, 2017.

[9] Ronan Collobert. Word embeddings through hellinger pca. In in Proceedings of the 14th Conference of the European Chapter of the Association for Computational Linguistics. 2014.

[10] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In International Conference on Machine Learning, ICML. 2008.