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

Chapter 1

Project Progress report II

1.1 Project objectives and motivations

The general sub-domain which the project is based on is the **AI- Enabled data Management**. The category that this project comes under is **The extended system with additional components (category-3)**

The major objectives this project is as follows:

1. Implement the new functionalities such as delete and update which is not implemented in existing DBEst++ Query Parsing engine.

2. After implementing the new functionalities delete and update feed the engine with series of delete and update queries to test the performance of these new implemented functionalities into DBEst++.

3. Analyze the engine once these new functionalities have been embedded and analyze the behaviour of this engine by calculating the relative error and response time. The relative error is the found by identifying the difference between the predicted answer from the model and the actual answer expected. This can be in the form of mean squared error.

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

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 starting time in which the query goes into the model and the ending time when the result comes out of the model and finding the difference between both.

4. The data that is used is the TPC-DS data and Flight data which was used in the previous work to for bench-marking queries. The tables will be provided in form of the csv files and the queries to be executed are sent to executor function like how queries are passed in JDBC for example. We will be also adding the deleting and update queries along with the other queries which were sent previously once we create the delete and update components into the parser.

The reason that our project falls under this category because in the previous work the functionalities that were done were select, create, drop, and aggregate functions only but the delete and update queries which are also important functionalities in the database execution has not been done. So, our aim is to bring in these missing components into the DBEst++ engine.

1.2 Literature Review

The tuning of queries and other parameters in a database management system has been a recent topics of research. It has been proved that Machine Learning has been in help in providing effective results in tuning the parameter for the database (Van Aken et al., 2017). One of the functionality 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 (Ma and Triantafillou, 2019) and DeepDB (Hilprecht et al., 2020).

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) is 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 uses Machine Learning algorithms in order 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.

These improvements were made by extending the work of DBEst model called the DBEst++ (Ma et al., 2021). This engine used 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 (Graves et al., 2016) and especially in bigger products like the apple (Capes et al., 2017). 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 (Mikolov et al., 2013). Skip Gram model is used to transform group attribute values and other categorical attributes into a real valued vector representation (Mikolov et al., 2013). 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 (Nambiar and Poess, 2006) and the flight data set.

1.3 Work Completed

1.3.1 System Architecture

In the existing DBEst++ model, 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 1.1 the components that are circled are the new components are to be made in this project.

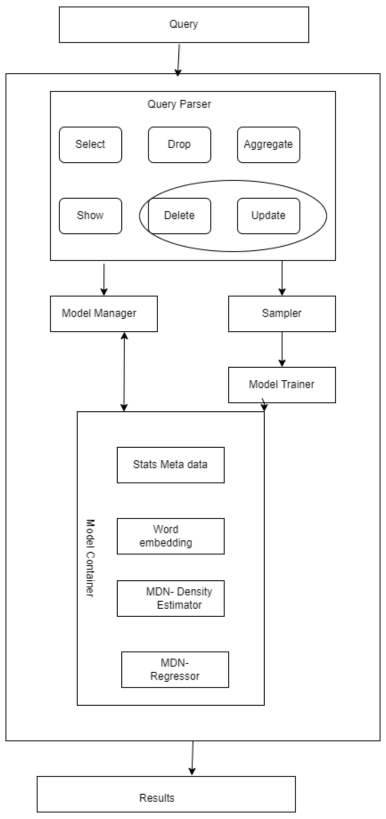


Figure 1.1: System architecture for DBEst++

1.3.2 System Diagram Component Descriptions

The **Query Parser** 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. We are using the same in this project as well. Whereas the delete and the update components has not been done in the previous work which will be done in this project.

For the **delete component**, 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 also 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 to be built by us.

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 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 also 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 to be built by us. 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 . So the parser 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** create samples of tables, based on which Machine learning models will be built. 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**. 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.

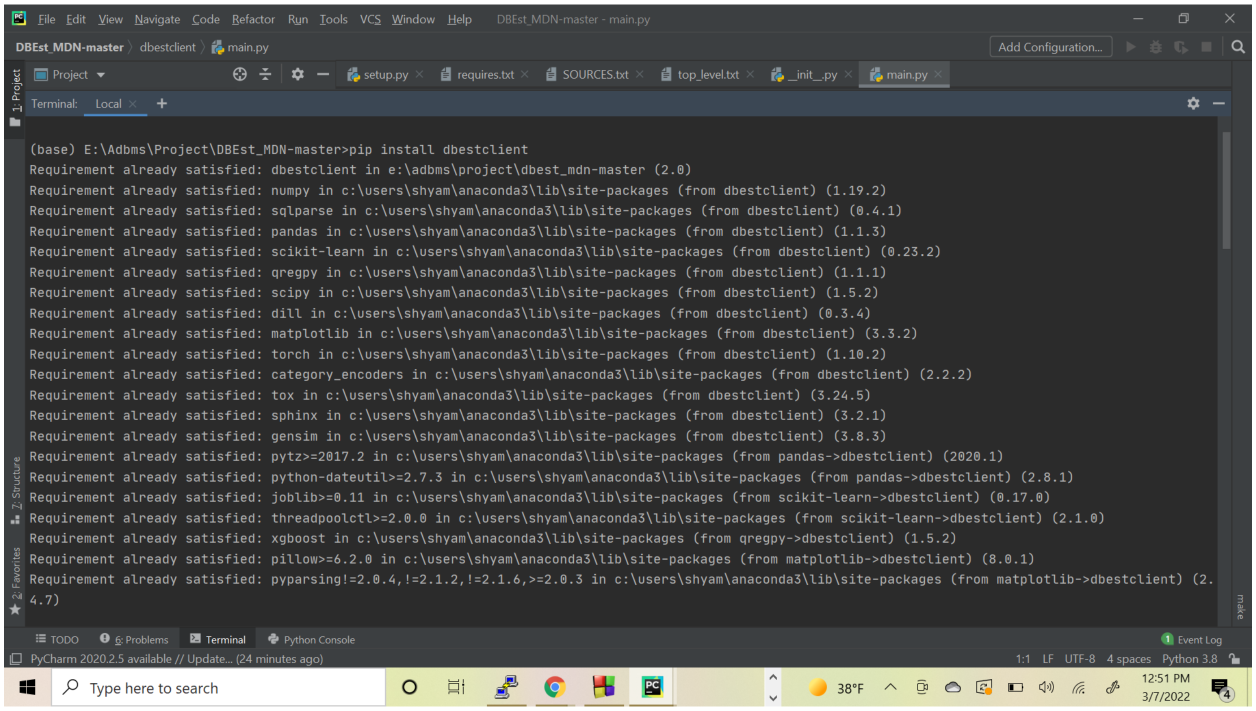
For example, let us consider the salary information of staffs in a university. Let us consider Tom and Alan are given almost the same salaries let us in this case consider 80-83k per year. Therefore the embedding vectors for Tom and Alan will be similar.

To use this approach, the training data has to 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 dat aset is created in such a way that pairs of categorical attributes values comes 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 has to 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.

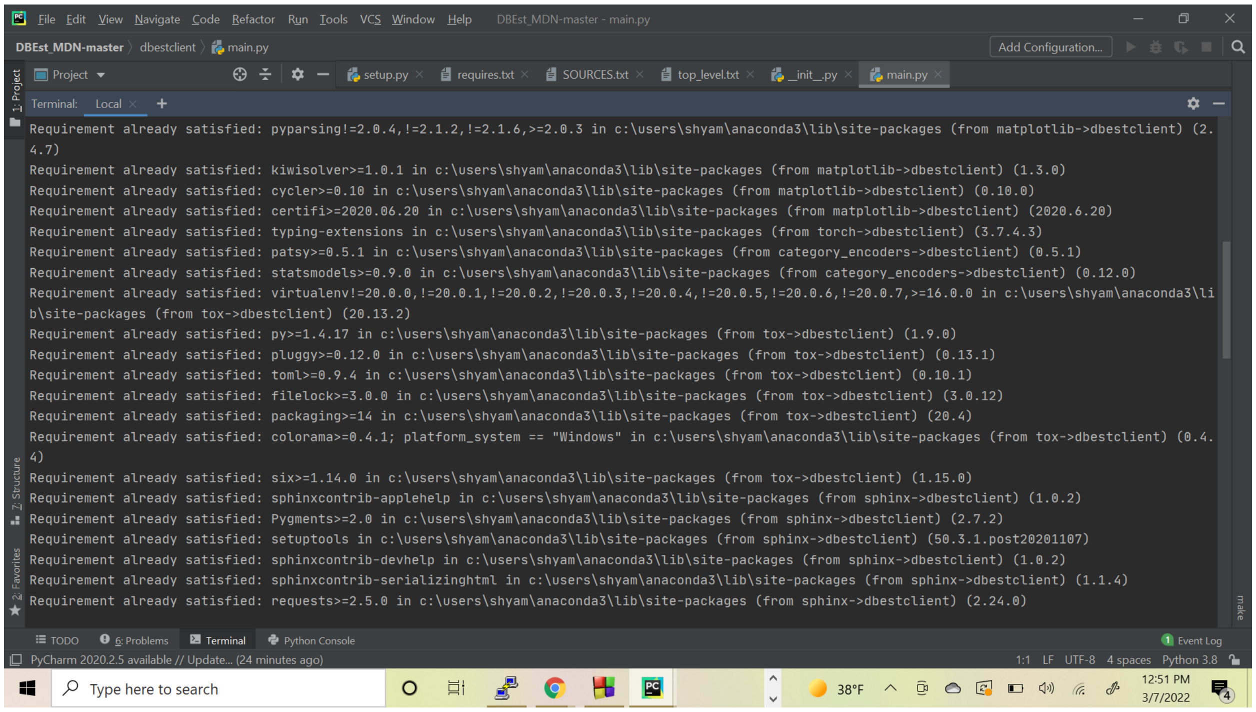
For regression estimation in the **MDN - regressor** will take the inputs features as the he word embedded data or vectorized data of the information that are selected to display and the categorical attributes given after the WHERE command. The label will be the resultant answer to the given query. The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

Since all these components are set once the DBEst++ environment used is setup which is given in github as open source we have setup the environment successfully and the screenshots supporting the successful installation of DBEst++ environment. [Note: The screenshots provided in progress report 1 is provided again since last the outputs were not be seen.] The figure 1.5 and figure 1.6 shows the execution of show component and select component in the DBEst++ engine.



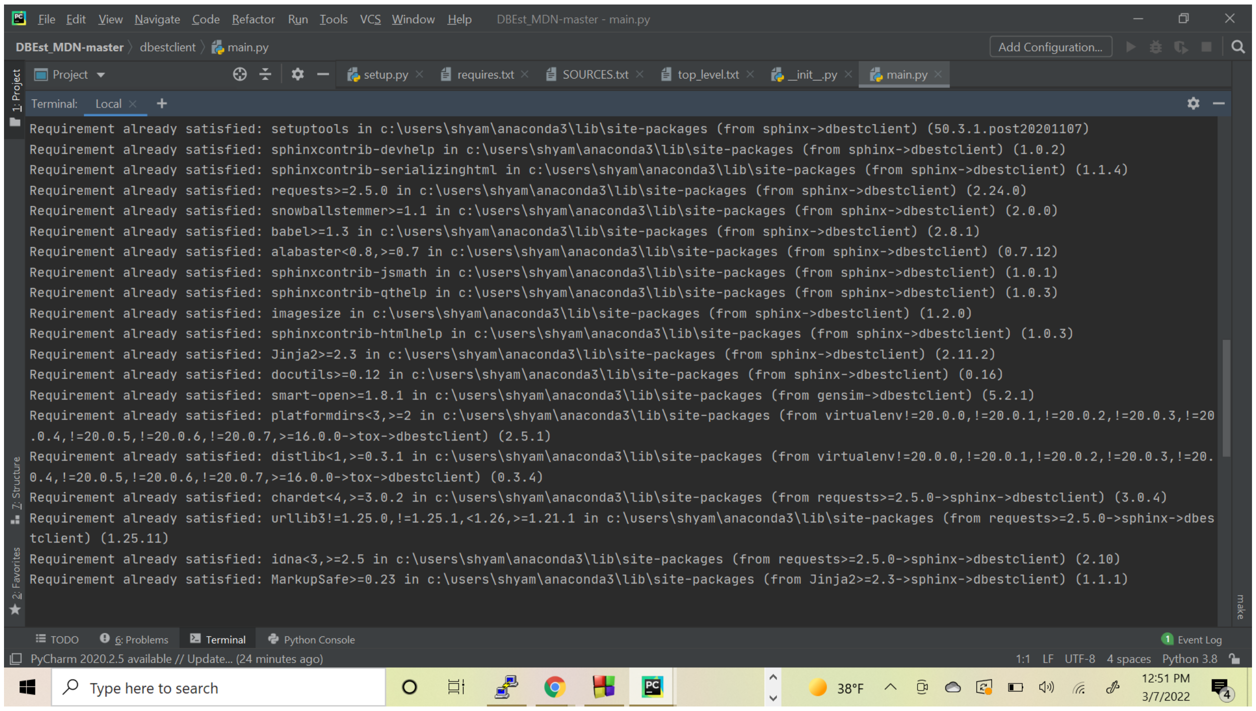
```
(base) E:\Adbms\Project\DBEst_MDN-master>pip install dbestclient
Requirement already satisfied: dbestclient in e:\adbms\project\dbest_mdn-master (2.0)
Requirement already satisfied: numpy in c:\users\shyam\anaconda3\lib\site-packages (from dbestclient) (1.19.2)
Requirement already satisfied: sqlparse in c:\users\shyam\anaconda3\lib\site-packages (from dbestclient) (0.4.1)
Requirement already satisfied: pandas in c:\users\shyam\anaconda3\lib\site-packages (from dbestclient) (1.1.3)
Requirement already satisfied: scikit-learn in c:\users\shyam\anaconda3\lib\site-packages (from dbestclient) (0.23.2)
Requirement already satisfied: qregpy in c:\users\shyam\anaconda3\lib\site-packages (from dbestclient) (1.1.1)
Requirement already satisfied: scipy in c:\users\shyam\anaconda3\lib\site-packages (from dbestclient) (1.5.2)
Requirement already satisfied: dill in c:\users\shyam\anaconda3\lib\site-packages (from dbestclient) (0.3.4)
Requirement already satisfied: matplotlib in c:\users\shyam\anaconda3\lib\site-packages (from dbestclient) (3.3.2)
Requirement already satisfied: torch in c:\users\shyam\anaconda3\lib\site-packages (from dbestclient) (1.10.2)
Requirement already satisfied: category_encoders in c:\users\shyam\anaconda3\lib\site-packages (from dbestclient) (2.2.2)
Requirement already satisfied: tox in c:\users\shyam\anaconda3\lib\site-packages (from dbestclient) (3.24.5)
Requirement already satisfied: sphinx in c:\users\shyam\anaconda3\lib\site-packages (from dbestclient) (3.2.1)
Requirement already satisfied: gensim in c:\users\shyam\anaconda3\lib\site-packages (from dbestclient) (3.8.3)
Requirement already satisfied: pytz>=2017.2 in c:\users\shyam\anaconda3\lib\site-packages (from pandas->dbestclient) (2020.1)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\shyam\anaconda3\lib\site-packages (from pandas->dbestclient) (2.8.1)
Requirement already satisfied: joblib>=0.11 in c:\users\shyam\anaconda3\lib\site-packages (from scikit-learn->dbestclient) (0.17.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\shyam\anaconda3\lib\site-packages (from scikit-learn->dbestclient) (2.1.0)
Requirement already satisfied: xgboost in c:\users\shyam\anaconda3\lib\site-packages (from qregpy->dbestclient) (1.5.2)
Requirement already satisfied: pillow>=6.2.0 in c:\users\shyam\anaconda3\lib\site-packages (from matplotlib->dbestclient) (8.0.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\shyam\anaconda3\lib\site-packages (from matplotlib->dbestclient) (2.
4.7)
```

Figure 1.2: Screenshot 1



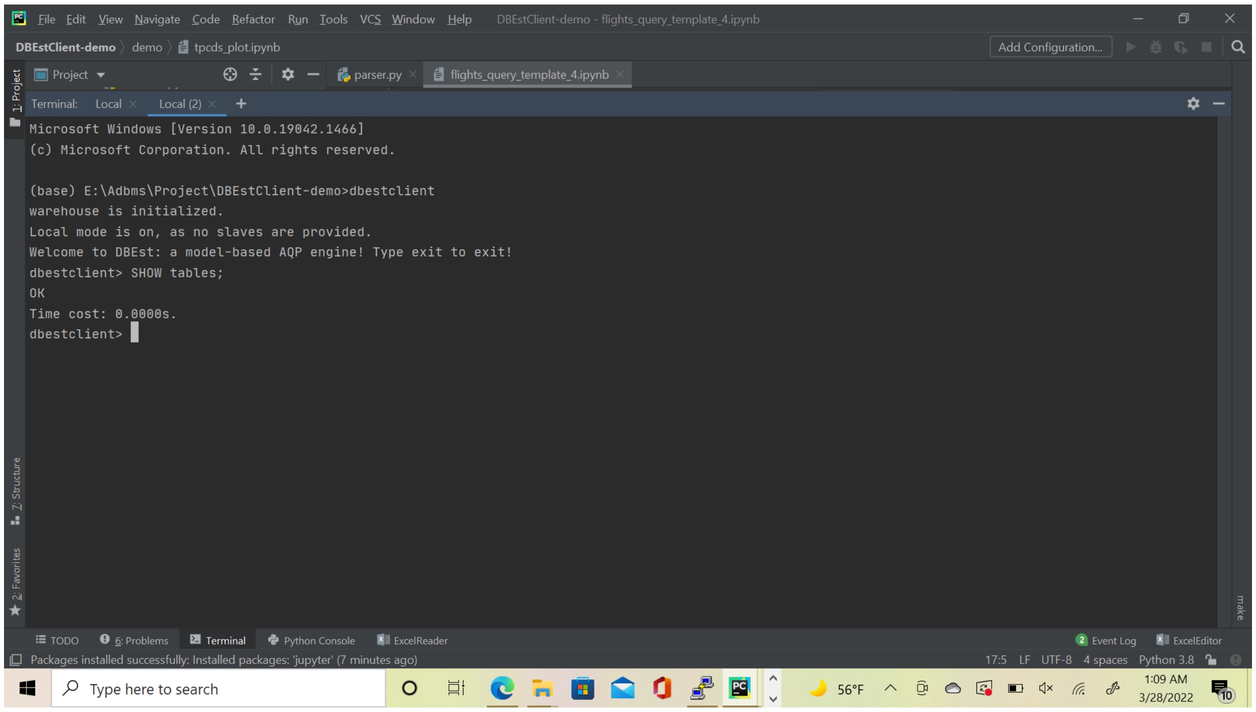
```
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\shyam\anaconda3\lib\site-packages (from matplotlib->dbestclient) (2.
4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\shyam\anaconda3\lib\site-packages (from matplotlib->dbestclient) (1.3.0)
Requirement already satisfied: cycler>=0.10 in c:\users\shyam\anaconda3\lib\site-packages (from matplotlib->dbestclient) (0.10.0)
Requirement already satisfied: certifi>=2020.06.20 in c:\users\shyam\anaconda3\lib\site-packages (from matplotlib->dbestclient) (2020.6.20)
Requirement already satisfied: typing-extensions in c:\users\shyam\anaconda3\lib\site-packages (from torch->dbestclient) (3.7.4.3)
Requirement already satisfied: patsy>=0.5.1 in c:\users\shyam\anaconda3\lib\site-packages (from category_encoders->dbestclient) (0.5.1)
Requirement already satisfied: statsmodels>=0.9.0 in c:\users\shyam\anaconda3\lib\site-packages (from category_encoders->dbestclient) (0.12.0)
Requirement already satisfied: virtualenv!=20.0.0,!=20.0.1,!=20.0.2,!=20.0.3,!=20.0.4,!=20.0.5,!=20.0.6,!=20.0.7,>=16.0.0 in c:\users\shyam\anaconda3\li
b\site-packages (from tox->dbestclient) (20.13.2)
Requirement already satisfied: py>=1.4.17 in c:\users\shyam\anaconda3\lib\site-packages (from tox->dbestclient) (1.9.0)
Requirement already satisfied: pluggy>=0.12.0 in c:\users\shyam\anaconda3\lib\site-packages (from tox->dbestclient) (0.13.1)
Requirement already satisfied: toml>=0.9.4 in c:\users\shyam\anaconda3\lib\site-packages (from tox->dbestclient) (0.10.1)
Requirement already satisfied: filelock>=3.0.0 in c:\users\shyam\anaconda3\lib\site-packages (from tox->dbestclient) (3.0.12)
Requirement already satisfied: packaging>=14 in c:\users\shyam\anaconda3\lib\site-packages (from tox->dbestclient) (20.4)
Requirement already satisfied: colorama>=0.4.1; platform_system == "Windows" in c:\users\shyam\anaconda3\lib\site-packages (from tox->dbestclient) (0.4.
4)
Requirement already satisfied: six>=1.14.0 in c:\users\shyam\anaconda3\lib\site-packages (from tox->dbestclient) (1.15.0)
Requirement already satisfied: sphinxcontrib-applehelp in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (1.0.2)
Requirement already satisfied: Pygments>=2.0 in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (2.7.2)
Requirement already satisfied: setuptools in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (50.3.1.post20201107)
Requirement already satisfied: sphinxcontrib-devhelp in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (1.0.2)
Requirement already satisfied: sphinxcontrib-serializinghtml in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (1.1.4)
Requirement already satisfied: requests>=2.5.0 in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (2.24.0)
```

Figure 1.3: Screenshot 2



```
Requirement already satisfied: setuptools in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (50.3.1.post20201107)
Requirement already satisfied: sphinxcontrib-devhelp in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (1.0.2)
Requirement already satisfied: sphinxcontrib-serializinghtml in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (1.1.4)
Requirement already satisfied: requests>=2.5.0 in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (2.24.0)
Requirement already satisfied: snowballstemmer>=1.1 in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (2.0.0)
Requirement already satisfied: babel>=1.3 in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (2.8.1)
Requirement already satisfied: alabaster<0.8,>=0.7 in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (0.7.12)
Requirement already satisfied: sphinxcontrib-jsmath in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (1.0.1)
Requirement already satisfied: sphinxcontrib-qthelp in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (1.0.3)
Requirement already satisfied: imagesize in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (1.2.0)
Requirement already satisfied: sphinxcontrib-htmlhelp in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (1.0.3)
Requirement already satisfied: Jinja2>=2.3 in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (2.11.2)
Requirement already satisfied: docutils>=0.12 in c:\users\shyam\anaconda3\lib\site-packages (from sphinx->dbestclient) (0.16)
Requirement already satisfied: smart-open>=1.8.1 in c:\users\shyam\anaconda3\lib\site-packages (from gensim->dbestclient) (5.2.1)
Requirement already satisfied: platformdirs<3,>=2 in c:\users\shyam\anaconda3\lib\site-packages (from virtualenv!=20.0.0,!=20.0.1,!=20.0.2,!=20.0.3,!=20
.0.4,!=20.0.5,!=20.0.6,!=20.0.7,>=16.0.0->tox->dbestclient) (2.5.1)
Requirement already satisfied: distlib<1,>=0.3.1 in c:\users\shyam\anaconda3\lib\site-packages (from virtualenv!=20.0.0,!=20.0.1,!=20.0.2,!=20.0.3,!=20.
0.4,!=20.0.5,!=20.0.6,!=20.0.7,>=16.0.0->tox->dbestclient) (0.3.4)
Requirement already satisfied: chardet<4,>=3.0.2 in c:\users\shyam\anaconda3\lib\site-packages (from requests>=2.5.0->sphinx->dbestclient) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in c:\users\shyam\anaconda3\lib\site-packages (from requests>=2.5.0->sphinx->dbes
tclient) (1.25.11)
Requirement already satisfied: idna<3,>=2.5 in c:\users\shyam\anaconda3\lib\site-packages (from requests>=2.5.0->sphinx->dbestclient) (2.10)
Requirement already satisfied: MarkupSafe>=0.23 in c:\users\shyam\anaconda3\lib\site-packages (from Jinja2>=2.3->sphinx->dbestclient) (1.1.1)
```

Figure 1.4: Screenshot 3



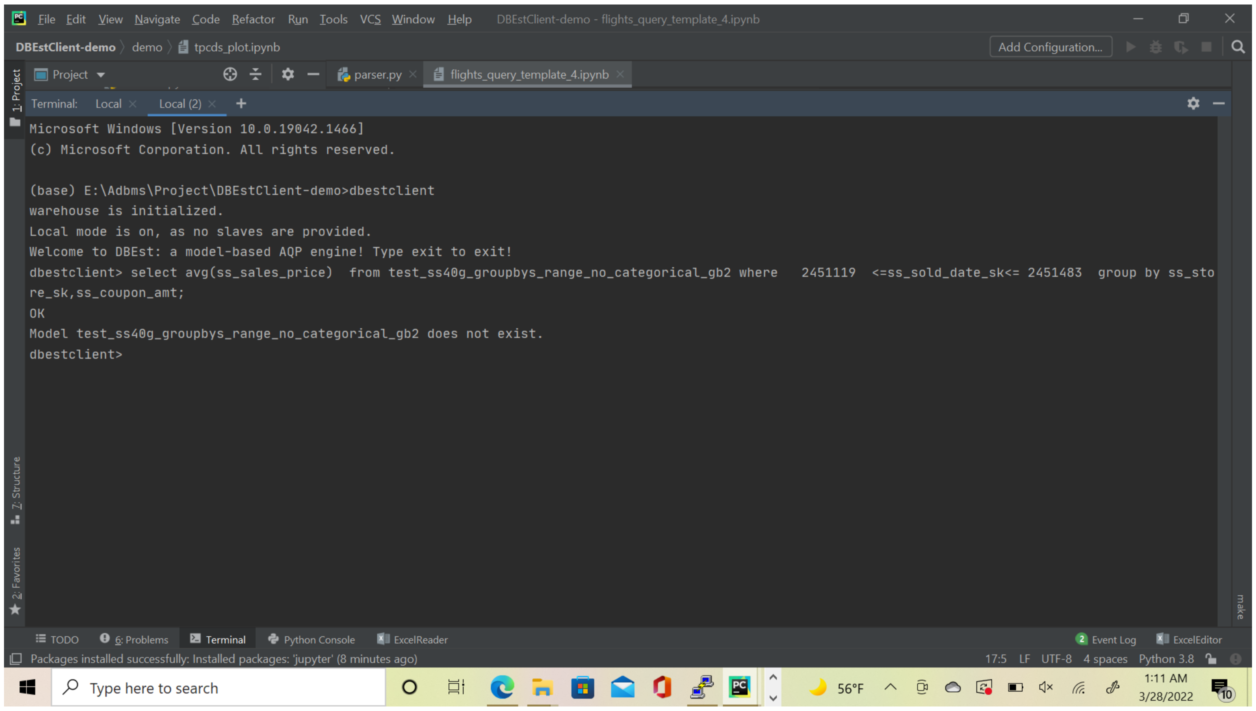
```
Microsoft Windows [Version 10.0.19042.1466]
(c) Microsoft Corporation. All rights reserved.

(base) E:\Adbms\Project\DBEstClient-demo>dbestclient
warehouse is initialized.
Local mode is on, as no slaves are provided.
Welcome to DBEst: a model-based AQP engine! Type exit to exit!
dbestclient> SHOW tables;
OK
Time cost: 0.0000s.
dbestclient>
```

DBEstClient-demo - flights_query_template_4.ipynb
DBEstClient-demo 〉 demo 〉 tpcds_plot.ipynb
Add Configuration...
parser.py
flights_query_template_4.ipynb
Terminal: Local Local (2)
Packages installed successfully: Installed packages: 'jupyter' (7 minutes ago)
Type here to search
56°F
1:09 AM
3/28/2022

Figure 1.5: show table execution

1.3.3 Project schedule status

As mentioned in the time table of the progress report 1 the data has been collected which was already provided from the people who has created the DbEst++ architecture. The



```
Microsoft Windows [Version 10.0.19042.1466]
(c) Microsoft Corporation. All rights reserved.

(base) E:\Adbms\Project\DBEstClient-demo>dbestclient
warehouse is initialized.
Local mode is on, as no slaves are provided.
Welcome to DBEst: a model-based AQP engine! Type exit to exit!
dbestclient> select avg(ss_sales_price)  from test_ss40g_groupbys_range_no_categorical_gb2 where   2451119  <=ss_sold_date_sk<= 2451483  group by ss_sto
re_sk,ss_coupon_amt;
OK
Model test_ss40g_groupbys_range_no_categorical_gb2 does not exist.
dbestclient>
```

Figure 1.6: select query execution

data are the Filght data and TPC-DS. TPC-DS 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. The initial setup of the architecture is accomplished as provided in screenshots "The same as in Project Progress Report 1". Able to figure out the issues with the version and able to set up and use the environment with different types of queries. Also updated the existing DBEST++ environment to be available for the current versions. The part we are behind is writing the code which adds in the delete and update component to the query parser for which now we also know the design to build the code which is explained in section 1.3.2 and run it with the data set which is explained above.

1.4 Work to be done and Time Table

The work to be done in the following weeks is that of writing the code which adds in the delete and update component to the query parser and run it with the the benchmarked data by adding in the delete and update queries to the data which was not tired before as well.

The new model after adding these components is put to analysis which is done through by finding the relative error and response time. The relative error is the found by identifying the difference between the predicted answer from the model and the actual answer expected. This can be in the form of mean squared error.

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 starting time in which the query goes into the model and the ending time when the result comes out of the model and finding the difference between both.

So once these work is done the difference between the previous model and this model is that the new model now will be well equipped to execute the delete and update statements as well which was not possible in the previous versions.

The time table for the rest of the project is as follows

Tasks	Starting date	Ending date	Deliverables	Person-in-charge
Report Preparation	03/21/2022	03/28/2022	Submit report	Ramya, Shyam
Coding for delete	03/28/2022	04/06/2022	Delete component created	Shyam
Coding for update	04/06/2022	04/13/2022	update component created	Ramya
Rerun algorithm	04/13/2022	04/19/2022	Executing objective	Shyam, Ramya
Analyzing algorithm	04/19/2022	04/26/2022	Analyzing objective	Shyam, Ramya
Final Report	04/26/2022	05/02/2022	Submit report	Shyam,Ramya
Final Presentation	04/26/2022	05/02/2022	Presentation	Ramya,Shyam

Bibliography

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.

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.

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.

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.

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.

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

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

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.