## Name & Title: - Big Data Analytics Management System

**Introduction**: -Big Data Analytics Management System is a powerful tool for organisations that deal with massive amounts of data. It involves the use of advanced analytics techniques to process, analyse and derive insights from large and complex data sets. This system enables businesses to make data-driven decisions and gain a competitive advantage in their respective industries. It encompasses various technologies and platforms that aid in data ingestion, storage, processing, visualisation, and reporting. With its ability to handle big data, the Big Data Analytics Management System has become a crucial tool for modern businesses seeking to leverage the power of data.

### Enums Used:

| Enum Name | Description |
|---|---|
| INT | Represents integer data type |
| FLOAT | Represents floating-point data type |
| STRING | Represents string or character data type |

### Structures Used:

Data Point:

| Data Member | Description |
|---|---|
| type | A variable of type DataType that represents the data type of the value stored in the DataPoint. |
| value | A pointer to a memory location that stores the value of the DataPoint. The data type of the value stored in this memory location is determined by the type data member. |

Data Set:

| Data Member | Description |
|---|---|
| size | An integer variable representing the size of the data set |
| data | A pointer to an array of DataPoint structures that store the actual data points |

**Problem Description:**

The following functions will be used for the assignment:

| Function Name | Parameter | Return Type | Description |
|---|---|---|---|
| createDataPoint | DataType type, void *value | DataPoint * | This function creates a new data point with the given data type and value. It allocates memory for the new data point and value, copies the value to the data point and sets the fields of the new data point. It returns a pointer to the newly created data point. If the data type is not valid or if there is a memory allocation failure, the function returns NULL. |
| createDataSet | int size | DataSet * | This function creates a new data set with the given size. It allocates memory for the data set struct and for the data points, initialises the data points to NULL, and sets the size and data fields of the data set. It returns a pointer to the newly created data set. If the size is not valid or if there is a memory allocation failure, the function returns NULL. |
| addDataPoint | DataSet *dataset, int index | void | This function adds a new data point to the specified index of the given data set. It checks if the dataset and point are not NULL, if the index is within the bounds of the dataset, and if the data point type matches the dataset type. If the checks pass, it frees the existing data point at the specified index, if any, allocates memory for the data point value, and copies it from the input point. If there is a memory allocation failure, the function does not modify the dataset and returns. |
| getDataPoint | DataSet *dataset, int index | DataPoint * | This function returns a pointer to the data point at the specified index of the given data set. It checks if the dataset and index are valid and if the data point and value are not NULL. If all checks pass, it returns a pointer to the data point. If any of the checks fail, the function returns NULL. |
| freeDataSet | DataSet *dataset | void | This function frees the memory allocated for the given data set. If the data set has no data points, it frees the data set struct itself. If the data set has data points, it frees each data point and the value pointed to by the data point. |
| filterByType | DataSet *dataset, DataType type | DataSet* | This function takes a pointer to a DataSet structure and a DataType as input parameters. It creates a new DataSet structure to hold the filtered data points and returns a pointer to this new structure. The function loops through all the data points in the input DataSet, checks if the data point is of the specified type, creates a new data point with the same type and value, and adds it to the filtered DataSet. The function returns NULL if the input DataSet is NULL or if the creation of the filtered DataSet fails. The function also checks for NULL |

| | | | |
|---|---|---|---|
| | | | pointers before accessing or adding data points to the data sets. |

**Test Cases** :

| Test case name | Input | Output | Explanation |
|---|---|---|---|
| testCreateDataPointInt | value = 42 | point != NULL | Creates a data point with INT data type and a value of 42. It checks if the point is not null, if its data type is INT, and if its value matches the input value. It also frees the memory of the data point and its value. |
| testCreateDataPointFloat | value = 3.14f | point != NULL | Creates a data point with FLOAT data type and a value of 3.14. It checks if the point is not null, if its data type is FLOAT, and if its value matches the input value. It also frees the memory of the data point and its value. |
| testCreateDataPointString | value = "hello" | point != NULL | Creates a data point with STRING data type and a value of "hello". It checks if the point is not null, if its data type is STRING, and if its value matches the input value. It also frees the memory of the data point and its value. |
| testCreateDataSetWithPositiveSize | size = 5 | dataset != NULL | Creates a data set with a size of 5. It checks if the dataset is not null and if its size matches the input size. It also checks if each data point in the dataset has a data type of INT and a null value. It frees the memory of the data set. |
| testCreateDataSetWithZeroSize | size = 0 | dataset == NULL | Creates a data set with a size of 0. It checks if the dataset is null. |
| testCreateDataSetWithNegativeSize | size = -5 | dataset == NULL | Creates a data set with a size of -5. It checks if the dataset is null. |
| testAddIntegerDataPoint | dataset size = 3, value = 10 | result->type = INT, *((int*)*result->value) = 10 | Creates a data set with a size of 3 and a data point with an INT data type and a value of 10. It adds the data point to the dataset at index 0. It checks if the result |

| | | | |
|---|---|---|---|
| | | | has a data type of INT and if its value matches the input value. It frees the memory of the data set and the data point. |
| testAddDataPointWithInvalidIndex | dataset size = 2, value = 12 | result = NULL | Creates a data set with a size of 2 and a data point with an INT data type and a value of 12. It adds the data point to the dataset at index 2, which is an invalid index. It checks if the result is null. It frees the memory of the data set and the data point. |
| testAddDataPointWithExistingValue | createDataSet(1), value=6, valueI=15 | result->type=INT, *((int *)result->value) =15 | A dataset is created with a size of 1. Two data points are added to this dataset at index 0, one with a value of 6 and another with a value of 15. The getDataPoint function is then used to retrieve the data point at index 0, which should have a value of 15 after the first data point with a value of 6 is freed. |
| testGetDataPointWithNullDataset | NULL, 0 | result=NULL | The getDataPoint function is called with a null dataset and index 0, which should return null since there are no data points in the dataset. |
| testGetDataPointFromNullDataSet | NULL, 0 | result=NULL | The getDataPoint function is called with a null dataset and index 0, which should return null since there are no data points in the dataset. |
| testGetDataPointWithInvalidIndex | createDataSet(3), index=10; createDataSet(3), index=3 | result=NULL; result=NULL | The getDataPoint function is called with an index that is out of range for the dataset, which should return null since there is no data point at that index. |
| testFreeDataSetWithNoDataPoints | createDataSet(0) | dataset=NULL | A dataset with size 0 is created and then freed. The freeDataSet function should set the dataset pointer to null. |
| testFreeDataSetWithOneDataPoint | createDataSet(1), value=10 | dataset!=NULL | A dataset with size 1 is created and then freed. The freeDataSet function should not set the dataset pointer to null because there is still one data point that needs to be freed. |
| testFreeDataSetWithDifferentDataTypes | createDataSet(3), value1=10, value2=5.5f | dataset!=NULL | A dataset with size 3 is created and then freed. The freeDataSet function should not set the dataset pointer to null because |

| | | | |
|---|---|---|---|
| | , value3="hello" | | there are still three data points that need to be freed. |
| testFilterByTypeWithNullDataset | NULL, INT | filteredData=NULL | The filterByType function is called with a null dataset and an integer data type, which should return null since there are no data points in the dataset. |
| testFilterByTypeWithEmptyDataset | createDataSet(0), FLOAT | filteredData=NULL | A dataset with size 0 is created and then filtered by the float data type. The filterByType function should return null since there are no data points in the dataset. |
| testDatasetWithNoMatchingType | createDataSet(3), value1=10, value2=20, value3=30.0, type=STRING | filteredData!=NULL, filteredData->size=3 | A dataset with size 3 is created and three data points are added to it: two integers and one float. The filterByType function is then called with the string data type, which should return a dataset with size 3 since none of the data points match the requested data type |