Department of Computer Science

Faculty of Engineering, Built Environment & IT

University of Pretoria

COS110 - Program Design: Introduction

Assignment 2 Specifications

Release Date: 01-09-2025 at 06:00

FitchFork Submissions Open: 08-09-2025

Due Date: 10-10-2025 at 23:59

Late Deadline: 11-10-2025 at 00:59

Total: 400

Read the entire specification before starting with the assignment.

Changes to the specification are indicated in orange.

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you begin coding.***
- This assignment should be completed individually.
- **Every submission will be inspected with the help of dedicated plagiarism detection software.**
- Be ready to upload your assignment well before the deadline. There is a late deadline which is 1 hour after the initial deadline which has a penalty of 20% of your achieved mark. **No extensions will be granted.**
- If your code does not compile, you will be awarded a mark of 0. The output of your program will be primarily considered for marks, although internal structure may also be tested (eg. the presence/absence of certain functions or structure).
- Failure of your program to successfully exit will result in a mark of 0.
- Note that plagiarism is considered a very serious offence. Plagiarism will not be tolerated, and disciplinary action will be taken against offending students. Please refer to the University of Pretoria's plagiarism page at `https://portal.cs.up.ac.za/files/departmental-guide/`.
- Unless otherwise stated, the usage of C++11 or additional libraries outside of those indicated in the assignment, will **not** be allowed. Some of the appropriate files that you have submit will be overwritten during marking to ensure compliance to these requirements. **Please ensure you use C++98**
- All functions should be implemented in the corresponding `cpp` file. No inline implementation in the header file apart from the provided functions.
- The usage of ChatGPT and other AI-Related software to generate submitted code is strictly forbidden and will be considered as plagiarism.

2 Overview

Databases are a cornerstone of any business in the modern age. In this assignment you will be implementing a single table database, which will illustrate the power of inheritance or operator overloading.

3 Your Task:

You are required to implement the following class diagram illustrated in Figure 1. Pay close attention to the function signatures as the `h` files will be overwritten, thus failure to comply with the UML, will result in a mark of 0.

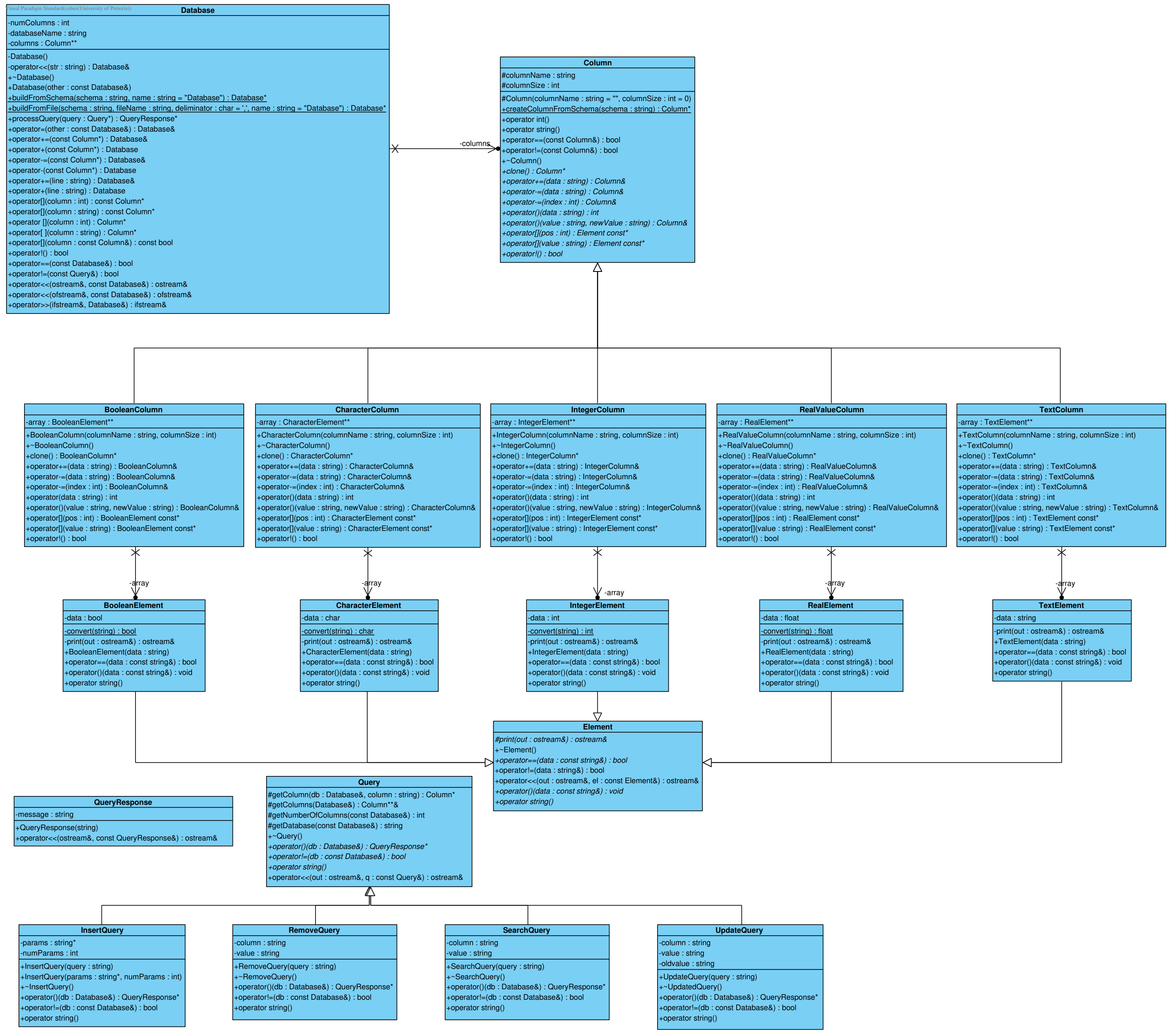

Visual Paradigm Standard(cobus(University of Pretoria))
Database
-numColumns : int
-databaseName : string
-columns : Column**
-Database()
-operator<<(str : string) : Database&
+~Database()
+Database(other : const Database&)
+buildFromSchema(schema : string, name : string = "Database") : Database*
+buildFromFile(schema : string, fileName : string, deliminator : char = ',', name : string = "Database") : Database*
+processQuery(query : Query*) : QueryResponse*
+operator=(other : const Database&) : Database&
+operator+=(const Column*) : Database&
+operator+(const Column*) : Database
+operator-=(const Column*) : Database&
+operator-(const Column*) : Database
+operator+=(line : string) : Database&
+operator+(line : string) : Database
+operator[](column : int) : const Column*
+operator[](column : string) : const Column*
+operator [](column : int) : Column*
+operator[ ](column : string) : Column*
+operator[](column : const Column&) : const bool
+operator!() : bool
+operator==(const Database&) : bool
+operator!=(const Query&) : bool
+operator<<(ostream&, const Database&) : ostream&
+operator<<(ofstream&, const Database&) : ofstream&
+operator>>(ifstream&, Database&) : ifstream&
-columns
Column
#columnName : string
#columnSize : int
#Column(columnName : string = "", columnSize : int = 0)
+createColumnFromSchema(schema : string) : Column*
+operator int()
+operator string()
+operator==(const Column&) : bool
+operator!=(const Column&) : bool
+~Column()
+clone() : Column*
+operator+=(data : string) : Column&
+operator-=(data : string) : Column&
+operator-=(index : int) : Column&
+operator()(data : string) : int
+operator()(value : string, newValue : string) : Column&
+operator[](pos : int) : Element const*
+operator[](value : string) : Element const*
+operator!() : bool
BooleanColumn
-array : BooleanElement**
+BooleanColumn(columnName : string, columnSize : int)
+~BooleanColumn()
+clone() : BooleanColumn*
+operator+=(data : string) : BooleanColumn&
+operator-=(data : string) : BooleanColumn&
+operator-=(index : int) : BooleanColumn&
+operator(data : string) : int
+operator()(value : string, newValue : string) : BooleanColumn&
+operator[](pos : int) : BooleanElement const*
+operator[](value : string) : BooleanElement const*
+operator!() : bool
CharacterColumn
-array : CharacterElement**
+CharacterColumn(columnName : string, columnSize : int)
+~CharacterColumn()
+clone() : CharacterColumn*
+operator+=(data : string) : CharacterColumn&
+operator-=(data : string) : CharacterColumn&
+operator-=(index : int) : CharacterColumn&
+operator()(data : string) : int
+operator()(value : string, newValue : string) : CharacterColumn&
+operator[](pos : int) : CharacterElement const*
+operator[](value : string) : CharacterElement const*
+operator!() : bool
IntegerColumn
-array : IntegerElement**
+IntegerColumn(columnName : string, columnSize : int)
+~IntegerColumn()
+clone() : IntegerColumn*
+operator+=(data : string) : IntegerColumn&
+operator-=(data : string) : IntegerColumn&
+operator-=(index : int) : IntegerColumn&
+operator()(data : string) : int
+operator()(value : string, newValue : string) : IntegerColumn&
+operator[](pos : int) : IntegerElement const*
+operator[](value : string) : IntegerElement const*
+operator!() : bool
RealValueColumn
-array : RealElement**
+RealValueColumn(columnName : string, columnSize : int)
+~RealValueColumn()
+clone() : RealValueColumn*
+operator+=(data : string) : RealValueColumn&
+operator-=(data : string) : RealValueColumn&
+operator-=(index : int) : RealValueColumn&
+operator()(data : string) : int
+operator()(value : string, newValue : string) : RealValueColumn&
+operator[](pos : int) : RealElement const*
+operator[](value : string) : RealElement const*
+operator!() : bool
TextColumn
-array : TextElement**
+TextColumn(columnName : string, columnSize : int)
+~TextColumn()
+clone() : TextColumn*
+operator+=(data : string) : TextColumn&
+operator-=(data : string) : TextColumn&
+operator-=(index : int) : TextColumn&
+operator()(data : string) : int
+operator()(value : string, newValue : string) : TextColumn&
+operator[](pos : int) : TextElement const*
+operator[](value : string) : TextElement const*
+operator!() : bool
-array
-array
-array
-array
-array
BooleanElement
-data : bool
-convert(string) : bool
-print(out : ostream&) : ostream&
+BooleanElement(data : string)
+operator==(data : const string&) : bool
+operator()(data : const string&) : void
+operator string()
CharacterElement
-data : char
-convert(string) : char
-print(out : ostream&) : ostream&
+CharacterElement(data : string)
+operator==(data : const string&) : bool
+operator()(data : const string&) : void
+operator string()
IntegerElement
-data : int
-convert(string) : int
-print(out : ostream&) : ostream&
+IntegerElement(data : string)
+operator==(data : const string&) : bool
+operator()(data : const string&) : void
+operator string()
RealElement
-data : float
-convert(string) : float
-print(out : ostream&) : ostream&
+RealElement(data : string)
+operator==(data : const string&) : bool
+operator()(data : const string&) : void
+operator string()
TextElement
-data : string
-print(out : ostream&) : ostream&
+TextElement(data : string)
+operator==(data : const string&) : bool
+operator()(data : const string&) : void
+operator string()
Element
#print(out : ostream&) : ostream&
+~Element()
+operator==(data : const string&) : bool
+operator!=(data : string&) : bool
+operator<<(out : ostream&, el : const Element&) : ostream&
+operator()(data : const string&) : void
+operator string()
QueryResponse
-message : string
+QueryResponse(string)
+operator<<(ostream&, const QueryResponse&) : ostream&
Query
#getColumn(db : Database&, column : string) : Column*
#getColumns(Database&) : Column**&
#getNumberOfColumns(const Database&) : int
#getDatabase(const Database&) : string
+~Query()
+operator()(db : Database&) : QueryResponse*
+operator!=(db : const Database&) : bool
+operator string()
+operator<<(out : ostream&, q : const Query&) : ostream&
InsertQuery
-params : string*
-numParams : int
+InsertQuery(query : string)
+InsertQuery(params : string*, numParams : int)
+~InsertQuery()
+operator()(db : Database&) : QueryResponse*
+operator!=(db : const Database&) : bool
+operator string()
RemoveQuery
-column : string
-value : string
+RemoveQuery(query : string)
+~RemoveQuery()
+operator()(db : Database&) : QueryResponse*
+operator!=(db : const Database&) : bool
+operator string()
SearchQuery
-column : string
-value : string
+SearchQuery(query : string)
+~SearchQuery()
+operator()(db : Database&) : QueryResponse*
+operator!=(db : const Database&) : bool
+operator string()
UpdateQuery
-column : string
-value : string
-oldvalue : string
+UpdateQuery(query : string)
+~UpdatedQuery()
+operator()(db : Database&) : QueryResponse*
+operator!=(db : const Database&) : bool
+operator string()

Figure 1: Class diagrams

Note, the importance of the arrows between the classes are not important for COS 110 and will be expanded on in COS 214.

In essence, the basic idea of the assignment is to implement a **database** which contains a series of **columns**. Each **column** contains a specific **element** that is closely linked with the data type that the column stores. In order to interact with the **database**, a series of **queries** can be passed to the **database**. A row can be thought of as a collection of related elements in the database, while a element can be the intersection between a column and a row.

All of the inheritance used in the assignment is **public**. In the derived classes, you will notice that some functions have been **omitted**. This is to help reduce the length of the specification. The description used in the base class should be followed to implement the function in the derived class. Use the provided UML diagrams to determine when a function is `static` or `pure-virtual`.

3.1 Element Hierarchy

The `Element` class is the base class for the element hierarchy and is used to constrain the derived classes. Figure 2 contains the classes that form the element hierarchy.

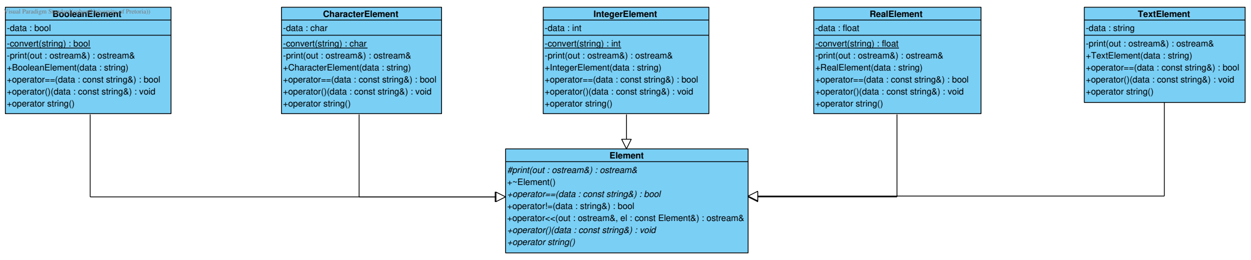

Visual Paradigm Standard(cobus(University of Pretoria))
BooleanElement
-data : bool
-convert(string) : bool
-print(out : ostream&) : ostream&
+BooleanElement(data : string)
+operator==(data : const string&) : bool
+operator()(data : const string&) : void
+operator string()
CharacterElement
-data : char
-convert(string) : char
-print(out : ostream&) : ostream&
+CharacterElement(data : string)
+operator==(data : const string&) : bool
+operator()(data : const string&) : void
+operator string()
IntegerElement
-data : int
-convert(string) : int
-print(out : ostream&) : ostream&
+IntegerElement(data : string)
+operator==(data : const string&) : bool
+operator()(data : const string&) : void
+operator string()
RealElement
-data : float
-convert(string) : float
-print(out : ostream&) : ostream&
+RealElement(data : string)
+operator==(data : const string&) : bool
+operator()(data : const string&) : void
+operator string()
TextElement
-data : string
-print(out : ostream&) : ostream&
+TextElement(data : string)
+operator==(data : const string&) : bool
+operator()(data : const string&) : void
+operator string()
Element
#print(out : ostream&) : ostream&
+~Element()
+operator==(data : const string&) : bool
+operator!=(data : string&) : bool
+operator<<(out : ostream&, el : const Element&) : ostream&
+operator()(data : const string&) : void
+operator string()

Figure 2: Element Hierarchy

3.1.1 Element

The `Element` class, which has been given to you, has the following functions. DO NOT MODIFY THIS FILE.

3.1.1.1 Functions

- Function 1
  - This function will be used to populate the `ostream` parameter, with a string representation of the relevant derived class.
  - In each of the derived classes, the function should be a constant function as well as a virtual function.
- Function 2
  - This is the destructor for the `Element` class.
- Function 3
  - This function needs to determine if the passed-in parameter, when converted using the derived class' `convert` function, is the same as the member variable or not.
  - If they are the same, the function should return `true`, else `false`.
- Function 4
  - This function is used to determine if an `Element` object is not equal to the passed-in `string`.
  - This function has been implemented for you.
- Function 5
  - This function is the overloaded stream insertion operator.
  - This function has been implemented for you.

- Function 6
  - This function is used to update the value of the element to the passed-in `string`.
- Function 7
  - This is a virtual function, which is also constant.
  - This is a conversion operator which will convert the `Element` object to a string.
  - This function should return a `string` representation of the data member, using the reverse rules of the `convert` function.

3.1.2 BooleanElement

The `BooleanElement` class is used to store elements whose data is of type `bool`. The `BooleanElement` class has the following member variables and functions. The member and function order is the same as shown in Figure 2.

3.1.2.1 Members

- Member 1
  - This is the data stored inside objects of the class.

3.1.2.2 Functions

- Function 1
  - This function needs to convert the `string` to a `bool`.
  - In order to do this, if the `string` is equal to `"T"`, then the function should return `true` else `false`.
- Function 3
  - This is the constructor and should populate the member variable using the passed-in parameter.

3.1.3 CharacterElement

The `CharacterElement` class is used to store elements whose data is of type `char`. The `CharacterElement` class has the following member variables and functions. The member and function order is the same as shown in Figure 2.

3.1.3.1 Members

- Member 1
  - This is the data stored inside objects of the class.

3.1.3.2 Functions

- Function 1
  - This function needs to convert the `string` to a `char`.
  - In order to do this, take the first character in the passed-in `string` as the result.
  - If the string is empty, return a space (`" "`).
- Function 3
  - This is the constructor and should populate the member variable using the passed-in parameter.

3.1.4 IntegerElement

The `IntegerElement` class is used to store elements whose data is of type `int`. The `IntegerElement` class has the following member variables and functions. The member and function order is the same as shown in Figure 2.

3.1.4.1 Members

- Member 1
  - This is the data stored inside objects of the class.

3.1.4.2 Functions

- Function 1
  - This function needs to convert the `string` to a `int`.
- Function 3
  - This is the constructor and should populate the member variable using the passed-in parameter.

3.1.5 RealElement

The `RealElement` class is used to store elements whose data is of type `float`. The `RealElement` class has the following member variables and functions. The member and function order is the same as shown in Figure 2.

3.1.5.1 Members

- Member 1
  - This is the data stored inside objects of the class.

3.1.5.2 Functions

- Function 1
  - This function needs to convert the `string` to a `float`.
- Function 3
  - This is the constructor and should populate the member variable using the passed-in parameter.

3.1.6 TextElement

The `TextElement` class is used to store elements whose data is of type `string`. The `TextElement` class has the following member variables and functions. The member and function order is the same as shown in Figure 2.

3.1.6.1 Members

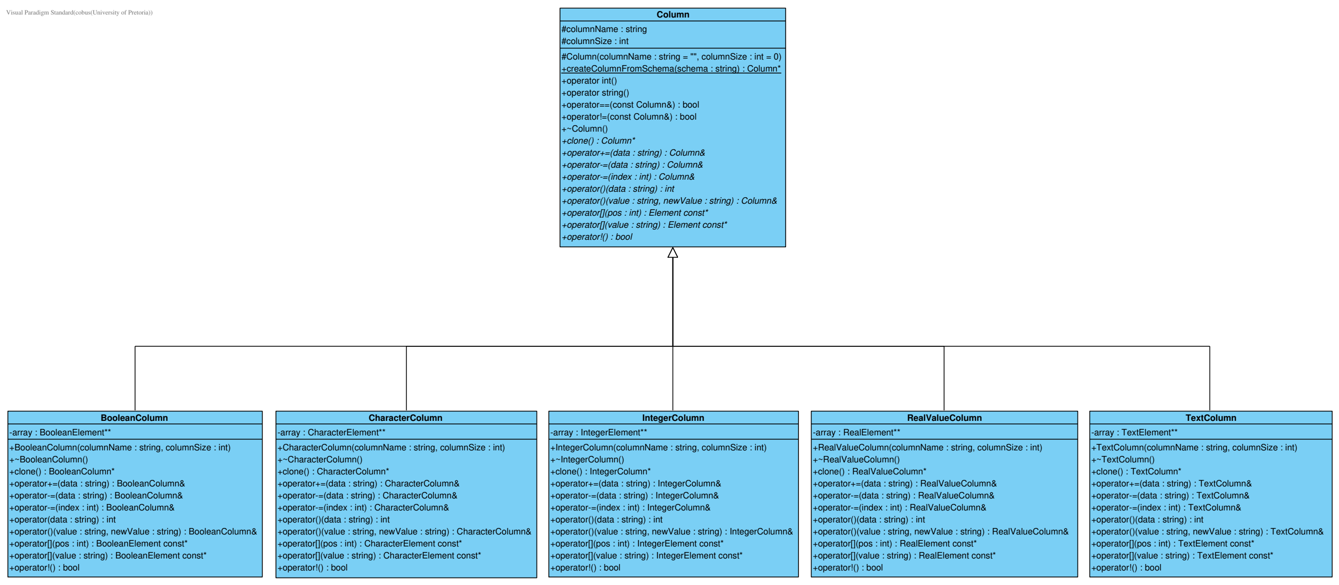
- Member 1
  - This is the data stored inside objects of the class.

3.1.6.2 Functions

- Function 2
  - This is the constructor and should populate the member variable using the passed-in parameter.
- Function 5
  - This is a virtual function.
  - This function should return the data member.

3.2 Column Hierarchy

The `Column` class is the base class for the column hierarchy and is used to constrain the derived classes. Figure 3 contains the classes that form the column hierarchy.



Visual Paradigm Standard(cobus(University of Pretoria))
Column
#columnName : string
#columnSize : int
#Column(columnName : string = "", columnSize : int = 0)
+createColumnFromSchema(schema : string) : Column*
+operator int()
+operator string()
+operator==(const Column&) : bool
+operator!=(const Column&) : bool
+~Column()
+clone() : Column*
+operator+=(data : string) : Column&
+operator-=(data : string) : Column&
+operator-=(index : int) : Column&
+operator()(data : string) : int
+operator()(value : string, newValue : string) : Column&
+operator[](pos : int) : Element const*
+operator[](value : string) : Element const*
+operator!() : bool
BooleanColumn
-array : BooleanElement**
+BooleanColumn(columnName : string, columnSize : int)
+~BooleanColumn()
+clone() : BooleanColumn*
+operator+=(data : string) : BooleanColumn&
+operator-=(data : string) : BooleanColumn&
+operator-=(index : int) : BooleanColumn&
+operator(data : string) : int
+operator()(value : string, newValue : string) : BooleanColumn&
+operator[](pos : int) : BooleanElement const*
+operator[](value : string) : BooleanElement const*
+operator!() : bool
CharacterColumn
-array : CharacterElement**
+CharacterColumn(columnName : string, columnSize : int)
+~CharacterColumn()
+clone() : CharacterColumn*
+operator+=(data : string) : CharacterColumn&
+operator-=(data : string) : CharacterColumn&
+operator-=(index : int) : CharacterColumn&
+operator()(data : string) : int
+operator()(value : string, newValue : string) : CharacterColumn&
+operator[](pos : int) : CharacterElement const*
+operator[](value : string) : CharacterElement const*
+operator!() : bool
IntegerColumn
-array : IntegerElement**
+IntegerColumn(columnName : string, columnSize : int)
+~IntegerColumn()
+clone() : IntegerColumn*
+operator+=(data : string) : IntegerColumn&
+operator-=(data : string) : IntegerColumn&
+operator-=(index : int) : IntegerColumn&
+operator()(data : string) : int
+operator()(value : string, newValue : string) : IntegerColumn&
+operator[](pos : int) : IntegerElement const*
+operator[](value : string) : IntegerElement const*
+operator!() : bool
RealValueColumn
-array : RealElement**
+RealValueColumn(columnName : string, columnSize : int)
+~RealValueColumn()
+clone() : RealValueColumn*
+operator+=(data : string) : RealValueColumn&
+operator-=(data : string) : RealValueColumn&
+operator-=(index : int) : RealValueColumn&
+operator()(data : string) : int
+operator()(value : string, newValue : string) : RealValueColumn&
+operator[](pos : int) : RealElement const*
+operator[](value : string) : RealElement const*
+operator!() : bool
TextColumn
-array : TextElement**
+TextColumn(columnName : string, columnSize : int)
+~TextColumn()
+clone() : TextColumn*
+operator+=(data : string) : TextColumn&
+operator-=(data : string) : TextColumn&
+operator-=(index : int) : TextColumn&
+operator()(data : string) : int
+operator()(value : string, newValue : string) : TextColumn&
+operator[](pos : int) : TextElement const*
+operator[](value : string) : TextElement const*
+operator!() : bool

Figure 3: Column Hierarchy

The UML for BooleanColumn's `operator()` was updated to include the correct signature.

3.2.1 Column

The `Column` class has the following members and functions.

3.2.1.1 Members

- Member 1
  - This is the name associated with the current object.
- Member 2
  - This is the number of elements that can be stored in the current object.

3.2.1.2 Functions

- Function 1
  - This is the constructor and should initialise the member variables to the passed-in parameters.
  - If the `columnSize` is less than 0, set the `columnSize` member variable to 0.
- Function 2
  - This function is used to create a column based on a predefined format (also known as a schema).
  - The format is as follows:

    ```
    Column Name:Column Type
    ```

    An example of a schema is provided below:

    ```
    firstName:text
    ```

  - It can be assumed that the format will be correct, but the column type needs to be converted to lower case.

- Using Table 1, create the correct column type and populate it with the passed in column name and a size of 0.

Column type in schema	**Column type class**
`bool`	`BooleanColumn`
`char`	`CharacterColumn`
`int`	`IntegerColumn`
`real`	`RealValueColumn`
`text`	`TextColumn`

Table 1: Column Type Association Rules

- If the column type does not match any of the types in Table 1, set the column type to type `text`.
- Return the newly created column.

- Function 3
  - This is a constant function.
  - This conversion operator should return the size of the column.
- Function 4
  - This is a constant function.
  - This conversion operator should return the name of the column.
- Function 5
  - This is a constant function.
  - This function should determine if the passed-in column is the same as the current column.
  - For two columns to be equal, the following properties must hold:
    1. The column names should match.
    2. The column size should match.
    3. Each element at index i in the one column should be equal to the element at index i in the other column.
  - If the columns are equal, return `true`, else return `false`.
- Function 6
  - This is a constant function.
  - This function should determine if the two columns are not equal using the same properties as the above function.
  - If the two columns are not equal, the function should return `true`, else the function should return `false`.
- Function 7
  - This is a virtual function.
  - This destructor should deallocate any dynamic memory allocated, if any was allocated.
- Function 8
  - This is a constant function.
  - This function will make a clone of a column object.
- Function 9

  - This function will append the new element to the column by increasing the array and then inserting.
  - The function should then return the current object.

- Function 10
  - This function will remove the element that contains the passed-in data from the column.
  - The function should only remove the first match.
  - The function should then return the current object.
  - If an element was removed, this function should resize the array so there are no gaps.

- Function 11
  - This function will remove the element that is at the passed-in index.
  - If the index is invalid, the function should not modify the column.
  - The function should then return the current object.
  - If an element was removed, this function should resize the array so there are no gaps.

- Function 12
  - This is a constant function.
  - This function should return the index of the passed-in data in the column.
  - If the data is not contained in the column, the function should return -1.

- Function 13
  - This function should iterate through the column and update only the first occurrence of the element in the column whose data is equal to the first parameter, by setting the element's data to the second parameter.
  - The function should then return the current object.

- Function 14
  - This is a constant function.
  - This function should return the `Element` pointer of the element at the passed-in index.
  - If the index is invalid, the function should return `NULL`.

- Function 15
  - This is a constant function.
  - This function should return the `Element` pointer of the element whose data member matches the passed-in parameter.
  - If no match can be found, the function should return `NULL`.

- Function 16
  - This is a constant function.
  - This function needs to determine if the column is valid or not.
  - For a column to be valid, the following properties must hold:
    1. The array of the specific subclass should not be `NULL`.
    2. The size of the column should be greater than or equal to 0.
    3. The column should be full, i.e., not have any gaps.
  - If the column is valid, the function should return `true`; else return `false`.

3.2.2 BooleanColumn

The `BooleanColumn` is used to store pointers to dynamic objects of type `BooleanElement`. The `BooleanColumn` contains the following members and functions:

3.2.2.1 Members

- Member 1
  - This is a dynamic 1D array of dynamic `BooleanElement` objects of size `columnSize`.

3.2.2.2 Functions

- Function 1
  - This is the class's constructor and should initialise the relevant member variables appropriately.
- Function 2
  - This is the destructor for the class and should deallocate any dynamic memory allocated to the class, if any was allocated.
- Function 3
  - This is a constant function.
  - This function should return a new `BooleanColumn` object that is a deep copy of the current object.
- Function 10
  - This function works exactly as described in the `Column` class' description of the function, except that the return type is `BooleanElement`.

3.2.3 CharacterColumn

The `CharacterColumn` is used to store pointers to dynamic objects of type `CharacterElement`. The `CharacterColumn` contains the following members and functions:

3.2.3.1 Members

- Member 1
  - This is a dynamic 1D array of dynamic `CharacterElement` objects of size `columnSize`.

3.2.3.2 Functions

- Function 1
  - This is the class's constructor and should initialise the relevant member variables appropriately.
- Function 2
  - This is the destructor for the class and should deallocate any dynamic memory allocated to the class, if any was allocated.
- Function 3
  - This is a constant function.
  - This function should return a new `CharacterColumn` object that is a deep copy of the current object.
- Function 10
  - This function works exactly as described in the `Column` class' description of the function, with the exception that the return type is `CharacterElement`.

3.2.4 IntegerColumn

The `IntegerColumn` is used to store pointers to dynamic objects of type `IntegerElement`. The `IntegerColumn` contains the following members and functions:

3.2.4.1 Members

- Member 1
  - This is a dynamic 1D array of dynamic `IntegerElement` objects of size `columnSize`.

3.2.4.2 Functions

- Function 1
  - This is the class's constructor and should initialise the relevant member variables appropriately.
- Function 2
  - This is the destructor for the class and should deallocate any dynamic memory allocated to the class, if any was allocated.
- Function 3
  - This is a constant function.
  - This function should return a new `IntegerColumn` object that is a deep copy of the current object.
- Function 10
  - This function works exactly as described in the `Column` class' description of the function, except that the return type is `IntegerElement`.

3.2.5 RealValueColumn

The `RealValueColumn` is used to store pointers to dynamic objects of type `RealElement`. The `RealValueColumn` contains the following members and functions:

3.2.5.1 Members

- Member 1
  - This is a dynamic 1D array of dynamic `RealElement` objects of size `columnSize`.

3.2.5.2 Functions

- Function 1
  - This is the class's constructor and should initialise the relevant member variables appropriately.
- Function 2
  - This is the destructor for the class and should deallocate any dynamic memory allocated to the class, if any was allocated.
- Function 3
  - This is a constant function.
  - This function should return a new `RealValueColumn` object that is a deep copy of the current object.
- Function 10
  - This function works exactly as described in the `Column` class' description of the function, with the exception that the return type is `RealElement`.

3.2.6 TextColumn

The `TextColumn` is used to store pointers to dynamic objects of type `TextElement`. The `TextColumn` contains the following members and functions:

3.2.6.1 Members

- Member 1
  - This is a dynamic 1D array of dynamic `TextElement` objects of size `columnSize`.

3.2.6.2 Functions

- Function 1
  - This is the class's constructor and should initialise the relevant member variables appropriately.
- Function 2
  - This is the destructor for the class and should deallocate any dynamic memory allocated to the class, if any was allocated.
- Function 3
  - This is a constant function.
  - This function should return a new `TextColumn` object that is a deep copy of the current object.
- Function 10
  - This function works exactly as described in the `Column` class' description of the function, with the exception that the return type is `TextElement`.

3.3 QueryReponse

The `QueryResponse` class is used to communicate the result of querying the database back to the user. Figure 4 contains the UML for the `QueryResponse` class.

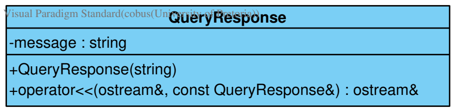


Figure 4: QueryResponse Class

3.3.1 QueryResponse

The `QueryResponse` class has the following members and functions:

3.3.1.1 Members

- Member 1
  - This is the message that contains the response to the query.

3.3.1.2 Functions

- Function 1
  - This is the constructor for the class and should initialise the member variables.
- Function 2
  - This function should populate the passed-in `ostream` member with the `message` member variable, and then return the appropriate passed-in parameter.

3.4 Query Hierarchy

The Query class is the base class for the query hierarchy and is used to constrain the derived classes. Figure 5 contains the classes that form the query hierarchy.

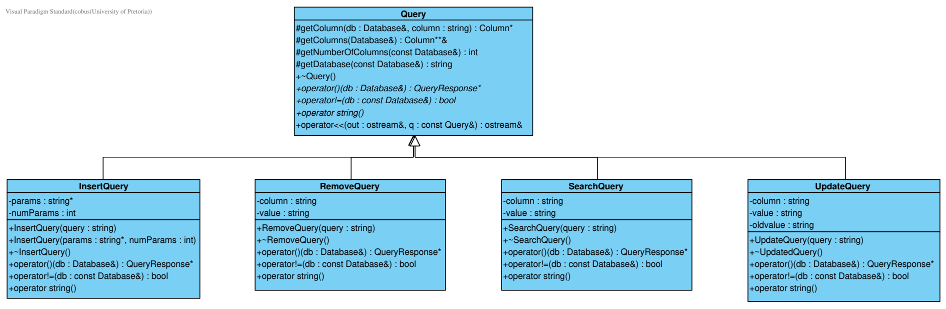

Visual Paradigm Standard(cobus(University of Pretoria))
Query
#getColumn(db : Database&, column : string) : Column*
#getColumns(Database&) : Column**&
#getNumberOfColumns(const Database&) : int
#getDatabase(const Database&) : string
+~Query()
+operator()(db : Database&) : QueryResponse*
+operator!=(db : const Database&) : bool
+operator string()
+operator<<(out : ostream&, q : const Query&) : ostream&
InsertQuery
-params : string*
-numParams : int
+InsertQuery(query : string)
+InsertQuery(params : string*, numParams : int)
+~InsertQuery()
+operator()(db : Database&) : QueryResponse*
+operator!=(db : const Database&) : bool
+operator string()
RemoveQuery
-column : string
-value : string
+RemoveQuery(query : string)
+~RemoveQuery()
+operator()(db : Database&) : QueryResponse*
+operator!=(db : const Database&) : bool
+operator string()
SearchQuery
-column : string
-value : string
+SearchQuery(query : string)
+~SearchQuery()
+operator()(db : Database&) : QueryResponse*
+operator!=(db : const Database&) : bool
+operator string()
UpdateQuery
-column : string
-value : string
-oldvalue : string
+UpdateQuery(query : string)
+~UpdatedQuery()
+operator()(db : Database&) : QueryResponse*
+operator!=(db : const Database&) : bool
+operator string()

Figure 5: Query Hierarchy

3.4.1 Query

The Query class has the following functions. There exists a friend relation between the Database class and the Query class, such that the private members of Database can be accessed.

3.4.1.1 Functions

- Function 1
  - This is a constant function.
  - This function should return the column whose name matches the passed-in string parameter from the passed-in Database object.
  - If no column matches, the function should return NULL.
- Function 2
  - This is a constant function.
  - This function should return the dynamic array of columns that is stored in the passed-in Database object.
- Function 3
  - This is a constant function.
  - This function should return the number of columns that are contained in the passed-in Database object.
- Function 4
  - This is a constant function.
  - This function should return the name of the database as stored in the passed-in Database object.
- Function 5
  - This is a virtual function.
  - This is the destructor for the Query class and should deallocate any dynamic memory allocated, if any was allocated.
- Function 6
  - This function will perform the query on the passed in Database object and return a QueryResponse pointer.
  - If the query was unsuccessful, the function should return NULL.
- Function 7

  - This is a constant function.
  - This function should validate if the query can be performed on the passed-in `Database` object.
  - If the query can be performed, the function should return `true`, else it should return `false`.

- Function 8
  - This is a constant function.
  - This conversion operator will return a string representation of the query.

- Function 9
  - This function should populate the passed-in `ostream` object with the string representation of the passed-in `Query` object.
  - There should be a new line after the string representation of the `Query` object.
  - The function should return the populated `ostream` object.

3.4.2 InsertQuery

The `InsertQuery` class will be used to insert data into the database and has the following members and functions.

3.4.2.1 Members

- Member 1
  - This is a 1D array that contains the data that needs to be inserted into the database.
  - Each value in the array will be formatted as follows:

    ```
    Column_name:Data
    ```

    An example of a value will be:

    ```
    LastName:Bond
    ```

- Member 2
  - This is the number of parameters in the parameters array.

3.4.2.2 Functions

- Function 1
  - This is the constructor for the `InsertQuery` class and receives a string with the parameters.
  - The format of the string is as follows:

    ```
    Column_1:value1,Column_2:value2,...,Column_N:valueN
    ```

  - Note that the order of the columns in the input string does **not** need to be in the same order as the columns in the database.
  - An example of the input:

    ```
    firstName:James,lastName:Bond,codeName:007
    ```

  - The constructor should populate the member variables accordingly.

- Function 2
  - This is another constructor for the `InsertQuery` class and receives an array and the size of the array as parameters.

- The constructor needs to populate the member variables with the passed-in parameters using deep copies where applicable.
- If the passed-in parameter is invalid (NULL or less than 0), initialise the array to an array of size 0.

- Function 3
  - This is a virtual function.
  - This is the destructor for the InsertQuery class and should deallocate any dynamically allocated memory, if any was allocated.
- Function 4
  - This function should execute the query on the passed-in Database object.
  - This is done by parsing each of the parameters in the array and inserting the relevant data into the specific column of the passed-in Database object.
  - If the query cannot be performed on the passed-in Database object, the function should not modify the passed-in parameter and simply return NULL.
  - If the query was successfully executed, the function should return a new QueryResponse object with the message: "Row successfully inserted".
- Function 5
  - This is a constant function.
  - This function should determine if the query can be performed on the passed-in Database object.
  - In order for the query to be performed, the following properties need to hold:
    1. The number of parameters in the query should match the number of columns in the Database object.
    2. Each of the columns in the query should match with each of the columns in the passed-in Database object.
  - If the query cannot be performed the function should return true, else the function should return false.
- Function 6
  - This is a virtual function, which is also constant.
  - This function should return the string representation of the query.
  - The format is as follows:

```
INSERT INTO DATABASE (Column_1:value1,Column_2:value2,...,ColumnN:valueN)
```

An example of the string is as follows:

```
INSERT INTO DATABASE (firstName:James,lastName:Bond,codeName:007)
```

3.4.3 RemoveQuery

The RemoveQuery class will be used to remove data from the database given some condition, and has the following members and functions.

3.4.3.1 Members

- Member 1
  - This is the name of the column which will be used as the condition for removal.
- Member 2
  - This is the value that needs to match in the column, such that the row will be removed from the database.

3.4.3.2 Functions

- Function 1
  - This is the constructor for the RemoveQuery class and receives a string with the parameters.
  - The format of the string is as follows:
    ```
    Column_1:value_1
    ```
  - An example of the input:
    ```
    firstName:James
    ```
  - The constructor should populate the member variables accordingly.
  - It can be assumed that the format of the input string is correct.
- Function 2
  - This is a virtual function.
  - This is the destructor for the RemoveQuery class and should deallocate any dynamically allocated memory, if any was allocated.
- Function 3
  - This function deletes the first row in the database where the element in the specified column matches the value member variable.
  - Only the first match should be removed.
  - If the column is *firstName* and the value is *James*, then the function should go through the *firstName* column and check if any element matches with *James*, and remove that entire row from the database.
  - The function should initially perform a validation to ensure that the query is valid for the passed-in Database object.
  - If the query is invalid, the function should not modify the passed-in Database object and simply return NULL.
  - If the query was successfully executed and a row was removed, the function should return a new QueryResponse object with the message: "Row successfully removed".
  - If the query was successfully executed, but a row was not removed, the function should return a new QueryResponse object with the message: "No rows were removed".
- Function 4
  - This is a constant function.
  - This function should determine if the query can be performed on the passed-in Database object.
  - For the query to be able to be performed, the following property needs to hold:
    1. The column whose name is stored in the member variable is a column in the passed-in Database object.
  - If the query cannot be performed, the function should return true, else the function should return false.
- Function 5
  - This is a virtual function, which is also constant.
  - This function should return the string representation of the query.
  - The format is as follows:
    ```
    DELETE FROM DATABASE WHERE Column_1 = value_1
    ```
    An example is as follows:
    ```
    DELETE FROM DATABASE WHERE firstName = James
    ```

3.4.4 SearchQuery

The `SearchQuery` class will be used to search for a certain row in the database which will be based on some given condition, and has the following members and functions.

3.4.4.1 Members

- Member 1
  - This is the name of the column which will be used as the condition for searching.
- Member 2
  - This is the value that needs to match in the column, such that the row will be returned from the database.

3.4.4.2 Functions

- Function 1
  - This is the constructor for the `SearchQuery` class and receives a `string` with the parameters.
  - The format of the `string` is as follows:

    ```
    Column_1:value_1
    ```

  - An example of the input:

    ```
    firstName:James
    ```

  - The constructor should populate the member variables accordingly.
  - It can be assumed that the format of the input string is correct.
- Function 2
  - This is a virtual function.
  - This is the destructor for the `SearchQuery` class and should deallocate any dynamically allocated memory, if any was allocated.
- Function 3
  - This function searches for a row in the database where the element in the specified `column` matches the `value` member variable.
  - Only the first match should be returned.
  - The function should initially perform a validation to ensure that the query is valid for the passed-in `Database` object.
  - If the query is invalid, the function should not modify the passed-in `Database` object and simply return `NULL`.
  - If the query was successfully executed and a match was found, the function should return a new `QueryResponse` object with the message formatted as follows: `"value_1,value_2,value_3"`. The order should be the same order as the columns in the passed-in `Database` object.
  - If the query was successfully executed and no match was found, the function should return a new `QueryResponse` object with the message: `"No records found"`.
- Function 4
  - This is a constant function.
  - This function should determine if the query can be performed on the passed-in `Database` object.

  - For the query to be able to be performed, the following property needs to hold:
    1. The column whose name is stored in the member variable is a column in the passed-in `Database` object.
  - If the query cannot be performed, the function should return `true`, else the function should return `false`.

- Function 5
  - This is a virtual function, which is also constant.
  - This function should return the `string` representation of the query.
  - The format is as follows:

```
SELECT FROM DATABASE WHERE Column_1 = value_1
```

An example is as follows:

```
SELECT FROM DATABASE WHERE firstName = James
```

3.4.5 UpdateQuery

The `UpdateQuery` class will be used to update a certain value for a certain row in the database that will be based on some given condition, and has the following members and functions.

3.4.5.1 Members

- Member 1
  - This is the name of the column which will be used as the condition for updating.
- Member 2
  - This is the new value that will replace the value of the element that matches with the `oldValue` member variable, if a match is found.
- Member 3
  - This is the value that needs to match in the column, such that the value can be updated.

3.4.5.2 Functions

- Function 1
  - This is the constructor for the `UpdateQuery` class and receives a string with the parameters.
  - The format of the string is as follows:

```
Column_1:old_value_1;value_1
```

  - An example of the input:

```
firstName:Johannes;James
```

  - The constructor should populate the member variables accordingly.
  - It can be assumed that the format of the input string is correct.
- Function 2
  - This is a virtual function.
  - This is the destructor for the `UpdateQuery` class and should deallocate any dynamically allocated memory, if any was allocated.

- Function 3
  - This function searches for a row in the database where the element in the specified `column` matches the `value` member variable. Once the element has been found, your code should change the element's data to the data contained in `value`.
  - Only the first match should be updated.
  - The function should initially perform a validation to ensure that the query is valid for the passed-in `Database` object.
  - If the query is invalid, the function should not modify the passed-in `Database` object and simply return `NULL`.
  - If the query was successfully executed and an element was updated, the function should return a new `QueryResponse` object with the message: `"Updated a row"`.
  - If the query was successfully executed and no element was updated, the function should return a new `QueryResponse` object with the message: `"No row was updated"`.
- Function 4
  - This is a constant function.
  - This function should determine if the query can be performed on the passed-in `Database` object.
  - For the query to be able to be performed, the following property needs to hold:
    1. The column whose name is stored in the member variable is a column in the passed-in `Database` object.
  - If the query cannot be performed, the function should return `true`, else the function should return `false`.
- Function 5
  - This is a virtual function, which is also constant.
  - This function should return the `string` representation of the query.
  - The format is as follows:

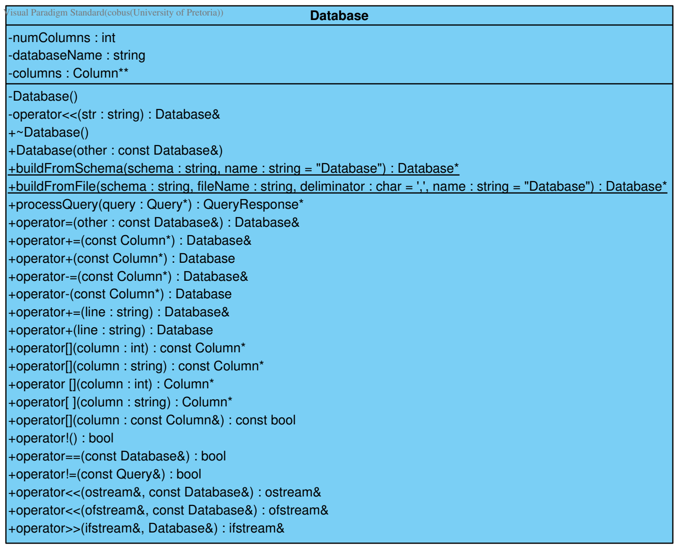
    ```
    UPDATE DATABASE SET Column_1 = value_1 WHERE Column_1 = oldValue_1
    ```

    An example is as follows:

    ```
    UPDATE DATABASE SET firstName = James WHERE firstName = Johannes
    ```

3.5 Database

The `Database` class is the class that will act as the context for the entire assignment. Figure 6 contains the UML for the `Database` class. The `Database` class has a `friend` relation with the `Query` class such that the `Query` class can access the private members of the `Database` class.



```
-numColumns : int
-databaseName : string
-columns : Column**
-Database()
-operator<<(str : string) : Database&
+~Database()
+Database(other : const Database&)
+buildFromSchema(schema : string, name : string = "Database") : Database*
+buildFromFile(schema : string, fileName : string, deliminator : char = ',', name : string = "Database") : Database*
+processQuery(query : Query*) : QueryResponse*
+operator=(other : const Database&) : Database&
+operator+=(const Column*) : Database&
+operator+(const Column*) : Database
+operator-=(const Column*) : Database&
+operator-(const Column*) : Database
+operator+=(line : string) : Database&
+operator+(line : string) : Database
+operator[](column : int) : const Column*
+operator[](column : string) : const Column*
+operator [](column : int) : Column*
+operator[ ](column : string) : Column*
+operator[](column : const Column&) : const bool
+operator!() : bool
+operator==(const Database&) : bool
+operator!=(const Query&) : bool
+operator<<(ostream&, const Database&) : ostream&
+operator<<(ofstream&, const Database&) : ofstream&
+operator>>(ifstream&, Database&) : ifstream&
```

Figure 6: Database Class

3.5.1 Database

The `Database` class has the following members and functions:

3.5.1.1 Members

- Member 1
  - This member indicates the number of columns in the database.
- Member 2
  - This member contains the name of the database.
- Member 3
  - This member is a dynamic 1D array of dynamic `Column` objects which represent the columns stored in the database.

3.5.1.2 Functions

- Function 1
  - This is the constructor for the Database class and should initialise the respective member variables to the following values: `-1`, `"Database"`, `NULL`.
- Function 2
  - This function receives a string representation and needs to insert it into the database.
  - The format for the `string` is as follows:

```
value1,value2,...,valueN
```

  - Note that the order of the values in the input string can be assumed to be the same order as the columns in the database.
  - It can be assumed that the input string will be valid.
  - The function should return the modified `Database` object.
- Function 3

- This is the destructor for the `Database` class and should deallocate any dynamic memory that was allocated, if any was allocated.

- Function 4
  - This is the copy constructor for the `Database` class and should make a deep copy of the passed-in `Database` object.

- Function 5
  - This function is used to create a `Database` object from the passed-in schema and return it.
  - The schema has the following format:

    ```
    Column_1_Name:Column_1_Type,...,Column_N_Name:Column_N_Type
    ```

    An example is given below:

    ```
    firstName:text,lastName:text,age:int
    ```

  - If the passed-in schema is empty(`""`), return a new `Database` object with the name initialised to the passed-in parameter. The number of columns and columns should be 0.
  - The function needs to create a new `Database` object and add each of the columns, with an initial size of 0, in the order they were provided in the schema.
  - The function should also initialise all the other member variables of the `Database` object to the appropriate values/parameters.
  - It can be assumed that the schema's format is valid.
  - Ensure this function works perfectly as it is used during testing to build your database.

- Function 6
  - This function is similar to the previous function in the sense that a database needs to be built from the passed-in schema and name, using the same rules as previously mentioned.
  - This function also needs to populate the `Database` object using the contents of the passed-in file.
  - The content in the file is separated by the passed-in delimiter.
  - It can be assumed that the file is valid, and that the contents of the file are in the same order as the columns in the database. Ensure this function works perfectly as it is used during testing to build and populate your database.

- Function 7
  - This function needs to have the passed-in query be performed on the current `Database` object.
  - The function should return the `QueryResponse` result that was obtained from executing the query.
  - If the passed-in query is `NULL`, the function should simply return `NULL`.
  - The function also needs to verify that the query is valid before the query is performed on the database.

- Function 8
  - This is the assignment operator and should create a deep-copy of the passed-in `Database` object using the methods discussed in class.

- Function 9
  - This operator needs to **append** a **deep-copy** of the passed-in column to the columns of the database, and return the current object.

  - If a column with the same name already exists in the database or the passed-in column is NULL, do not modify the current Database object.

- Function 10
  - This is a constant function.
  - This operator needs to create a copy of the current Database object and append a deep-copy of the passed-in Column object to the copy.
  - The function should then return the copy.
  - If a column with the same name already exists in the copy or the passed-in column is NULL, do not modify the copy.

- Function 11
  - This operator needs to remove the column from the current Database object, whose name matches the name of the passed-in column.
  - If the passed-in parameter is NULL, or is not in the database, the function should **not** modify the current Database object.
  - The function should return the current Database object.
  - Note that after the column has been removed, the columns array needs to be resized, so that no "gaps" remain.
  - Reminder to deallocate the column being removed.

- Function 12
  - This is a constant function.
  - This operator needs to create a copy of the current Database object and remove the passed-in column from the copy.
  - The function should then return the copy.
  - If a column with the same name does not exist in the copy or the passed-in parameter is NULL, do not modify the copy.
  - Note that after the column has been removed, the columns array needs to be resized, so that no "gaps" remain.
  - Reminder to deallocate the column being removed.

- Function 13
  - This operator receives a string representation and needs to insert it into the database.
  - The format for the string is as follows:

```
Column_1:value1,Column_2:value2,...,Column_N:valueN
```

  - Note that the order of the columns in the input string does **not** need to be in the same order as the columns in the database.
  - It can be assumed that the input string will be valid.
  - The function should return the current Database object.

- Function 14
  - This is a constant function.
  - This operator receives a string representation and needs to insert it into a copy of the current Database object.

- The format for the string is as follows:

```
Column_1:value1,Column_2:value2,...,Column_N:valueN
```

- Note that the order of the columns in the input string does **not** need to be in the same order as the columns in the database.
- It can be assumed that the input will be valid.
- The function should return the modified copy of the `Database` object.

- Function 15
  - This is a constant function.
  - This operator needs to return the column at the passed-in index.
  - If the index is invalid, the operator should return `NULL`.
- Function 16
  - This is a constant function.
  - This operator needs to return the column whose name matches the passed-in parameter.
  - If no column matches, the operator should return `NULL`.
- Function 17
  - This operator needs to return the column at the passed-in index.
  - If the index is invalid, the operator should return `NULL`.
- Function 18
  - This operator needs to return the column whose name matches the passed-in parameter.
  - If no column matches, the operator should return `NULL`.
- Function 19
  - This is a constant function.
  - This operator needs to determine if a column in the database has the same name as the passed-in column or not.
  - If there is return `true`, else return `false`.
- Function 20
  - This is a constant function.
  - This operator needs to determine if the database is valid or not.
  - For the database to be valid, the following properties must hold:
    1. Number of columns must be greater than or equal to 0.
    2. The columns array cannot be `NULL`.
    3. The columns array cannot contain any gaps.
    4. None of the columns are invalid.
    5. All of the columns have the same number of elements.
  - If the database is invalid, return `true`, else return `false`.
- Function 21
  - This is a constant function.

  - This operator needs to determine if the current database object and the passed-in database object are equal or not.
  - For two databases to be equal, the following properties must hold:
    1. Both databases have the same number of columns.
    2. The order of the columns in the databases is the same.
    3. Each column in the one database is the same as the corresponding column in the other database.
  - If the two databases are equal, return `true`, else return `false`.
  - If both of the column arrays in both of the databases are `NULL`, the databases are also seen as equal.

- Function 22
  - This is a constant function.
  - This operator needs to determine if the passed-in query is valid for the database or not, using the query validation rules specified in the query section.

- Function 23
  - This operator needs to print out the database to the passed-in `ostream` parameter as a comma-delimited list, including a trailing comma at the end of each line of output.
  - The first line should be the names of each of the columns in the database.
  - The remaining lines should be each of the rows in the database.
  - There should be a `newline` at the end of each of the lines.
  - If the database is invalid, the operator should insert the following message into the `ostream` parameter (without a `newline` at the end) and then return the parameter:

    ```
    "Invalid Database"
    ```

  - An example of the output is as follows:

    ```
    firstName,lastName,studentNumber,
    Elizabeth,Thomas,904729,
    Joseph,Gonzalez,790563,
    ```

- Function 24
  - This operator needs to print out the database to the passed-in `ofstream` parameter as a markdown table[1].
  - The first line of text in the table should be the names of the columns in the order they are stored in the database.
  - The remaining lines should be each of the rows in the database.
  - If the database is invalid, the operator should insert the following message into the `ofstream` parameter (without a `newline` at the end) and then return the parameter:

    ```
    "Invalid Database"
    ```

  - An example of the markdown table is as follows:

    ```
    |firstName|lastName|studentNumber|
    |-|-|-|
    |Elizabeth|Thomas|904729|
    |Joseph|Gonzalez|790563|
    ```

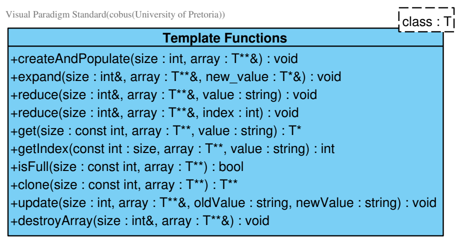
[1]More information about markdown tables can be found at: `https://www.codecademy.com/resources/docs/markdown/tables`

- Function 25
  - This operator needs to read the data in from a CSV file, which can be assumed to be in the correct format, into the database.
  - It can be assumed that this function will only be called with valid arguments.

3.6 Template Functions

This namespace contains a series of template functions that **will not be** explicitly tested during evaluation on FitchFork. Due to the high amount of functions in the different inheritance hierarchies sharing the same algorithm but working with different types, these template functions can be used to help reduce the amount of code that needs to be written. Note these functions **do not** need to be implemented and can be stubbed, such that they can compile without complications.

Figure 7 contains the functions that are contained in the namespace. The description of each of the functions contains a suggestion on how the function should be implemented. Note, although you do not need to implement the functions, you **can not** change the function signatures.



```
Template Functions
+createAndPopulate(size : int, array : T**&) : void
+expand(size : int&, array : T**&, new_value : T*&) : void
+reduce(size : int&, array : T**&, value : string) : void
+reduce(size : int&, array : T**&, index : int) : void
+get(size : const int, array : T**, value : string) : T*
+getIndex(const int : size, array : T**, value : string) : int
+isFull(size : const int, array : T**) : bool
+clone(size : const int, array : T**) : T**
+update(size : int, array : T**&, oldValue : string, newValue : string) : void
+destroyArray(size : int&, array : T**&) : void
```

Figure 7: Template Functions Namespace

3.6.0.1 Functions

- Function 1
  - This function should populate the `array` parameter with a new dynamic 1D array of size `size`, and assign each index in the array to `NULL`.
- Function 2
  - This function should expand the `array` parameter by 1, update the `size` parameter and add the `new_value` to the new open position.
- Function 3
  - This function should remove the first occurrence of the `value` in the `array`, resize the `array` and update the `size`.
- Function 4
  - Function added to UML.
  - This function should remove the element at the passed-in `index`, resize the `array` and update the `size`.
- Function 5
  - This function should return the pointer at the index where the dereferenced value at that index is equal to the passed in `value`.
- Function 6
  - This function should return the index where the dereferenced value at that index is equal to the passed in `value`.

- Function 7
  - This function should determine if the passed-in `array` contains any "gaps".
- Function 8
  - This function returns a deep copy of the passed-in array.
- Function 9
  - This function needs to update the first occurrence of the `oldValue` with the `newValue`.
  - *Hint: you may need to dereference the pointer and use an operator.*
- Function 10
  - This function should deallocate the passed-in array.
  - The function has a second template parameter, which is used to indicate if the array contains dynamic objects or dynamic arrays.

4 Memory Management

As memory management is a core part of COS110 and C++, each task on FitchFork will allocate approximately 10% of the marks to memory management. The following command is used:

```
valgrind --leak-check=full ./main
```

Please ensure, at all times, that your code *correctly* de-allocates *all* the memory that was allocated.

5 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, 10% of the assignment marks will be allocated to your testing skills. To do this, you will need to submit a testing main (inside the `main.cpp` file) that will be used to test an Instructor Provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the gcov [2] tool, specifically the following version *gcov (Debian 8.3.0-6) 8.3.0*, will be used. The following set of commands will be used to run gcov:

```
g++ --coverage *.cpp -o main
./main
gcov -f -m -r -j ${files}
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

We will scale this ration based on class size.

The mark you will receive for the testing coverage task is determined using Table 2:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%

[2]For more information on gcov please see `https://gcc.gnu.org/onlinedocs/gcc/Gcov.html`

40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 2: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the functions stipulated in this specification will be considered to determine your testing mark. Remember that your main will be testing the Instructor Provided code and as such, it can only be assumed that the functions outlined in this specification are defined and implemented.

As you will be receiving marks for your testing main, we will also be doing plagiarism checks on your testing main.

6 Implementation Details

- You must implement the functions in the header files exactly as stipulated in this specification. Failure to do so will result in compilation errors on FitchFork.
- You may only use **c++98**.
- You may only utilize the specified libraries. Failure to do so will result in compilation errors on FitchFork.
- You may only use the following libraries:
  - `string`
  - `iostream`
  - `fstream`
  - `sstream`
- You are supplied with a **trivial** main demonstrating the basic functionality of this assessment.

7 Upload Checklist

The following C++ files (including the ones listed in Table 3) should be in a zip archive named uXXXXXXXX.zip where XXXXXXXX is your student number:

- Any text files/CSV used by your `main.cpp`

The files should be in the root directory of your zip file. In other words, when you open your zip file you should immediately see your files. They should not be inside another folder.

Header (.h)	**Source (.cpp)**
BooleanColumn.h	BooleanColumn.cpp
BooleanElement.h	BooleanElement.cpp
CharacterColumn.h	CharacterColumn.cpp
CharacterElement.h	CharacterElement.cpp
Column.h	Column.cpp
Database.h	Database.cpp
Element.h	
InsertQuery.h	InsertQuery.cpp
IntegerColumn.h	IntegerColumn.cpp
IntegerElement.h	IntegerElement.cpp
Query.h	Query.cpp
QueryResponse.h	QueryResponse.cpp
RealElement.h	RealElement.cpp
RealValueColumn.h	RealValueColumn.cpp
RemoveQuery.h	RemoveQuery.cpp
SearchQuery.h	SearchQuery.cpp
TemplateFunctions.h	TemplateFunctions.cpp
TextColumn.h	TextColumn.cpp
TextElement.h	TextElement.cpp
UpdateQuery.h	UpdateQuery.cpp
	main.cpp

Table 3: Header and source files in the project

8 Submission

You need to submit your source files on the FitchFork website (`https://ff.cs.up.ac.za/`). All methods need to be implemented (or at least stubbed) before submission. Your code should be able to be compiled with the following command:

```
g++ -g -std=c++98 -Wall -Werror QueryResponse.cpp BooleanElement.cpp
    CharacterElement.cpp Column.cpp IntegerElement.cpp RealElement.cpp
    TextElement.cpp BooleanColumn.cpp CharacterColumn.cpp Query.cpp
    IntegerColumn.cpp RealValueColumn.cpp TextColumn.cpp Database.cpp
    InsertQuery.cpp RemoveQuery.cpp SearchQuery.cpp UpdateQuery.cpp  main.cpp -o
    main
```

and run with the following command:

```
./main
```

Remember your **h** file will be overwritten, so ensure you do not alter the provided **h** files.
You have 5 submissions, and your best mark will be your final mark. Upload your archive to the Assignment 2 slot on the FitchFork website. If you submit after the deadline but before the late deadline, a 20% mark deduction will be applied. **No submissions after the late deadline will be accepted!**

9 Change Log

Change	Date
Added additional information to Functions 5 and 6 of Database	07/09/2025
Updated the UML for `Template Functions`, `Column Hierarchy` and `Combined UML`.	08/09/2025
Added the missing `sstream` library to the allowed libraries.	11/09/2025
Added the InsertQuery files to the upload checklist, and Function 19 of Database should be a const function. Functions 9, 10, and 11 of Column were also updated. Function 11 of database should not deallocate the column that is being removed.	14/09/2025