Библиотека DataStorage

Библиотека предназначена для работы с табличными данными. Все данные заносятся в переменную класса DataStore, в которой хранятся в виде «базы данных». По сути, переменная DataStore представляет из себя массив, состоящий из кортежей (tuple). Поэтому для получения элементов вы также можете пользоваться индексацией списка ([]) или встроенными методами для вытягивания. Например, GetColumn <column number> ().

Многие функции взяты из LINQ запросов и SQL команд. Поэтому вы можете получать не только конкретные значения, но и сортировать данные, вводить запросы и получать конечный результат, а также трансформировать существующие таблицы.

1. Объявление экземпляра класса.

DataStore содержит несколько конструкторов, которые позволяют создавать, как пустые, так и заполненные таблицы.

---- Пример ----

```
DataStore<int, int> example1; // пустая таблица
DataStore<char, string> example2(10); // пустая таблица с 10 строками
DataStore<wstring, long double> example3 = { L"Wstr", 4.12 }; // таблица с одной записью

DataStore<int, int> example4 = {
	{1,1},
	{7,7}
	}; // таблица с двумя записями
```

---- Конец примера ----

2. Вытягивание данных

Для получения данных о таблице вы можете воспользоваться следующими методами:

LineCount и ColumnCount – возвращают количество записей и столбцов в таблице.

First и Last – возвращают в виде строки первую и последнюю запись в таблице.

Contains (перегрузка) – осуществляет проверку на наличие записей в кортеже. Возвращает булево значение. Проверка производится самих записей, а для поиска элемента по столбцам смотрите пункт «Агрегатные функции».

ColumnToBox, ColumnToDeque, ColumnToList, ColumnToSet, ColumnToVector – вытягивание столбца, но не в виде строки, как это делает GetColumn<number column>, а в виде контейнера. ColumnToBox – заполняет контейнер ColumnContainer, предназначенный для обработки столбцов агрегатными функциями.

GetLine<number line> и GetColumn<number column> – возвращают в виде строки либо конкретную запись, либо столбец. При работе с колонной следует учитывать, что строка записывает каждый элемент столбца с новой строки.

GetElement<number line, number column> – позволяет вытянуть конкретный элемент из таблицы, исходя из его положения по строке и столбцу. Указывается два параметра: номер линии и номер столбца. Следует учитывать, что индексация строк и столбцов начинается с 0.

GetType<number column> – позволяет узнать тип данных конкретного столбца. Таким образом, при наличии таблицы с неизвестными типами, вы можете использоваться этим методом, чтобы узнать какие типы принимает таблица.

---- Пример ----

```
DataStore<size_t, string, string, int, string> Students = {
	{__COUNTER__, "Vadim", "PI-02", 1, "8-918..."},
	{__COUNTER__, "Egor", "PI-02", 1, "8-964..."},
	{__COUNTER__, "Veronika", "PI-01", 4, "8-918..."},
	{__COUNTER__, "Felix", "PI-02", 2, "8-964..."},
	{__COUNTER__, "Danil", "ZPI-04", 3, "8-918..."},
};

	cout << Students.GetColumn<1>(); // получение списка имен всех студентов
```

---- Конец примера ----

3. Трансформация таблицы

Эта группа методов предназначена для трансформирования таблицы. Например, сюда относятся функции, которые изменяют элементы или структуру таблицы.

AddNote (перегрузка) – метод для добавление новых записей в таблицу. Имеет ряд перегрузок, благодаря которым вы можете добавлять записи как отдельно, так и группами.

AddColumn<type column> (Fill value) – добавляет новый столбец к существующей таблице и заполняет его аргументом. Но при этом следует учитывать, что трансформируется не существующая таблица, а возвращается новая с новым столбцом.

ПРИМЕЧАНИЕ. *Используйте* ***Remember*** *совместно с AddNote для закрепления новых записей в исходной таблице, если это необходимо. Remember может быть полезен, когда нужно изменить исходное значение таблицы.*

ChangeNote (перегрузка) – изменение (редактирование) записи. Также имеет ряд перегрузок, с помощью которых можно изменять данные, основываясь на существующих однородных таблицах или абсолютно новых данных.

ChangeElement<number line, number column> – изменяет элемент, исходя из его положения по строке и столбцу.

ChangeColumn<number column> – изменяет значения столбца, исходя из содержимого контейнера ColumnContainer или списка initializer_list. Замена происходит с первого значения столбца и заменяется первым значением контейнера. Этот процесс продолжается пока не закончится переборка ColumnContainer.

ПРИМЕЧАНИЕ. ChangeColumn<number column> (lambda-expression) *для изменения параметров всех элементов столбца использует ссылки в аргументах. Например, для замены целочисленного столбца в аргументах лямбда-выражения заносится int& {имя_переменной}. Без ссылки на переменную значения столбца не будут обновлены*

DeleteNote (перегрузка) – удаление записей. Возможно удаление по индексу в таблице (по строке) и удаление по значениям.

Resize – изменение размерности таблицы. Благодаря этому методу вы можете либо расширять количество записей в таблице, либо наоборот удалять лишние.

Restore – восстановление исходной таблицы. Данный метод крайне необходим, когда ведется работа со множеством запросов. Поскольку некоторые методы трансформируют таблицу и удаляют записи при сортировке, то метод Restore позволит восстановить исходные данные, если необходимо работать с изначальной таблицей при создании новых запросов.

Union (перегрузка) – в отличие от AddNote добавляет не просто значения или список значений, а целые таблицы. При объединении таблиц следует учитывать, что обе таблицы должны быть однородными (то есть содержать одинаковое количество столбцов, а также их типы данных должны совпадать).

Clear – очистка таблицы.

---- Пример ----

```
DataStore<int, int> SumTable = { {4,1}, {5,-2}, {10, 31} };

SumTable.AddNote({
      {0,0}, {-11, 7}, {13, 2}
      });

DataStore<int, int, int> TransformedTable =
      SumTable.AddColumn(NULL)
          .ChangeColumn([&](tuple<int, int, int>& SUM)
                              {return get<2>(SUM) = get<0>(SUM) +
get<1>(SUM); });

cout << TransformedTable.GetColumn<2>();

      SumTable.~DataStore();
```

---- Конец примера ----

4. Сортировка данных

OrderBy и OrderByDescending / OrderBy<number column> и OrderByDescending<number column> – сортирует имеющуюся таблицу по возрастанию или убыванию. Оба метода содержат перегрузки, позволяющие сортировать данные по определенному столбцу.

Unique и Unique<number column> – оставляет только уникальные данные таблицы, а остальные урезает.

Repeat и Repeat<number column> – оставляет только повторяющиеся элементы. При этом дубликаты не выводятся, а также урезаются вместе с уникальными элементами.

Reverse – переворачивает контейнер. Первые элементы станут последними, а последние – первыми.

Where и Where_i / Where<number column> и Where_i<number column> – Where позволяет рассматривать только те записи, которые удовлетворяют некоторому условию, которое содержится внутри анонимной функции. Приписка «_i» означает инверсию условия. Так, если вы рассматриваете условие «>10», то результатом будут все записи со значениями в столбце «<10».

Take (перегрузка) – берет только определенные записи, а остальные, которые находятся вне указанного диапазона, урезаются.

---- Пример ----

```
initializer_list<tuple<string, string, string>> list_check = {
	{"13.02.2021", "utility", "normal"},
	{"15.02.2021", "efficiency", "good"},
	{"14.02.2021", "safety", "bad"},
	{"14.02.2021", "reliability", "normal"},
	{"16.02.2021", "functionality", "normal"}
};

DataStore<string, string, string> System_Check = list_check;

typedef tuple<string, string, string> X;
System_Check.Where([](X bad_checker) {return get<2>(bad_checker) == "bad";
});
                        // обертка для сложных условий

System_Check.Where<2>([](string X) {return X == "bad"; });
                        // обертка для конкретного столбца

for (const auto& [date, param, status] : System_Check)
                cout << "Требующий изменения параметр - " << param << endl;
```

---- Конец примера ----

5. Агрегатные функции (ColumnContainer)

ColumnContainer – контейнер, предназначенный для хранения данных определенного столбца таблицы.

Count – подсчет количества элементов.

Median – вывод центрального элемента цепочки.

Max – вывод максимального значения.

Min – вывод минимального значения.

Sum – суммирование столбца.

Average – вывод среднего значения столбца

Contains (перегрузка) – поиск элемента.

Clear – очистка контейнера. Либо можете воспользоваться деструктором.

---- Пример ----

```
DataStore<int, string, double> player_stats = {
	{3715, "help_23l", 41.221},
	{0054, "user7809", 11.087},
	{4113, "DonLiL", 31.211}
};

int MaxPoint = player_stats.ColumnToBox<2>().Max();
	cout << "Максимальное число очков на уровне: " << MaxPoint << endl;
```

---- Конец примера ----

ПРИЛОЖЕНИЕ

DataStore

Перечень функций и их перегрузки:

1. template<typename type> DataStore<Types…, type> **AddColumn** (const type Fill)

2. DataStore& **AddNote** (const Types... note)

3. DataStore& **AddNote** (const tuple<Types...> note)

4. DataStore& **AddNote** (const initializer_list<tuple<Types...>>& note_list)

5. template<size_t Column> DataStore& **ChangeColumn** (initializer_list<T> list)

6. template<size_t Column> DataStore& **ChangeColumn** (ColumnContainer<T> variable_array)

7. DataStore& **ChangeColumn** (function<void (tuple<Types...>&)> change)

8. template<size_t Column> DataStore& **ChangeColumn** (function<void(T&)>change)

9. template<size_t Line, size_t Column> DataStore& **ChangeElement** (const auto& notePart)

10. template<size_t Line> DataStore& **ChangeNote** (DataStore& variable_note)

11. template<size_t Line> DataStore& **ChangeNote** (DataStore& variable_note, const size_t index)

12. template<size_t Line> DataStore& **ChangeNote** (const Types... note)

13. constexpr size_t **ColumnCount** ()

14. template<size_t Column> constexpr ColumnContainer<T> **ColumnToBox** ()

15. template<size_t Column> constexpr deque<T> **ColumnToDeque** ()

16. template<size_t Column> constexpr list<T> **ColumnToList** ()

17. template<size_t Column> constexpr set<T> **ColumnToSet** ()

18. template<size_t Column> constexpr vector <T> **ColumnToVector** ()

19. constexpr bool **Contains** (const tuple<Types...> found)

20. constexpr bool **Contains** (DataStore& variable_note, const size_t index)

21. constexpr bool **Contains** (DataStore& variable_note)

22. constexpr void **Clear** ()

23. DataStore& **DeleteNote** (const size_t index)

24. template<size_t Line> DataStore& **DeleteNote** ()

25. DataStore& **DeleteNote** (const Types... note)

26. DataStore& **DeleteNote** (DataStore& variable_note)

27. DataStore& **DeleteNote** (DataStore& variable_note, const size_t index)

28. constexpr string **First** ()

29. template<size_t Column> constexpr string **GetColumn** ()
30. template<size_t Line> constexpr string **GetLine** ()
31. template<size_t Line, size_t Column> constexpr auto **GetElement** ()
32. template<size_t Column> constexpr string **GetType** ()
33. constexpr string **Last** ()
34. constexpr size_t **LineCount** ()
35. DataStore& **OrderBy** ()
36. template<size_t Column> DataStore& **OrderBy** ()
37. DataStore& **OrderByDescending** ()
38. template<size_t Column> DataStore& **OrderByDescending** ()
39. DataStore& **Remember** ()
40. DataStore& **Repeat** ()
41. template<size_t Column> DataStore& **Repeat** ()
42. DataStore& **Resize** (size_t size)
43. DataStore& **Restore** ()
44. DataStore& **Reverse** ()
45. DataStore& **Take** (const size_t size)
46. DataStore& **Take** (const size_t start, const size_t end)
47. DataStore& **Union** (DataStore& variable_note)
48. DataStore& **Union** (DataStore& variable_note, const size_t index)
49. DataStore& **Unique** ()
50. template<size_t Column> DataStore& **Unique** ()
51. DataStore& **Where** (function<bool (tuple<Types...>&)> sorted)
52. DataStore& **Where_i** (function<bool (tuple<Types...>&)> sorted)
53. template<size_t Column> DataStore& **Where** (function<bool (T)> sorted)
54. template<size_t Column> DataStore& **Where_i** (function<bool (T)> sorted)

ColumnContainer

Перечень функций и их перегрузки:

1. constexpr double **Average** ()
2. constexpr bool **Contains** (const Type found)
3. constexpr bool **Contains** (const initializer_list<Type> found_list)
4. constexpr size_t **Count** ()
5. constexpr void **Clear** ()
6. constexpr Type **Max** ()
7. constexpr Type **Median** ()

8. constexpr Type **Min** ()
9. constexpr double **Sum** ()