Database
`Database();`
`Database(const Database&); //Copy constructor`
`~Database();`
 	
Constructor, Copy constructor, and Destructor for Database
	Parameters
	Copy constructor can take a reference to another database. The contents of the database are copied as the initial values of the created database.
	
	Return Value
	N/A

addTable `void addTable(Table table, std::string name);` *Adds a given table to the data under a given name.*	**Parameters** `Table table` - the table to add to the database `std::string name` - the name to give the table in the database **Return value** N/A

dropTable `void dropTable(std::string tableName);` *Removes a given table from the database.*	**Parameters** `std::string tableName` - the name of the table to be removed **Return value** N/A

listTables `std::list<std::string>* listTables();` *Returns a list of the names of all of the tables in the database.*	**Parameters** N/A **Return value** A pointer to a list of names of tables in the database.

getTables `std::list<Table>* getTables();` *Returns a list of all of the tables in the database.*	**Parameters** N/A **Return value** A pointer to a list of all tables in the database

query `Table query(std::list<std::string> tableAttributes ,std::string tableName, std::string condition);` *Returns a table containing requested attributes, from a given table name, that matches a condition.*	**Parameters** `std::list<std::string> tableAttributes` - list of attributes to have in the returned table `std::string tableName` - the name of a stored table to choose data from `std::string condition` - The condition the returned table must meet **Return value** A table that has all of the attributes as required by tableAttrubutes that also satisfies the condition parameter.

deleteRecord `void deleteRecord(std::string tableName, std::string whereArgument);` *Deletes a record stored in the named table that matches conditions dictated by the “where” parameter.*	**Parameters** `std::string tableName` - the name of the table to delete the record from `std::string whereArgument` - The condition(s) that must be met in order to delete a record from the table **Return value** N/A

Table

```
Table();
Table(const Table &in);
Table(std::list<std::string>*,
std::list<Type>*);
~Table();
```

Constructor, Copy Constructor, and Destructor for Table

Parameters

`list<std::string>` - a list of attribute name
`list<Type>` - a list of attribute type

Return value

N/A

addAttribute

```
void addAttribute(std::string name, Type
type);
```

Adding a column to the end of the table with new attribute which has NULL entry

Parameters

`std::string name` - The attribute name
`Type type` - The attribute type

Return value

N/A

deleteAttribute

```
bool deleteAttribute(std::string name);
```

Takes an attribute name and deletes it from the table

Parameters

`std::string name` - The attribute name.

Return value

`bool` - It returns true if the attribute name is found and deleted; returns false if the name is not found.

insertRecord

```
void insertRecord(Record record);
```

Takes a record and adds it to the table.

Parameters

`Record record` - The record value you want to add to the table.

Return value

N/A

getAttributes

```
std::list<std::string>* getAttributes();
```

Returns a list of the attributes and types for that table.

Parameters

N/A

Return value

`std::list<std::string>*` - A list of attributes and types

getSize

```
unsigned int getSize();
```

Returns the number of records in the table.

Parameters

N/A

Return value

`unsigned int` - The number of records in the table.

renameAttribute

```
bool renameAttribute(std::string oldName, std::string newName);
```

Renames and attribute.

Parameters

`std::string oldName` - The name of the attribute you want to rename.
`std::string newName` - The new name for the attribute you want to rename.

Return value

`bool` - Return true for success, and return false for fail.

crossJoin

```
Table crossJoin(Table firstTable, Table secondTable);
```

Takes two tables as input and produces one table as output.

Parameters

`Table firstTable` - The first table to be joined.
`Table secondTable` - The second table to be joined.

Return value

`Table` - A single table resulting from the joining two specified tables.

getSum

`float getSum(std::string attributeName);`

Takes in a single attribute name and calculates the sum of all record entry values under that attribute.

Parameters

`std::string attributeName` - The specified attribute name.

Return value

`float` - The sum of all record entries under the specified attribute.

getCount

`int getCount(std::string attributeName);`

Takes in a single attribute name and calculates the number of record entries under that attribute. NULL entries are not counted.

Parameters

`std::string attributeName` - The specified attribute name.

Return value

`int`- The number of record entries under the specified attribute.

getMin

`float getMin(std::string attributeName);`

Takes in a single attribute name and finds the record entry under that attribute with the smallest value

Parameters

`std::string attributeName` - The specified attribute name.

Return value

`float`- The value of the record entry with the smallest value.

getMax

`float getMax(std::string attributeName);`

Takes in a single attribute name and finds the record entry under that attribute with the largest value.

Parameters

`std::string attributeName` - The specified attribute name.

Return value

`float`- The value of the record entry with the largest value.

Record
`Record();`
`Record(const Record &in);`
`~Record();`
 	
Constructor, Copy Constructor, and Destructor for Record
	Parameters
	Copy constructor can take a reference to another record. The contents of the record are copied as the initial values of the created record.
	
	Return value
	N/A

accessRecordEntry `std::string accessRecordEntry(int entry);` *Allows access to an individual entry in the record.*	**Parameters** `int entry` - The ith entry in the record. **Return value** `std::string` - The value of the entry. (Hey, you have access to it!)

modifyRecordEntry `void modifyRecordEntry(int entry, std::string newEntryValue);` *Modifies an individual entry in the record.*	**Parameters** `int entry` - The ith entry in the record. `std::string newEntryValue` - The new value for the entry **Return value** N/A

retrieveRecordEntry `std::string retrieveRecordEntry(int entry);` *Retrieves an individual entry in the record.*	**Parameters** `int entry` - The ith entry in the record. **Return value** `string` - The value of the entry.