Xiaoyu Li | 225002283
Michael Wakin | 822002724
Huai Wu | 825000258
CSCE 315-503
Team Assignment 1 API documentation

# 1 Global variables

`enum VAR_TYPE` with values `INT`, `INTEGER`, `FLOAT`, `REAL`, `CHAR`, `VARCHAR`, `DATE`, `TIME`.
This `enum` stores the type of data contained in a `Table`.

# 2 Public classes

`struct Attribute`
An `Attribute` object is essentially a column within a `Table`.

- Public variables
  - `std::string name`
    Hold the name of the attribute.
  - `VAR_TYPE type`
    Store the type of the attribute.
- Public functions
  - `Attribute(VAR_TYPE t, const std::string& n)`
    Constructor with type and name as arguments.
  - `Attribute(const std::string& n)`
    Constructor with name as argument.

`class Database`
A `Database` object is the core of the entire database structure and holds `Table`s.

- Public functions
  - `Database()`
    Constructor with no arguments.
  - ∼`Database()`
    Destructor.
  - `void addTable(const std::string name, Table& table)`
    Add existing table to the database with a name and the table itself.
  - `void dropTable(const std::string name)`
    Drop a `Table` object from the `Database` using its name.
  - `std::vector<std::string> listTableNames()`
    Return a list (`vector`) of `Table` names in the `Database`.
  - `Table query(std::string SELECT, std::string FROM, std::string WHERE)`
    Query the `Database`.
    `SELECT` argument is an attribute, a list of attributes separated by commas, or an '*' for all attributes.
    `FROM` argument is the name of the `Table` to be queried.
    `WHERE` argument is a logical statement using comparison operators (=, !=, <, >, <=, >=), logical conjunctions (AND, OR, NOT), or parentheses.

  - `std::map<std::string, Table> getTables()`
    Return all the `Table` objects in the `Database`.

`class Record`
A `Record` object is essentially a row within a `Table`.

- Public functions
  - `Record(std::vector<std::string> v)`
    Constructor with a vector of strings as argument to populate `Record` object.
  - `Record(int size)`
    Constructor allowing creation of a `Record` object of given size and initialization of the entries to empty strings.
  - `~Record()`
    Destructor.
  - `std::vector<std::string> getValues()`
    Return a list (`vector`) of all values as strings.
  - `std::size_t getSize()const`
    Return the size of the `Record`.
  - `std::string getEntry(int index)`
    Return an entry by index.
  - `std::vector<std::string>::const_iterator begin()const`
    Return an iterator pointing to the first `Record` object.
  - `std::vector<std::string>::const_iterator end()const`
    Return an iterator pointing after the last `Record` object.
  - `void setEntry(int index, const std::string& entry)`
    Write an entry by index.
  - `void join(const Record& other)`
    Take another record as input and produce combined `Record` object as output.

`class Table`
A `Table` object holds `Attribute` and `Record` objects, which behave as columns and rows, respectively.

- Public functions
  - `Table()`
    Constructor with no arguments.
  - `Table(std::vector<std::string> names)`
  - `~Table()`
    Destructor.
  - `Table(const Table& table)`
    Copy constructor.
  - `Table& operator = (const Table& table)`
    Copy assignment.
  - `void addColumn(std::string name)`
    Add a column (`Attribute`) to end of the table.
  - `void addColumn(Attribute attr)`
    Add a column (`Attribute`) to end of the table.
  - `void deleteColumn(std::string name)`
    Delete a column (`Attribute`) from the table.

- `void deleteColumn(Attribute attr)`
  Delete a column (`Attribute`) from the table.
- `void insertRow(Record r)`
  Add a row (`Record`) to the table.
- `std::vector<Record>::const_iterator begin() const`
  Return an iterator points to first `Record` object.
- `std::vector<Record>::const_iterator end() const`
  Return an iterator to the point after the last `Record` object.
- `const Record& first() const`
  Return the first element of rows.
- `const Record& last() const`
  Return the last element of rows.
- `Table crossJoin(Table other)`
  Take another `Table` object as input and produce a combined `Table` as output.
- `Table naturalJoin(Table other)`
  Create one entry for each row of the first `Table` object, with the additional columns from the matching key in the second `Table` object.
- `std::size_t getIndexByName(const std::string& name) const`
  Returns the index of a column by its name.
- `std::size_t getRowSize() const`
  Return the number of rows in a `Table` object.
- `std::size_t getColumnSize() const`
  Return the number of columns in a `Table` object.
- `std::vector<Attribute> getColumns() const`
  Return a list of columns in a `Table` object.
- `std::vector<Attribute> getKeys() const`
  Return a list of keys in `Table`.
- `std::vector<Record> getRows() const`
  Return a list of rows in a `Table`.
- `void setColumns(std::vector<Attribute> new_columns)`
  Assign new attributes to a `Table`.
- `void setKeys(std::vector<std::string> names)`
  Set the keys with a list (`vector`) of strings.
- `int count(std::string name) const`
  Return the number of non-null values in a give column.
- `std::string min(std::string name) const`
  Return the smallest value of a give column.
- `std::string max(std::string name) const`
  Return the largest value of a give column.
- `std::ostream& Print(std::ostream& os)`
  Output for test purposes.