Переписывание MillDB на скриптовый язык

ВЫПОЛНИЛ СТУДЕНТ ГРУППЫ ИУ9-72 Д. С. ЧУГУНОВ
РУКОВОДИТЕЛЬ КУРСОВОЙ РАБОТЫ А. В. КОНОВАЛОВ

Цели и задачи работы

Целью курсовой работы является:

1. Ознакомление с MillDB
2. Интеграция новых функциональностей
3. Реализация компилятора на скриптовом языке

Описание MillDB

- MillDB – генератор высокопроизводительной OLAP «write-only» базы данных
- Спецификация на собственном диалекте SQL
- На выходе библиотека на языке C позволяющая читать и записывать данные в базу

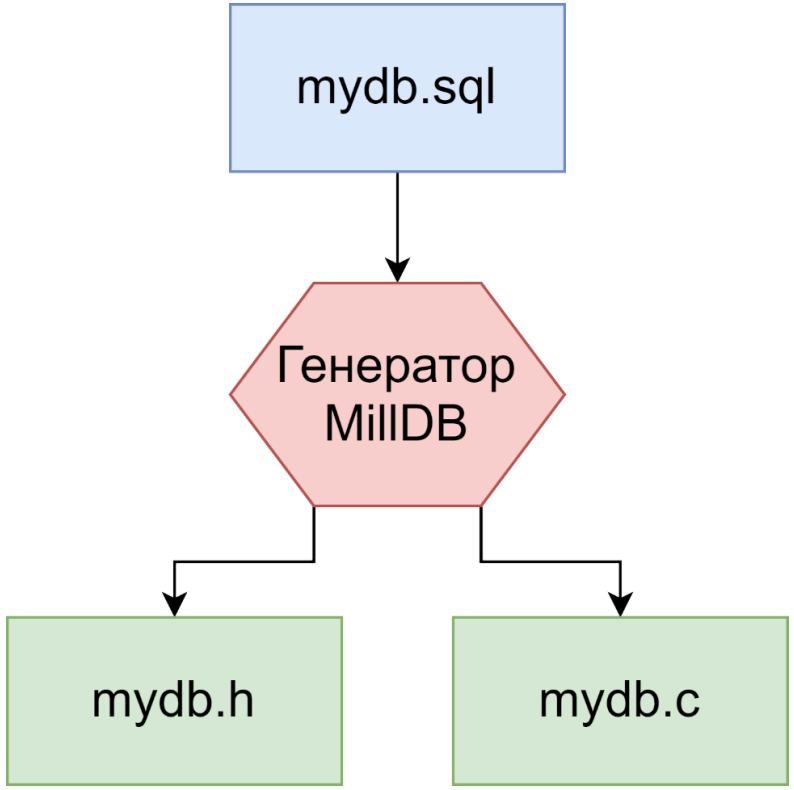

Схема работы

Интеграция новых функциональностей

- В курсовых работах были реализованы новые функциональности:
  1) Расширенный язык запросов
  2) Дополнительный индекс и фильтр Блума
- Разработка происходила независимо друг от друга
- Различие в классе колонок

Расширение логических операций

Column
+ name: str
+ type: DataType
+ pk: bool

Дополнительные индексы и фильтр Блума

Column
+ name: str
+ type: DataType
+ mod: int
+ fail_share: float

Различие в классах колонок

Реализация на C++

- Лексический анализатор – Flex
- Синтаксический анализатор – Bison
- Генерация – std::cout
- Недостатки:
  1) Сложность поддержки кода
  2) Плохая читаемость
  3) Нет сборщика мусора
  (необходимо следить за памятью)

```cpp
#include <iostream>
// Код для генерации
int main() {
    char table_name[] = "data";

    char key_type[] = "int";
    char key_name[] = "key";

    char value_type[] = "char*";
    char value_name[] = "value";

    std::cout << "struct " << table_name << " {" << std::endl;
    std::cout << "\t" << key_type << " " key_name << ";" << std::endl;
    std::cout << "\t" << value_type << " " << value_name << ";" << std::endl;
    std::cout << "}" << std::endl;

    return 0;
}
// Сгенерированный код
struct data {
    int key;
    char* value;
}
```

Генерация на языке C++

Разработка на скриптовом языке

- Лексический анализатор – на основе конечного автомата, имеет метод next_token
- Синтаксический анализатор – LL(1) (сверху вниз), вместо LR анализатора в Bison
- Компилятор – однопроходный
- Генерация – шаблонизируемые строки

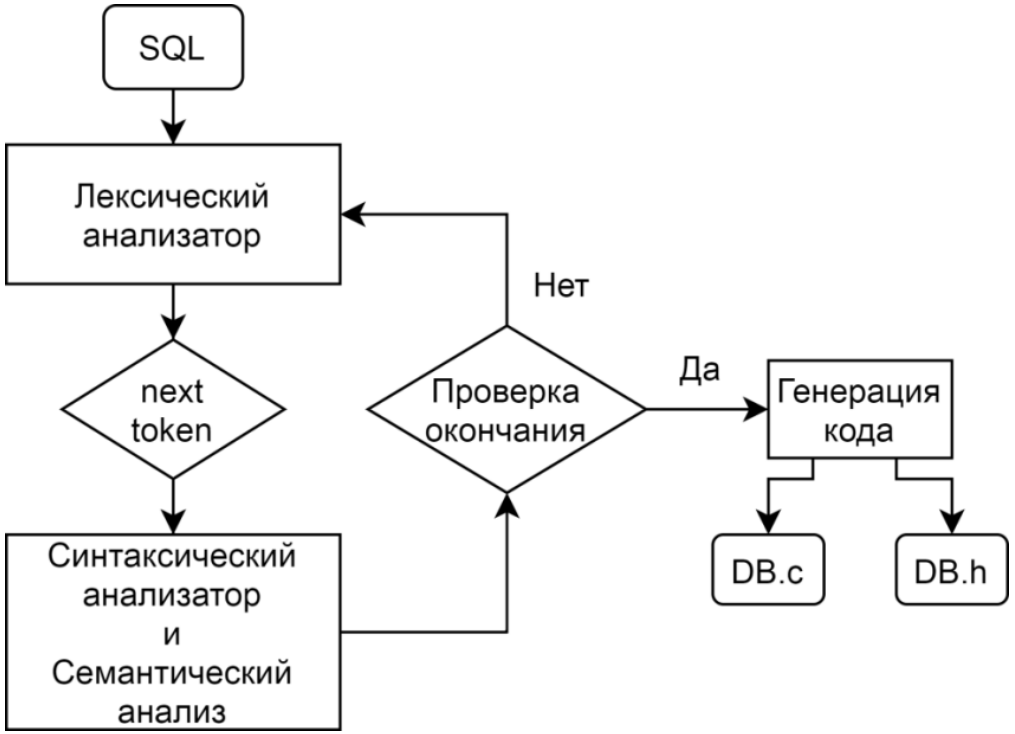

Схема работы генератора на скриптовом языке

Реализация на Python

- Лексический анализатор в основе содержит генератор
- Синтаксический анализатор основан на методе рекурсивного спуска
- Для шаблонизируемых строк используется библиотека Jinja2

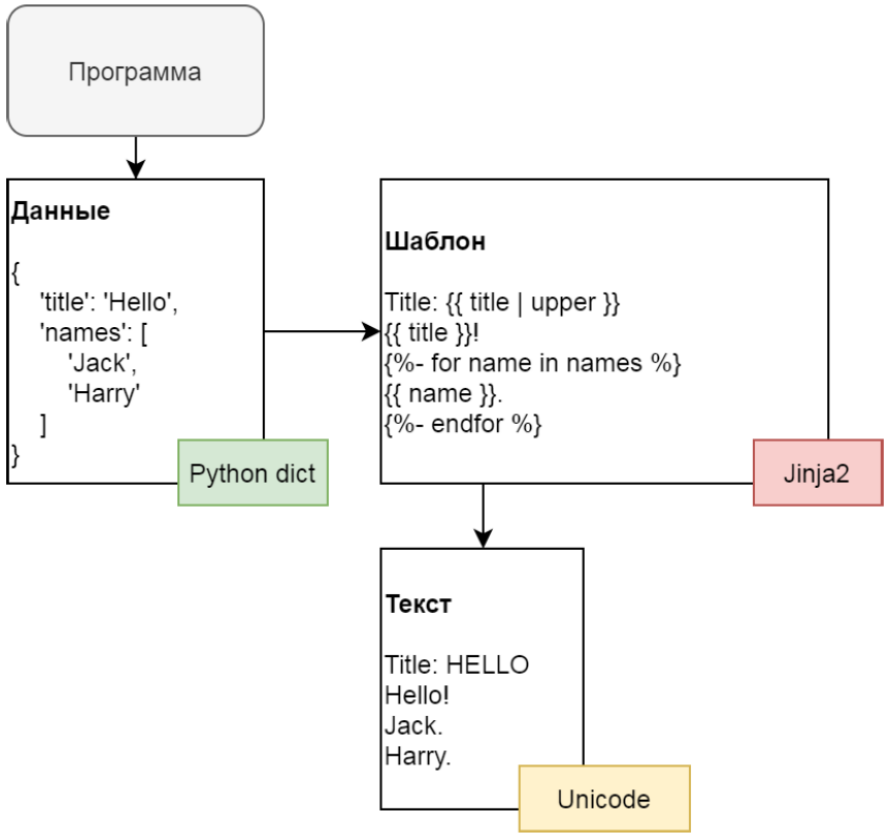

Схема работа Jinja2

Сравнение реализаций

```cpp
#include <iostream>
#include <vector>
#include <string>

int main()
{
  std::vector<std::string> cols = {"col1", "col2", "col3"};
  std::string name = "main";
  std::cout << "void " << name << "(";
  for (std::string col : cols) {
    std::cout << "int " << col << ", ";
  }
  std::cout << "col4) {" << std::endl;
  std::cout << "\t// pass" << std::endl;
  std::cout << "}" << std::endl;
  return 0;
}
```

Реализация на C++

```python
import jinja2

def main():
    st = """
void {{ name }}({% for col in cols %}{{ col }}, {% endfor %}col4) {
    // pass
}
    """
    tmp = jinja2.Template(st)
    print(tmp.render(name='main', cols=['col1', 'col2', 'col3']))
```

Реализация на Python

```
void main(col1, col2, col3, col4) {
    // pass
}
```

Сгенерированный код

Синтаксический анализатор

Для класса Token были перегружены операции сравнения (==, !=), и операция битового сдвига (>>)

Операции сравнения позволяют проверить текущий токена

Операция битового сдвига реализует комбинацию проверки токена, возврата значения токена и перехода на следующий токен

```
@log(tree_logger)
def parse_type(self):
    kind = self.token >> 'TYPE'
    if self.token == 'LPARENT':
        self.token.next()  # self.token >> 'LPARENT'
        size = self.token >> 'INTEGER'
        self.token >> 'RPARENT'
        return context.get_type_by_name(kind, size)
    return context.get_type_by_name(kind)
```

Установка и запуск в одну строку

- Реализация на Python позволяет пользоваться генератором практически на всех Unix системах, а также на Windows
- Для удобства пользователей шаги установки и использования были сведены к минимуму

```
(venv) C:\>pip install git+https://github.com/bmstu-iu9/mill-db.git
```

Установка

```
(venv) C:\>python -m pymilldb path/to/sql/file.sql
```

Использование

Выводы

- Изучен генератор баз данных MillDB
- Была произведена интеграция новый функциональностей
- Переписан генератор с языка C++ (+Flex, +Bison) на язык Python