SimpleDB: Course Project of Compiler Theory

Sun Jiacheng

November 22, 2014

Chapter 1

Information

1.1 Team Information

Sun Jiacheng	12330285	291624707@qq.com
Kuang Yuanyuan	12330153	596755905@qq.com
Qiu Zhilin	12330268	847300960@qq.com
Sun Dongliang	12330284	1255993541@qq.com
Wang Kaibin	12330305	wkbpluto@qq.com

1.2 Tarball Structure

```
/doc -- documents of project
/src -- source code
```

1.3 Compile

```
cd ./src
make -- compile SimpleDB
make test -- compile and run sample test
make tar -- tar the project.
```

Chapter 2

Project

2.1 Project Website

https://github.com/thomasking0529/SimpleDB

2.2 Code Style

K&R(CDT's default style)

2.3 Common

2.3.1 Lexer

```
enum TokenType:
    KEYWORD -- actions(create, insert, delete, select),
          primitives("table" only), case insensitive,
          "where", properties of table(default, primary
          key), case insensitive.

    ID -- id (identifier) is a sequence of digits, underline
          and letters. All identifiers should start with a
          letter or an underline. The maximum length of an
          identifier is 64.

    NUM -- num (number) is a sequence of digits. (of 32-bits)
    OP -- Arithmetical operators: +, -, *, /, unary -, unary +
          Relational operators: <, >, <>, ==, >=, <=
          Logical operators: &&, ||, !
          Assignment operator: =.
          Basic punctuation("(", ")", ",")

struct Token:
    TokeType type -- token type.
    std::string value -- token value, store the original string
          of token.
```

2.3.2 Parser

```
enum Action:
    CREATE -- create table
    INSERT -- insert one row to table
    DELETE -- delete row(s) from table
    SELECT -- query row(s) from table
    INVALID -- if unknown keywords detected

enum Op:
    PLUS, // +, both unary and binary
    MINUS, // -, both unary and binary
    MULTIPLY, // *
    DIVIDE, // /
    LT, // <
    GT, // >
    NE, // <>
    E, // ==
    GTE, // >=
    LTE, // <=
    AND, // &&
    OR, // ||
    NOT, // !
    EQ, // =
    LB, // (
    RB, // )
    COMMA, // ,

struct Condition:
    std::string lop -- left operand
    std::string rop -- right operand
    Op op -- operator

enum PropType:
    INT -- int(only)

struct Property:
    std::string id -- property id
    PropType type -- property type
    std::string default_value -- default value
    operator== -- operator overloading

struct Statement:
    Action act -- action.
    std::string table -- table to operate on.
    std::list<std::string> prop_list -- property to return or
        add, for create and select
    std::list<Condition> conds -- where clauses, for create,
        select and delete
```

2.3.3 Core

```
struct Table:
    std::string id -- table name
    std::set<Property> -- properties of table
    std::list<std::string> -- values of properties, in 1-D array
    void insert(const std::list<std::string>& record) -- insert
```

2.4 Design

How you implement. Time and space complexity.

2.4.1 Lexer

Lexer reads a single string of statement and extracts tokens from it.

APIs

```
Constructor:
    Initialization.
std::list<Token> GetTokens(const std::string& statement):
    Get a string of statement and return a list of tokens extracted.
    Do token validation.
```

2.4.2 Parser

Parser get a list of tokens and return a single statement.

APIs

```
Constructor:
    Initialization.
Statement Parse(std::list<Token> token_list)
    Parse token list to statement.
    Invoke separated parse function.
    Do grammar validation.
```

2.4.3 Core

Core part of SimpleDB is supposed to manage the ”true” database in memory.

APIs

```
Constructor:
    Initialization.
void Execute(std::string)
    Execute a single statement.
    Invoke private member function.
    Check consistency.
```

2.5 Distribution

2.5.1 Lexer implementation

2.5.2 parseCreate and parseInsert

2.5.3 parseDelete and parseSelect

2.5.4 parseWhere

2.5.5 Core

2.6 Test

A test sample contains: a single statement or several statements to test the program what is this test sample for what's the expected output does the program work correctly TAs will have their own test samples to test your program. Test document: does the interpreter work? Test samples and screenshots