SimpleDB: Course Project of Compiler Theory

Kuang Yuanyuan

January 2, 2015

Chapter 1

Information

1.1 Team Information

Name	Student ID	E-mail
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 APIs

2.3.1 Lexer

```
enum TokenType:
    KEYWORD -- select from where delete table create
  int values primary key default into insert
  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)
  only integers
```

```
    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.

class Lexer:
std::list<Token> GetTokens(const std::string& s);
get a list of tokens from input string
```

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:
Condition* lop-- left subtree
Condition* rop-- right subtree
    Op op -- operator
std::string opd -- operand

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

struct Statement:
    Action act -- action.
    std::string table -- table to operate on.
    std::list<Property> prop_list -- property to return or
        add, for create and select
std::vector<std::string> key_idx -- list of keys
std::vector<int> value_list -- list of values
    Conditio* cond -- where clauses, for select and delete

class Polish:
Polish() -- Constructor, initialize private variables
void neglect() -- set neglect flag, for neglect numbers
void Insert(const std::string& item) -- inset a token
int Calculate() -- calculate the expression
Condition* buildTree -- build a condition tree

class Parser:
Parser() -- Constructor, initialize private variables
Statement Parse(const std::string& s) -- main parser
```

2.3.3 Core

```
struct Row:
```

```
int key -- key value, for identify row
std::vector<int> -- key indices
std::vector<int> cols -- values

struct Table:
    std::string id -- table name
    std::vector<Property> -- properties of table
    std::vector<int> -- key indices of table
unsigned long long count -- if no primary key, use this
std::set<Row> rows -- set of rows, use Row.key for key
    void Insert(const std::list<int>& record) -- insert
void Delete(const Condition* cond) -- delete
std::vector<int> Query(const Condition* cond) -- query

class SimpleDB:
SimpleDB() -- Constructor, initialize private variables
void Execute(const std::string& stmt) -- execute raw input
```

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.

```
Finite State Automata:
read a charactor from input string;
check charactor for:
'a-z' 'A-Z' '0-9': concat to temp string
split characters: space, \t, \n, save temp string
symbols: save temp string
```

2.4.2 Parser

Design

Use LL1 parser.

Implementation

Construct a parsing table. Use recursive reduction and common factor extraction, to reduce the prediction table.
Initialize parsing stack.
Initialize another stack for saving the left terminals, thus when a terminal matched, we can get the context of where it produced.
Read an input token from token stream, match it with the top token of stack, rewrite the stack if matched.

Expression

Use a calculator to reduce the where clause. Use reverse polish to check the priority of braces. Set a flag for neglect numbers, if unary plus encountered, discard it. If a unary minus encountered, if a number followed, not the flag; if an id followed, expand the expression to (0-id); if a left brace followed, expand to "(0-", count the left braces, count unary braces followed by left brace; if a unary not encountered, insert another symbol to set it as binary operator.

2.4.3 Core

Core part of SimpleDB is supposed to manage the database in memory.
SimpleDB a set of Table, Table manage a set of Row. Do query, insert, delete on Table.

Chapter 3

Project Schedule

3.1 Stage 1

3.1.1 Project architecture and APIs design

Author	Deadline
Sun Jiacheng	12.5.2014

3.1.2 Lexer implementation

Author	Deadline
Sun Jiacheng	12.5.2014

3.1.3 Group Discussion

Contents:
Stage 2 and Stage 3 work distribution.
Date:
12.6.2014

3.2 Stage 2

3.2.1 Parser

Author	Deadline
Qiu Zhilin	12.23.2014

3.2.2 Core

Author	Deadline
Sun Dongliang	12.23.2014

3.3 Stage 3

3.3.1 Test Document

Author	Deadline
Wang Kaibin	1.2.2014

3.3.2 Design Document

Author	Deadline
Kuang Yuanyuan	1.2.2014