数据库系统实验报告

作业名称:	MiniSQL
姓 名:	王振阳
学 号:	3190104758
电子邮箱:	1194072799@qq.com
联系电话:	17799857316
指导老师:	孙建伶
日期:	2020年 6 月 28 日

一、 实验目的

设计并实现一个精简型单用户 SQL 引擎(DBMS) MiniSQL，允许用户通过字符界面输入 SQL 语句实现表的建立/删除；索引 的建立/删除以及表记录的插入/删除/查找。 通过对 MiniSQL 的设计与实现，提高学生的系统编程能力，加深对数据库系统原理的理解。

我在此次实验中负责Record Manager 和 GUI 网络通信部分的实现。

二、 系统需求

1. 基本需求：表的创建、删除，记录的创建、删除，索引的创建删除，基本的错误显示，重点部分是记录的查询（要求支持等值查询、区间查询、以及使用and链接的多个条件的查询）
2. 数据类型：要求至少支持int、float、char
3. 高标准要求（bonus）：中文支持、GUI、更加丰富的错误显示、更多的语句等。

在本次实验中，我们完全覆盖了基本需求和几乎全部的Bonus。

三、 实验环境

1. 操作系统

我个人使用Windows10 20H2(OS 19042)进行工作。

CPU Name: Intel CoreTM i7-9750H CPU @ 2.60GHz

Property	Value
Base Frequency	2.59 GHz
Max Turbo Frequency	4.10 GHz
Cache	12 MB Intel® Smart Cache
Cores Number	12
Threads	12
TDP	45W

RAM: 16GB DDR4 2666MHz with Two channel memory

SSD: SAMSUNG MZVLB1T0HBLR-000L2

2. 开发环境

下面是我个人的开发环境

软件包	版本号
go	1.16.3
node	14.16.0

3. 语言与语言规范

本次主要使用go和 javascript 作为主要语言进行开发。go用于实现DBMS本体和网络后端，js用于实现GUI界面。

四、模块设计

1. Record

i. 模块总体设计

因为 API 仅起到检查和转发的作用，事实上 Record Manager 调用Catalog Manager 和 Index Manager，并与 Buffer Manager 交互。

通过在 `RMUtils.go` 中的封装，实现了对 Buffer Manager 操作的高度封装，而针对API的转发，进行函数处理。

ii. 功能描述

Record Manger 主要承担执行操作的功能，对于 table 的建立删除，index 的建立删除，record 的插入，更新和删除进行处理。

对于所有的运行时错误进行错误处理。

对于各类操作进行完整性校验。

iii. 主要数据结构

模块中最为重要的是freelist的维持，通过链表的形式实现对空余位置的快速查找。

为了简化代码结构，freelist 采用 Index Manager 中定义的链表格式。

```
var FreeList IndexManager.FreeList
```

iv. 主要函数功能

Record Manager 中的函数列表

```
const freeListFileHotFix = "_list"
var FreeList IndexManager.FreeList
func CreateIndex(table *CatalogManager.TableCatalog, newIndex
CatalogManager.IndexCatalog) error
func CreateTable(tableName string) error
func DeleteRecord(table *CatalogManager.TableCatalog, where *types.Where)
(error, int)
func DeleteRecordWithIndex(table *CatalogManager.TableCatalog, where
*types.Where, ...) (error, int)
func DropDatabase(databaseId string) error
func DropIndex(table *CatalogManager.TableCatalog, indexName string) error
func DropTable(tableName string) error
func FlushFreeList() error
func InsertRecord(table *CatalogManager.TableCatalog, columnPos []int,
startBytePos []int, ...) error
func SelectRecord(table *CatalogManager.TableCatalog, columns []string, where
*types.Where) (error, []value.Row)
func SelectRecordWithIndex(table *CatalogManager.TableCatalog, columns []string,
where *types.Where, ...) (error, []value.Row)
func UpdateRecord(table *CatalogManager.TableCatalog, columns []string, values
[]value.Value, ...) (error, int)
```

```
func UpdateRecordWithIndex(table *CatalogManager.TableCatalog, columns []string,
values []value.Value, ...) (error, int)
func checkRow(record value.Row, where *types.Where, colPos []int) (bool, error)
func columnFilter(table *CatalogManager.TableCatalog, record value.Row, columns
[]string) (value.Row, error)
func deleteRecord(table *CatalogManager.TableCatalog, recordPosition dataNode)
error
func getColPos(table *CatalogManager.TableCatalog, where *types.Where) (colPos
[]int)
func getRecord(table *CatalogManager.TableCatalog, recordPosition dataNode)
(bool, value.Row, error)
func getRecordData(tableName string, recordPosition dataNode, length int)
([]byte, error)
func loadFreeList(tableName string) error
func setRecord(table *CatalogManager.TableCatalog, recordPosition dataNode,
columnPos []int, ...) error
func setRecordData(tableName string, recordPosition dataNode, data []byte,
length int) error
func updateRecord(table *CatalogManager.TableCatalog, columns []string, values
[]value.Value, ...) (bool, error)
func updateRecordData(table *CatalogManager.TableCatalog, recordPosition
dataNode, record value.Row) error
type dataNode = IndexManager.Position
```

首先考虑对外暴露的接口

大部分结构的功能相对比较清晰，这里需要特别说明的是，在 Insert 的过程中，因为需要检查unique是否满足，实际上需要进行一次select，为了保证效率，在每一个unique的位置上建立B+树索引，以提升相对的插入效率

其次是内部使用的结构

```
func loadFreeList(tableName string) error {
    fileName := CatalogManager.TableFilePrefix() + "_data/" + tableName +
freeListFileHotFix //文件名
    if FreeList.Name == fileName {
   //已经load了
        return nil
    } else if len(FreeList.Name) > 0 {//需要把旧的flush
        err := FlushFreeList()
        if err != nil {
            return err
        }
    }
    if !Utils.Exists(fileName) {  //如果没有这个文件 新建该文件并序列化写入初始name信息
        newfile, err := Utils.CreateFile(fileName)
        defer newfile.Close()
        if err!=nil {
            return err
        }
        wt:=msgp.NewWriter(newfile)
        FreeList.Name=fileName
        err=FreeList.EncodeMsg(wt)
        if err!=nil  {
            return err
        }
        return wt.Flush()
```

```
    }
    //存在该文件 直接读取即可
    existFile, err := os.Open(fileName)
    defer existFile.Close()
    if err != nil {
        return errors.New("打开free list文件失败")
    }
    rd := msgp.NewReader(existFile)
    return FreeList.DecodeMsg(rd)
}

//FlushFreeList 退出程序时候请不要忘记
func FlushFreeList() error {
    oldList, err := os.OpenFile(FreeList.Name, os.O_WRONLY|os.O_TRUNC, 0666) //写
入旧文件
    defer oldList.Close()
    if err != nil {
        return errors.New("free list文件打开失败")
    }
    wt := msgp.NewWriter(oldList)
    err = FreeList.EncodeMsg(wt)
    if err != nil {
        return errors.New("free list文件写入失败")
    }
    return wt.Flush()

}
```

通过 msgp 库进行 freelist 的持久化存储，将freelist开始程序时读入而在结束程序之后flush

```
func getRecordData(tableName string, recordPosition dataNode, length int)
([]byte, error) {
    block, err :=
BufferManager.BlockRead(CatalogManager.TableFilePrefix()+"_data/"+tableName,
recordPosition.Block)
    if err != nil {
        return nil, err
    }
    defer block.FinishRead()
    record := block.Data[int(recordPosition.Offset)*length :
int(recordPosition.Offset+1)*length]
    return record, nil
}

func setRecordData(tableName string, recordPosition dataNode, data []byte,
length int) error {
    block, err :=
BufferManager.BlockRead(CatalogManager.TableFilePrefix()+"_data/"+tableName,
recordPosition.Block)
    if err != nil {
        return err
    }
    block.SetDirty()
    defer block.FinishRead()

    record := block.Data[int(recordPosition.Offset)*length :
int(recordPosition.Offset+1)*length]
```

```
    copy(record, data)
    return nil
}
```

这里封装了对于 Buffer Manager的操作，实现一条记录的读取和写入

```
func getRecord(table *CatalogManager.TableCatalog, recordPosition dataNode)
(bool, value.Row, error) {
    data, err := getRecordData(table.TableName, recordPosition,
table.RecordLength)
    if err != nil {
        return false, value.Row{}, err
    }
    nullmapBytes:=data[0:len(table.ColumnsMap)/8+1]
    nullmap:=Utils.BytesToBools(nullmapBytes)

    if nullmap[0] == false {
        return false, value.Row{}, nil
    }
    record := value.Row{Values: make([]value.Value, len(table.ColumnsMap))}
    //思考顺序问题，Column是以什么顺序存储的
    for _, column := range table.ColumnsMap {
        startPos := column.StartBytesPos
        length := column.Type.Length //这个length是给char和string和null用的，所以其他
类型无用
        valueType := column.Type.TypeTag

        if nullmap[column.ColumnPos+1] == false {
            valueType = CatalogManager.Null
        }
        if record.Values[column.ColumnPos], err =
            value.Byte2Value(data[startPos:], valueType, length); err != nil {
            return true, value.Row{}, err
        }
    }
    return true, record, nil
}


func setRecord(table *CatalogManager.TableCatalog, recordPosition dataNode,
    columnPos []int, startBytePos []int, values []value.Value) error {
    data := make([]byte, table.RecordLength)
    nullmapBytes:=data[0:len(table.ColumnsMap)/8+1]
    nullmap:=Utils.BytesToBools(nullmapBytes)
    nullmap[0] = true
    for _, columnIndex := range columnPos {
        nullmap[columnIndex+1] = true
    }
    nullmapBytes=Utils.BoolsToBytes(nullmap)

    copy(data[:], nullmapBytes)
    for index, _ := range columnPos {
        tmp, err := values[index].Convert2Bytes()
        if err != nil {
            return err
        }
```

```
        copy(data[startBytePos[index]:], tmp)
    }
    if err := setRecordData(table.TableName, recordPosition, data,
table.RecordLength); err != nil {
        return err
    }
    return nil
}
```

更进一步的，这里实现了序列化的过程，将一条以结构存储的记录存入Buffer

```
func columnFilter(table *CatalogManager.TableCatalog, record value.Row, columns
[]string) (value.Row, error) {
    if len(columns) == 0 { //如果select* 则使用全部的即可
        return record, nil
    }
    var ret value.Row

    for _, column := range columns {
        ret.Values = append(ret.Values,
record.Values[table.ColumnsMap[column].ColumnPos])
    }

    return ret, nil
}
```

物理优化的一部分，通过截取所需的列实现效率的优化

2. GUI

i. 功能描述

通过 React-Ace 框架实现数据库的GUI界面。
这一GUI界面支持用户登录鉴权，权限管理；
SQL语句自动补全；
SQL语句语法静态检查；
查询数据回显。

ii. 主要结构

实现 SQL 语句查询的模块

```
import React, {useEffect, useRef, useState} from 'react';
import {VariableSizeGrid as Grid} from 'react-window';
import ResizeObserver from 'rc-resize-observer';
import classNames from 'classnames';
import {Table, Empty, Result} from 'antd';

function VirtualTable(props) {
    const {columns, scroll, className} = props;
    const [tableWidth, setTableWidth] = useState(0);
    const widthColumnCount = columns.filter(({width}) => !width).length;
```

```
    const mergedColumns = columns.map(column => {
        if (column.width) {
            return column;
        }

        return {...column, width: Math.floor(tableWidth / widthColumnCount)};
    });
    const gridRef = useRef();
    const [connectObject] = useState(() => {
        const obj = {};
        Object.defineProperty(obj, 'scrollLeft', {
            get: () => null,
            set: scrollLeft => {
                if (gridRef.current) {
                    gridRef.current.scrollTo({
                        scrollLeft,
                    });
                }
            },
        });
        return obj;
    });

    const resetVirtualGrid = () => {
        gridRef.current.resetAfterIndices({
            columnIndex: 0,
            shouldForceUpdate: false,
        });
    };

    useEffect(() => resetVirtualGrid, []);
    useEffect(() => resetVirtualGrid, [tableWidth]);

    const renderVirtualList = (rawData, {scrollbarSize, ref, onScroll}) => {
        ref.current = connectObject;
        return (
            <Grid
                ref={gridRef}
                className="virtual-grid"
                columnCount={mergedColumns.length}
                columnWidth={index => {
                    const {width} = mergedColumns[index];
                    return index === mergedColumns.length - 1 ? width -
scrollbarSize - 1 : width;
                }}
                height={scroll.y}
                rowCount={rawData.length}
                rowHeight={() => 54}
                width={tableWidth}
                onScroll={({scrollLeft}) => {
                    onScroll({
                        scrollLeft,
                    });
                }}
            >
                {({columnIndex, rowIndex, style}) => (
                    <div
                        className={classNames('virtual-table-cell', {
```

```
                            'virtual-table-cell-last': columnIndex ===
mergedColumns.length - 1,
                        })}
                        style={style}
                    >
                        {rawData[rowIndex]
[mergedColumns[columnIndex].dataIndex]}
                    </div>
                )}
            </Grid>
        );
    };

    return (
        <ResizeObserver
            onResize={({width}) => {
                setTableWidth(width);
            }}
        >
            <Table
                {...props}
                className={classNames(className, 'virtual-table')}
                columns={mergedColumns}
                pagination={false}
                components={{
                    body: renderVirtualList,
                }}
            />
        </ResizeObserver>
    );
}


function DataTable(props) {
    const {tableData, tableColumns} = props;
    const getData = (tableData) => {
        return tableData.map(
            (x, idx) => {
                const item = {}
                x.map(
                    (_x, _idx) => {

                        item[_idx] = _x;
                        return _x;
                    }
                )
                return item;
            }
        )
    }


    const _columns = tableColumns.map(
        (x, idx) => {
            return ({
                title: x,
                dataIndex: idx,
                key: x
```

```
            })
        }
    )
    const _data = getData(tableData)
    return (
        <VirtualTable
            columns={_columns}
            dataSource={_data}
            scroll={{
                y: 300,
                x: '100vw',
            }}
        />
    )
}

function Callback(props) {
    const {status, times, rows, data} = props
    if (status === true) {
        if (data !== undefined && data !== null && data !== []) {
            const tableColumns = data[0]
            const tableData = data.slice(1)
            return <div style={{
                width: '100%',
                height: '100%'
            }}>
                <Result
                    status="success"
                    title={`操作成功！ 本次操作影响了${rows}行数据，耗时${times}s`}
                    style={{
                        backgroundColor: '#FFF'
                    }}
                />
                <DataTable tableData={tableData} tableColumns={tableColumns}/>

            </div>
        }
        return (<div style={{
            width: '100%',
            height: '100%'
        }}>
            <Result
                status="success"
                title={`操作成功!耗时${times}s`}
                style={{
                    backgroundColor: '#FFF'
                }}
            />
        </div>)
    } else if (status === false) {
        return <Result
            status="error"
            title={`操作失败！ 后端反馈：${data}`}
            style={{
                backgroundColor: '#FFF'
            }}
        />
    } else if (status === undefined) {
```

```
        return <Empty/>
    } else if (times === undefined || times === null || times === []) {
        // TODO 退出登录
        return <Result
            status="success"
            title={'登出成功! '}
            style={{
                backgroundColor: '#FFF'
            }}
        />

    } else {
        return <Result
            status="error"
            title={`啊欧，失败了！可能是你的语句不太正常。`}
            style={{
                backgroundColor: '#FFF'
            }}
        />
    }


}

export default Callback;
```

实现数据回显的模块

```
import React, {useState,useRef} from "react";
import AceEditor from "react-ace";
import {Button, Empty, Layout, Menu, message, PageHeader} from 'antd';
import Axios from "axios";
import Callback from "./Callback";

import 'ace-builds/src-noconflict/ext-language_tools';
import 'ace-builds/src-noconflict/ext-searchbox';
import 'ace-builds/src-noconflict/mode-mysql';
// theme
import 'ace-builds/src-noconflict/theme-sqlserver';
import 'ace-builds/src-noconflict/theme-github';
import 'ace-builds/src-noconflict/theme-eclipse';
import 'ace-builds/src-noconflict/theme-monokai';
import 'ace-builds/src-noconflict/theme-clouds';
import 'ace-builds/src-noconflict/theme-chrome';
import 'ace-builds/src-noconflict/theme-solarized_dark';
import 'ace-builds/src-noconflict/theme-solarized_light';
import Redirect from "react-router-dom/es/Redirect";

function MiniSQL(props) {
    const {userName} = props


    const {SubMenu} = Menu;
    const {Content, Footer, Sider} = Layout;
```

```
    const themeList = ["sqlserver", "github", "eclipse", "monokai", "clouds",
"chrome", "solarized_dark", "solarized_light"]
    const [theme, setTheme] = useState(themeList[0])
    const EditorRef = useRef()
    const [queryData, setQueryData] = useState([])
    const [checkOn,setCheckOn] = useState()
    const sqlSplit = (texts) => {
        const dtFilter = require("dt-sql-parser").filter;

        const afterFilterComments = dtFilter.filterComments(texts)
        const afterSplit = dtFilter.splitSql(afterFilterComments)
        console.log(afterFilterComments)
        console.log(afterSplit)
        let res = []
        for (let i = 0; i < afterSplit.length; i++) {
            const item = afterSplit[i]
            if (item !== "" && item !== '\n' && item !== undefined) {
                res.push(item.replace(/[\r\n]/g, '').replace('undefined', ''))
            }
        }
        console.log(res)
        return res
    }
    const syntaxCheck = (text) => {
        // if(checkOn===undefined||checkOn===false){
        //     return false
        // }

        const dtSqlParser=require("dt-sql-parser").parser;
        return dtSqlParser.parseSyntax(text);
    }


    const doQuery = (data) => {

        if (userName === 'manager') {
            if (data.indexOf('delete') !== -1 || data.indexOf('drop') !== -1) {
                message.error('权限不足! 只有root账号才能进行delete和drop操作')
                return
            }
        } else if (userName === 'customer') {
            console.log(data)
            if (data.indexOf('select') === -1) {
                message.error('权限不足! 普通用户只能使用select操作')
                return
            }
        }
        const query = async (data) => {
            try {
                const res = await Axios(
                    'api/query',
                    {
                        method: 'POST',
                        data: {
                            'query': data
                        }
                    }
                );
```

```
                setQueryData(res.data.data)
            } catch (e) {
                message.error('后端提示：啊欧，失败了！可能是你的语句不太正常。')
            }
        }

        let texts = sqlSplit(data)
        if (texts === undefined || texts === null) {
            message.error('空代码可跑不了哦！')
            return
        }

        for (let i = 0; i < texts.length; i++) {
            const check = syntaxCheck(texts[i]);
            console.log(check)
            if(check!==false){
                message.error(
                    `前端语法检查:\n
                                错误类型: ${check.token}\n
                                错误位置: \n
                                        开始行数: ${check.loc.first_line}    结束行
数: ${check.loc.last_line}\n
                                        开始列数: ${check.loc.first_column}  结束列
数: ${check.loc.first_column}\n
                                修改建议: \n
                                        改为:
${check.expected!==null&&check.expected.length>0?check.expected[0].text:'暂无'}
                            `

                )
                return
            }
            query(texts[i])
        }


    }


    return (<div>
        {
            (userName === undefined || userName === null || userName === '') ?
<Redirect to="/"/>
                : <Layout>

                    <Content style={{padding: '0 50px'}}>
                        <PageHeader
                            className="site-page-header"
                            title="MinSQL Editor"
                            subTitle={"current user: " + userName}
                        />
                        <Layout className="site-layout-background" style=
{{padding: '24px 0'}}>
                            <Sider className="site-layout-background" width=
{200}>
                                <Menu
                                    mode="inline"
                                    style={{height: '100%'}}
                                    onClick={(param) => {
```

```
                                    setTheme(themeList[param["key"]])
                                }}
                            >
                                <Button
                                    type="primary"
                                    style={{
                                        textAlign: "center",
                                        width: "100%",
                                        marginBottom: "5px"
                                    }}
                                    onClick={(e) => {
                                        e.preventDefault();
                                        const context =
EditorRef.current.editor.getValue()
                                        doQuery(context)

 EditorRef.current.editor.setValue(context)
                                    }}
                                    ghost
                                >

                                    Run Code
                                </Button>
                                <SubMenu key="theme" title="Theme">
                                    <Menu.Item key={0}>sql
server</Menu.Item>
                                    <Menu.Item key={1}>github</Menu.Item>
                                    <Menu.Item key={2}>eclipse</Menu.Item>
                                    <Menu.Item key={3}>monokai</Menu.Item>
                                    <Menu.Item key={4}>clouds</Menu.Item>
                                    <Menu.Item key={5}>chrome</Menu.Item>
                                    <Menu.Item key=
{6}>solarized_dark</Menu.Item>
                                    <Menu.Item key=
{7}>solarized_light</Menu.Item>
                                </SubMenu>
                            </Menu>

                        </Sider>
                        <Content style={{padding: '0 24px', minHeight:
300}}>
                            <AceEditor
                                ref={EditorRef}
                                mode="mysql"
                                theme={theme}
                                fontSize={16}
                                style={{
                                    width: '100%',
                                    height: '100%',
                                    minHeight: 300,
                                    fontFamily: "Fira Code, Consolas,
monospace"
                                }}
                                setOptions={{
                                    enableBasicAutocompletion: false, //关闭基
本自动完成功能
                                    enableLiveAutocompletion: true,   //启用实
时自动完成功能
```

```
                                    enableSnippets: true,
                                    showLineNumbers: true,
                                    editorProps: {$blockScrolling: true},
                                    highlightActiveLine: true,
                                    tabSize: 4
                                }}
                            />

                        </Content>
                    </Layout>
                    {queryData === null || queryData === undefined
                        ? <Empty/>
                        :
                        <Callback status={queryData[0]} times={queryData[1]}
rows={queryData[2]} data={queryData[3]}/>}

                </Content>
                <Footer style={{textAlign: 'center'}}>MiniSQL ©2021 Created
by Wolfram</Footer>
            </Layout>
        }
    </div>)

}

export default MiniSQL;
```

六、 遇到的问题及解决方法

1. 团队协作

   可以说这个数据库管理程序是我至今为止做过的最具有挑战性的大程，因为各个部分的接口比较复杂，接口传入的参数要求也不太一样，因此总是会出现我对于一些接口的传入参数和返回参数的数据结构理解有误的情况。在debug期间也确实出现了很多因为队友间没有充分交流而导致的问题。这也给了我对于团队合作的一些启发，遇到不太确定的问题一定要及时沟通，需要与队友间频繁交流，了解互相的进度。

2. react-ace的使用

   React-Ace框架是一个简单易用的 TextEditor 框架， 通过对他的学习实现了一个功能强大的GUI界面

3. 协程的使用

   虽然我们要求是一个简单的串行程序，但是既然我们底层bm支持并发，那么光是串行程序将大大降低运行速度，通过管道等方式，实现了多条指令的并发。

七、 总结

这一次大程应该说是上大学以来做的最难的一次课程设计，不光是码力的锻炼，更是各种意义上的提升，但只要认真完成，也确实能从中学习到许多。同时要感谢孙老师课堂上的悉心指导为我们打下了良好的基础，感谢助教哥哥的耐心指导和细致的实验说明也帮助了我们许多，感谢我的两位队友与我的协助，和我一起解决了非常多的问题，很幸运能和这么可靠的两位同学合作。