# 暨南大学本科实验报告专用纸

课程名称 高级语言程序设计实验 成绩评定

实验项目名称 实验 16 Relational Database 指导教师 张鑫源

实验项目编号 实验项目类型 实验地点 机房

学生姓名 王浩宇 学号 2018054490

学院 智能科学与工程 系 专业 信息安全

实验时间 2021 年 6 月 21 日 下 午~ 6 月 21 日 下 午

本课程所有代码可从 GitHub 仓库：

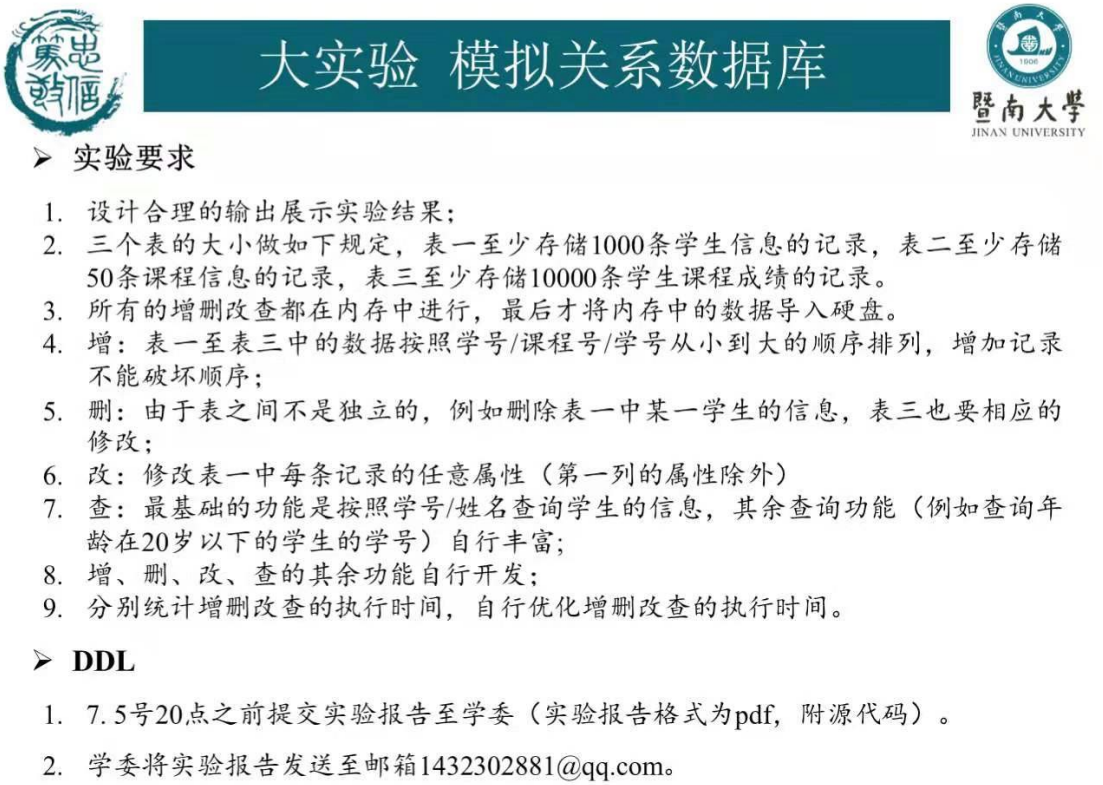
https://github.com/ChopperCP/CProgrammingLanguageExperiment

上找到。

## （一）实验目的

设计一个简单的关系数据库。

## （二）实验内容和要求



大实验 模拟关系数据库

暨南大學 JINAN UNIVERSITY

➢ 实验要求

1. 设计合理的输出展示实验结果；
2. 三个表的大小做如下规定，表一至少存储1000条学生信息的记录，表二至少存储50条课程信息的记录，表三至少存储10000条学生课程成绩的记录。
3. 所有的增删改查都在内存中进行，最后才将内存中的数据导入硬盘。
4. 增：表一至表三中的数据按照学号/课程号/学号从小到大的顺序排列，增加记录不能破坏顺序；
5. 删：由于表之间不是独立的，例如删除表一中某一学生的信息，表三也要相应的修改；
6. 改：修改表一中每条记录的任意属性（第一列的属性除外）
7. 查：最基础的功能是按照学号/姓名查询学生的信息，其余查询功能（例如查询年龄在20岁以下的学生的学号）自行丰富；
8. 增、删、改、查的其余功能自行开发；
9. 分别统计增删改查的执行时间，自行优化增删改查的执行时间。

➢ **DDL**

1. 7.5号20点之前提交实验报告至学委（实验报告格式为pdf，附源代码）。
2. 学委将实验报告发送至邮箱1432302881@qq.com。


➢ 实验内容

模拟关系数据库，实现增、删、改、查四项功能。数据库中的数据可以视为大量元组构成的一个集合。例如存储学生-课程的关系表由下面三个表构成（可以视为三份文件）：

表一. 学生信息

| 学号 | 姓名 | 性别 | 年龄 | 院系 |
|---|---|---|---|---|
| | | | | |
| | | | | |

表二. 课程信息

| 课程号 | 课程名 | 先行课 | 学分 |
|---|---|---|---|
| | | | |

表三. 学生-课程成绩信息

| 学号 | 课程号 | 成绩 |
|---|---|---|
| | | |

## （三）主要仪器设备

**仪器：**计算机

**实验环境：**Clion

## （四）实验原理

**由于源码过长，所以在此不列出，下面对应上面的要求讲解实现的原理，讲解过程中会附源码片段。**

### 一、 表的设计

本实验中将表的条目和表分别定义为结构体，三种表的 data 字段为其对应的条目类型的指针的数组。表的具体设计如下：

```
struct Student {
    unsigned int student_number;
    char student_name[30];
    char gender[10];
    unsigned int age;
    char school[100];
};
```

```c
struct StudentTable {
    char type[5];       // Identify table type
    unsigned int last_available;    // Last available index
    struct Student *data[MAX_STUDENT_TABLE_ENT_CNT];
};
```

```c
struct Course {
    unsigned int course_number;
    char course_name[100];
    unsigned int required_course_number;
    double credit;
};

struct CourseTable {
    char type[5];       // Identify table type
    unsigned int last_available;    // Last available index
    struct Course *data[MAX_COURSE_TABLE_ENT_CNT];
};
```

```c
struct StudentCourse {
    unsigned int student_number;
    unsigned int course_number;
    double grade;
};

struct StudentCourseTable {
    char type[5];       // Identify table type
    unsigned int last_available;    // Last available index
    struct StudentCourse *data[MAX_STUDENT_COURSE_TABLE_ENT_CNT];
};
```

## 二、 表的导入和导出设计

本实验使用 cJSON 库（https://github.com/DaveGamble/cJSON）将表的数据导出为 JSON 格式。下面以 StudentTable 的导入和导出为例：

```c
int export_student_table(char *path, struct StudentTable
*student_table) {
    cJSON *table = cJSON_CreateObject();

    cJSON_AddStringToObject(table, "type", student_table->type);
    cJSON_AddNumberToObject(table, "last_available",
student_table->last_available);
    cJSON *data = cJSON_AddArrayToObject(table, "data");

    for (int i = 0; i < student_table->last_available; ++i) {
        struct Student *student = student_table->data[i];
        cJSON *entry = cJSON_CreateObject();
        cJSON_AddNumberToObject(entry, "student_number",
```

```
student->student_number);
        cJSON_AddStringToObject(entry, "student_name",
student->student_name);
        cJSON_AddStringToObject(entry, "gender", student->gender);
        cJSON_AddNumberToObject(entry, "age", student->age);
        cJSON_AddStringToObject(entry, "school", student->school);

        cJSON_AddItemToArray(data, entry);
    }

    FILE *fd = fopen(path, "w");
    if (fd == NULL) {
        printf("[!] fopen() error!\n");
        return -1;
    }
    fputs(cJSON_Print(table), fd);
    cJSON_Delete(table);
    fclose(fd);
    printf("[*] Exported StudentTable to %s.\n", path);

    return 0;
}

int import_student_table(char *path, struct StudentTable
*student_table) {
    FILE *fd = fopen(path, "r");
    if (fd == NULL) {
        printf("[!] fopen() error!\n");
        return -1;
    }
    char* buffer=malloc(sizeof(char)*MAX_FILE_BUFFER_SIZE);
    fread(buffer, MAX_FILE_BUFFER_SIZE, 1, fd);

    cJSON *table_json = cJSON_Parse(buffer);
    strcpy(student_table->type,
cJSON_GetObjectItemCaseSensitive(table_json,
"type")->valuestring);
    // last_available will be determined by the real entry count
//    student_table->last_available =
cJSON_GetObjectItemCaseSensitive(table_json,
"last_available")->valueint;
    cJSON *table_json_data =
cJSON_GetObjectItemCaseSensitive(table_json, "data");

    cJSON *student_json = NULL;
    int i = 0;
    cJSON_ArrayForEach(student_json, table_json_data) {
        struct Student *student = malloc(sizeof(struct Student));
        student->student_number =
cJSON_GetObjectItemCaseSensitive(student_json,
```

```
"student_number")->valueint;
        strcpy(student->student_name,
cJSON_GetObjectItemCaseSensitive(student_json,
"student_name")->valuestring);
        strcpy(student->gender,
cJSON_GetObjectItemCaseSensitive(student_json,
"gender")->valuestring);
        student->age =
cJSON_GetObjectItemCaseSensitive(student_json, "age")->valueint;
        strcpy(student->school,
cJSON_GetObjectItemCaseSensitive(student_json,
"school")->valuestring);

        student_table->data[i] = student;
        student_table->last_available = i;
        ++i;
    }

    printf("[*] Imported StudentTable from %s.\n", path);
    free(buffer);
    return 0;
}
```

导出的 JSON：

```
{
    "type":  "S",
    "last_available":  1000,
    "data":  [{
            "student_number":  1,
            "student_name": "ChopperCP",
            "gender": "Male",
            "age": 21,
            "school": "School of Intelligence Engineering
Science"
        }, {
            "student_number":  2,
            "student_name": "ChopperCP",
            "gender": "Male",
```

```
        "age": 21,
        "school": "School of Intelligence Engineering
Science"
    }, {
        "student_number":  3,
        "student_name": "ChopperCP",
        "gender": "Male",
        "age": 21,
        "school": "School of Intelligence Engineering
Science"
    },
    …
```


使用 export_student_table()时，需将想要导出的 json 文件的目录作为 path 参数，想要导出的表作为 student_table 参数输入即可。

使用 import_student_table()时，需将想要导入的 json 文件的目录作为 path 参数，然后再新创建一个 StudentTable 类型的变量，将其地址提供给 student_table 参数即可。

## 三、 增

本实验中的三张表的两张以主键排序，StudentCourseTable 表根据 student_number 排序，增加条目后会重新排序整个表。下面以 StudentTable 表的增加条目为例：

```
int add_student(struct StudentTable *table, struct Student
*student) {
    if (table->last_available >= MAX_STUDENT_TABLE_ENT_CNT) {
        printf("[!] Not enough space in the table.");
        return -1;
    }

    // Insert the student to the last place.
```

```
    table->data[table->last_available++] = student;
    // Sort the table.
    qsort(table->data, table->last_available, sizeof(struct
Student *), compare_student);

    return 0;
}
```

使用时，将要插入的表作为 table 参数，要插入的学生的地址作为 student 参数即可。

## 四、 删

由于本实验中 StudentTable 表和 CourseTable 表分别和 StudentCourseTable 有关联，所以删除一个学生/一个课程的话要在两个表中都要删除。

下面以 remove_student 为例，此函数根据 student_number 移除学生：因为 StudentTable 表是关于 student_number 有序且唯一的，所以可以先使用二分搜索在 StudentTable 表中 找到学生并移除，然后再处理 StudentCourseTable 的情况。

因为在 StudentCourseTable 表中有可能出现一个学生选修多门课程的情况，所以二分搜索行不通（因为不唯一），只可以遍历找出对应项删除（排序后对应项是连续的）。

```
int remove_student(struct StudentTable *student_table, struct
StudentCourseTable *student_course_table,
                   struct Student *target_student) {
    // Only student_number in target_student is searched, other
elements in target_student are ignored.
    struct Student **result_student = bsearch(&target_student,
student_table->data, student_table->last_available,
                                              sizeof(struct Student *),
compare_student);
    if (result_student == NULL) {
        printf("[!] Student with the number %u not found!\n",
target_student->student_number);
        return -1;
    } else {
```

```
        printf("[*] Student with the number %u found, name: %s.\n",
target_student->student_number,
               (*result_student)->student_name);
    }

    // Remove student from student_table
    _clean_Student_from_StudentTable(result_student,
student_table);

    // Can't use binary search in SC because there are more than 1
entry which has the same student_number.
    struct StudentCourse **result_student_course =
&student_course_table->data[0];
    struct StudentCourse **last_table_entry =
&(student_course_table->data[student_course_table->last_available
- 1]);
    while ((result_student_course <= last_table_entry) &&
           ((*result_student_course)->student_number !=
target_student->student_number)) {
        ++result_student_course;
    }
//    struct  StudentCourse**
result_student_course=bsearch(&target_student,student_course_tabl
e->data,student_course_table->last_available,sizeof(struct
StudentCourse*),compare_student_student_course);
    if (result_student_course > last_table_entry) {
        // Result not found
        result_student_course = NULL;
    }
    if (result_student_course == NULL) {
        printf("[*] Student with the number %u not found in SC\n",
target_student->student_number);
        return -1;
    } else {
        printf("[*] Student with the number %u found in SC, course
number: %u.\n", target_student->student_number,
               (*result_student_course)->course_number);
        // Remove student from student_course table.

_clean_Student_from_StudentCourseTable(result_student_course,
student_course_table);
    }

    return 0;
}
```

使用时需 StudentTable、StudentCourseTable 的指针，并提供一个 target_student 的指针 ，这个 target_student 中只有

student_number 会被用于搜索，所以在初始化此结构体对象的时候其它元素可以不管。

## 五、 改

因为本实验并不要求除了第一列的修改，所以所有的修改都只涉及一张表，通过简单的二分搜索、遍历就可以完成。

下面以 update_student_name()为例：

```
int update_student_name(struct StudentTable *student_table,
struct Student *target_student) {
    // Only student_number in target_student is searched, other
elements in target_student are ignored.
    // From student_table
    struct Student **result_student = bsearch(&target_student,
student_table->data, student_table->last_available,
                                        sizeof(struct Student *),
compare_student);
    if (result_student == NULL) {
        printf("[!] Student with the number %u not found!\n",
target_student->student_number);
        return -1;
    } else {
        printf("[*] Student with the number %u found, name: %s.\n",
target_student->student_number,
               (*result_student)->student_name);
    }

    // Operations
    strcpy((*result_student)->student_name,
target_student->student_name);

    return 0;
}
```

## 六、 查

本实验提供 Student 表根据 student_number、student_name 的查询，Course 表根据 course_number、course_name 的查询，StudentCourse 表根据 student_number、course_number 的查询。和更改相同，所有的查询都可以在一张表中完成。

下面以 select_student_number 为例：

```
int select_student_number(struct StudentTable *student_table,
struct Student *target_student) {
    struct Student **result_student = bsearch(&target_student,
student_table->data, student_table->last_available,
                                        sizeof(struct Student *),
compare_student);
    if (result_student == NULL) {
        printf("[!] Student with the number %u not found!\n",
target_student->student_number);
        return -1;
    } else {
        printf("[*] Student with the number %u found, name: %s.\n",
target_student->student_number,
               (*result_student)->student_name);
    }

    return 0;
}
```

## （五）实验步骤与调试

本实验的内容和功能较多，花费时间较长。调试指针和选择合适的 JSON 库时花费了大量时间。

## （六）实验结果与分析

本实验的功能较多，将每个功能的测试结果一一列出太过繁杂，在本实验的 main.c 的 main（）函数中有许多功能的演示，需要的可自行运行来查看结果。