



# I\*DB—演示与实现

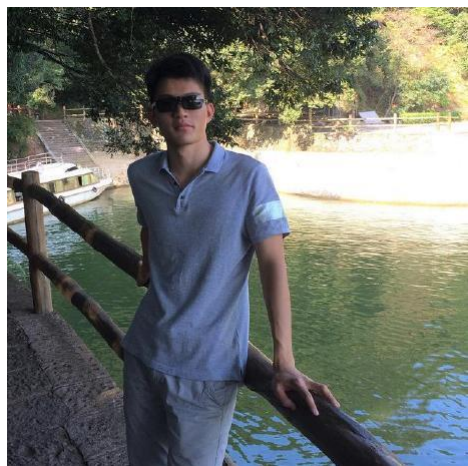
小组成员：张恒，陶友贤，赵旺，修晔良

Github : <https://github.com/taoyouxian/I-DB>

# 组员特色



张恒



陶友贤



赵旺



修晔良



# 组员分工

1. 〈实验一〉 存储管理：陶友贤、张恒(参与)
2. 〈实验二〉 索引模块：赵旺，张恒(Update)
3. 〈实验三〉 SPJ：陶友贤
4. 〈实验四〉 查询分析：修晔良、赵旺(参与)
5. 〈实验五〉 查询优化与执行：张恒，陶友贤(参与)
6. QT系统整合与功能模块处理：陶友贤
7. Github(包含Wiki)资源更新：组员



# Outline

1. 存储管理
2. 索引模块
3. SPJ
4. 查询分析
5. 查询优化与执行
6. 演示

SQL解析（词法、语法、语义）

查询优化（基于规则）

查询操作符（投影、选择、连接）

数据字典

映射表

B+树索引（Hash）

缓存管理

段页式存储管理

文件系统



# 1. 存储管理

- 数据文件组织
  - 段页式表示
- 缓冲区管理
  - LRU算法
- 空闲空间管理
  - Bitmap

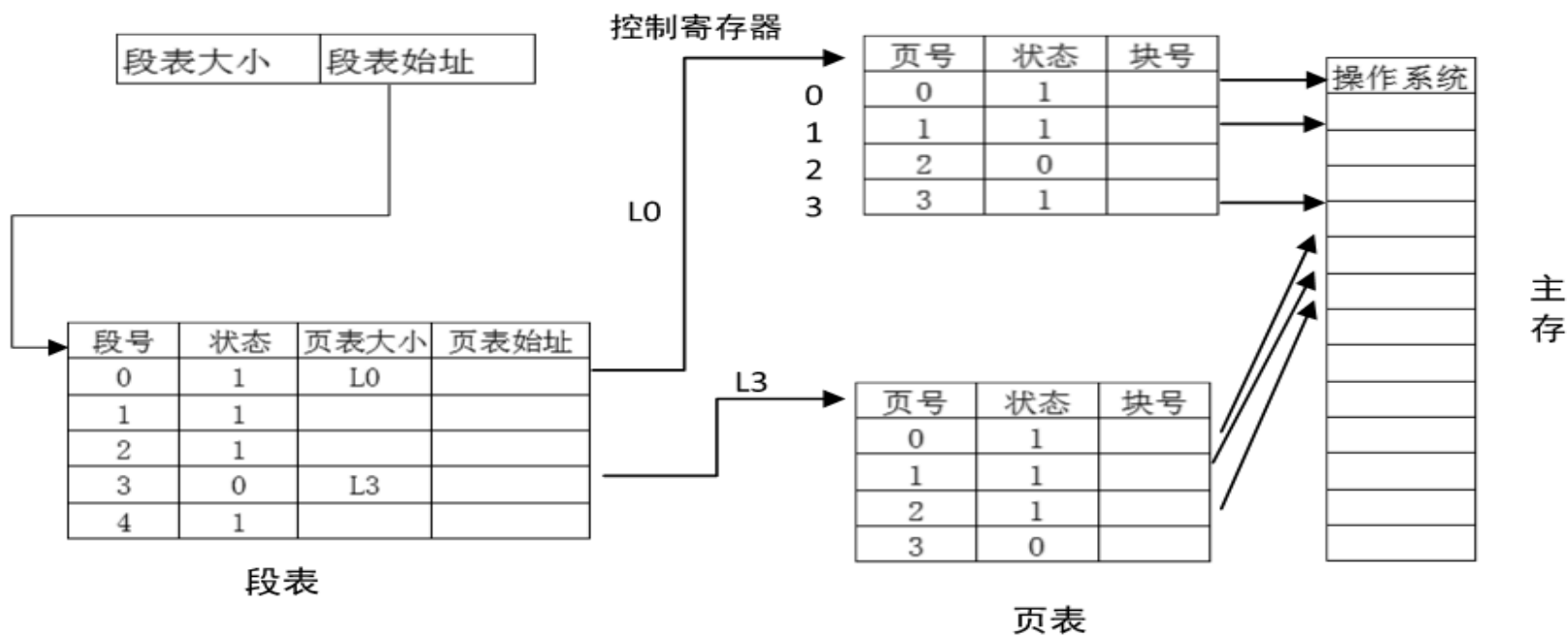


# 1.1 数据文件组织

- 与数据库系统相关的信息如段表始址，段表大小等信息存放在数据库文件的头部。
- 采用段-页-记录的存储层次，页面大小4KB
- 包括元数据段，用户数据段
- 段的大小动态增加，每次从空闲表bitmap中获取空闲页面
- 在每段的起始位置保存页数，起始页的地址等信息

# 1.1 数据文件组织

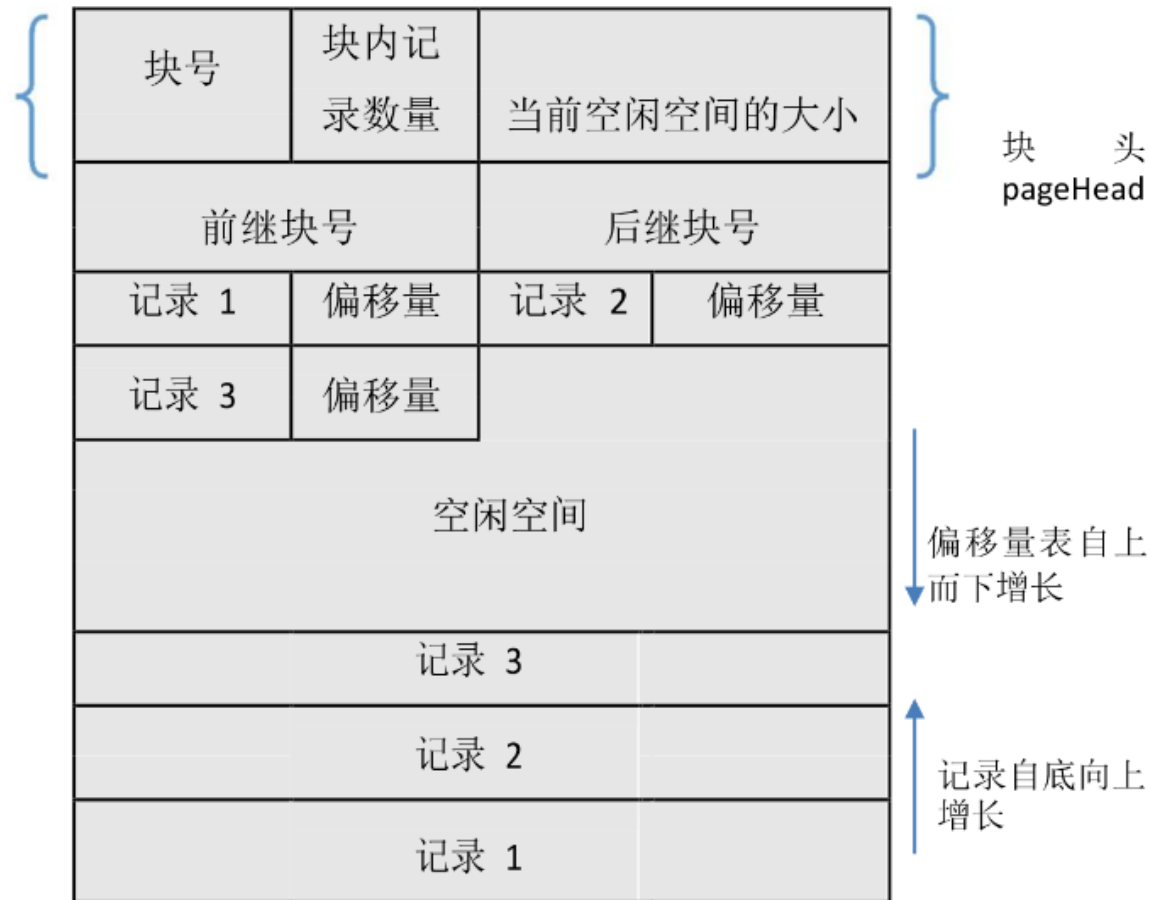
- 段页式结构图





# 1.1 数据文件组织

- 页块结构图

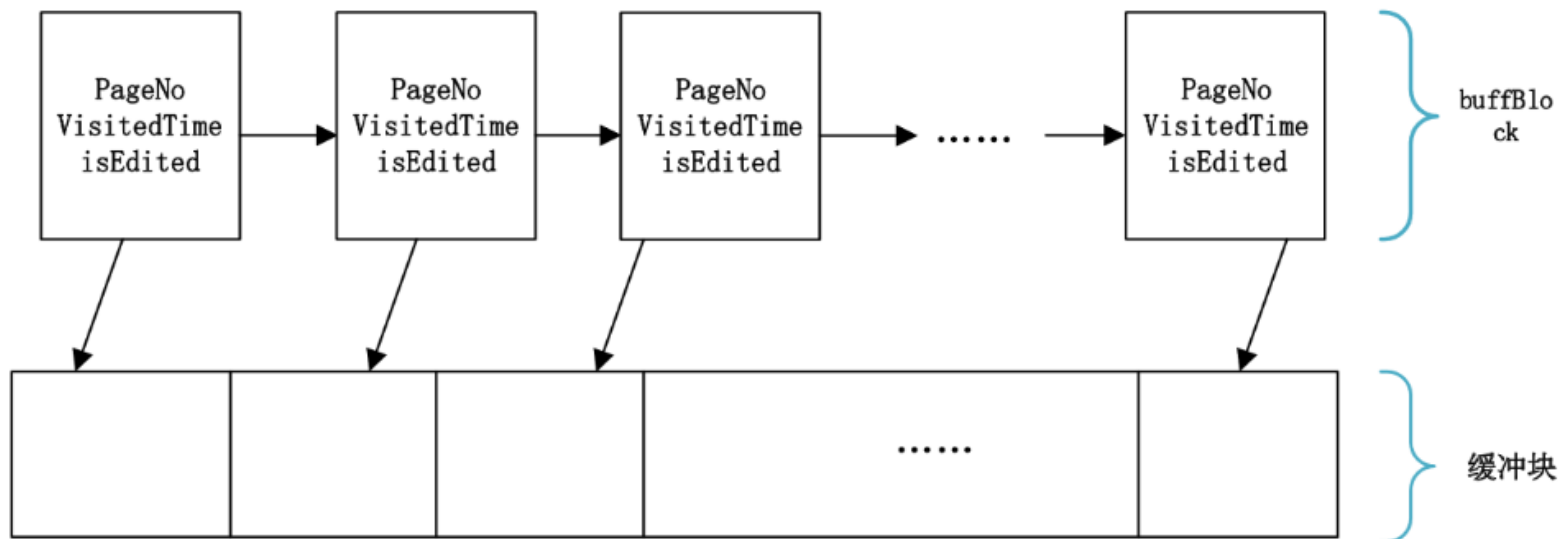




## 1.2 缓冲区管理

- LRU算法
- 缓冲区块与页大小相同

```
/* (缓冲块信息)
*/
struct bufferBlock{
    long pageNo;
    long visitTime;
    bool isEdited;
};
```





## 2 索引模块(B+ Tree)

- 索引的初始化
  - 根节点常驻内存
- 索引维护（增、删、改）
  - Insert
  - Delete
  - Split
  - Merge
  - Pushup
  - Re\_distribute
- 索引的查找
  - 等值/范围

## 2.1 B+树定义

- 树的根节点和叶子结点使用同样的类BTNode
- 非叶子节点pointers = keys+1
- 叶子节点pointers[0,...,count-1]保存record pointers[count]指向其后继点。
- 节点最小可容纳key的数目为2，最大为4

```
struct BTNode
{
    int type; // 1是叶子, 0是内节点, -1是根节点。
    int count; // key 的个数;
    int keys[MAX];
    int pointers[MAX+1]; // 地址
    int parent; // 父节点所在的位置

    Node(): type(-1), count(0), parent(-1) {
        memset(pointers, -1, sizeof(pointers));
    }
};

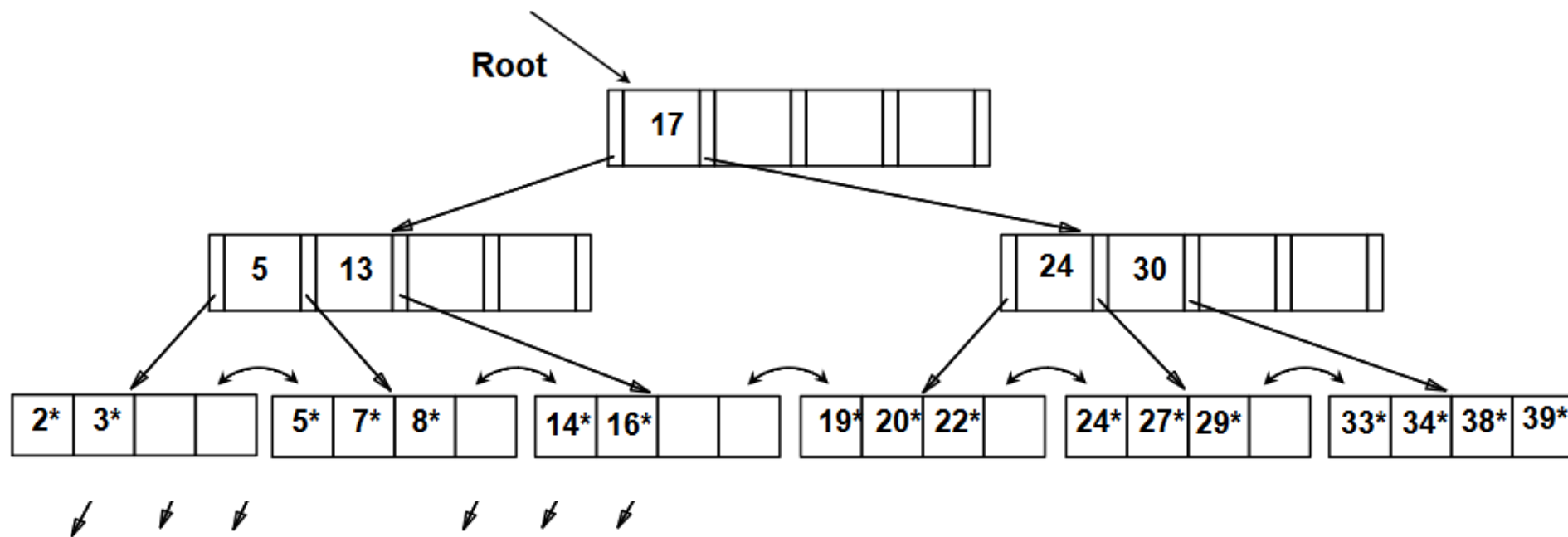
void init_index(FILE *fp, BTNode &root) { }

class BP_tree
{
public:
    BTNode Root;
    FILE *fp;
    long Root_addr;
    BP_tree() {
        init_index(fp, Root, Root_addr);
    }

    void Insert(Record x);
    void Search(int key);
    void Delete(int key);
    int B_serach(int key);
    void ShowTree();
    void Forcepage();
    // ...
};
```

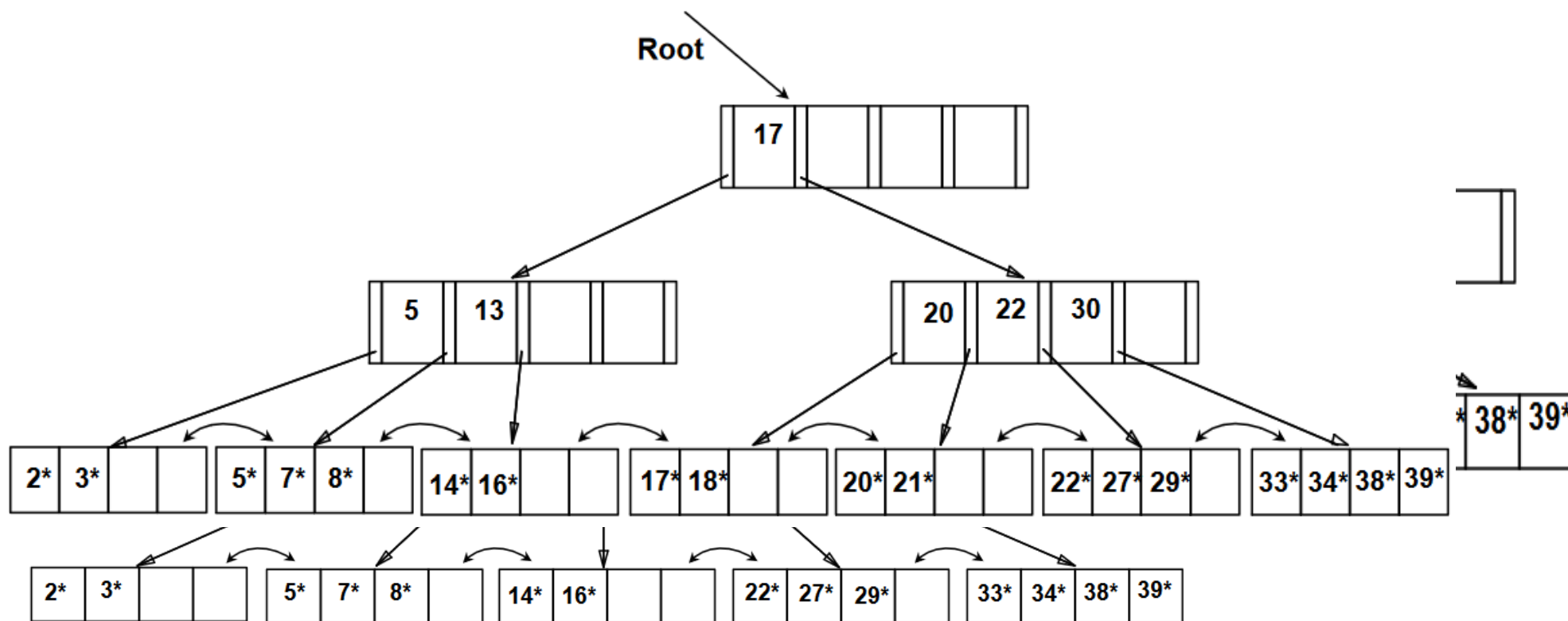
## 2.2 B+树插入举例

### Example B+ Tree After Inserting 8\*



## 2.3 B+树删除举例

- It suffices to re-distribute index entry with key 20; we've re-distributed 17 as well





### 3. 查询执行

- 查询的结果放在一个临时表中，临时表不写回硬盘
- 选择
  - 表扫描
  - 索引扫描
- 投影
- 连接
  - 嵌套循环连接
  - 基于排序的等值连接
  - 基于散列的等值连接



## 3.1 选择

```
int tableScanEqualSelector(struct dbSysHead *head, int dictID,  
char* attribute_name, char* value); //等值选择  
  
int tableScanRangeSelector(struct dbSysHead *head, int dictID,  
char* attribute_name, char* min, char* max); //范围选择
```

扫描就是将关系的元组从磁盘读入到内存中的操作，是其他复杂操作的基础与关键。数据库的磁盘中存放了关系 `employee` 的所有元组，分布在多个数据块中，而数据库管理系统知道哪些块中包含了 `R` 的元组。以块为单位，将包含关系 `R` 的元组的数据块依次读入内存中，这种操作称为表扫描。选择操作是一元操作符，其特点是不需要一次在内存中装入整个关系，而是一次读一个块即可。描述基于表扫描的选择操作可用 SQL 语句 “`select * from employee where age = 20`” 为例，其中 “`where age = 20`” 即为选择的条件，而 “`from employee`” 意味着需要对表 `employee` 进行扫描，这就是一个基于表扫描的等值选择，将 `where` 后面的等值条件改为范围条件，即变成了范围选择。该功能的伪代码大致如下：



## 3.2 连接

- 嵌套循环连接

```
int nestedLoopJoin(struct dbSysHead *head, int employee_dictID, int  
department_dictID);
```

- 基于排序的等值连接

```
int SortJoin(struct dbSysHead *head, int employee_dictID, int  
department_dictID);
```

- 基于散列的等值连接

```
int HashJoin(struct dbSysHead *head, int employee_dictID, int  
department_dictID);
```





## 4. 查询分析

- Flex对SQL语句进行词法分析， Bison进行语法分析
- 功能实现：
  - 增
  - 删
  - 改
  - 查
- 将Qt的查询执行放入in.txt中， 解析结果放在out.txt中， 实现功能的解耦



## 4.1 增-CREATE

- 新建一个表，并设置主键

```
create table stu(id int primary key,name char(50));
```

```
D:\学校课程\数据库原理与实现\work\实验四\0108\exp4.exe  
create table stu(id int primary key,name char(50));  
6  
create  
table:stu  
column:id,name,  
data_type:,INTEGERCHARACTER 50  
primary key:id> Success.
```



## 4.1 增-INSERT

- 向表中插入元组

```
insert into stu(id,name) values(1045,'zhao');
```

```
insert into stu(id,name) values(1045,' zhao' );
5
insert
table:stu
column: id, name
listnode: 1045, ' zhao' > Success.
```



## 4.2 删-DELETE

- 删除表中的一个元组

```
delete from stu where id>1000;
```

```
delete from stu where id>1000;  
4  
delete  
table:stu  
condition:, id>1000> Success.
```



## 4.2 删-DROP

- 删除某个表

```
drop table stu;
```

```
drop table stu;  
3  
drop  
table:stu> Success.
```



## 4.3 改-Update

- 更改表中某相应条件的元组信息

```
update stu set id=1043 where name= 'xiu';
```

```
update  
table:, stu  
setlist:, id=1043  
condition:, name=xiu  
> Success.
```



## 4.4 查-SELECT

- 查询表中满足条件的元组信息
  - 支持多个条件查询: =, <>, !=, >, >=, <, <=。WHERE条件如果有多个，可以AND连接
  - 单表\*查询，并根据ORDER BY条件排序

```
select count(*) from stu where age>20;
```

```
select count(*) from stu where age>20;
6
select
selects:count(*)
table:,stu
where_clause:,age>20
order_clause:id> Success.
```



## 4.4 双表查询

```
select name,age from stu,course where stu.id=course.id order by id ;
```

```
select name,age from stu,course where stu.id=course.id order by id ;
6
select
selects:, name, age
table:, stu, course
where_clause:, stu.id=course.id
order_clause:id> Success.
```





# 5. 查询优化与执行

- SQL执行效果、多表连接、查询优化方法
- 功能实现：
  - 6大SQL执行（Create、Select、Insert、Update、Delete、Drop）
  - 选择、投影下推
  - 运行代价的估计（简单示例）
  - 多表连接（通过动态规划来选择连接顺序和分组）



# 6 亮点

- 6.1 对B+树索引的优化

我们将B+树的根节点常驻在内存中，在插入删除过程中先找到叶子节点，在找的过程中将路径进行压栈处理，则避免更新时候重新从文件中读取各个节点。

- 6.2 数据字典和映射表

我们实现了对数据字典和映射表的内存化，这样避免了对文件不必要的读写。当用户请求一张表的时候，可以直接返回数据表的指针给用户，节省IO，提高效率。

- 6.3 查询解析文件化

我们将需要解析的SQL写入至文件in.sql中，这样可以与解析模块解耦，然后将解析结果按照一定的格式写入out.sql中，一定程度上方便了操作。



- 6.4 Delete操作、Update操作

我们为了后期数据的恢复，在元组记录时保存记录该元组是否被删除isDeleted，如果是，则在读取该元组内容前就可以知道，同时将页中数据个数，删除个数保存起来，这样在执行count(\*)操作时节约时间。

- 6.5 基于迭代器的扫描表和连接算法

这样操作使得上层的程序可以逐条地获取需要的元组数据，不会占用过多的内存空间，提高了效率；同时基于迭代器的连接算法避免了中间结果的输出和读入，使得整个查询执行过程中，从连接→选择→投影成为一条流水线，提高了速度。

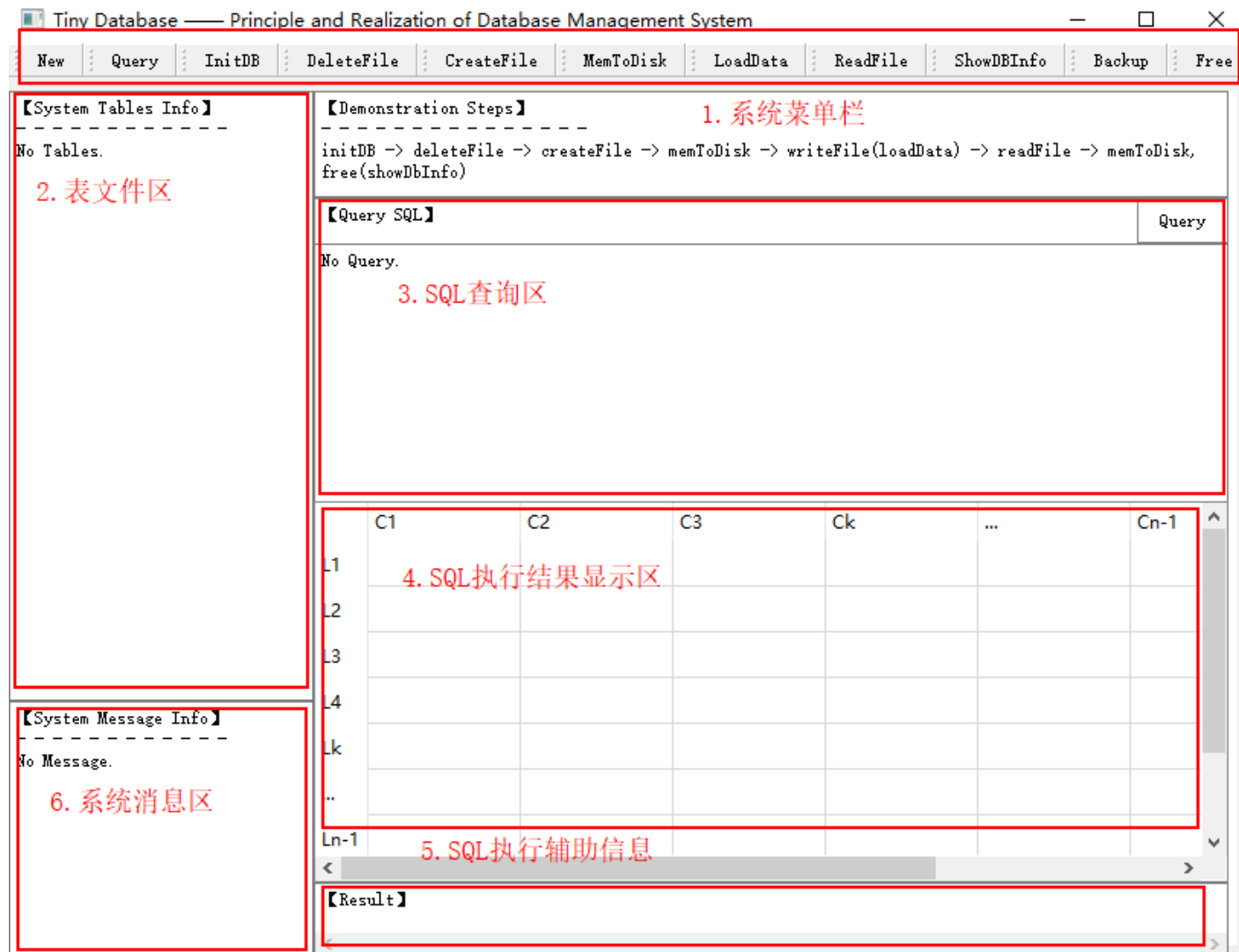
- 6.6良好的UI交互性

我们使用QT管理【I\*DB】中的相关操作，显示运行时间，但是由于内部操作复杂，部分功能未加上，但是已经满足了前四个实验基本的内容，实验五由于时间问题仍待补充。



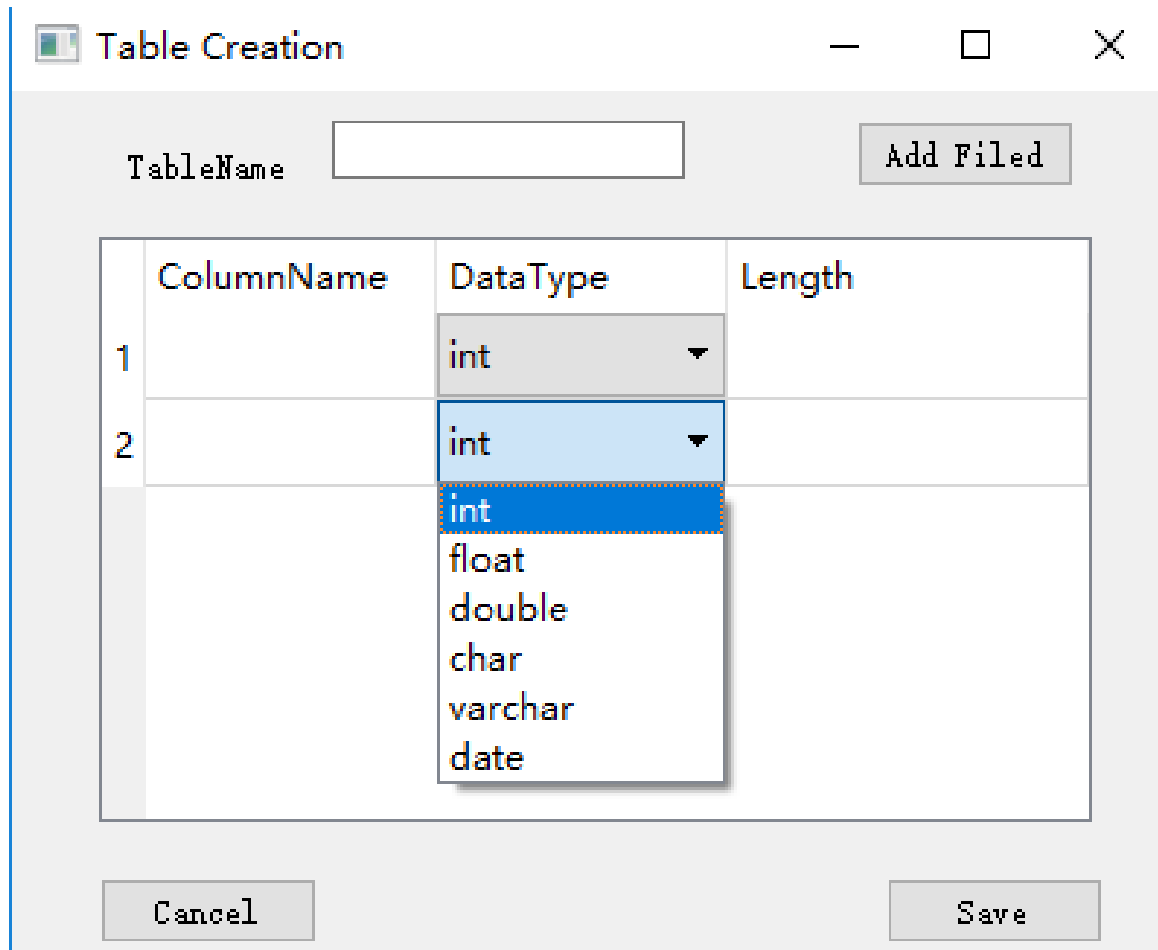
# 7 演示

- 操作界面
- vs2013
- QT5.7.1



# 7 演示

- 新建表



A screenshot of a 'Table Creation' dialog box. At the top, there's a title bar with a small icon, the text 'Table Creation', and standard window controls (minimize, maximize, close). Below the title bar, on the left, is a label 'TableName' followed by an empty text input field. To the right of this field is a button labeled 'Add Filed'. The main area of the dialog contains a table with three columns: 'ColumnName', 'DataType', and 'Length'. The table has three rows. The first row has an index '1' in the first column, an empty 'ColumnName' cell, a dropdown menu in the 'DataType' column showing 'int', and an empty 'Length' cell. The second row has an index '2' in the first column, an empty 'ColumnName' cell, a dropdown menu in the 'DataType' column showing 'int', and an empty 'Length' cell. The third row is currently selected, with its first column cell highlighted in blue. The 'DataType' dropdown for this row is open, showing a list of data types: 'int' (highlighted in blue), 'float', 'double', 'char', 'varchar', and 'date'. At the bottom of the dialog, there are two buttons: 'Cancel' on the left and 'Save' on the right.

	ColumnName	DataType	Length
1		int	
2		int	
		int float double char varchar date	



- Github: <https://github.com/taoyouxian/I-DB>

The screenshot shows the GitHub repository page for 'taoyouxian / I-DB'. The repository has 3 forks, 2 stars, and 3 unwatchers. The 'Wiki' tab is selected, showing a 'Home' page. The page content includes a welcome message, member information, catalog information, and a sidebar with a table of contents.

**Home** Edit New Page

Youxian Tao edited this page 9 hours ago · 23 revisions

Welcome to the I\*DB wiki!

**Member information (Order By Name Letter)**

- 陶友贤 ( [taoyouxian@ruc.edu.cn](mailto:taoyouxian@ruc.edu.cn) )
- 修晔良 ( [xiuyeliang@ruc.edu.cn](mailto:xiuyeliang@ruc.edu.cn) )
- 张恒 ( [zhangheng@ruc.edu.cn](mailto:zhangheng@ruc.edu.cn) )
- 赵旺 ( [zhaowanghappy@163.com](mailto:zhaowanghappy@163.com) )

**Catalog information**

- [Storage Manage](#)
- [B+ Tree Index](#)
- [SPJ](#)
- [Query Execution](#)
- [Query Optimization](#)

Please refer to the user manual for additional information on how to use [I\*DB]:

- Getting Started
  - [Manual Installation](#)
  - [Uploading new datasets](#)
- User Manual
  - [System Overview](#)

**Pages 7**

- Getting Started
  - [Manual Installation](#)
  - [Uploading new datasets](#)
- User Manual
  - [System Overview](#)
  - [Functionalities and Options](#)
  - [Algorithm](#)
- Report
  - [Exp1: Storage Manage](#)
  - [Exp2: B+ Tree Index](#)
  - [Exp3: SPJ Algorithm](#)
  - [Exp4: Query](#)



THANKS FOR WATCHING