

# 数据库系统设计

## Database Design



李文栋 3110104353

鲁懿帆 3110101132

张力乾 3110103937

# 数据库系统设计

## Database Design

### 综述

为了更好的理解数据库的原理，获得实践经验，我们组编写了这个模仿 mysql 的 miniSql 小型数据库引擎，它能够实现最基本的 SQL 语句。

### 编译环境

vs2012 , window7&8

### 系统概述

#### 1. 交互模式

命令行

#### 2. 支持数据类型

- 1) int
- 2) char(n)，其中 char(n)满足  $1 \leq n \leq 255$  。
- 3) float

#### 3. 支持操作

##### 1) 表的定义和删除

一个表最多可以定义 32 个属性，各属性可以指定是否 unique；支持单属性的主键定义。

##### 2) 索引的建立和删除

对于表的主属性自动建立 B+树索引，对于声明为 unique 的属性可以通过 SQL 语句由用户指定建立/删除 B+树索引（因此，所有的 B+树索引都是单属性单值的）。

##### 3) 查找记录

可以通过指定用 and 连接的多个条件进行查询，支持等值查询和区间查询。

##### 4) 插入和删除记录

支持每次一条记录的插入操作；支持每次一条或多条记录的删除操作。

## 4. 语法说明

语法与 SQL 语句相符

## 测试

### 先进行 minisql 约束完整性测试

首先验证表的列出现不支持类型的情况，如图所示，输出了错误信息

```
C:\Users\luyifan1993\Desktop\MiniSqlTest\Release>MiniSqlTest
create table cuobiao{
chucuo errortype,
zheng int,
};
An error as follow had occurred :
Unsupported type : errortype
```

下面验证创建表操作，如图所示，输出了成功信息

```
create table biao{
zheng int,
fu float,
chuan char(8),
weizheng int unique,
weifu float unique,
primary key (weifu)
};
Table biao created.
```

下面测试创建表时，表名重复的情况，如图所示，错误信息如下：

```
create table biao{
zheng int,
fu float,
primary key (zheng)
};
An error as follow had occurred :
Already exists table : biao
```

下面执行创建索引，索引对应表不存在的情况，如图所示，错误信息如下：

```
create index biaoindex on nonexistbiao (weizheng);
An error as follow had occurred :
No such table : nonexistbiao
```

下面执行创建索引，索引对应所在列不存在的情况，如图所示，错误信息如下：

```
create index biaoindex on biao (nonexistattribute);
An error as follow had occurred :
No such column in table biao : nonexistattribute
```

下面执行创建索引操作，如图所示，正确创建结果如下：

```
create index biaoindex on biao (weizheng);
Index biao created.
```

下面执行删除不存在的索引操作，如图所示，错误信息如下：

```
drop index nonexistindex;
An error as follow had occurred :
No such index : nonexistindex
```

下面执行删除正确索引操作，如图所示，正确删除索引提示如下：

```
drop index biaoindex;  
Index biaoindex dropped.
```

下面执行删除不存在表操作，如图所示，错误信息如下：

```
drop table nonexistentbiao;  
An error as follow had occurred :  
No such table : nonexistentbiao
```

下面执行删除存在的表操作，如图所示，删除成功结果如下：

```
drop table biao;  
Index biaoPindex dropped.  
Table biao dropped.
```

下面建立一张新的表，如图所示，建表成功

```
create table withindex(  
  ifield int,  
  ffield float,  
  cfield char(10) unique,  
  primary key (ifield)  
);  
Table withindex created.
```

下面执行插入数据到不存在的表，如图所示，错误信息如下：

```
insert into withoutindex values(123, 1.23, '1234567890');  
An error as follow had occurred :  
No such table exists : withoutindex
```

下面执行插入数据类型和表不匹配的情况，如图所示，错误信息如下：

```
insert into withindex values(12.3, 123, '1234567890');  
An error as follow had occurred :  
Type doesn't match : 12.3
```

下面插入两条数据到存在的表中，如图所示，插入成功提示如下：

```
insert into withindex values(123, 1.23, '1234567890');  
Insertion succeeded  
  
insert into withindex values(1234, 1.23, '2134567890');  
Insertion succeeded
```

下面执行插入数据不满足 unique 条件约束，如图所示，错误信息如下：

```
insert into withindex values(123, 1.23, '1234567890');  
An error as follow had occurred :  
Already exists value '123' at unique column : ifield
```

下面测试表的选择操作，如图所示，执行选择操作成功结果如下：

```
select ifield, cfield  
from withindex  
where ifield=123 and ffield>=1.0 and cfield!='nooonexist';  
ifield      cfield  
123         1234567890  
Totally 1 item(s).
```

下面执行表选择操作，数据不符合类型要求，如图所示，错误信息如下：

```
select ifield, cfield  
from withindex  
where ifield=1.36;  
An error as follow had occurred :  
Type mismatch at ' 1.36 '
```

下面执行删除操作，约束中存在表不存在的列，如图所示，错误信息如下：

```
delete from withindex
where nonexistentattribute = 3;
An error as follow had occurred :
No such column exists in table withindex : nonexistentattribute
```

下面执行删除操作，当没有一条满足删除的约束条件，如图所示，删除数目信息如下：

```
delete from withindex
where ifield=1;
0 item(s) deleted
```

下面删除表中的所有数据，删除成功信息，如下所示：

```
delete from withindex
where ifield>100;
2 item(s) deleted
```

下面 drop table，结束约束测试，进行 index 提升操作效率的测试：

```
drop table withindex;
Index withindexPindex dropped.
Table withindex dropped.

quit;
```

然后进行 Index 提升效率测试

#### 1) 测试方法

为了测试操作执行效率，我们在程序中加入了时间的变量记录开始和结束时间：

```
#ifdef TOTAL_TIME
#include "ctime"
time_t tottimestart;
time_t tottimeend;
#endif
```

然后在程序末尾输出总共的执行时间：

```
int main(){
#ifdef TOTAL_TIME
    tottimestart=clock();
#endif
//insert 10000
#ifdef TOTAL_TIME
    tottimeend=clock();
    cout<<"Total time : "<<{tottimeend-tottimestart}*1.0/CLOCKS_PER_SEC<<endl;
#endif
} //end of main
```

## 2) 插入测试

创建两个文件(可在附件中找到)，内容都是建立一张如下的表，然后插入 10000 条数据，不同的是，在第一个文件(with.sql)中，插入语句之前先在 cfield 列上建立索引；而第二个文件(without.sql)则不建立该索引。

```
create table withindex{
ifield int,
ffield float,
cfield char(10) unique,
primary key (ifield)
};
```

然后分别执行这两个文件，结果用时如下：

有索引：(详细结果在 result1 中)

```
Total time : 16.589
```

无索引：(详细结果在 result2 中)

```
Total time : 522.822
```

可以看出没有建索引时插入操作明显慢很多。这是因为每次插入操作时，为了保证唯一性，系统先要执行选择操作确保无重复值存在于表中，而建立了索引以后的 B+树查询自然比不建索引线性扫描快。

## 3) 选择测试

在建立好表并插入 10000 条数据的基础上，我们继续进行选择操作时的测试：

创建两个文件(可在附件中找到)，内容都是选择出最后插入的 10 条记录。不同的是，在第一个文件(indexselect.sql)中，选择操作条件的列上有索引，而第二个文件(select.sql)在开头将索引删除，因此选择操作条件的列上无索引。

```
select * from withinindex where cfield='NBQLRWDXEH';  
select * from withinindex where cfield='YVCBKVKSKL';  
select * from withinindex where cfield='YUILXUADAM';  
select * from withinindex where cfield='MGWNRCKTTB';  
select * from withinindex where cfield='VZSVKDMIEH';  
select * from withinindex where cfield='VPRYCFFMQC';  
select * from withinindex where cfield='WLRUCIYYSG';  
select * from withinindex where cfield='SJNZIOTGTI';  
select * from withinindex where cfield='AAVRWDSMAX';  
select * from withinindex where cfield='ELWWXQOIYD';
```

然后依次执行这两个文件，结果用时如下：

有索引：(详细结果在 result3 中)

```
Total time : 0.011
```

无索引：(详细结果在 result4 中)

```
Total time : 1.052
```

可以看出，在加入索引之后，选择操作的耗时明显下降。

#### 4) 总结

在所有涉及选择操作的操作中，用 B+树进行查询会显著提高效率。

## 小组成员及分工

李文栋：API Catalog Interpreter

鲁懿帆：RecordManager， BufferManager

张力乾：IndexManager