

Kermit Address DB 系统说明书

作者：陈科名 (Kermit)

一、功能说明

Kermit Address DB 实现了一个简单的用于存储通信簿的关系型数据库，使用二进制文件存储数据，并实现了简化的 SQL 的 INSERT 语句和 SELECT 语句，且两种语句均只用了单个元组大小的内存缓冲区。该数据库支持四种类型的字段：定点数 int、浮点数 float、定长字符串 char、变长字符串 varchar，并且在查询语句中支持多字段查询，还可以使用 =、!=、>、<、>=、<= 运算符。

本实验的数据库仅要求使用一个关系模式，用于实现通信簿，故本系统的关系模式设置为了：

字段名	name	age	phone	email	resume	description
类型	char(20)	int	char(40)	char(40)	char(128)	varchar(1024)
备注	名字， 20 字节 定长字符串	年龄，4 字节的定 点数	电话， 40 字节 的定长字 符串	邮箱， 40 字节 的定长字 符串	简历， 128 字节 的定长字 符串	描述，最长 1024 字节的可 变长字符串

二、使用说明

1、启动数据库

命令如下

```
addressdb # 使用当前文件夹下的 data 数据作为通信簿数据源，没有则会自动创建

addressdb -p <data path> # 使用 p 参数指定通信簿数据源的路径，支持同一个计算机上存在多个数据源

addressdb --help # 查看使用说明
```

启动后的界面如下

```
~/MyCode/@ruc/@database/addressdb/dist master ./addressdb
#####
#   Kermit Address DB   #
#####

warning: this program uses gets(), which is unsafe.
addressdb> █
```

查看使用说明如下

```
~/MyCode/@ruc/@database/addressdb/dist master ?1 ./addressdb --help
Usage: addressdb [options]

Options:
  -p, --path=<data path>  The path the data locates in.
  -h, --help               Show usage.
```

2、数据库操作

进入数据库后可使用 `help` 命令查看数据库的操作帮助，界面如下

```
~/MyCode/@ruc/@database/addressdb/dist master ?1 ./addressdb
#####
#   Kermit Address DB   #
#####

warning: this program uses gets(), which is unsafe.
addressdb> help
Commands:
  insert value (value0,value1,...),(value0,value1,...),...
    Insert into address database.
  select field0,field1,... where field0<op>value0 and field1<op>value1 ...
    Find value from address database. 'op': =,>,<,>=,<=,!=.
  help    Show usage.
addressdb> █
```

当输入命令不正确，将会提示

```
addressdb> jn
Command is not found. Input `help` to check available commands.
addressdb> █
```

当语法不正确，将会提示

```
addressdb> insert value (12
Syntax error.
```

3、数据库 INSERT 操作

数据库插入命令的语法为简化后的 INSERT 指令，语法如下

```
insert value (value0,value1,...),(value0,value1,...),...
```

由于本实验的关系模式已经确定，故插入值应当如下

```
insert value (<name>,<age>,<phone>,<email>,<resume>,<description>),  
(<name>,<age>,<phone>,<email>,<resume>,<description>),...
```

往数据库插入操作的示例如下

```
~/MyCode/@ruc/@database/addressdb/dist master +2 !1 ./addressdb  
#####  
#   Kermit Address DB   #  
#####  
  
warning: this program uses gets(), which is unsafe.  
addressdb> insert value (kermit,23,18080008000,keming@ruc.edu.cn,My resume.,My desc.)  
1 item has been inserted.  
addressdb> insert value (dbiir,18,18108274362,dbiir@ruc.edu.cn,My name is DBIIR.,I'm outstanding.)  
1 item has been inserted.  
addressdb> insert value (deke,28,18827461893,deke@ruc.edu.cn,I am DEKE lab of RUC.,State of art!)  
1 item has been inserted.  
addressdb> |
```

注：这三条数据所对应的数据文件已放置在项目根目录，以作示例

4、数据库 select 操作

数据库查询命令的语法为简化后的 SELECT 指令，语法如下

```
select field0,field1,... where field0<op>value0 and field1<op>value1 ...  
# 其中的 <op> 为运算符，包括：=,>,<,>=,<=,!=
```

查询操作的示例如下，使用的数据源为上方 insert 操作插入的数据

```

~/My/@ruc/@database/addressdb/dist master +2 !2 ./addressdb -p ../data
#####
# Kermit Address DB #
#####

warning: this program uses gets(), which is unsafe.
addressdb> select * where name!=kermit and name!=dbiir
-----(1)-----
name: deke
age: 28
phone: 18827461893
email: deke@ruc.edu.cn
resume: I am DEKE lab of RUC.
description: State of art!
-----
1 item has been selected.
addressdb> select name,age,resume where age>15 and age<=23
-----(1)-----
name: kermit
age: 23
resume: My resume.
-----
(2)-----
name: dbiir
age: 18
resume: My name is DBIIR.
-----
2 item has been selected.
addressdb>

```

三、技术解析

1、代码结构

```

.
├── data                # 实例数据源，里面存储了三条数据供展示
├── dist                # 编译出的二进制程序存放处
│   ├── aarch64_linux  # aarch64 GNU/Linux
│   ├── arm64_darwin   # arm64 Darwin (MacOS)
│   └── x86_64_linux   # x86_64 GNU/Linux
└── src                # 代码
    ├── client.c        # 客户端交互的模块，实现 SQL 命令的交互与解析
    ├── client.h        # 客户端交互的模块的头文件
    ├── io.c            # 磁盘 IO 的模块，与二进制数据库文件交互
    ├── io.h            # 磁盘 IO 的模块的头文件
    ├── main.c          # 入口文件，命令行交互、启动、查看帮助指令等的实现
    ├── operate.c       # SQL 操作的模块，实现了 SQL 语句的解析与执行逻辑
    ├── operate.h       # SQL 操作的模块的头文件
    ├── schema.c        # 关系模式相关的模块，用于创建关系模式、提供关系模式查询、
                        # 字段查询、往缓冲区读写等
    └── schema.h        # 关系模式相关的模块的头文件

```

其中编译出的二进制文件都在 ./dist 里，我手边有三台机器，分别编译成了三份：

1. aarch64_linux: 使用 gcc 在 arm ubuntu 20.04 上编译

2. arm64_darwin: 使用 clang 在 arm MacOS 13 上编译
3. x86_64_linux: 使用 gcc 在 x86 ubuntu 20.04 上编译

2、核心模块解析

核心模块主要是 `operate.c` 与 `schema.c`

(1) schema.c

该模块定义了模式的二进制结构，对于每个字段来讲，模式结构如下

```
struct address_schema_field_item
{
    /**
     * 一些字段标志
     * * 第一位: 为 0 则定长, 为 1 则变长
     * * 第二、三位: 00 为 int, 01 为 float, 10 为 char, 11 为 varchar
     * * 后几位: 保留标志
     */
    unsigned char sign;
    int length;           // 字段长度, varchar 的实际长度由字段前 4 字节 (int 长度) 确定
    char name[field_name_length]; // 字段名
};
```

- 前 8 位：标志位，其中第一位表示该字段是定长字段或者变长字段，第二和第三位共同表示字段类型，后 5 位为保留位
- 第 9-40 位：字段长度，单位字节，表示该字段的数据的长度，若是定长字段则是实际长度，若是变长字段（varchar），则表示最大长度，其实际长度在数据中由字段数据的前 32 位动态确定。
- 第 41-200 位：字段名

该模块还确定了元组的二进制结构，分为长度段和数据段，如下所示

| 元组长度位: 32 位 | 数据段 |

对于数据段，分为变长字段和定长字段。对于定长字段而言，仅保存数据，其长度由上方关系模式确定；对于变长字段而言，结构如下

| 数据 | # 定长字段数据
| 字段长度位: 32 位 | 数据 | # 变长字段数据

该模块内的所有函数及注释如下

```

/** 创建通信簿用到的模式定义 */
void create_address_schema(const int address_schema_length);
/** 根据字段名获取字段项索引号 */
int get_address_schema_index_from_name(char *field_name);
/** 根据索引号获取字段项定义 */
struct address_schema_field_item *get_address_schema_from_index(int
index);
/** 根据字段名获取字段项定义 */
struct address_schema_field_item *get_address_schema_from_name(char
*field_name);
/** 根据字段名获取字段项类型 */
int get_address_schema_type_from_name(char *field_name);
/** 根据索引号获取字段名 */
char *get_address_schema_field_name_from_index(int index);
/** 获取缓冲区应当的长度 */
int get_buf_length(int tuple_length);
/** 获取缓冲区表示长度的长度 */
int get_buf_tuple_length_length();
/** 从缓冲区获取元组长度 */
int get_tuple_length(unsigned char *buf);
/** 往缓冲区写元组长度 */
void set_tuple_length(unsigned char *buf, int length);
/** 从缓冲区获取数据段 */
unsigned char *get_tuple_data(unsigned char *buf);
/** 创建一个空元组缓冲区 */
unsigned char *create_one_buf();
/** 删除一个缓冲区 */
void delete_one_buf(unsigned char *buf);
/** 从缓冲区获取字段值 */
void *get_tuple_data_field(unsigned char *buf, char *field_name);
/** 往缓冲区写数据 */
int write_tuple(unsigned char *output_buf, int valuec, char **value);

```

(2) operater.c

该模块为主要操作模块，实现了 insert 和 select 语句。

对于 insert 语句，首先解析语法，然后查询关系模式，将输入数据转化为对应数据类型，之后调用 schema.c 模块构造缓冲区二进制数据，最后将缓冲区通过 io.c 模块写入到二进制文件中。

对于 select 语句，首先解析语法，然后将筛选条件和输出字段条件转化为程序可读的结构体，之后依次从二进制文件读入一个元组到缓冲区，缓冲区大小为 1 个元组的大小，调用 schema.c 模块获取关系模式信息，根据筛选操作符进行匹配；若匹配成功则进入到输出阶段，根据指定的输出字段输出信息，然后读入下一个；若失败，则直接读入下一个元组到缓冲区。

(3) 其他模块

其他模块实现的功能是命令行的实现、SQL 命令的实现、磁盘 IO 的抽象等功能，较为简单且不是本实验的核心目标，故不作详细解析。但代码中记录了注释，可参考注释~