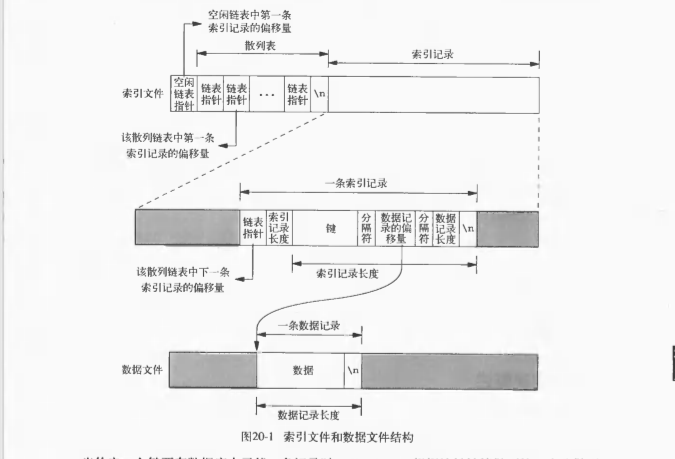
基于前人的注释和项目总结，我试着加入我自己的需求和理解，对前人的成果进行补充说明，将给还未开始数据库项目的同学作为参考（因为很多东西只可意会不可言传，具体要根据个人情况去理解）：

项目步骤：

1. 先对书本20章节浏览一遍，有个大概的认识，函数源码不用看，对数据库有个大概的认识之后就可以了，再阅读前人的总结文档，旨在理解数据库的结构： 索引文件 + 数据文件的构造和了解数据库创建的大体过程即可，目的是该下列这个图给理解透了，看源码时候更容易理解（看代码时候有疑惑可以随时回来看这个结构图）



个人理解的索引文件构造：索引文件可以看作一个顺序表（底层是数组），在散列表部分，每一个链表指针都是一条单链表的头结点，每个结点用于存储插入的索引记录的地址（相对于索引文件头的偏移量），通过地址可以查找到相应的索引记录（索引记录在散列表之后），一条索引记录的整体构造如图所示，自己理解，而索引记录中的数据记录的偏移量和数据记录长度则用于往数据文件中查询和修改数据记录

1. 理解之后节可以去看源码（方式有很多，找适合自己的方式去看代码，我比较喜欢顺着整体脉络去看代码，遇到还没学的函数可以用跳转的方式去看需要的代码，再回到主体脉络；也可以安利另外一种方法：先把数据库函数分类看懂，再串成一个完整的程序）
2. 下面写一下我对数据库源码用到的函数所查到的资料和自己的理解，先看一下也是可以的，不过还是建议自己先去看代码，有不懂的理解不了的再回来找需要的函数的知识点，这样对能力的提升更有帮助，这里放出来是为了减少各位去百度的时间，以及我理顺一下自己的对函数理解的思路。

主要几个函数部分：

* 1. 错误函数部分：

1. 可变参数处理宏定义

#include <stdarg.h>

void va\_start(va\_list ap, last);

type va\_arg(va\_list ap, type);

void va\_end(va\_list ap);

void va\_copy(va\_list dest, va\_list src);

有关VA\_LIST的用法

VA\_LIST 是在C语言中解决变参问题的一组宏

VA\_LIST的用法：

       （1）首先在函数里定义一具VA\_LIST型的变量，这个变量是指向参数的指针

       （2）然后用VA\_START宏初始化变量刚定义的VA\_LIST变量，这个宏的第二个参数是第一个可变参数的前一个参数，是一个固定的参数。

       （3）然后用VA\_ARG返回可变的参数，VA\_ARG的第二个参数是你要返回的参数的类型。

       （4）最后用VA\_END宏结束可变参数的获取。然后你就可以在函数里使用第二个参数了。如果函数有多个可变参数的，依次调用VA\_ARG获取各个参数。

（2）vsnprintf函数

头文件：#include  <stdarg.h>

函数原型：int vsnprintf(char \*str, size\_t size, const char \*format, va\_list ap);

函数说明：将可变参数格式化输出到一个字符数组

参数：str输出到的数组，size指定大小，防止越界，format格式化参数，ap可变参数列表函数用法

（3）int snprintf(char\*str, size\_t size,constchar\*format, ...);

函数说明:

最多从源串中拷贝size－1个字符到目标串中，然后再在后面加一个0。所以如果目标串的大小为size的话，将不会溢出。

函数返回值: 若成功则返回欲写入的字符串长度，若出错则返回负值。

size是限定最终生成的dest的字符数,最多拷贝size-1个字符; 一般情况下size会取sizeof(dest),这是为了dest不溢出.(destination)

snprintf(str,sizeof(str),"%02x",a);

* 1. 加解锁函数部分：

调用的核心函数是

int lock\_reg(int fd, int cmd, int type, off\_t offset, int whence, off\_t len) {

struct flock lock;

lock.l\_len = len; //锁的区域长度

lock.l\_start = offset; //锁的开始字节的偏移量

lock.l\_type = type; //锁的类型（F\_RDLCK共享读锁，排他写锁RLCK，解锁F\_UNLCK）

lock.l\_whence = whence;//设置开始的字节

return (fcntl(fd, cmd, &lock)); // F\_GETLK 获取锁的属性，如果没有目标区域没有锁，将l\_type设置为F\_UNLOCK

F\_SETLK， F\_SETLKW 均为设置锁，如果目标区域已加锁，前者直接退出，后者阻塞，直到锁被释放。

}

3，DBHANDLE db\_open(const char \*pathname, int oflag, ...);

函数功能：创建数据库结构体DB实例，根据参数去创建索引文件和数据文件，初始化索引文件函数部分

1. 构造hash（散列顺序表）大小的字符串和链表指针大小的字符串
2. 为数据库实例分配空间，并为DB结构体的成员赋值，
3. 根据参数构造数据库名字和索引文件名字
4. 根据参数去和用open()函数创建和打开数据库文件名。用返回的文件描述符赋值给db,如果失败则清空db和其他操作
5. 根据参数给索引文件加锁，并用系统函数fstat函数去获取文件的状态字，根据状态字去判断是否需要初始化索引文件的空闲链表和散列链表并写到索引文件（具体步骤看代码）
6. 将db结构的db->idx 定位到散列表后第一条索引记录的位置
7. 返回db结构的指针（句柄,一种特殊的指针）

4，根据给定的键读取一条数据 \*/

char \*db\_fetch(DBHANDLE h, const char \*key)

1. 调用(\_db\_find\_and\_lock(db, key, 0)函数，根据键值key在散列链中查找索引记录，并对相应的散列表加锁，返回0 表示成功，返回-1 表示失败
2. 调用\_db\_hash(db, key)获取键所在链表的hash值（相当与数组的下标）
3. 获取索引记录所在散列链的链头指针，根据最后一个参数对当前的散列链加锁
4. 获取当前链表的第一个指针的整型值
5. 进入while循环，遍历散列链，比较每个结点的key值和要寻找的key值，db->ptroff保存上一个结点的指针值，以便删除当前结点
6. 找不到则查找错误计数器自增，找到则读数据文件的数据
7. 读取结束后对索引文件中索引记录所在的散列表进行解锁
8. 返回所读取数据的字符串

5, static off\_t \_db\_readidx(DB \*db, off\_t offset)

1. 函数功能：从索引文件的指定偏移量的位置取索引记录，成功则返回链表中下一条记录的偏移量
2. 将读写位置定位当前的记录在索引文件的偏移量的位置，并返回偏移量
3. 读取并获取当前记录的指向当前散列链表下一条的索引记录的偏移量存进db->ptrval和当前索引记录的长度存进db->idxlen
4. 读取当前的索引剩下的不定长部分读进db的db->idxbuf
5. 从db->idxbuf中获取当前索引记录对应的数据记录的偏移量赋值给db->datoff和数据记录长度db->datlen
6. 返回下一条索引记录的偏移量

6,db\_writeidx(DB \*db, const char \*key, off\_t offset, int whence, off\_t ptrval)

（1）如果被删除的索引记录中指向下一条记录的偏移量有效则根据相应的格式将被删除索引的下半部分赋值给索引记录缓冲区，并获取长度db->idxbuf

1. 根据相应的格式构造要写入索引记录的头部存入asciiptrlen
2. 根据whence判断是否对索引文件加锁，是的话在哈希链表后对索引文件加锁(此出要用一条索引去写入off\_t offset的位置所以需要加锁)
3. 将读写位置定位到被offset的位置并返回给db->idxoff
4. 通过iovec结构体写入索引记录
5. 解锁

7,t db\_delete(DBHANDLE h, const char \*key)

（一）调用\_db\_find\_and\_lock(db, key, 1)，如果找到相应键值的索引记录，对其所在的散列表加写锁

（二）找到索引记录的偏移量调用\_db\_dodelete(db); //删除是指用宏SPACE填充，对于索引，填充键

需要的操作：删除对应数据、删除对应索引（只删除key）、在空闲链表中添加已删除索引

1. 获取db结构中数据记录缓冲区和索引记录缓冲区的指针，对其指向的内容用空格替换，达到置空的效果
2. 对索引文件中的空闲链表加写锁（排它锁）
3. 调用\_db\_writedat(db, db->datbuf, db->datoff, SEEK\_SET);先对数据文件加写锁，然后将读写位置定位到要删除的数据记录的位置，用db结构体的数据记录缓冲区（已经置为空格）和数据记录长度构造iovec结构体，将结构体的内容写到数据文件中要删除的记录位置，达到删除的效果
4. 获取空闲列表的指针
5. 获取被删除索引记录指向下一条记录的偏移量
6. 调用\_db\_writeidx(db, db->idxbuf, db->idxoff, SEEK\_SET, freeptr);去重写被删除索引记录
7. 调用\_db\_writeptr将被删除结点的偏移量存入空闲链表的新的结点中（先判断指针的合法性，在定位读写位置到空心啊链表的前面，写入偏移量到空闲链表前面）
8. 调用\_db\_writeptr(db, db->ptroff, saveptr)将被删除下一条索引记录的偏移量的ASCII值写到被删除所以前一条索引记录中头部指向下一条索引偏移量的的指针中（单链表删除一个结点的操作）
9. 对空闲链表解锁

8,int db\_store(DBHANDLE h, co nst char \*key, const char \*data, int flag)

函数功能：修改数据 根据参数 DB\_INSERT、DB\_REPLACE、DB\_STORE决定是插入，替代还是两者合一

（一）验证参数，判断其有效型，无效则退出修改函数

（二）获取键长，数据长并对其检验

（三）根据键值在索引文件中找相应的索引记录，找不到则

（1）如果参数是替代索引记录则报错释放对索引记录的锁并退出

（2）db\_readptr读取当前要添加的记录处在的hash链的偏移量

（3）根据键长和数据记录长度去空链表中查找是否存在键长和数据长相等的键

找不到则(但单链表前插入法插入新的结点)->将新的索引数据写入数据文件末尾-> 将新的索引文件写入索引文件末尾-> 将新添加的索引记录中的指针指向未添加这条记录前的hash链的头索引记录（改变索引记录的第一个位置的指针即可）-> 将新记录加到对应的散列链的链首

找到则 -> 将匹配到的索引记录的空间从空闲链表中删除(单链表删除结点)并将数据记录偏移量和长度赋值非db结构体-> 向数据文件中写入要添加的数据记录 -> 将新添加的索引记录中的指针指向未添加这条记录前的hash链的头索引记录（改变索引记录的第一个位置的指针即可）-> 将新记录加到对应的散列链的链首

1. 根据键值在索引文件中找相应的索引记录，找到则
2. 如果要插入记录则插入失败
3. 修改数据时候

如果长度不等-> 将旧的索引删除-> 将新数据添加到索引文件和数据文件的末尾 -> 将新记录添加到对应散列链的链首 –

如果长度相等则更新数据记录即可

（3）对索引文件相应散列链进行解锁，返回文件状态码

课外百度补充：

用到的其他函数

（1）int open(constchar\*pathname,int flags,mode\_tmode);

系统调用函数

作用：打开和创建文件。

返回值：成功则返回文件描述符，否则返回-1

对于open函数来说，第三个参数仅当创建新文件时（即 使用了O\_CREAT 时）才使用，用于指定文件的访问权限位（access permission bits）。

pathname 是待打开/创建文件的POSIX路径名（如/home/user/a.cpp）；flags 用于指定文件的打开/创建模式，这个参数可由以下常量（定义于fcntl.h）通过逻辑位或逻辑构成。

O\_RDONLY只读模式

O\_WRONLY只写模式

O\_RDWR读写模式

打开/创建文件时，至少得使用上述三个常量中的一个。以下常量是选用的：

O\_APPEND每次写操作都写入文件的末尾

O\_CREAT如果指定文件不存在，则创建这个文件

O\_EXCL如果要创建的文件已存在，则返回-1，并且修改errno的值

O\_TRUNC如果文件存在，并且以只写/读写方式打开，则清空文件全部内容(即将其长度截短为0)

O\_NOCTTY如果路径名指向终端设备，不要把这个设备用作控制终端。

O\_NONBLOCK如果路径名指向FIFO/块文件/字符文件，则把文件的打开和后继

（2）函数原型及说明：

void \*malloc(long NumBytes)：该函数分配了NumBytes个字节，并返回了指向这块内存的指针。如果分配失败，则返回一个空指针（NULL）。

关于分配失败的原因，应该有多种，比如说空间不足就是一种。

（3）void free(void \*FirstByte)： 该函数是将之前用malloc分配的空间还给程序或者是操作系统，也就是释放了这块内存，让它重新得到自由。

（4）static DBHASH \_db\_hash(DB \*db, const char \*key)

类比数组，我们知道数组由下标可取得对应值，而散列表根据key生成的hash值，hash值可以简单理解为数组中的下标。通过key生成hash值函数成为hash函数。本例中，hash值的算法如下：

while ( c = str [i] )

{

Hash\_val +=(int)c\*i;

i++;

}

hash%hash\_table\_len;

最后一步取模是hash算法中常用的方法，可以将hash值限制在散列表范围内。但不同的key可能产生同一hash值，这种现象称为hash冲突。而本例的散列链（链地址法）就是解决hash冲突的一种方法。

采用这种方法就必须考虑如何能使数据在散列表均匀分布（充分利用散列表的长）。如果某些链过长，而某些链过短甚至没有数据，设置的散列表大小也就没有意义了。但由于数据来源key是随机不稳定的，这种情况不可能完全消除，经测试发现散列表的长度如果为素数，可以使数据最大限度的均匀分布，这个结果也是和%运算有关的。

本例中散列表实际大小为137，因为插入数据时，根据取模运算，hash值为0-137。

但使用时候因为要跳过第0条链表，所以+1

(4) 头文件：#include <sys/types.h> #include <unistd.h>

定义函数：off\_t lseek(int fildes, off\_t offset, int whence);

函数说明：

每一个已打开的文件都有一个读写位置, 当打开文件时通常其读写位置是指向文件开头,

若是以附加的方式打开文件(如O\_APPEND), 则读写位置会指向文件尾.

当read()或write()时, 读写位置会随之增加,lseek()便是用来控制该文件的读写位置.

参数fildes 为已打开的文件描述词, 参数offset 为根据参数whence来移动读写位置的位移数.

参数 whence 为下列其中一种:

SEEK\_SET 参数offset 即为新的读写位置.

SEEK\_CUR 以目前的读写位置往后增加offset 个位移量.

SEEK\_END 将读写位置指向文件尾后再增加offset 个位移量. 当whence 值为SEEK\_CUR 或

SEEK\_END 时, 参数offet 允许负值的出现.

下列是较特别的使用方式:

1) 欲将读写位置移到文件开头时:lseek(int fildes, 0, SEEK\_SET);

2) 欲将读写位置移到文件尾时:lseek(int fildes, 0, SEEK\_END);

3) 想要取得目前文件位置时:lseek(int fildes, 0, SEEK\_CUR);

返回值：当调用成功时则返回目前的读写位置, 也就是距离文件开头多少个字节.

若有错误则返回-1, errno 会存放错误代码

（5）1 .struct iovec定义了一个向量元素。通常，这个结构用作一个多元素的数组。对于每一个传输的元素，

指针成员iov\_base指向一个缓冲区，这个缓冲区是存放的是readv所接收的数据或是writev将要发送的数据。

成员iov\_len在各种情况下分别确定了接收的最大长度以及实际写入的长度。

struct iovec {

// Starting address (内存起始地址）

void \*iov\_base;

//Number of bytes to transfer（这块内存长度）

size\_t iov\_len;

};

ssize\_t readv(int fd, const struct iovec \*iov, int iovcnt);

ssize\_t writev(int fd, const struct iovec \*iov, int iovcnt);

这两个函数需要三个参数：

要在其上进行读或是写的文件描述符fd

读或写所用的I/O向量(vector)

要使用的向量元素个数(count)

这些函数的返回值是readv所读取的字节数或是writev所写入的字节数。如果有错误发生，就会返回-1，而errno存有错误代码。

注意，也与其他I/O函数类似，可以返回错误码EINTR来表明他被一个信号所中断。

（6）C 库函数 char \*strchr(const char \*str, int c) 在参数 str 所指向的字符串中搜索第一次出现字符 c（一个无符号字符）的位置。

(7) 文件：#include <stdio.h>

sprintf()函数用于将格式化的数据写入字符串，其原型为：

int sprintf(char \*str, char \* format [, argument, ...]);

【参数】str为要写入的字符串；format为格式化字符串，与printf()函数相同；argument为变量。

除了前两个参数类型固定外，后面可以接任意多个参数。而它的精华，显然就在第二个参数--格式化字符串--上。

printf()和sprintf()都使用格式化字符串来指定串的格式，在格式串内部使用一些以“%”开头的格式说明符

（format specifications）来占据一个位置，在后边的变参列表中提供相应的变量，

最终函数就会用相应位置的变量来替代那个说明符，产生一个调用者想要的字符串。