SQLite源码学习(38) 对表的一些处理_sqlite3源码分析 rootpage-CSDN博客

blog.csdn.net/pfysw/article/details/108012409

SQLite 专栏收录该内容
43 篇文章 48 订阅

订阅专栏

1.删除的记录在表的第2页，但是进入balance()函数后，为什么pCur->iPage是0？

sqlite3BtreeFirst—>moveToRoot里会把pCur->iPage设为0，**pCur->iPage不是数据库文件的页面，而是pCur->apPage的索引**。

2.cursor是什么时候创建的

在allocateCursor里，中有OpenRead或OpenWrite时才创建。

3.allocateCursor和sqlite3BtreeCursor有什么区别

前者只是分配空间，后者是初始化的一些补充

4. insert里打印的变量，pCur->pgnoRoot和pPage->pgno的区别

```
pPage = pCur->pPage;
TRACE(("INSERT: table=%d nkey=%lld ndata=%d page=%d %s\n",
        pCur->pgnoRoot, pX->nKey, pX->nData, pPage->pgno,
        loc==0 ? "overwrite" : "new entry"));
```

- 1
- 2
- 3
- 4

pCur->pgnoRoot是当前表根节点所在页面，pPage->pgno是当前要插入的页面序号，虚拟机在执行NewRowid命令时会调用
`sqlite3BtreeLast()------->moveToRoot()--------->getAndInitPage()`
来给pCur->pPage赋值，其中ppPage就是pCur->pPage的地址

```
rc = sqlite3PagerGet(pBt->pPager, pgno, (DbPage**)&pDbPage, bReadOnly);
if( rc ){
  goto getAndInitPage_error1;
}
*ppPage = (MemPage*)sqlite3PagerGetExtra(pDbPage);
if( (*ppPage)->isInit==0 ){
  btreePageFromDbPage(pDbPage, pgno, pBt);
  rc = btreeInitPage(*ppPage);
  if( rc!=SQLITE_OK ){
    goto getAndInitPage_error2;
  }
}
```

一个表里可能有很多页，pCur根据pCur->apPage记录的路径来跳转到相应的也，类似的代码如

```
if( pCur ){
  pCur->iPage--;
  pCur->pPage = pCur->apPage[pCur->iPage];
}
```

5.什么时候创建sqlite_master表

在newDatabase的zeroPage，写完数据库的头100个自己后，就会跟着建立一张sqlite_master表，这张表的数据更新也和普通表一样，在insertCell函数里

```
data = pPage->aData;
assert( &data[pPage->cellOffset]==pPage->aCellIdx );
rc = allocateSpace(pPage, sz, &idx);
if( rc ){ *pRC = rc; return; }
/* The allocateSpace() routine guarantees the following properties
** if it returns successfully */
assert( idx >= 0 );
assert( idx >= pPage->cellOffset+2*pPage->nCell+2 || CORRUPT_DB );
assert( idx+sz <= (int)pPage->pBt->usableSize );
pPage->nFree -= (u16)(2 + sz);
if( iChild ){
  /* In a corrupt database where an entry in the cell index section of
  ** a btree page has a value of 3 or less, the pCell value might point
  ** as many as 4 bytes in front of the start of the aData buffer for
  ** the source page.  Make sure this does not cause problems by not
  ** reading the first 4 bytes */
  memcpy(&data[idx+4], pCell+4, sz-4);
  put4byte(&data[idx], iChild);
}else{
  memcpy(&data[idx], pCell, sz);
}
pIns = pPage->aCellIdx + i*2;//更新cell索引数组
memmove(pIns+2, pIns, 2*(pPage->nCell - i));
put2byte(pIns, idx);
pPage->nCell++;
/* increment the cell count */
//更新这一页的cell数量
if( (++data[pPage->hdrOffset+4])==0 ) data[pPage->hdrOffset+3]++;
```

allocateSpace里更新第一个cell的偏移地址

```
top -= nByte;
put2byte(&data[hdr+5], top);
assert( top+nByte <= (int)pPage->pBt->usableSize );
*pIdx = top;
```

那么pPage->aCellIdx和pPage->cellOffset是哪里来的呢，在zeroPage函数里，会被newDatabase()和btreeCreateTable（）函数调用

```
data[hdr] = (char)flags;
first = hdr + ((flags&PTF_LEAF)==0 ? 12 : 8);
memset(&data[hdr+1], 0, 4);
data[hdr+7] = 0;
put2byte(&data[hdr+5], pBt->usableSize);
pPage->nFree = (u16)(pBt->usableSize - first);
decodeFlags(pPage, flags);
pPage->cellOffset = first;
pPage->aDataEnd = &data[pBt->usableSize];
pPage->aCellIdx = &data[first];
```

5.为什么insert之后却没有写入到数据库文件

相关测试代码如下

```
do_execsql_test btree01-1.1 {
  PRAGMA page_size=1024;
  PRAGMA vdbe_trace = 1;
  CREATE TABLE t1(a INTEGER PRIMARY KEY, b BLOB);
  WITH RECURSIVE
     c(i) AS (VALUES(1) UNION ALL SELECT i+1 FROM c WHERE i<1)
  INSERT INTO t1(a,b) SELECT i, zeroblob(6500) FROM c;
  UPDATE t1 SET b=zeroblob(3000);
  UPDATE t1 SET b=zeroblob(64000) WHERE a=2;
  PRAGMA integrity_check;
} {ok}
```

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11

看打印日志，c(i) AS (VALUES(1) UNION ALL SELECT i+1 FROM c WHERE i<1)对应的应该是

```
INSERT: table=1 nkey=1 ndata=2 page=1 new entry
```

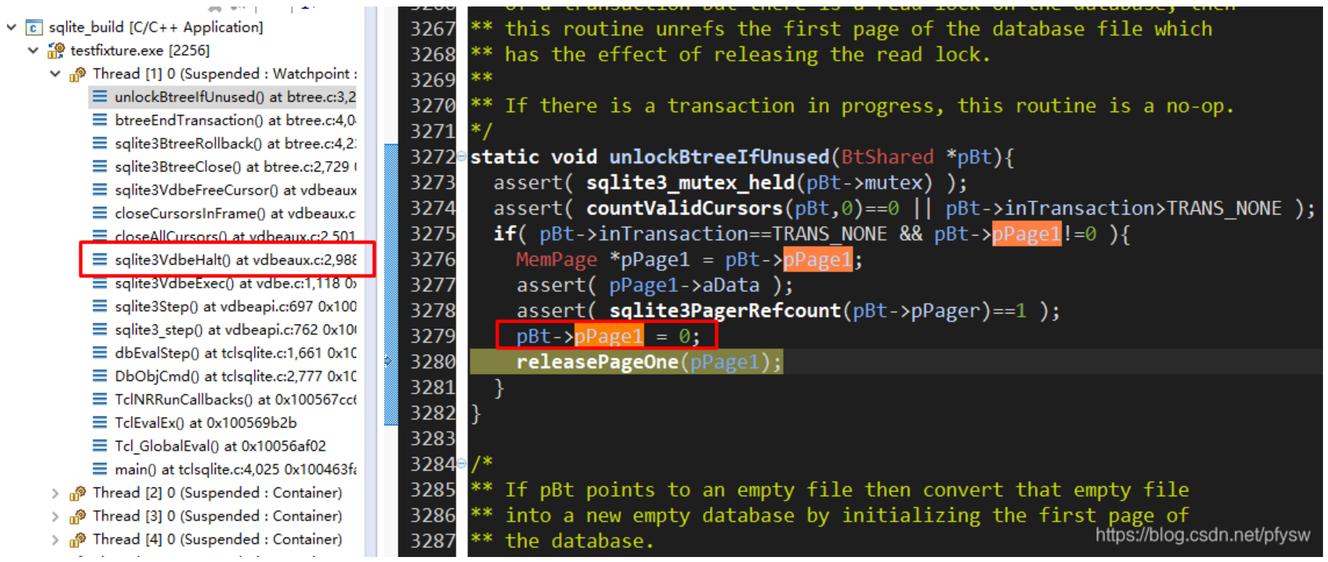
1

大概会在第1页的1020字节处写入数据，之所以没有写入到数据库的原因是，因为这个是临时的，可以看字节码后面会执行到Delete指令，还没存到文件之前就把这个cell给删了，即把这些数据清0，调用关系
sqlite3BtreeDelete()----->dropCell()------>freeSpace()
但是我们知道第一页在之前已经向文件里写入了sqlite_master的数据了，而现在第一页在内存里的内容（pBt->pPage1）是清零的，为什么最后在pager层写文件时

```
rc = sqlite3OsWrite(pPager->fd, pData, pPager->pageSize, offset);
```

1

这里的pData是原来文件里的数据，而不是清0后的数据

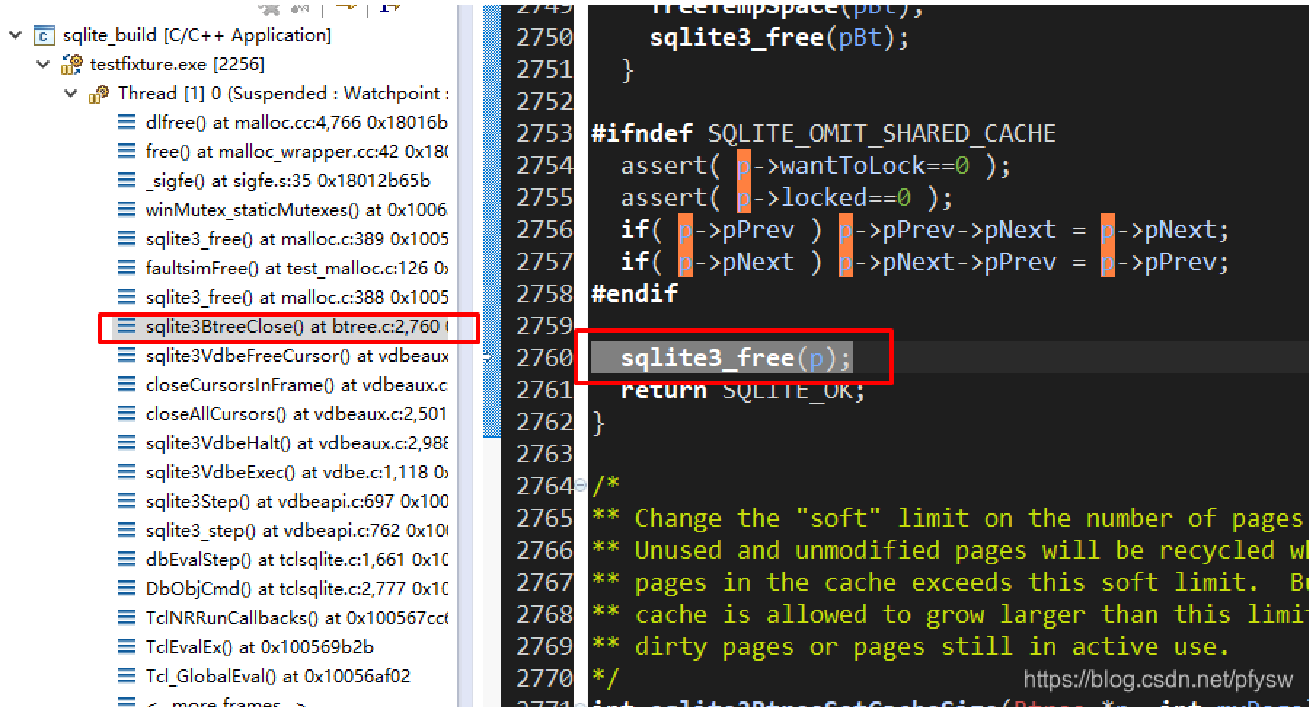


```
sqlite_build [C/C++ Application]
  testfixture.exe [2256]
    Thread [1] 0 (Suspended : Watchpoint :
      unlockBtreeIfUnused() at btree.c:3,2
      btreeEndTransaction() at btree.c:4,0
      sqlite3BtreeRollback() at btree.c:4,2
      sqlite3BtreeClose() at btree.c:2,729 (
      sqlite3VdbeFreeCursor() at vdbeaux
      closeCursorsInFrame() at vdbeaux.c
      closeAllCursors() at vdbeaux.c:2,501
      sqlite3VdbeHalt() at vdbeaux.c:2,98
      sqlite3VdbeExec() at vdbe.c:1,118 0x
      sqlite3Step() at vdbeapi.c:697 0x100
      sqlite3_step() at vdbeapi.c:762 0x10
      dbEvalStep() at tclsqlite.c:1,661 0x1C
      DbObjCmd() at tclsqlite.c:2,777 0x1C
      TclNRRunCallbacks() at 0x100567cc
      TclEvalEx() at 0x100569b2b
      Tcl_GlobalEval() at 0x10056af02
      main() at tclsqlite.c:4,025 0x100463f
    Thread [2] 0 (Suspended : Container)
    Thread [3] 0 (Suspended : Container)
    Thread [4] 0 (Suspended : Container)
```

```c
** this routine unrefs the first page of the database file which
** has the effect of releasing the read lock.
**
** If there is a transaction in progress, this routine is a no-op.
*/
static void unlockBtreeIfUnused(BtShared *pBt){
  assert( sqlite3_mutex_held(pBt->mutex) );
  assert( countValidCursors(pBt,0)==0 || pBt->inTransaction>TRANS_NONE );
  if( pBt->inTransaction==TRANS_NONE && pBt->pPage1!=0 ){
    MemPage *pPage1 = pBt->pPage1;
    assert( pPage1->aData );
    assert( sqlite3PagerRefcount(pBt->pPager)==1 );
    pBt->pPage1 = 0;
    releasePageOne(pPage1);
  }
}

/*
** If pBt points to an empty file then convert that empty file
** into a new empty database by initializing the first page of
** the database.
```

https://blog.csdn.net/pfysw

其实看上面这个图片就可以知道pBt->pPage1 被设为了0，而且p->pBt对象页被释放了



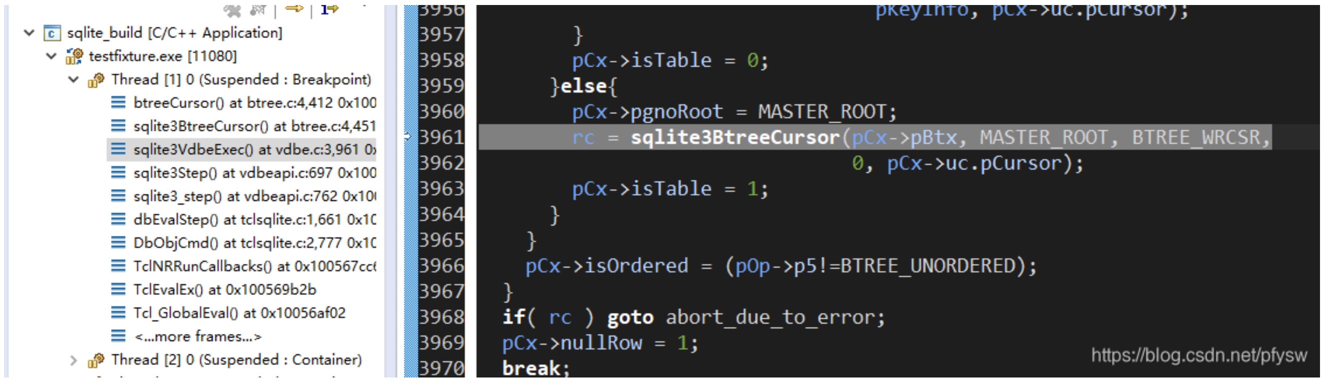
```
sqlite_build [C/C++ Application]
  testfixture.exe [2256]
    Thread [1] 0 (Suspended : Watchpoint :
      dlfree() at malloc.cc:4,766 0x18016b
      free() at malloc_wrapper.cc:42 0x18
      _sigfe() at sigfe.s:35 0x18012b65b
      winMutex_staticMutexes() at 0x1006
      sqlite3_free() at malloc.c:389 0x1005
      faultsimFree() at test_malloc.c:126 0x
      sqlite3_free() at malloc.c:388 0x1005
      sqlite3BtreeClose() at btree.c:2,760
      sqlite3VdbeFreeCursor() at vdbeaux
      closeCursorsInFrame() at vdbeaux.c
      closeAllCursors() at vdbeaux.c:2,501
      sqlite3VdbeHalt() at vdbeaux.c:2,98
      sqlite3VdbeExec() at vdbe.c:1,118 0x
      sqlite3Step() at vdbeapi.c:697 0x100
      sqlite3_step() at vdbeapi.c:762 0x10
      dbEvalStep() at tclsqlite.c:1,661 0x1C
      DbObjCmd() at tclsqlite.c:2,777 0x1C
      TclNRRunCallbacks() at 0x100567cc
      TclEvalEx() at 0x100569b2b
      Tcl_GlobalEval() at 0x10056af02
      <...more frames...>
```

```c
    freeTempSpace(pBt);
    sqlite3_free(pBt);
  }

#ifndef SQLITE_OMIT_SHARED_CACHE
  assert( p->wantToLock==0 );
  assert( p->locked==0 );
  if( p->pPrev ) p->pPrev->pNext = p->pNext;
  if( p->pNext ) p->pNext->pPrev = p->pPrev;
#endif

  sqlite3_free(p);
  return SQLITE_OK;
}

/*
** Change the "soft" limit on the number of pages
** Unused and unmodified pages will be recycled w
** pages in the cache exceeds this soft limit.  B
** cache is allowed to grow larger than this limi
** dirty pages or pages still in active use.
*/
```

https://blog.csdn.net/pfysw

甚至在commit时，btree对象也被换掉了

我们再回过头来看当时insert时的btree对象是哪来的


```
sqlite_build [C/C++ Application]
  testfixture.exe [11080]
    Thread [1] 0 (Suspended : Breakpoint)
      btreeCursor() at btree.c:4,412 0x100
      sqlite3BtreeCursor() at btree.c:4,451
      sqlite3VdbeExec() at vdbe.c:3,961 0x
      sqlite3Step() at vdbeapi.c:697 0x100
      sqlite3_step() at vdbeapi.c:762 0x10
      dbEvalStep() at tclsqlite.c:1,661 0x10
      DbObjCmd() at tclsqlite.c:2,777 0x10
      TclNRRunCallbacks() at 0x100567cc
      TclEvalEx() at 0x100569b2b
      Tcl_GlobalEval() at 0x10056af02
      <...more frames...>
    Thread [2] 0 (Suspended : Container)
```

```
                                    pKeyInfo, pCx->uc.pCursor);
      }
      pCx->isTable = 0;
    }else{
      pCx->pgnoRoot = MASTER_ROOT;
      rc = sqlite3BtreeCursor(pCx->pBtx, MASTER_ROOT, BTREE_WRCSR,
                              0, pCx->uc.pCursor);
      pCx->isTable = 1;
    }
  }
  pCx->isOrdered = (pOp->p5!=BTREE_UNORDERED);
}
if( rc ) goto abort_due_to_error;
pCx->nullRow = 1;
break;
```
https://blog.csdn.net/pfysw

pCx->pBtx好像是一个临时对象

```
struct VdbeCursor {
 。。。。。。
  Btree *pBtx;            /* Separate file holding temporary table */
```

- 1
- 2
- 3

所以这2个根本就不是同一个btree对象

```
Btree *pBt = db->aDb[i].pBt;//真实数据库文件对应的表
VdbeCursor *pCx;
pCx->pBtx//临时表，只在内存中存在
```

- 1
- 2
- 3