HSQLDB存储课后作业

贾元昊 软件51 2015013193

1

缓存的替换机制是怎样的？缓存的容量是如何维护的？

解答：

对于Cached Table来说，最开始首先会在DataFileCache中调用get方法根据指定key在cache（Cache）中寻找数据，当寻找不到的时候会调用getFromFile方法（参数与get一样）从文件中读取。getFromFile方法最开始同样会根据指定key在cache中寻找数据。发现找不到时，会调用readObject将test.data中的数据读取到rowIn（RowInputInterface）中，再使用store（RowAVLDisk）的get方法将rowIn中的数据变为object（CachedObject）（这里有一个两次的循环，原因不明）。找到后会将object用put方法保存到cache中。对于Text Table来说，最开始会使用TextCache作为缓存，存储方式差不多。（addInit函数功能成迷）

默认Cache大小是10000KB（here），实际占用的内存大小会比这个要大1~3倍。显然10000KB的缓存填数据太麻烦了，让我们把默认的缓存改小一点才能触发替换机制。顺便借此机会看一下缓存容量是怎么变的。

```
SET FILES CACHE SIZE 100; -- 100, 100KB, is the minimum value
SHUTDOWN;
```

重新启动server，追踪到上面这条命令的执行，有如下语句：

```
/*org.hsqldb.persist.Logger.setCacheSize*/
public void setCacheSize(int value) {
    propCacheMaxSize = value * 1024L;
}
```

这里我们看到logger中的名为propCacheMaxSize的变量用来维护cache的最大容量。这个在cache里会对应为bytesCapacity。

最大容量确定之后需要确定当前容量，追踪得到如下语句：

```
/*org.hsqldb.persist.Cache.put*/
void put(CachedObject row) {

    int storageSize = row.getStorageSize();

    if (preparePut(storageSize)) {
        putNoCheck(row);
    } else {
        long value = size() + reserveCount >= capacity ? capacity
                                                        : bytesCapacity
                                                          / 1024L;
        throw Error.error(ErrorCode.DATA_CACHE_IS_FULL,
                          String.valueOf(value));
    }
}
private void putNoCheck(CachedObject row) {
```

```
17
18      if (accessCount > ACCESS_MAX) {
19          updateAccessCounts();
20          resetAccessCount();
21          updateObjectAccessCounts();
22      }
23
24      Object existing = addOrRemoveObject(row, row.getPos(), false);
25
26      if (existing != null) {
27          dataFileCache.logSevereEvent("existing object in Cache.put() "
28                                       + row.getPos() + " "
29                                       + row.getStorageSize(), null);
30      }
31
32      row.setInMemory(true);
33
34      cacheBytesLength += row.getStorageSize();
35  }
```

可以看到缓存的实际容量储存在cacheByteLength变量中，每次插入一个row的时候，都会把row对应的size加到上面。

然后研究一下这个preparePut函数就可以看到hsqldb是如何在cache满的情况下清理cache的。

```
1   /*org.hsqldb.persist.Cache.preparePut*/
2   boolean preparePut(int storageSize) {
3
4       boolean exceedsCount = size() + reserveCount >= capacity;
5       boolean exceedsSize  = storageSize + cacheBytesLength > bytesCapacity;
6
7       if (exceedsCount || exceedsSize) {
8           cleanUp(false);
9
10          exceedsCount = size() + reserveCount >= capacity;
11          exceedsSize  = storageSize + cacheBytesLength > bytesCapacity;
12
13          if (exceedsCount || exceedsSize) {
14              clearUnchanged();
15          } else {
16              return true;
17          }
18
19          exceedsCount = size() + reserveCount >= capacity;
20          exceedsSize  = storageSize + cacheBytesLength > bytesCapacity;
21
22          if (exceedsCount || exceedsSize) {
23              cleanUp(true);
24          } else {
25              return true;
26          }
27
28          exceedsCount = size() + reserveCount >= capacity;
29          exceedsSize  = storageSize + cacheBytesLength > bytesCapacity;
30
31          if (exceedsCount) {
32              dataFileCache.logInfoEvent(

33                  "dataFileCache CACHE ROWS limit reached");
```

```
        }

        if (exceedsSize) {
            dataFileCache.logInfoEvent(
                "dataFileCache CACHE SIZE limit reached");
        }

        if (exceedsCount || exceedsSize) {
            return false;
        }
    }

    return true;
}
```

我们可以很显然的看到这里的清理过程分为了三步，三步之后依然exceed的话就会返回false，在后续报错误：

```
cleanUp(false);
cleanUnchanged();
cleanUp(true);
```

其中cleanUp函数会遍历每一条记录的访问记数，然后会把访问记数不够多的记录删掉，参数为true表示全部删除，参数为false表示删除一半；cleanUnchanged函数会把内存里没有变化过的记录删除。这两种删除方式都是一种预测，相当于将最不可能会访问到的行删掉。至此也就实现了缓存交换机制。

2

数据是怎样实现内外存交换的？在什么时候进行？

解答：

Cached Table和Text Table都涉及到了内存和外存的同时使用。所以针对这两个表进行研究即可。

初始化：

对于Cached Table来说，读入缓存发生在`SET TABLE <table_name> INDEX '... ... ...'`的时候。此时会调用DataFileCache中的getFromFile方法。具体内容第一问中有详细描述。

对于Text Table来说，读入缓存发生在`SET TABLE <table_name> SOURCE '...'`的时候。此时会使用setDataSource（org.hsqldb.TextTable，下同）方法来设置源文件。该函数会调用openCache方法来创建并打开缓存。openCache中会调用connect方法将缓存连接到文件上。所有Text Tables与其cache的对应关系都会储存在org.hsqldb.TextTableStorageManager.textCacheList中。connect方法中的如下代码实现了从文件中读取数据到cache中：

```
private void connect(Session session, boolean withReadOnlyData) {
    ...
    cache =
        (TextCache) database.logger.textTableManager
            .openTextFilePersistence(this, securePath, readOnly,
                                     isReversed);

    store.setCache(cache);

    reader = cache.getTextFileReader();

```

```
    // read and insert all the rows from the source file

    if (cache.isIgnoreFirstLine()) {
        reader.readHeaderLine();
        cache.setHeaderInitialise(reader.getHeaderLine());
    }

    readDataIntoTable(session, store, reader);
    ...
}

private void readDataIntoTable(Session session, PersistentStore store,
                               TextFileReader reader) {
    while (true) {
        RowInputInterface rowIn = reader.readObject();

        if (rowIn == null) {
            break;
        }

        Row row = (Row) store.get(rowIn);

        if (row == null) {
            break;
        }

        Object[] data = row.getData();

        systemUpdateIdentityValue(data);
        enforceRowConstraints(session, data);
        store.indexRow(session, row);
    }
}
```

查询时：

这里两种缓存表的操作与第一题讲的差不多，基本上就是如果cache命中，则返回结果，计数+1；反之从文件中读取，如果缓存空间不够则会出现缓存替换的情况。

数据保存：

数据库在checkpoint或者shutdown的时候或cache里有modified数据即将被清理的时候会将数据进行保存。

对于Text Table来说，TextTableStorageManager会调用closeAllTextCaches方法来讲所有注册在其中的Text Table所使用的缓存保存并关闭。具体则是会调用org.hsqldb.persist.Cache的saveAll函数，其实现如下：

```
 /**
  * Writes out all modified cached Rows.
  */
void saveAll() {

    int savecount = 0;

    objectIterator.reset();

    for (; objectIterator.hasNext(); ) {
        if (savecount == rowTable.length) {
            saveRows(savecount);
```

```
13
14              savecount = 0;
15          }
16
17          CachedObject r = (CachedObject) objectIterator.next();
18
19          if (r.hasChanged()) {
20              rowTable[savecount] = r;
21
22              savecount++;
23          }
24      }
25
26      saveRows(savecount);
27  }
28
29  private synchronized void saveRows(int count) {
30
31      if (count == 0) {
32          return;
33      }
34
35      rowComparator.setType(CachedObjectComparator.COMPARE_POSITION);
36      ArraySort.sort(rowTable, 0, count, rowComparator);
37      dataFileCache.saveRows(rowTable, 0, count);
38
39      saveRowCount += count;
40  }
```

可以看到保存策略基本上就是判断每一个在缓存中的row是否被更改过，如果被更改过的话就将该列保存在rowTable中，再使用saveRows方法将其保存到文件中；否则不处理该row。最终会调用DataFileCache.dataFile.write来将其写到文件中，具体调用层级比较深就不详细写了。

对于Cached Table来说，则是会在DataFileCache的close方法中调用reset方法，在reset方法中调用cache.saveAll方法，具体形式与上面完全一样，不再赘述。