BadgerDB项目的目标是为您提供有关RDBMS关键组件的实践教育。 在此项目中，您需要在提供的存储管理器之上实现一个布尔管理器

1.1 BadgerDB I / O层

BadgerDB数据库系统的最底层是I / O层。该层允许系统高层创建/销毁文件，分配/取消分配文件内的页面以及读取/写入文件的页面。该层由两个类组成：文件（文件类）和页面（页面类）类。这些类使用C ++异常来处理任何意外事件的发生。向您提供了File类，Page类和异常类的实现。要启动此项目，您可以将压缩文件夹BufMgr.zip下载到您的私有工作区，并使用以下命令解压缩该文件夹：unzip BufMgr.zip

对该代码进行了充分的注释，以帮助您了解其功能和作用。请使用Doxygen如下所示为您的代码生成文档。在bufmgr目录中，运行以下命令以生成文档文件。

>make doc

请注意，>是Linux机器上的shell提示，不是命令的一部分。文档文件将在docs目录中生成。现在，您可以在浏览器中打开docs / index.html文件，并通过对类及其方法的描述来更好地理解其实现。

1.2 BadgerDB Bu ﬀ er管理器

数据库缓冲区池是一组固定大小的内存缓冲区的数组，称为帧，用于保存已从磁盘读取到内存中的数据库页面（也称为磁盘块）。页面是磁盘和驻留在主内存中的缓冲区池之间的传输单位。大多数现代DBMS使用的页面大小至少为8,192字节。另一个需要注意的重要事项是，内存中的数据库页面是首次读取时磁盘上相应页面的精确副本。一旦将页面从磁盘读取到缓冲区池中，DBMS软件就可以更新存储在磁盘上的信息。页面，导致缓冲区池中的副本与磁盘上的副本有所不同。这些页面被称为“脏”。

由于磁盘上的数据库本身通常大于缓冲区池可用的主内存量，因此在任何给定时间，只有一部分数据库页面可以装入内存。缓冲区管理器用于控制哪些页面驻留在内存中。每当购买者管理器收到对数据页的请求时，购买者管理器都会检查以查看所请求的页面是否已经在构成购买者池的帧之一中。如果是这样，则购买管理器仅返回指向页面的指针。如果不是，则缓冲区管理器释放一个帧（如果页面脏了，则可以通过将其包含的页面写入磁盘），然后从磁盘将请求的页面读入已释放的帧中。

在进一步阅读之前，您应该首先阅读描述BadgerDB I / O层的文档，以便您了解其功能（在上一节中进行了介绍）。简而言之，I / O层通过打开/关闭文件以及读取/写入文件页面的方法，为Unix文件提供了一个面向对象的接口。现在，您需要知道的关键是打开文件（通过输入字符串名称）会返回File类型的对象。此类具有读取和写入File页面的方法。您将使用这些方法在磁盘和缓冲区池之间移动页面。

1.2.1布尔替换策略和时钟算法

有许多方法可以确定需要空框架时要替换的页面。操作系统中常用的策略是FIFO，MRU和LRU。尽管LRU是最常用的策略之一，但是它具有高开销，并且不是在数据库系统中发生的许多常见情况下使用的最佳策略。取而代之的是，许多系统使用近似LRU行为的时钟算法，并且速度更快。

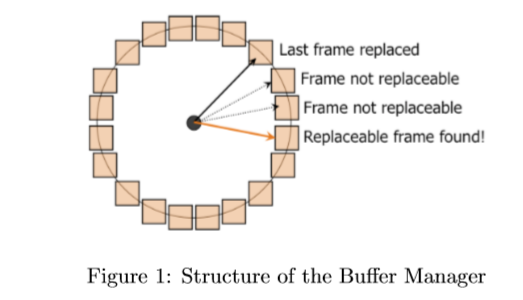
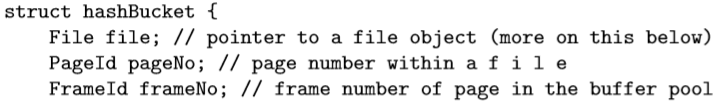


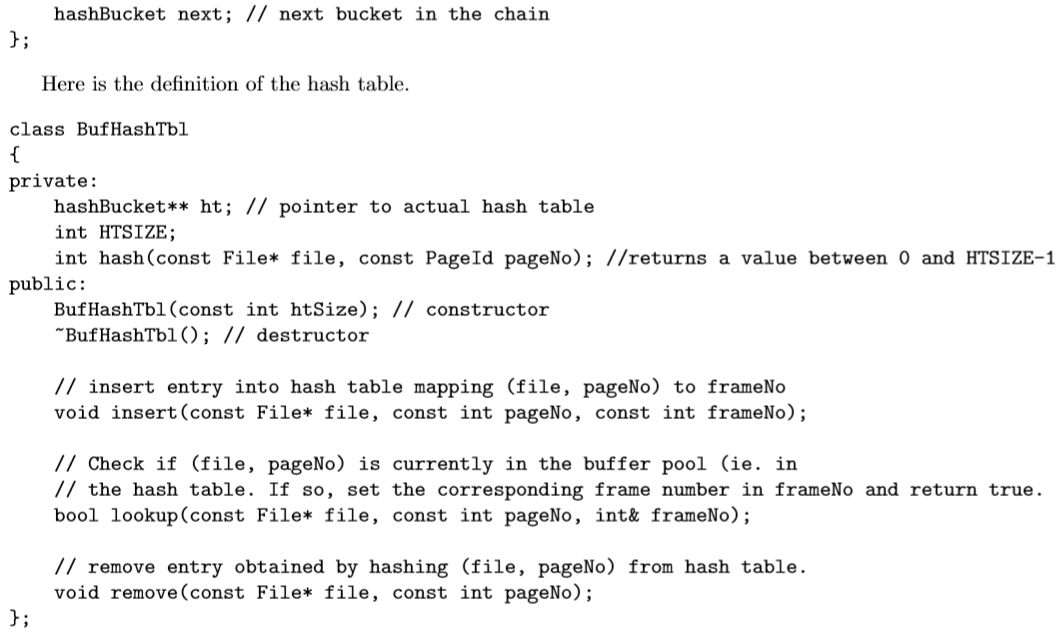
图1显示了缓冲池的概念布局。在图1中，每个方框对应于缓冲池中的一个框架。假设缓冲区池包含numBufs帧，编号为0到numBufs-1。从概念上讲，布尔池中的所有框架都以循环列表的形式排列。与每个帧相关的位称为refbit。每次访问缓冲池中的页面（通过对缓冲管理器的readPage（）调用）时，相应帧的refbit都设置为true。在任何时间点，时钟指针（其值在0到numBufs − 1之间的整数）都会以顺时针方向前进（使用模块化算术，以使其不会超过numBufs − 1）。对于时针经过的每一帧，都会检查refbit，然后将其清除。如果设置了该位，则相应的帧已被“最近”引用，并且不会被替换。另一方面，如果refbit为false，则选择要替换的页面（假设它没有固定-固定页面将在下面讨论）。如果选定的缓冲区框架变脏（即已被修改），则当前占据该框架的页面将写回到磁盘。否则，将仅清除框架，并从该位置读取磁盘中的新页面。图2说明了时钟算法的执行。时钟算法的详细信息如下。

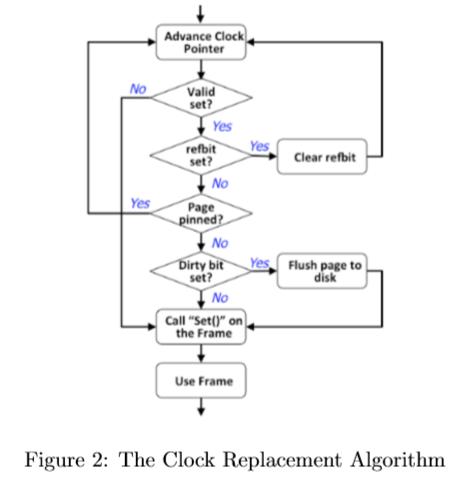
1.2.2买方经理的结构

BadgerDB缓冲区管理器使用三个C ++类：BufMgr，BufDesc和BufHashTbl。 BufMgr类只有一个实例。该类的关键组成部分是实际的缓冲区池，该缓冲区池由numBufs帧数组组成，每个帧的大小与数据库页面的大小相同。除了此数组之外，BufMgr实例还包含BufDesc类的numBufs实例的数组，该数组用于描述缓冲区池中每个帧的状态。哈希表用于跟踪缓冲区池中当前驻留的页面。该哈希表由BufHashTbl类的实例实现。该实例是BufMgr类的私有数据成员。这些类将在下面详细描述。

BufHashTbl类。 BufHashTbl类用于将文件和页码映射到缓冲池帧，并使用链式桶式哈希实现。我们提供了此类的实现供您使用。



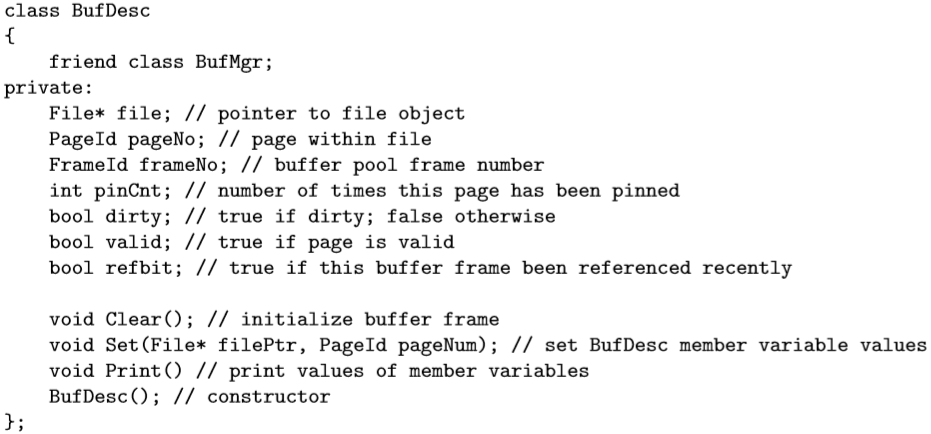




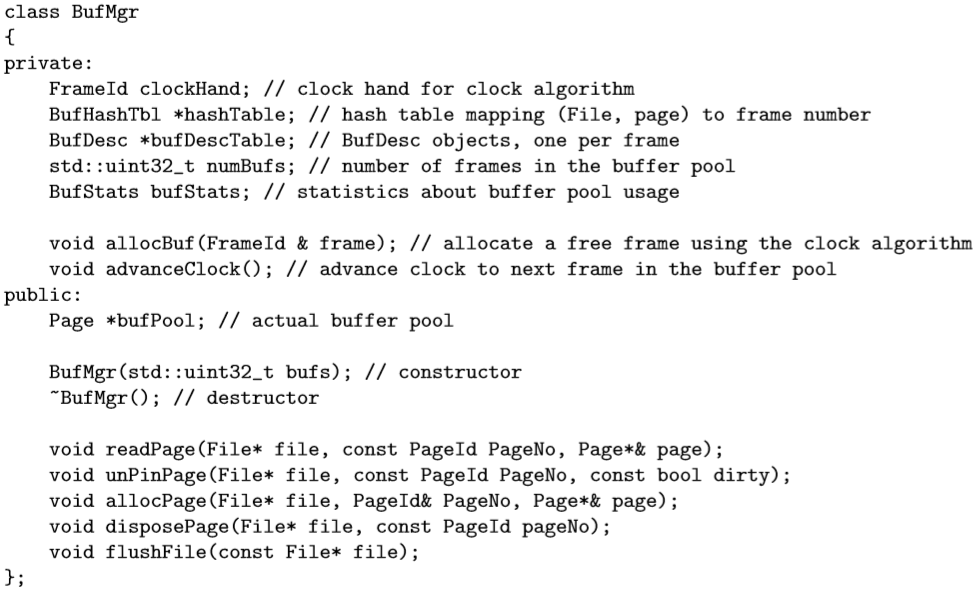
BufDesc类别。 BufDesc类用于跟踪缓冲区池中每个帧的状态。 定义如下。

首先请注意，BufDesc类的所有属性都是私有的，并且BufMgr类被定义为朋友。 尽管这看起来有些奇怪，但是这种方法将对BufDesc私有变量的访问限制为仅对BufMgr类。 另一种选择（将所有内容公开）使访问权限过于宽泛。

BufDesc类的大多数属性的目的应该很明显。 脏位（如果为true）表示该页面是脏的（即已被更新），因此必须在该帧用于容纳另一页之前将其写入磁盘。 pinCnt指示页面已被固定多少次。 时钟算法使用refbit。 有效位用于指示框架是否包含有效页面。 无需在此类中实现任何方法。 但是，如果您愿意，可以随意以任何方式进行扩充。



BufMgr类。 BufMgr类是买方经理的核心。 在这里您应该为该项目编写新代码。



此类的定义如下：

•BufMgr（const int bufs）这是类的构造函数。 为缓冲区池分配带有bufs页面框架和对应的BufDesc表的数组。 分配缓冲池时，所有帧的设置方式将处于清除状态。 哈希表也将以空状态开始。 我们提供了构造函数。

•〜BufMgr（）刷新所有脏页并取消分配缓冲区池和BufDesc表。

•void advanceClock（）将时钟提前到缓冲区池中的下一帧。

•void allocBuf（FrameId＆frame）使用时钟算法分配一个空闲帧； 如有必要，将脏页写回磁盘。 如果固定了所有缓冲区帧，则抛出BufferExceededException。 该私有方法将由下面描述的readPage（）和allocPage（）方法调用。 确保如果分配的缓冲区框架中具有有效页，请从哈希表中删除相应的条目。

•void readPage（File \* file，const PageId PageNo，Page \*＆page）首先通过调用lookup（）方法检查页面是否已在缓冲区池中，当页面不在缓冲区池中时，该方法可能会抛出HashNotFoundException。哈希表以获取帧号。根据lookup（）调用的结果，有两种情况要处理：

–情况1：页面不在缓冲区池中。调用allocBuf（）分配缓冲区框架，然后调用方法file-> readPage（）将页面从磁盘读取到缓冲区池框架中。接下来，将页面插入哈希表。最后，在框架上调用Set（）以正确设置它。 Set（）将使页面的pinCnt设置为1。通过page参数返回一个指向包含页面的框架的指针。

–情况2：页面位于缓冲区池中。在这种情况下，设置适当的refbit，增加页面的pinCnt，然后通过page参数将指针返回到包含页面的框架。

•void unPinPage（File \* file，const PageId PageNo，const booldirty）递减包含（file，PageNo）的帧的pinCnt，如果dirty == true，则设置dirty位。如果引脚数已经为0，则抛出PAGENOTPINNED。如果在哈希表查找中未找到页面，则不执行任何操作。

•void allocPage（File \* file，PageId＆PageNo，Page \*＆page）此方法的第一步是通过调用file-> allocatePage（）方法在指定的文件中分配一个空页。此方法将返回新分配的页面。然后调用allocBuf（）获得缓冲区池帧。接下来，将一项插入到哈希表中，并在框架上调用Set（）以正确设置它。该方法既通过pageNo参数将新分配的页面的页号返回给调用方，又通过page参数将指针分配给为该页面分配的缓冲区帧。

•void disposePage（File \* file，const PageId pageNo）此方法从文件中删除特定页面。从文件中删除页面之前，请确保如果要在缓冲区池中为要删除的页面分配了一个帧，则该帧将被释放，并且相应地也删除哈希表中的条目。

•void flushFile（File \* file）应该在bufTable中扫描属于该文件的页面。对于遇到的每个页面，它应该：（a）如果页面是脏的，则调用file-> writePage（）将页面刷新到磁盘，然后将该页面的脏位设置为false，（b）从哈希表中删除该页面。 （无论页面是干净还是脏的），并且（c）为页面框架调用BufDesc的Clear（）方法。如果文件的某个页面被固定，则抛出PagePinnedException。如果遇到属于该文件的无效页面，则引发BadBufferException。

2入门

解压缩BufMgr.zip时，将有一个名为bufmgr的目录。在此目录中，您将找到以下文件：

•Makefile：一个make文件。您可以通过在外壳上键入make来创建项目。

•main.cpp：驱动程序文件。显示如何使用File和Page类。还包含供买方经理使用的简单测试用例。您必须使用更严格的测试套件来扩展这些测试。

•buffer.h：采购经理的类定义

•buffer.cpp：方法的最基本实现。在此处提供您的实际实现。

•bufHash.h：缓冲区池哈希表类的类定义。不要换。

•bufHash.cpp：缓冲区池哈希表类的实现。不要换。

•file.h：File类的类定义。您不应更改该文件。

•file.cpp：File类的实现。您不应更改该文件。

•文件iterator.h：文件中页面迭代器的实现。不要换。

•page.h：页面类的类定义。不要换。

•page.cpp：页面类的实现。不要换。

•page iterator.h：页面中记录的迭代器的实现。

•exceptions目录：所有异常类的实现。如果需要，请随时在此处添加更多文件。

3编码和测试

我们已经定义了这个项目，以便您可以了解和利用C ++进行面向对象编程的全部好处。您的编码风格应通过定义明确的类和干净的接口来继续进行。建议不要恢复为C（低级过程式）编程风格，这将受到惩罚。该代码应使用Doxygen样式注释进行详细记录。每个文件都应以您的姓名和学生证开头，并应说明文件的用途。每个函数之前应加几行注释，以描述函数并解释输入和输出参数以及返回值。

4上交

您需要在一个压缩文件夹中提交所有必要的材料（使用GZip或WinZip）。您的文件夹应命名为“ Proj2 <StudentName> <StudentID>”，并且仅包含源代码文件（不包含二进制文件）。我们将编译您的客户经理实施，将其与我们的测试驱动程序链接，然后运行测试。由于我们应该能够使用任何有效的驱动程序来测试您的代码，因此务必忠实于此处指定的接口的确切定义。如果您更改这些接口并且您的代码无法编译，您将受到处罚。

5物流

BadgerDB用C ++编码，并在Linux机器上运行。以下是一些物流要点：

•警告：C ++的优势之一是它可以编译时间代码检查（因此减少了运行时错误）。尝试通过打开尽可能多的编译器警告来利用此优势。我们将提供的Makefile文件将默认启用-Wall。

•辅助工具：始终在寻找可以简化您工作的工具。示例：make用于编译和构建项目，makedepend用于自动生成依赖关系，perl或python或bash用于编写测试脚本，valgrind用于跟踪内存错误，gdb用于调试，以及git用于版本控制。

•软件工程：诸如此类的大型项目需要大量的软件设计。在开始编写任何代码之前，请花一些时间思考您的总体方法。