1. 用户模块：

用户信息存在名为user的非聚集索引表里。用户注册就添加一条用户信息。登陆和查用户就从文件中读取，改用户数据和改权限就改写文件

1. 列车管理模块：

有两个堆表：存放站点时间票价（下称堆表1），存放剩余票数（下称堆表2）。

列车信息放在一个非聚集索引表里。Key: trainid, Value:{列车名称，类别，有几个站点，有几种票，RID1，RID2，票种}。

为了便于查票，我们还需要一个聚集索引表，key:{站点，id}，value：{站点编号，类别，日期修正量，到达时间，发车时间}，因为这样可以让同一个站点的记录连在一起。

添加车次也就是向列车信息索引表里插入，向堆表1插入，向查票索引表插入每个站点的信息

贩卖车次就是向堆表2插入初始车票情况。（确认是否贩卖就是看堆表2的rid是否为-1）

修改车次和删除车次就很简单了，可以类推，在这里不赘诉了。

1. 查票模块：

直接查询：

找出经过loc1的车次序列和经过loc2的车次序列（范围查找），因为trainid是以字典序排列的，两个扫一遍就可以找到相同的trainid且找到的也是按字典序排列的。然后根据这些trainid去列车信息索引表和2个堆表里找需要的信息

中转查询：

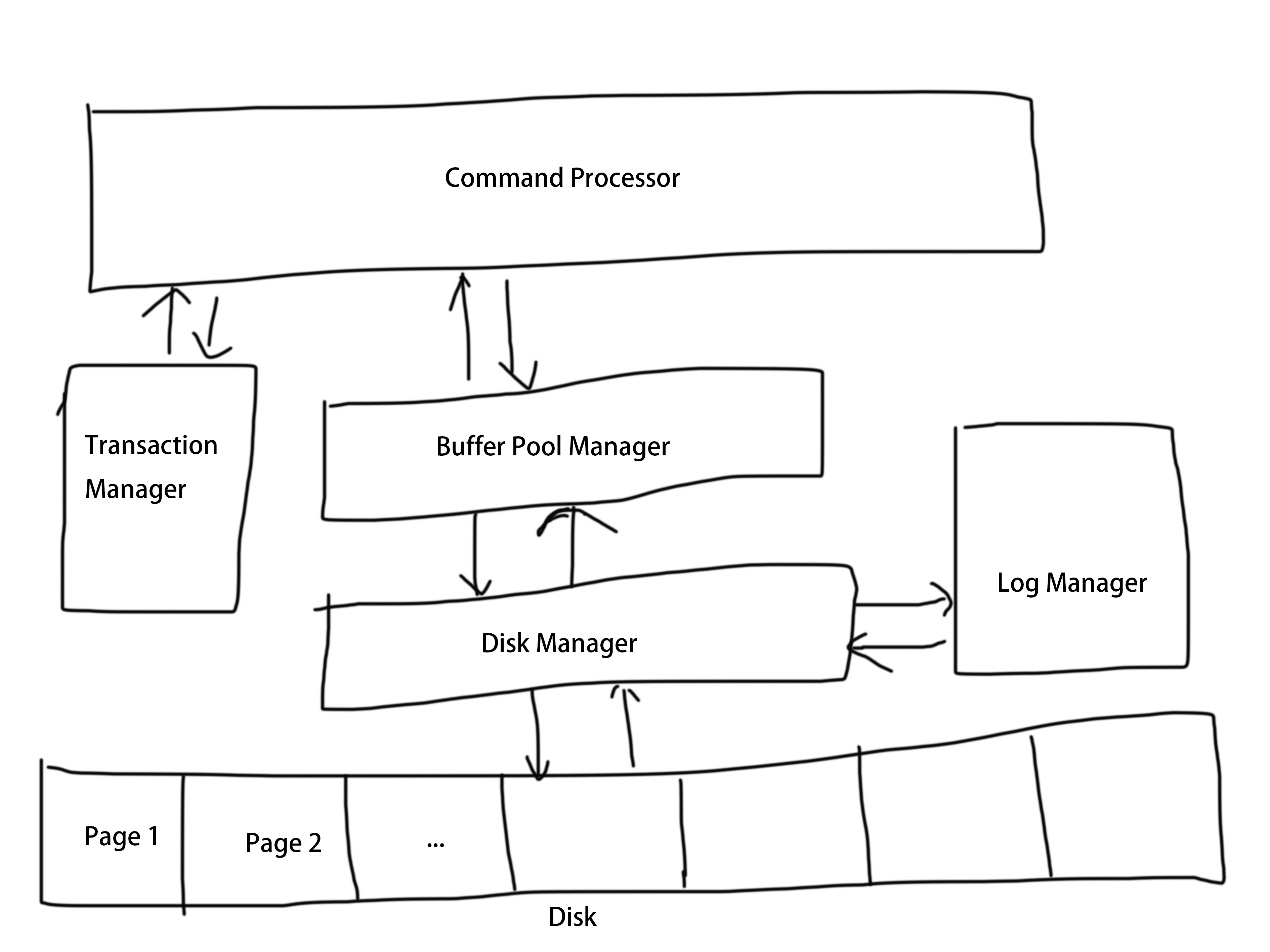
找出经过loc1的车次序列和经过loc2的车次序列，然后找到loc1可以到达的站点，可以到达loc2的站点，找到他们重合的一些站点，然后找出其中用时最短的路线。

购票模块

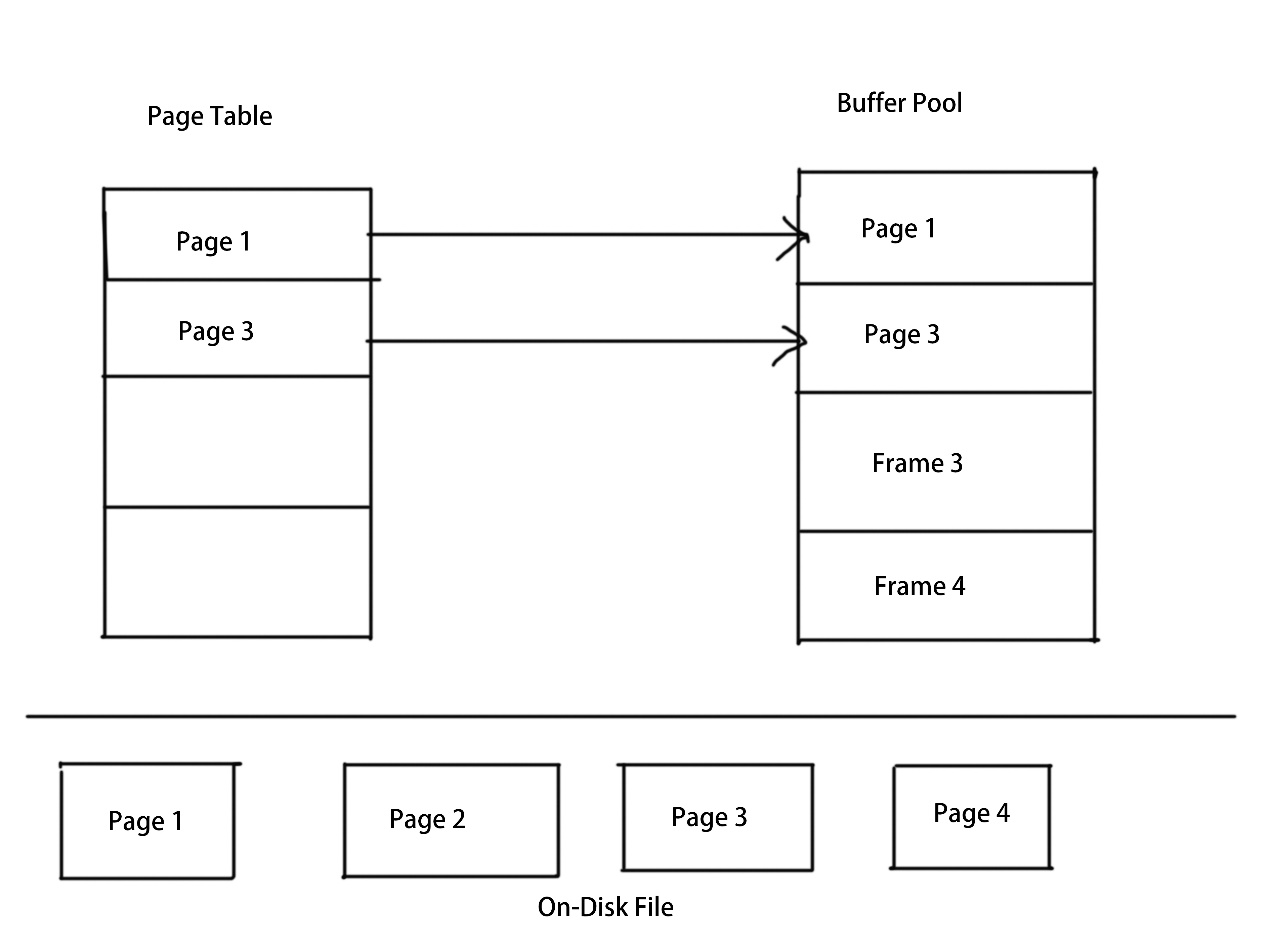
购票信息储存在聚集索引表里key:{用户id，该列车发车日期，类别，trainid，站点1，站点2}，value：{每个票种购买数量，出发日期，结束日期，出发时间，到达时间，票价，票种，站点1编号，站点2编号}，定义大小就是对key中的每个元素从前到后分别比大小，相当于广义的字典序

买票和退票就是修改列车的剩票记录，对索引表进行插入操作，查票操作就是对索引表范围查找

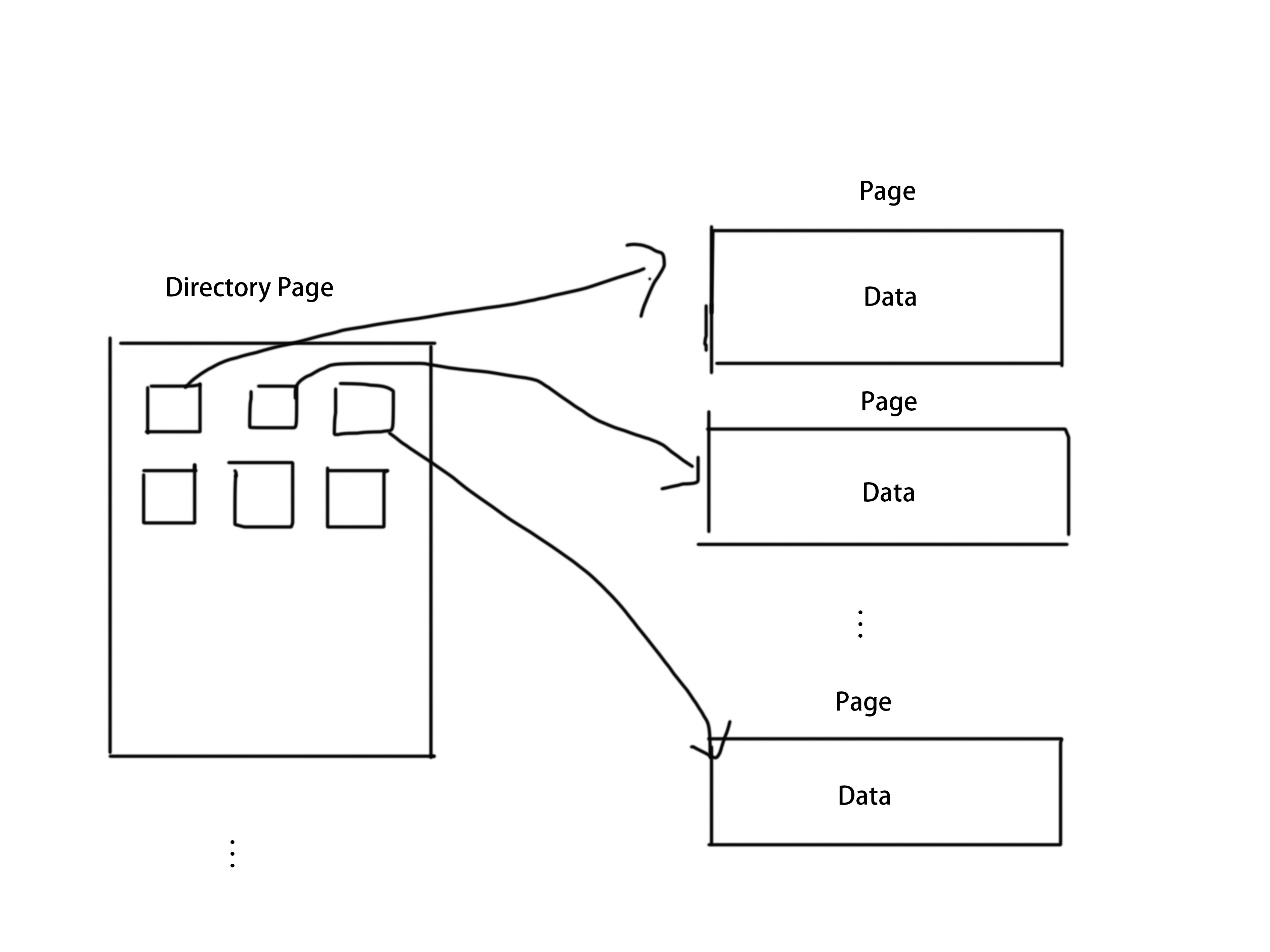
底层存储结构：



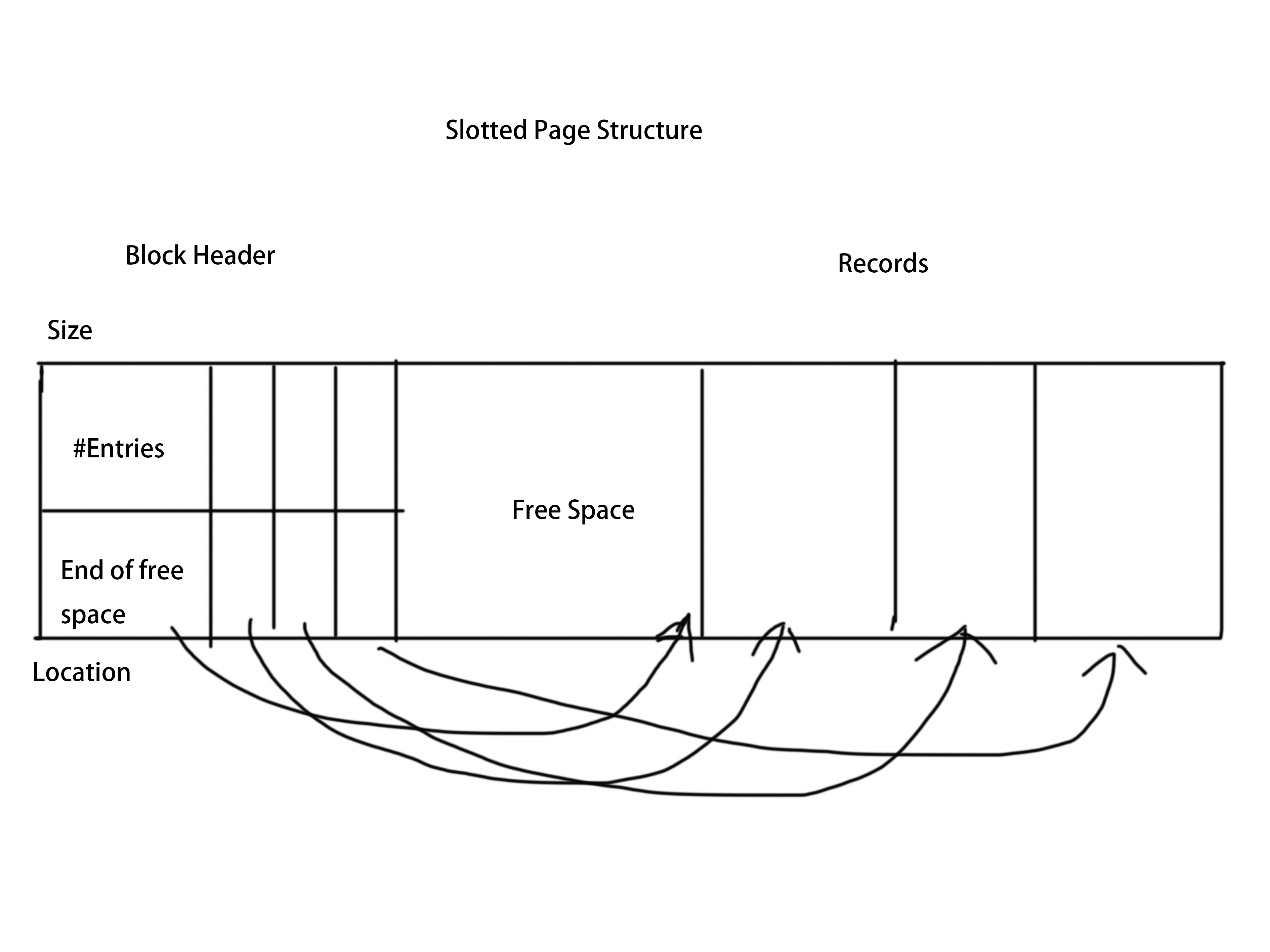
Buffer pool manager结构：



下图是堆表的结构，用了目录页来管理每个页的使用情况。



每一堆表页中存储结构，其他页的访问都是通过RID间接访问：



ARIES的实现：

1.log的实现：log一共有二十多个类别，分别记录了对各种页的插入、删除、更新操作，Transaction的Begin、End、Commit、Abort标识符。一个操作序列（Transaction）以Begin开头，以End结尾。End前若有Commit标记，说明本次Transaction生效。若End前没有Commit标记或有Abort标记，说明本次Transaction被废弃，需要回滚本次Transaction。有一个特殊的log类别，叫Checkpoint。Checkpoint之前的所有log记录的操作都保证写入了磁盘。

2.文件恢复的实现：程序开始进行文件操作前，读取log文件中的最靠后的Checkpoint的位置，从Checkpoint位置后开始Analysis，然后开始逐个redo各条Checkpoint之后的log。redo完所有的log后，我们对没有Commit的Transaction里的所有log从后往前undo，恢复到程序崩溃前的状态。