Urna Semper
Instructor's Name
June 15, 2018

火车订票系统后端开发文档

由“好快”小组开发

功能实现清单：

用户部分：

1. 用户注册：register name password email phone
2. 用户登录：login id password
3. 查询用户信息：query_profile id
4. 修改用户信息：query_profile id name password email phone
5. 修改用户权限：modify_privilege id1 id2 privilege

车票相关：

1. 查询车票：loc1 loc2 date catalog
2. 带中转查询车票：query_transfer loc1 loc2 date catalog
3. 订购车票：buy_ticket id num train_id loc1 loc2 date ticket_kind
4. 查询购票信息：query_order id date catalog
5. 退订车票：refund_ticket id num train_id loc1 loc2 date ticket_kind

车次相关

1. 新建车次：

- 格式

`add_train *train_id* *name* *catalog* *num(station)* *num(price)* *(name(price) ) x num(price)*`

`*[name time(arriv) time(start) time(stopover) (price) x num(price) ] x num(station)*`

2. 公开车次：sale_train train_id
3. 查询车次：query_train train_id
4. 删除车次 delete_train train_id

5. 修改车次

```
modify_train *train_id* *name* *catalog* *num(station)* *num(price)* *(name(price) num(price)*
*[name time(arriv) time(start) time(stopover) (price) * num(price) (\n)] *num(stati
```

管理相关

1. 删库命令： clean
2. 关闭系统：exit

思路阐述：

用户部分：

1. 基本的查询和修改，实现思路简单直接。

车票相关：

1. 查询车票：

   1.1. 对每一种类型的列车单独处理，重复多次2-4步骤。

   1.2. 分别找到经过始末站的所有列车id，并对这两个集合结构取交集，就得到了所有经过始末站点的列车。

   1.3. 遍历列车集合，筛除始发站和终点站在时间顺序上与用户要求相矛盾的列车。

   1.4. 对每个符合要求的列车，查找在特定日期的剩余票数。

   1.5. 输出信息。

2. 带中转查询车票：

   2.1. 对每种类型的列车单独处理，重复多次以下步骤。

   2.2. 找到所有到达终点站的列车，对每辆列车，做以下操作：

   2.3. 遍历当前列车的当前站点之前的站点，对每个可能的换乘车站，做一下操作：

   2.4. 调用函数1，查找始发站到当前可能的换乘车站的车票，暂存下来。

   2.5. 输出车票信息。

3. 订购车票：

   3.1. 找到当前列车，做鲁棒性判断。

   3.2. 修改列车剩余票数。

   3.3. 插入或修改用户交易信息。

4. 查询购票信息： 查询用户所有购票信息。

5. 退订车票：修改用户交易信息。

车次相关：

1. 加入、查找、删除列车：

   1.1. 长度不固定的插入和修改，建议用B树而不是B+树来实现。

   1.2. 注意记录各个站点之间日期的偏移量。

2. 公开车次：将列车能被按地点查询。

管理相关：

1. 重置、关闭系统：删除文件，退出循环。注意强制关闭时信息的保存。

问题分解：

用户部分：

1. 所有被要求的功能都可以被转化为固定长度数据的单个点查询、点修改。

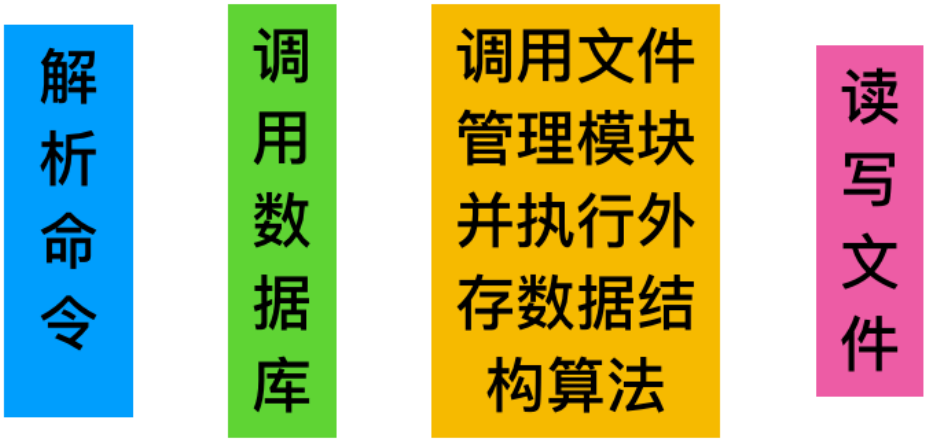
车票相关：

1. 将不定长度的存储内容转换为定长的，使用不足以唯一确定集合中元素的关键字进行区间查询，这里应当用索引顺序文件的数据结构B+树。

车次相关：

1. 将列车作为索引文件存储，做好不固定长度的单点插入、查询、删除。

系统架构：



解析命令部分：interpreter类

调用数据库部分：processor类

数据库健值对(key value)定义部分：station_kv ticket_kv train_id_kv, train_info_kv, user_kv, bill_kv.

B+树部分：BPlusTree类

B+树底层文件交互封装：BufferManager类

堆文件底层交互封装：PileManager类

B+树实现部分：

1. 用户信息B+树 “user_db”.
   - key: user_id (int);
   - val: privilege + name + password + email + phone
2. 列车信息堆文件“train_info_db”.
   - key: offset (int);
   - val: train_info
   -

3. 列车索引B+树“train_id_db”.

- key: train_id (char[]);
- val: pile file offset (int) + whether on sale (bool)

4. 列车经过哪些站点B+树 “station_db”.

- key: station name (char[]) + train_cat + train_id (char[]); 40+B
- val: train_id + location index (char) 20+B

5. 列车还剩多少票B+树“ticket_db”.

- key: train_id + date (head start) (int) + station index (char); 20+B
- val: (ticket sold) * 5 (short) 10B

6. 用户交易信息B+树“bill_db”.

- key: user_id + date + catalog. 40+B
- val: train_id + location index1/2 + (ticket bought) * 5 30+B

时间性能优化：

1. 将信息在不超过所需空间的范围内使用小数据类型存储，缩小了关键字大小。

2. 对于变长数据使用了范围查询的方式，将关键字组织为一个结构体，用类似字典序的方式重定义不等号，然后B+树提供给定上下界，返回元素顺序表的接口。就可以使用这样的方式处理后端中经常出现的变长数据问题。所有B+树的关键字大小都在30-40byte左右，一个4096byte的块可以装载约100-200个关键字。

3. 数据类型卡常。

4. 数据库组织上，在不能减小查询次数的情况下，避免重复存储信息

5. 一行神奇的编译命令。

空间性能优化：

1. 数据结构中B+树的实现完成了回收站的功能，它会记录下被删除的废弃文件位置，当有这些位置时优先用完它们，之后才会新增文件大小。

关于数据安全：

1. 在与前端配套的后端中，每次对文件进行读写时，都将文件缓冲区当中的数据flush到当前文件，以防信息不同步。

关于B+树实现中的优化：

1. 自定义了顺序存储容器vector，按块大小开数组，不会doublespace，较少浪费内存空间，查找范围较大时使用二分查找。

2. 自定义了缓存类BufferManager，封装底层文件操作，结构清楚。

3. 每个B+树在创建时内存中以数组的形式维护一个节点池，用来节约操作时新建内存节点的开销。

4. 在操作失败或者无需删改的情况下特判，不将节点数据写入外存。