





DBTrain Lab 1 Report

刘泓尊 2018011446 计84

CI job id: 83249

Status	Job	Pipeline
 passed	#83249  master  19fb78ba	#58876 by 

Record: 记录的序列化和反序列化

实现了 `FixedRecord` 的 `Load` 和 `Store` 接口。

注意到 `FixedRecord` 构造的时候已经提供了 `_iTypeVec` 和 `_iSizeVec` 的初始化，所以序列化/反序列化的时候只需要转换 `_iFields` 字段就可以了。

Store: 对 `_iFields` 中的 `Field*` 类型遍历，判断类型，通过 `dynamic_cast` 转换为具体的子类，调用 `Field` 的抽象方法 `GetData` 将数据写入到 `uint8_t*` 的 buffer 中。

Load: 根据 `_iTypeVec` 中的类型信息，构建对应的 `Field*` 子类，然后调用 `Field` 的抽象方法 `SetData` 将 `uint8_t*` 的 buffer 数据反序列化为 `Field*` 类型。

Page: 页面的整体组织

实现了 `LinkPage` 的 `PushBack` 和 `PopBack` 接口，以提供基于双向链表的页管理。

PushBack: 通过 `LinkPage::GetPageID()` 获得页编号。如果本节点的下一个节点为空，直接创建 `LinkPage` 对象，添加在后面即可；如果下一个节点非空，则需要修改下一个节点的前驱和后继，总共需要修改4个指向。

PopBack: 如果下一个节点为空，直接返回即可。否则判断下一个节点是否有后续页面。如果有，需要修改下下个节点的前驱。之后修改本节点的后继为空或下下个节点即可。在最后调用

`MiniOS::DeletePage()` 在内存中删除下个页面。需要注意创建一个 `Page` 对象要及时析构，以完成对页面的实际写入。

实现了 `RecordPage` 的 `InsertRecord`, `GetRecord`, `DeleteRecord`, `UpdateRecord` 接口，实现了单个页面内 `Record` 的增删改查。

InsertRecord: 寻找空的槽位插入，只需要遍历 `[0, _nCap)` 区间内的槽编号是否被占用(利用 `HasRecord(SlotID)` 接口)。找到空槽位之后，调用 `Page::SetData()` 接口在 `BITMAP_OFFSET + BITMAP_SIZE + nSlotID * _nFixed` 的地方插入 `_nFixed` 大小的数据即可。

GetRecord: 对于未被占用的槽，抛出 `RecordPageException` 异常。创建长度为 `_nFixed` 的 `uint8_t*` 的 buffer，利用 `Page::GetData()` 读出序列化后的数据即可。

DeleteRecord: 对于未被占用的槽，抛出 `RecordPageException` 异常。直接将 `Bitmap` 中 `nSlotID` 位置0，同时 `_bModified = true` 以完成修改。不需要修改 `Slot` 内部数据。

UpdateRecord: 对于已被占用的槽, 抛出 `RecordPageException` 异常。调用 `Page::SetData()` 写入数据。

Table: 增删改查记录的接口

实现了Table的`GetRecord`, `InsertRecord`, `DeleteRecord`, `UpdateRecord`, `SearchRecord`, `NextNotFull` 接口。

NextNotFull: 用于在页满的时候获取新的可用页面。我使用最先匹配算法以达到较高的效率。每次页满时, 从 `_nNotFull` 开始向后遍历到表尾, 遇到第一个空页面作为新的 `_nNotFull`; 每次页 `pageid` 中有槽被删除时, 都将 `_nNotFull` 置为 `min(pageid, _nNotFull)`, 以保证 `_nNotFull` 始终是未满页面编号的最小者。全满时, 创建新页面添加到链表尾部, 需要用到 `PushBack` 接口。最后析构掉 `Page*` 对象, 以完成对内存的写入。在随机数据下的均摊复杂度是 $O(1)$ 的。

InsertRecord: 创建 `_nFixed` 大小的buffer, 从传入的 `Record*` 对象序列化到buffer。之后创建 `RecordPage` 对象并通过 `InsertRecord` 接口向页面写入数据。注意, 如果页满, 需要调用 `NextNotFull`, 利用最先匹配来更新 `_nNotFull`。

GetRecord: 创建 `RecordPage` 对象并通过 `GetRecord` 接口, 从页面获得 `uint8_t*` 的无格式记录数据。然后通过 `EmptyRecord` 接口创建新 `Record` 对象, 调用 `Load` 方法完成反序列化。最后析构所有不返回的内容。

DeleteRecord: 创建页面管理对象 `RecordPage`, 调用 `DeleteRecord` 析构对应页面即可。注意将 `_nNotFull` 修改为 `min(_nNotFull, nPageID)`

UpdateRecord: 创建 `RecordPage` 对象并通过 `GetRecord` 接口, 从页面获得 `uint8_t*` 的无格式记录数据。然后通过 `EmptyRecord` 接口创建新 `FixedRecord` 对象, 调用 `Load` 方法完成反序列化。之后遍历 `iTrans` 数组, 通过 `GetPos` 和 `GetField` 获得更新位置和更新后数据, 通过 `FixedRecord::SetField` 更新结构体字段, 之后调用 `FixedRecord::Store()` 重新序列化, 通过 `RecordPage::UpdateRecord()` 更新页面数据即可。为了代码简洁, 我直接用了一次序列化和一次反序列化, 读出和写入了一些没有update的部分, 可能效率不是很高, 也可以只对改动的部分进行写入, 效率更高但是实现复杂。

SearchRecord: 遍历所有页面的所有槽。通过 `RecordPage::GetRecord` 接口获得无格式数据, 之后创建新 `FixedRecord` 对象完成反序列化, 通过 `Condition::Match` 方法判断是否满足, 加入结果向量即可。如果 `pCond` 为空, 则返回所有数据。

测试结果

```
[=====] Running 10 tests from 1 test suite.
[-----] Global test environment set-up.
[-----] 10 tests from Lab1
[ RUN      ] Lab1.LinkedListTest
[      OK  ] Lab1.LinkedListTest (4 ms)
[ RUN      ] Lab1.RandomSqlTest
[      OK  ] Lab1.RandomSqlTest (112 ms)
[ RUN      ] Lab1.RecordPageTest
[      OK  ] Lab1.RecordPageTest (1 ms)
[ RUN      ] Lab1.InsertSelectTest
[      OK  ] Lab1.InsertSelectTest (0 ms)
[ RUN      ] Lab1.UpdateTest
[      OK  ] Lab1.UpdateTest (1 ms)
[ RUN      ] Lab1.DeleteTest
[      OK  ] Lab1.DeleteTest (1 ms)
[ RUN      ] Lab1.UpdateDeleteTest
[      OK  ] Lab1.UpdateDeleteTest (1 ms)
[ RUN      ] Lab1.DeleteInsertTest
[      OK  ] Lab1.DeleteInsertTest (1 ms)
[ RUN      ] Lab1.StartTest
[      OK  ] Lab1.StartTest (0 ms)
[ RUN      ] Lab1.TableTest
[      OK  ] Lab1.TableTest (1 ms)
[-----] 10 tests from Lab1 (122 ms total)
[-----] Global test environment tear-down
[=====] 10 tests from 1 test suite ran. (122 ms total)
[ PASSED  ] 10 tests.
Passed lab1 test
```