MiniSQL阶段报告2

——Record Manager

> 第七小组

2.1 实验概述

在MiniSQL的设计中，Record Manager负责管理数据表中所有的记录，它能够支持记录的插入、删除与查找操作，并对外提供相应的接口。

2.2 记录与模式

为了能够持久化存储上面提到的Row、Field、Schema和Column对象，我们需要提供一种能够将这些对象序列化成字节流（char*）的方法，以写入数据页中。与之相对，为了能够从磁盘中恢复这些对象，我们同样需要能够提供一种反序列化的方法，从数据页的char*类型的字节流中反序列化出我们需要的对象。

实现

- Row::SerializeTo(*buf, *schema)
  - 依次将Row中field的个数，bitmap，以及各个field的内容序列化，写入buf指向的内容中，其中调用已经完成的field的SerializeTo函数。
- Row::DeserializeFrom(*buf, *schema)
  - 将buf指向的内容依次解释为field的个数，bitmap以及field的序列化数据。其中调用已经完成的field的 DeserializeFrom函数。
- Row::GetSerializedSize(*schema)
  - 返回此row序列化后的字节偏移量。
- Column::SerializeTo(*buf)
  - 依次将Column中的 COLUMN_MAGIC_NUM,name_.length(),name_,type_,len_,table_ind_,nullable_,unique_写入buf指向的内容。
- Column::DeserializeFrom(*buf, *&column, *heap)
  - 将buf指向的内容依次解释为 COLUMN_MAGIC_NUM,name_.length(),name_,type_,len_,table_ind_,nullable_,unique_。其中需要验证COLUMN_MAGIC_NUM。
- Column::GetSerializedSize()
  - 返回Column序列化的字节偏移量
- Schema::SerializeTo(*buf)
  - 依次将SCHEMA_MAGIC_NUM,columns_.size(),以及各个Column内容序列化，写入buf指向的内容，其中调用Column::SerializeTo(*buf)。
- Schema::DeserializeFrom(*buf, *&schema, *heap)
  - 依次将buf指向的内容解释为SCHEMA_MAGIC_NUM,columns_.size()以及各个column的序列化数据，其中调用Column::DeserializeFrom(*buf, *&column, *heap)且须验证 SCHEMA_MAGIC_NUM。
- Schema::GetSerializedSize()
  - 返回Schema序列化的字节偏移量

测试：

```
jsc@LAPTOP-J4D78JQ7:/mnt/d/ZJU_course/minisql_github/build$ ./test/tuple_test
I20220526 20:37:25.480017   179 main_test.cpp:10] This is an info log!
W20220526 20:37:25.480870   179 main_test.cpp:11] This is a warning log!
E20220526 20:37:25.481467   179 main_test.cpp:12] This is an error log!
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from TupleTest
[ RUN      ] TupleTest.FieldSerializeDeserializeTest
[       OK ] TupleTest.FieldSerializeDeserializeTest (0 ms)
[ RUN      ] TupleTest.RowTest
[       OK ] TupleTest.RowTest (0 ms)
[----------] 2 tests from TupleTest (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (2 ms total)
[  PASSED  ] 2 tests.
```

2.3 通过堆表管理记录

实现:

- `TableHeap:InsertTuple(&row, *txn)`
  - 插入一条新记录。检查此记录是否超过`TablePage::MaxTupleSize()`，若超过上限，打印错误信息。否则找寻第1个能够容纳此记录的页并插入。若所有已分配页均无法容纳，新申请一页并插入，将新页插入页链表。
- `TableHeap:UpdateTuple(&new_row, &rid, *txn)`
  - 更新`rid`处的记录。找到`rid`所在页，并调用`update()`。若更新失败，返回错误信息返回。若错误信息指出是新记录无法容纳，则在原页中删除此记录，再调用`TableHeap:InsertTuple(&row, *txn)`。
- `TableHeap:ApplyDelete(&rid, *txn)`
  - 找到此`rid`的位置，调用删除函数。
- `TableHeap:GetTuple(*row, *txn)`
  - 获取RowId为row->rid_的记录。
- `TableHeap:FreeHeap()`
  - 依次遍历页链表，删除所有页
- `TableHeap::Begin()`
  - 获取第一页的第一条记录，返回堆表的首迭代器。迭代器中记录`rid`信息，此记录所在页指针，和`tableheap*`。
- `TableHeap::End()`
  - 获取堆表的尾迭代器。`rid`中的成员均为无效成员，页指针为`nullptr`
- `TableIterator::operator++()`
  - 移动到下一条记录，通过++iter调用。检查此记录是否为此页最后一条记录，若不是则移动到下一条记录。若是，则找寻下一页直到遇到无效页或一个含有有效记录的的页。
- `TableIterator::operator++(int)`
  - 移动到下一条记录，通过iter++调用。调用`TableIterator::operator++()`实现

测试：

```
jsc@LAPTOP-J4D78JQ7:/mnt/d/ZJU_course/minisql_github/build$ ./test/table_heap_test
I20220526 20:38:27.078791   236 main_test.cpp:10] This is an info log!
W20220526 20:38:27.079851   236 main_test.cpp:11] This is a warning log!
E20220526 20:38:27.081494   236 main_test.cpp:12] This is an error log!
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from TableHeapTest
[ RUN      ] TableHeapTest.TableHeapSampleTest
[       OK ] TableHeapTest.TableHeapSampleTest (35 ms)
[----------] 1 test from TableHeapTest (37 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (41 ms total)
[  PASSED  ] 1 test.
jsc@LAPTOP-J4D78JQ7:/mnt/d/ZJU_course/minisql_github/build$
```