数据库专题训练 · Lab1

计01　容逸朗　2020010869

实验目的

1. 阅读代码，对于记录管理模块有一个结构性的理解
2. 设计底层记录页面组织，完成记录管理的各项基本功能

基础实验内容

1. TableMeta 的序列化和反序列化

- 首先将 `TableMeta` 中固定长度的私有成员序列化，对于长度不定的 `cols` 向量，我先保存（还原）了它的大小，然后再顺序遍历向量中的内容，并将其序列化。
- 注意到表元项内包含了字符串类型的列名，因此也需要先保存其长度，再按长度恢复数据即可。

2. Record 的序列化和反序列化

- 在基础实验中，我们只需实现定长数据存储，因此只要顺序遍历表格中列，并调用 `StoreField` 存储（或 `LoadField` 取出）对应字段数据，同时要注意维护与初始指针的偏移量大小，避免数据错位。
- 字段的序列化和反序列化则需要根据表项类型（int, float, string 等，可以由 tablemeta 中取得）分类，分别调用其 `Store` （或 `Load`）接口来恢复数据。

3. 页面内的记录插入、更新、删除

- **插入：** 首先通过位图在页面中找到第一个空槽，然后利用 `RecordFactory::StoreRecord` 方法将记录的内容序列化并记入页面中，再在位图中标记此槽位为已使用，最后把页面记为 dirty 即完成插入。
- **删除：** 考虑到我们存放的是定长数据，因此只需要把位图中的对应位置标记为未使用即可。
- **更新：** 只需要在对应的位置利用 `RecordFactory::StoreRecord` 写入新的内容复盖旧记录，完成操作后将页面记为 dirty 即完成更新操作。

4. 上层记录插入、更新、删除的接口函数

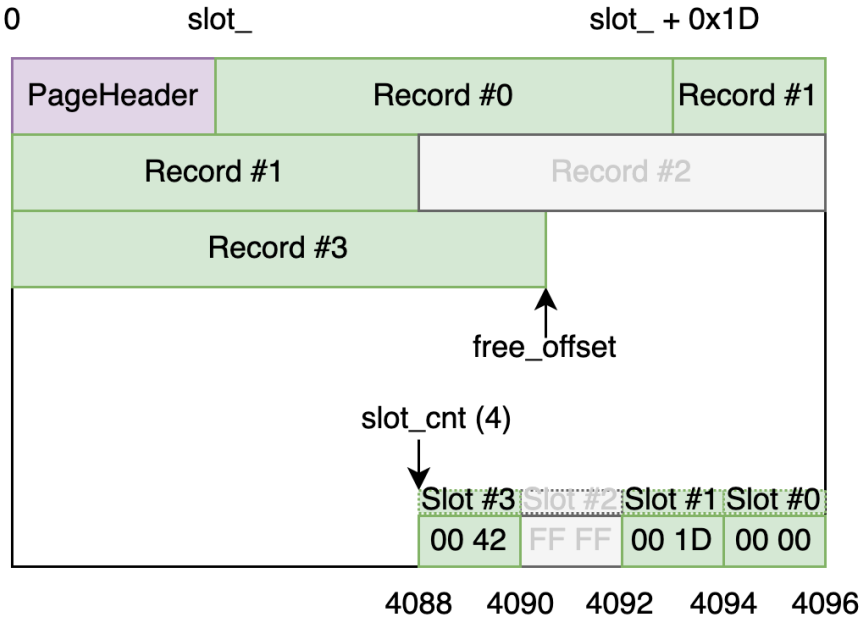
- **插入：** 首先判断当前的空闲页面 `first_free_` 是否为空页
  - 若为空，则调用 `CreatePage` 创建一个新的页面；
  - 否则调用 `GetPage` 得到对应的页面。
- 得到页面后可以利用 `InsertRecord` 方法插入记录，最后判断页面是否已满，若是则把空闲页面设为下一个有空位的页面，并标记 meta 为被修改状态便可。
- **删除：** 先找到对应页面，调用 `DeleteRecord` 删除记录，最后更改空闲页为当前页即可。
- **更新：** 先找到对应页面，再调用 `UpdateRecord` 更新记录即可。

高级功能

本次实验中，我还实现了变长数据（VARCHAR）的存储。

1. 设计方案

- **页内布局：**由于数据不是定长的，因此我删除了位图，改以偏移槽的方式维护页面空间。
  - 具体来说，每一条纪录会在页面中，从前往后依次存放记录。偏移槽则以由后往前的方式依次存放对应数据在页面中的偏移量，每个偏移槽的大小为 2 。（可存放 2^{16} 以内的偏移量，而页面大小为 4 KB，足够存下）
  - 由于不再使用位图维护页面空间，因此我在页首中加入了当前分配到的槽号 `slot_cnt`，页面剩余空间 `free_size` 和新记录存放位置偏移量 `free_offset`。



- **序列化与反序列化：**由于数据中含有可变长项，因此需要在变长的数据前插入一个两位的长度指示符，再把变长数据存入即可，恢复时首先根据表元项的类型判断，若为 varchar 则需要先读出长度，再读数据。
  - 注 1：VARCHAR 的长度上限为 65535，故使用 2B 大小储存长度是足够的。
  - 注 2：RecordFactory 的 `StoreField` 和 `StoreRecord` 需要改为有返回值的版本，其返回值分别为对应表项及记录所用之空间大小。
- **数据插入与删除：**
  - 数据插入时，首先计算数据要占用的空间大小（含变长数据者需额外计算长度项的空间），然后把得到的结果加上偏移槽所需的 2 位，然后利用此值判断页面的剩余空间是否足够存下，否则不断换页，直至为空（此时创造新页）或命中，则把数据存在对应页下便成。
  - 删除时，为实现简便，我们只需把对应槽位设置为 65535 (unsigned short max) 便认为对应槽位的数据已被删除。

    注：我们不考虑删除后的垃圾回收方案，数据对应存储位置作废。（如上图中 Record #2）

2. 改进效果

为测试改进效果，我设计了如下测例：

- 此测例中，共有两个表，其中一个储存 VARCHAR 类型，另一个为 CHAR 类型；
- 每个储存格的长度上限为 1000；
- 分别存入长为 5, 10, 20, 50, 100 的数据各十条，最后查看表格大小。

```
use dbtrain_test;

create table varchar_test (id int, name varchar(1000));

-- insert 10 rows with length 5
insert into varchar_test values (1, 'abcde'), (2, 'abcde'), ...

-- insert 10 rows with length 10
insert into varchar_test values (11, 'abcdefghij'), ...

-- insert 10 rows with length 20
insert into varchar_test values (21, 'abcdefghijklmnopqrst'),...

-- insert 10 rows with length 50
insert into varchar_test values (31, 'abcdefghijklmnopqrstuvwxy...'),...

-- insert 10 rows with length 100
insert into varchar_test values (41, 'abcdefghijklmnopqrstuvwxyz...'),...

-- create a table with char type to compare
create table char_cmp (id int, name char(1000));

-- 下面重复 5-18 行操作，放入 char_cmp 表内做对比
-- insert 10 rows with length 5
insert into char_cmp values (1, 'abcde'), (2, 'abcde'), ...

-- 此处略 --

-- insert 10 rows with length 100
insert into char_cmp values (41, 'abcdefghijklmnopqrstuvwxyz...'),...
```

实验结果如下：

- 使用基础版本得到的结果如下，可以看到两个表的大小都为 52 KB。

```
BoxWorld:dbtrain_test boxworld$ du -csh *
  4.0K    LOGDATA
  4.0K    LOGIDX
  4.0K    MASTER
   52K    char_cmp.data
  4.0K    char_cmp.meta
   52K    varchar_test.data
  4.0K    varchar_test.meta
```

- 为变长数据设计新的存储方式后，`varchar_test` 表的大小变为 4 KB：

```
BoxWorld:dbtrain_test boxworld$ du -csh varchar_test.*
  4.0K    varchar_test.data
  4.0K    varchar_test.meta
  8.0K    total
```

- 由此可以认为提高实验成功完成。

总结

- 高级功能 Commit ID：`16e37cf92e1a1117e8c1e7dcffefbd6b584f76eb`（位于 `ch1a` 分支上）
- 实验耗时如下：

文件	合计用时/hrs
table/table_meta.cpp	1.5
record/record_factory.cpp	1
table/page_handle.cpp	1
table/table.cpp	0.5
高级功能	6
总计	10