#4 CATALOG MANAGER

#4.1 实验概述

Catalog Manager 负责管理和维护数据库的所有模式信息，包括：

- 数据库中所有表的定义信息，包括表的名称、表中字段（列）数、主键、定义在该表上的索引。
- 表中每个字段的定义信息，包括字段类型、是否唯一等。
- 数据库中所有索引的定义，包括所属表、索引建立在那个字段上等。

这些模式信息在被创建、修改和删除后还应被持久化到数据库文件中。此外，Catalog Manager还需要为上层的执行器Executor提供公共接口以供执行器获取目录信息并生成执行计划。

#4.2 目录元信息

数据库中定义的表和索引在内存中以 `TableInfo` 和 `IndexInfo` 的形式表现，它们分别定义于 `src/include/catalog/table.h` 和 `src/include/catalog/indexes.h` ，其维护了与之对应的表或索引的元信息和操作对象。以 `IndexInfo` 为例，它包含了这个索引定义时的元信息 `meta_data_` ，该索引对应的表信息 `table_info_` ，该索引的模式信息 `key_schema_` 和索引操作对象 `index_` 。除元信息 `meta_data_` 外，其它的信息（如 `key_schema_` 、 `table_info_` 等）都是通过反序列化后的元信息生成的。也就是说，为了能够将所有表和索引的定义信息持久化到数据库文件并在重启时从数据库文件中恢复，我们需要为表和索引的元信息 `TableMetadata` 和 `IndexMetadata` 实现序列化和反序列化操作。它们与 `TableInfo` 和 `IndexInfo` 定义在相同文件中。在序列化时，为了简便处理，我们为每一个表和索引都分配一个单独的数据页用于存储序列化数据。因此，在这样的设计下，我们同样需要一个数据页和数据对象 `CatalogMeta` （定义在 `src/include/catalog/catalog.h` ）来记录和管理这些表和索引的元信息被存储在哪个数据页中。 `CatalogMeta` 的信息将会被序列化到数据库文件的第 `CATALOG_META_PAGE_ID` 号数据页中（逻辑意义上）， `CATALOG_META_PAGE_ID` 默认值为0。

因此，在本节中，你需要实现与Catalog相关的元信息的序列化和反序列化操作：

- `CatalogMeta::SerializeTo(*buf)`
- `CatalogMeta::GetSerializedSize()`
- `CatalogMeta::DeserializeFrom(*buf, *heap)`
- `IndexMetadata::SerializeTo(*buf)`
- `IndexMetadata::GetSerializedSize()`
- `IndexMetadata::DeserializeFrom(*buf, *&index_meta, *heap)`
- `TableMetadata::SerializeTo(*buf)`
- `TableMetadata::GetSerializedSize()`
- `TableMetadata::DeserializeFrom(*buf, *&table_meta, *heap)`
- `IndexInfo::Init(*index_meta_data, *table_info, *buffer_pool_manager)` : 传入事先创建好的 `IndexMetadata` 和从 `CatalogManager` 中获取到的 `TableInfo` ，创建索引本身的 `key_schema_` 和 `Index` 对象。这里的 `key_schema_` 可以通过 `Schema::ShallowCopySchema` 来创建，且 `key_schema_` 中包含的列与 `TableSchema` 中的列共享同一份存储。

提示：

- 与之前 `RecordManager` 中的序列化和反序列化类似，你需要通过魔数 `MAGIC_NUM` 来确保序列化和反序列化的正确性。
- 在 `CatalogManager` 、 `TableInfo` 、 `IndexInfo` 中都通过各自的 `heap_` 维护和管理与自身相关的内存分配和回收。如 `CatalogManager` 中，所有创建的 `CatalogMeta` 、 `TableInfo` 和 `IndexInfo` 对象都通过 `CatalogManager` 自身的 `heap_` 分配空间，随着 `CatalogManager` 对象的析构，这些对象自然而然也会被释放，而不需要我们手动去管理。同理， `TableInfo` 中所维护的 `TableSchema` 以及 `TableHeap` ，以及该表反序列化时创建的 `TableMetadata` 则需要通过 `TableInfo` 自身的 `heap_` 来分配空间。以下是反序列化一个表并生成 `TableInfo` 对象的例子：

C++ | 复制代码

```cpp
dberr_t CatalogManager::DoSomething() {
    /* table_info is created by CatalogManager using CatalogManager's heap_ */
    auto *table_info = TableInfo::Create(heap_);
    TableMetadata *table_meta = nullptr;
    TableMetadata::DeserializeFrom(/* some args */,
                                   table_info->GetMemHeap());
    if (table_meta != nullptr) {
        auto *table_heap = TableHeap::Create((/* some args */,
                                             table_info->GetMemHeap());
    }
    /* thus table_meta and table_heap are created by table_info */
    table_info->Init(table_meta, table_heap);
}
```

Bonus: 在序列化和反序列化时，通过校验算法（如 `CRC32` ）来对写入的数据内容（也可以是整个数据块）进行校验以确保数据正确性，或是采取其它方式来确保数据的正确性可以获得一定的加分。

#4.3 表和索引的管理

在实现目录、表和索引元信息的持久化后，你需要在 `src/include/catalog/catalog.h` 和 `src/catalog/catalog.cpp` 中实现整个 `CatalogManager` 类。 `CatalogManager` 类应具备维护和持久化数据库中所有表和索引的信息。 `CatalogManager` 能够在数据库实例（ `DBStorageEngine` ）初次创建时（ `init = true` ）初始化元数据；并在后续重新打开数据库实例时，从数据库文件中加载所有的表和索引信息，构建 `TableInfo` 和 `IndexInfo` 信息置于内存中。此外， `CatalogManager` 类还需要对上层模块提供对指定数据表的操作方式，如 `CreateTable` 、 `GetTable` 、 `GetTables` 、 `DropTable` 、 `GetTableIndexes` ；对上层模块提供对指定索引的操作方式，如 `CreateIndex` 、 `GetIndex` 、 `DropIndex` 。

Note: 在目前这种架构设计下，Catalog和实际数据存放在同一个数据库文件中。这使得Catalog Manager既依赖于Disk Manager 、Buffer Pool Manager，又为Record Manager、Index Manager、Executor(甚至Parser)提供接口支持，从而导致整个MiniSQL项目的内聚度相当地高。为此，一种妥协式的方案是：Catalog Manager不依赖于Disk Manager和Buffer Pool Manager，而是存放在独立的文件中，通过持久化手段直接落盘，但这种做法同样存在缺点，即Catalog的信息完全不受事务管理控制，存在一致性和可恢复性的问题。同学们可以根据实际需要自行选择Catalog Manager的实现方式。

#4.4 模块相关代码

- `src/include/catalog/catalog.h`
- `src/catalog/catalog.cpp`
- `src/include/catalog/indexes.h`
- `src/catalog/indexes.cpp`
- `src/include/catalog/table.h`
- `src/catalog/table.cpp`
- `test/catalog/catalog_test.cpp`

#4.5 开发提示

1. 推荐在**夏学期第5周前**完成本模块的设计。

#4.6 诚信守则

1. 请勿从其它组或在网络上找到的其它来源中复制源代码，一经发现抄袭，成绩为 `0`；
2. 请勿将代码发布到公共Github存储库上。

url=https%3A%2F%2Fwww.yuque.com%2Fyingchengjun%2Fpcp6qx%2Fyu3rrg&pic=null&title=%234%20CATA