数据库大作业 系统设计文档

潘乐怡 2020012374

徐浩博 2020010108

和嘉晅 2019010297

1. 概述

1.1 系统简介

本次大作业实现了一个简易版本的数据库，用户可以通过输入sql语句来进行数据库操作、表操作、查询操作等，同时支持多用户并发和重启恢复数据。

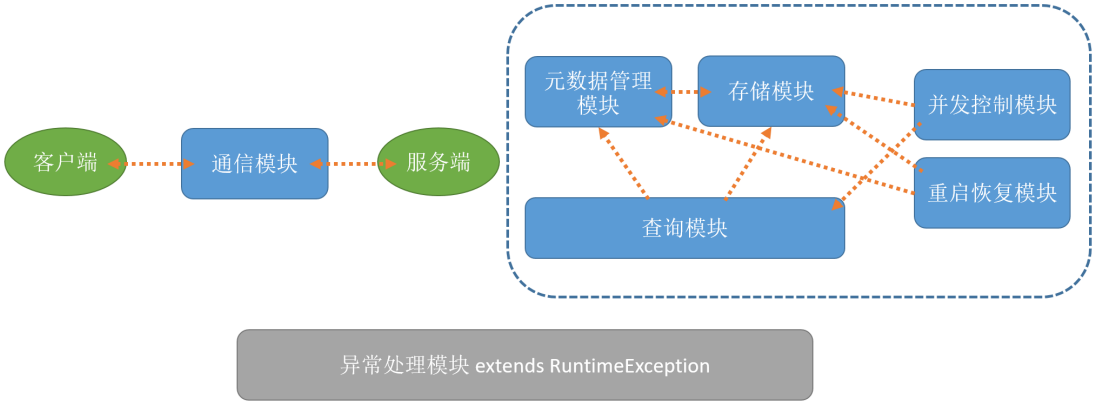
1.2 实现的功能点

- 所有基础要求
- **进阶要求**共7项
  - 扩展SQL语句：展示所有数据库名称、展示某一表的结构
  - 外键约束：在insert/update语句执行时查询父表B+树索引，检查引用完整性
  - 页式存储：将B+树每个内部节点和叶子结点均存储在文件的一张页上，读写树上节点均需访问对应页的位置；首个页面存储Header和空闲页面，运行时使用空闲页面链表分配页面
  - 提高I/O效率：数据库全局设立缓存池，采用写回缓存的工作方式，并采用LRU的页面置换算法，命中的页面可以直接在内存中访问，修改时也直接在内存中修改，待页面将置换下去之后再写回磁盘对应位置
  - 三张表以上的join：采用递归的方式，两张表先求join生成cross_table，再继续和其他表递归求join
  - 查询优化：对join查询进行索引嵌套循环连接，大幅度降低特定情况的join查询复杂度
  - checkpoint功能：留有单一事务的checkpoint接口，调用时将缓存池内的数据全部写回磁盘并更改log日志，并能够成功恢复

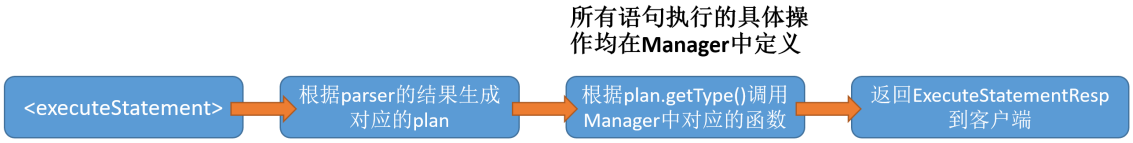
1.3 系统整体架构图

- 系统几大重要模块及其相互之间的依赖调用关系图如下：

  如用户在执行查询操作时，服务端查询模块进行操作时会调用元数据管理模块和存储模块里的内容



1.4 executeStatement流程



2. 核心模块

2.1 元数据存储模块

2.1.1 数据管理结构

- Manager.java中定义了成员`HashMap<String, Database> databases`，保存系统中所有数据库
- Manager.java中定义了成员`Database curDatabase`，存储现在正在使用的数据库
- Database类核心数据成员：
  - `databaseName`
  - `HashMap<String, Table> tables`
- Table类核心数据成员：（暂不考虑外键约束和btree修改）
  - `tableName`
  - `ArrayList<Column> columns`

2.1.2 create database

> 对应Manager.java中的函数`createDatabaseIfNotExists`

- 实现逻辑
  - 获取databaseName后判断是否已存在同名的database，如存在触发DuplicateDatabaseException
  - 如果没有重名问题，调用`databases.put`方法将新建的database添加到databases的hashMap中
  - 如果成功添加了新的database，调用`persistDatabases`将它的名字写入txt文件，实现database的数据记录持久化

    （注：仅database的数据记录保持是用txt存储实现的，table相关信息的记录保持见2.2.4）
- 涉及的exceptions

Exception	Description
DuplicateDatabaseException	如果用户尝试创建一个已经存在的database，触发该exception

2.1.3 drop database

> 对应Manager.java中的函数`deleteDatabase`

- 实现逻辑
  - 获取databaseName后判断是否已存在该database，如不存在触发DatabaseNotExistException

    - 如database存在，调用`databases.remove`方法将该database从hashMap中删除
    - 如果成功删除了database，调用`persistDatabases`重新将现有的databases写入txt文件
- 涉及的exception

Exception	Description
DatabaseNotExistException	如果用户尝试删除一个不存在的database，触发该exception

2.1.4 use database

> 对应Manager.java中的函数`switchDatabase`

- 实现逻辑
    - 获取databaseName后判断是否已存在该database，如不存在触发DatabaseNotExistException
    - 如database存在，设置curDatabase为该database即可
- 涉及的exception

Exception	Description
DatabaseNotExistException	如果用户尝试删除一个不存在的database，触发该exception

2.1.5 create table

> 对应Manager.java中的函数`createTableIfNotExist`

- 实现逻辑
    - 获取tableName后判断curDatabase中是否已存在该table，如存在触发DuplicateTableException
    - 如不存在重名的table，则取出create table语句中的每个column项依次解析：
        - column项数据类型
        - column项名字
        - 长度限制
        - Not Null 限制
        - 约束
            - primary constraint（判断是否有主键不存在或多列主键的情况）
            - foreign constraint（在 insert 和 update 语句执行时，检查引用完整性）
    - 将table各个column项存储在`_meta`文件中，重启恢复时根据该文件恢复出表的columns
- 涉及的exception

Exception	Description

Exception	Description
DuplicateTableException	如果用户尝试创建一个已经存在的table，触发该exception
MultiPrimaryKeyException	如果用户尝试创建一个包含多列主键的table，触发该exception
NoPrimaryKeyException	如果用户在创建table时没有声明主键，触发该exception

2.1.6 show table

> 对应Manager.java中的函数`showTable`

- 实现逻辑
  - 获取到table后将其名字、各column的数据类型、column名、长度限制、Not Null限制以及约束情况格式化输出，如下图所示：

```
Table Content
--------------------------------------------------------------------
table student
name(256)   STRING  PRIMARY KEY
age INT NOT NULL
id  INT
```

- 涉及的exception

Exception	Description
TableNotExistException	如果用户尝试展示一个不存在的table，触发该exception

2.1.7 drop table

> 对应Manager.java中的函数`deleteTable`

- 实现逻辑
  - 从curDatabase的table列表中删去该表，如果不存在则触发TableNotExistException
  - 删去表的`_meta`文件、B+树持久化文件`.bin`文件和，同时修改log，将log中有关该表的内容全部删去；如果文件不存在则触发FileNotExistException

  涉及的exception

Exception	Description
TableNotExistException	如果尝试删去一个不存在的table，触发该exception
FileNotExistException	如果尝试删去一个不存在的table对应存储文件，触发该exception

2.2 存储模块

2.2.1 insert

- 实现逻辑
  - 通过 Parser 获取表名，列名与值列表
  - 首先，若指定了列名称，检查加上可空列后能否形成一个完成的行
  - 之后按照列名称在表 scheme 中找出对应的列，把值 token 解析为带有 Java 类型信息的 Entry，形成 Row。
  - 最后以主键作为 Key，Row 作为 Value，插入 B+ 树
- 涉及的exception
Exception	Description
TableNotExistException	如果向不存在的table插入数据，触发该exception
SchemaLengthMismatchException	插入值（包括缺省的可空值）与列长度不匹配
NoPrimaryKeyException	数据未指定 primary key
DuplicateKeyException	插入后该表出现两个重复的primary key值

2.2.2 delete

- 实现逻辑
  - 通过 Parser 获取表名、是否有K_WHERE；如果有K_WHERE再获取条件语句的列名、比较运算符和值
  - 如果无K_WHERE，则将每行均删去
  - 如果有K_WHERE，则对于每个 Row 先获得对应条件语句列的值，然后与 sql 语句给的值进行大小比较，符合要求的删去
- 涉及的exception
Exception	Description
TableNotExistException	如果向不存在的table插入数据，触发该exception
AttributeNotExistException	如果条件语句中出现不存在的列名，触发该exception

2.2.3 update

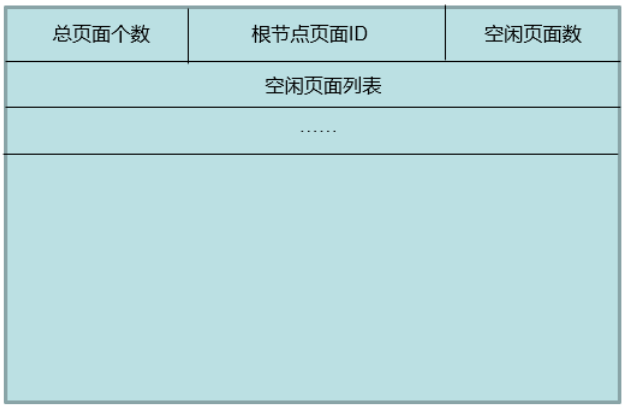
- 实现逻辑
  - 通过 Parser 获取表名、更新的列和值、是否有K_WHERE；如果有K_WHERE再获取条件语句的列名、比较运算符和值
  - 如果无K_WHERE，则对每行通过修改B+树对应primary key修改Value
  - 如果有K_WHERE，则对于每个 Row 先获得对应条件语句列的值，然后与 sql 语句给的值进行大小比较，符合要求的进行如上的修改
- 涉及的exception
Exception	Description
TableNotExistException	如果向不存在的table插入数据，触发该exception
DuplicateKeyException	修改后该表出现两个重复的primary key值

2.2.4 特色功能：数据持久化(页式存储+提高I/O效率)

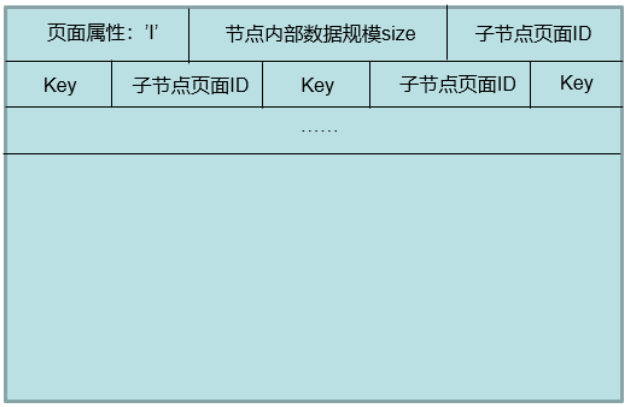
我们将整棵B+Tree均存入磁盘文件中以实现数据持久化，需要访问内部节点或叶子结点时均需从磁盘文件中读取。由于读写磁盘文件的效率较低，我们还对该方案进行了优化

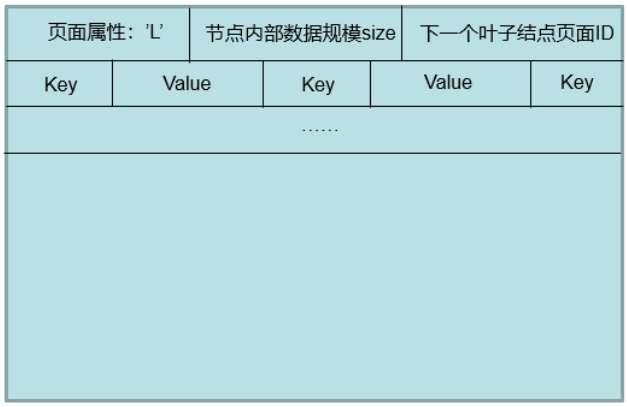
- 页式存储：将B+树每个内部节点和叶子结点均存储在文件的一张页上，读写树上节点均需访问对应页的位置；首个页面存储Header和空闲页面，运行时使用空闲页面链表分配页面
- 提高I/O效率：数据库全局设立缓存池，采用写回缓存的工作方式，并采用LRU的页面置换算法，命中的页面可以直接在内存中访问，修改时也直接在内存中修改，待页面将置换下去之后再写回磁盘对应位置

具体而言，我们将磁盘文件分为若干页，第一页固定为header，格式如下



之后的所有页均存储内部节点或叶子结点，格式如下：





对于页面分配，我们采取空闲页面链表的方式，将空闲（未分配或已经解除分配）的页面连接成为链表，欲分配时从链表表头取出一个空闲页面分配给新页面，而页面回收时，将空闲页面加至链表末尾。

如果遇到文件已满的情况，我们采用文件倍增的方式，将文件大小扩充为原来的两倍，由此产生出多一倍的空闲页面；这种方法既可以很好地防止空闲页面过多使得磁盘文件过大的问题，也可以降低需要扩增文件的次数，从而提高数据库运行效率。

为了测试我们作出的种种优化带来的效率提升，我们使用大作业自带的BPlusTreeTest.java进行读写测试，测试结果请见第四部分

可以看到我们的效率优化为B+树读写带来了巨大的性能提升

2.3 查询模块

> 对应Manager.java中的`select`函数

- 实现逻辑

  select语句形如：

  ```
  SELECT tableName1.AttrName1, tableName1.AttrName2…, tableName2.AttrName1,
  tableName2.AttrName2,…
  FROM tableName1 JOIN tableName2
  ON tableName1.attrName1=tableName2.attrName2
  WHERE tableName1.attrName1 = attrValue;
  ```

  因此我们的实现分为以下三个部分：

  - 计算join后的cross_table （对应from .... join ... on ...的部分）

    > 通过语句`finalTable = getFinalQueryTable(query);`求join后的table，具体求解方法在函数`getFinalQueryTable`内

    - 当出现三张表以上的join时，递归求解cross_table
    - 每两张表求cross_table时，先找出on条件左右是否有与这两张表相关的条件，如果有则进行join连接，否则直接求笛卡尔积
  - 根据where的条件选出符合条件的row（对应where...的部分）
    - 在得到第一步的`finalTable`后，通过`getRowsSatisfyWhereClause`函数选出符合条件的`newRows`
    - 使用`finalTable.rows = newRows;`来更新`finalTable`
  - 根据select后面的属性筛选出用户需要的column
    - 通过`filteredOnColumns`函数来筛选用户需要的column
- **进阶功能**：对join查询进行索引嵌套循环连接，大幅度降低特定情况的join查询复杂度。我们对比了查询优化前后的效率提升，测试结果详见第4部分。
- 涉及的exception

Exception	Description
TableNotExistException	如果用户尝试访问一个不存在的table，触发该exception
AttributeNotExistException	如果用户尝试访问一个不存在的attribute，触发该exception

2.4 并发控制模块

2.4.1 实现概述

- 在并发控制模块，我们实现了`READ_COMMITTED`和`SERIALIZABLE`两个隔离级别
- 可以通过在Manager中设置变量`seperateLevel`：值为`READ_COMMITTED`或`SERIALIZABLE`来区分两个隔离级别
- 我们采用表级锁的方式来进行隔离，其中：
  - Manager中保存数据成员`HashMap<Long, ArrayList<String>> s_lock_dict`（记录每个session取得了哪些表的s锁）和`HashMap<Long, ArrayList<String>> x_lock_dict`（记录每个session取得了哪些表的x锁）
  - Table中保存数据成员`ArrayList<Long> s_lock_list`（存储该表的s锁被哪些session持有）和`x_lock_list`（存储该表的x锁被哪些session持有）

2.4.2 不同隔离级别的特性和加锁方式

隔离级别	丢失更新	脏读	不可重复读	幻读	加锁方式
`READ_UNCOMMITTED`	否	是	是	是	update之前尝试获取x锁，commit的时候释放
`READ_COMMITTED`	否	否	是	是	update之前尝试获取x锁， commit的时候释放 select的时候要获取该表s锁，执行完就释放
`Repeatable Read`	否	否	否	是	update之前尝试获取x锁，commit的时候释放 select的时候要获取该表s锁，commit的时候释放

隔离级别	丢失更新	脏读	不可重复读	幻读	加锁方式
`SERIALIZABLE`	否	否	否	否	update之前尝试获取x锁，commit的时候释放 select的时候要获取该表s锁，commit完释放 insert和delete之前尝试获取x锁，commit的时候释放

2.4.3 具体实现

- Table类中设计以下函数来尝试获得、释放锁
  - `get_x_lock`
  - `get_s_lock`
  - `free_x_lock`
  - `free_s_lock`
- Manager类中用 `s_lock_dict.put()`、`s_lock_dict.remove()`、`x_lock_dict.put()`、`x_lock_dict.remove()` 来更新各个session对各个表级锁的持有状况
- session_queue 会话阻塞队列：

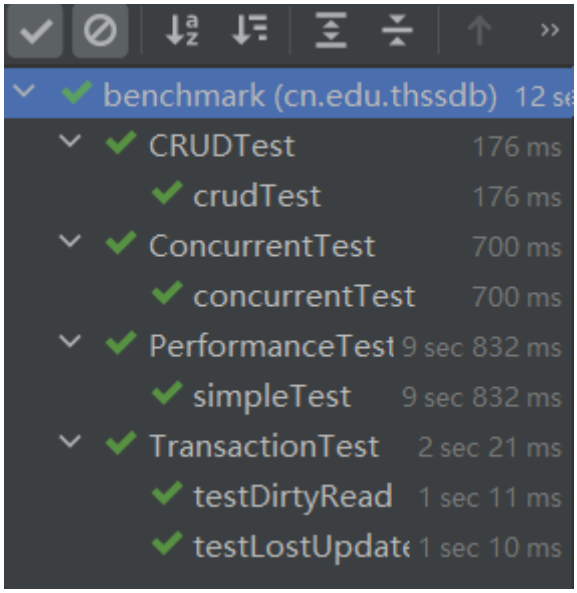
  记录因为暂时没有拿到锁而被阻塞的会话；被阻塞的函数内部使用while循环尝试拿锁，等锁释放，可以拿到锁后继续执行；

2.5 重启恢复模块

- 我们实现了单一事务的WAL机制，在数据库进行 `INSERT`、`UPDATE`、`DELETE` 时，记录对应的数据库名、表名和增删改的数据表项；欲恢复时，逐行读取log进行数据恢复。
- **进阶功能：**我们还实现了单一事务的checkpoint API；具体来说，我们在调用checkpoint的API时，在log中记录CHECKPOINT并且将缓存池中的数据全部写入磁盘文件，并将磁盘文件进行快照备份；由于磁盘文件中储存完整的B+树树状数据，因此只需从log中最后一个CHECKPOINT行之后，对最后一次checkpoint数据备份文件进行操作即可。在实际应用中，可以考虑采用定期调用或者关闭数据库时调用checkpoint API的方法，这样可以有效提高重启恢复的速度。
- 具体配置：`cn.edu.thssdb.utils.Global` 内可以调整 `public static final Boolean RECOVER_FROM_DISC` 以改变. 设置为 `false` 时不借助 `TABLE.bin` ，此时可以没有B树的二进制文件目录，恢复时直接从 LOG 文件中恢复。设置为 `true` 时会借助 `TABLE.bin` 和 `log/DATABASE` 日志文件。默认值为 `false`。

3. 测试结果

3.1 功能测试



3.2 性能测试

> 测试程序： benchmark.PerformanceTest
>
> 测试参数：
>
> *CLIENT_NUMBER* = 5，*TABLE_NUMBER* = 3，*OPERATION_RATIO* = "50:10:10:20:10"

Operation规模	Performance Test用时
1000	1.8s
2000	3.1s
5000	14.5s
10000	54.9s
20000	419s

3.3 进阶效率优化消融实验

- 页式存储+I/O效率优化

  > 测试程序： index.BPlusTreeTest
  >
  > page setting: page size = 8KB, fanout = 65
  >
  > compilation setting: java 8 -ea -Xmx32M -Xms16M

数据规模	1k	10k	100k
RandomAccessFile定位位置边偏移边读写	2.392s/4.223s/2.453s	31.980s/44.954s/23.344s	10min34s/-/-
RandomAccessFile整页读写	434ms/635ms/330ms	4.796s/7.556s/3.727s	45.927s/76s/45.901s
采用缓存池，buffer pool size = 4	173ms/146ms/86ms	1.428s/2.100s/1.164s	13.600s/22.685s/33.803s
采用缓存池，buffer pool size = 32	128ms/74ms/49ms	1.054s/1.533s/0.967s	10.415s/15.826s/30.455s
采用缓存池，buffer pool size = 1024	152ms/73ms/65ms	767ms/781ms/658ms	9.478s/13.725s/29.912s
内存 (内存不受限的原始版本)	11ms/3ms/3ms	25ms/13ms/88ms	0.251s/0.224s/7.454s

> 注：X/Y/Z分别代表testGet/testRemove/testIterator的时间

- 查询优化

> 测试程序： benchmark.PerformanceTest
>
> 测试参数：
>
> *CLIENT_NUMBER* = 5，*TABLE_NUMBER* = 3，*OPERATION_NUMBER* = 2000，*OPERATION_RATIO* = "50:10:10:20:10"

优化前	优化后
5.3s	3.1s

3.4 其他进阶功能

- checkpoint：经过反复测试，THSSDB 可以从二进制文件联合 log 中恢复数据，如果没有 log 也可以单独从二进制文件中恢复数据。
- 扩展SQL指令、外键约束、多表查询（三张表以上的join）：经过多次测试，符合预期

4. 项目分工

4.1 基础功能

模块	负责人
元数据管理模块	徐浩博: Database相关 潘乐怡: Table相关
存储模块	和嘉晅：insert 徐浩博：delete & update
查询模块	和嘉晅：单表select 潘乐怡：多表select
并发控制模块	潘乐怡、和嘉晅
重启恢复模块	徐浩博

4.2 进阶功能

功能	负责人
外键约束	和嘉晅
页式存储	徐浩博
I/O查询优化	徐浩博
查询优化	和嘉晅
三张表以上的join	潘乐怡
checkpoint	徐浩博

5. 加分点自评

本组除了完成全部基础功能之外，还附加完成了7项进阶功能。这7项中，**有3项优化了数据库效率**，使得数据库效率比原始版本至少有了一到两个数量级提升（详见3.3部分），这样的提升是巨大的。

另外，**本组一个特色功能是实现了checkpoint功能。**通过一个二进制文件，整个数据库的数据可以被恢复。一般情况下，内存受限的实现仅需存储B+树叶子结点，因此只能依靠 log 恢复数据。然而，在现实生活中，**通过文件恢复整个数据库的数据**是十分必要的，真实的服务器关闭或崩溃后，我们并不能通过 log 文件重新执行所有之前的操作来恢复，这种方法效率过低且不安全；依靠我们的方法，结合定期checkpoint或者手动checkpoint，可以最大程度上保存数据。实现这个功能需要将整个B+树结构储存在文件中，包括各种文件头、各种节点类型以及与文件管理和操作相关的内容。因此，相对于其他团队的实现，这个功能的难度较大。因此，我认为这个功能为我们的项目增加了不少亮点。