

通信模块

通信模块采用 CS 架构，使用 thrift 进行通信。根据接口协议实现与服务端 sql 处理对接即可

基础要求：

- ✓ 实现 connect、disconnect 和 executeStatement 服务。

异常处理模块

- ✓ ColumnNotExistException：列不存在
- ✓ PrimaryKeyEmptyException：主键不能为空
- ✓ TableNotExistException：表不存在

存储模块

存储接口主要体现在对 Table 类的记录插入、删除、修改和查询上

- void insert(Row row)
插入记录 row
- void delete(Row row)
删除记录 row
- void update(Row newRow, Row oldRow)
更新记录 oldRow 至 newRow，会首先检查原记录和新纪录主键是否相同，若相同，则直接更新即可；
若不同，首先删除 oldRow，接着插入 newRow
- Row get(Entry entry)
根据主键获取记录
- void persist()
持久化所有记录
- void recover()
根据持久化的文件恢复所有的记录

记录持久化

记录持久化主要使用 java 的 serialize 和 deserialize，在持久化时，serialize 所有记录保存到文件中，在恢复时，对文件进行 deserialize 获取所有记录。

测试

在 TableTest 文件中对上述功能进行了测试。

基础要求：

- ✓ 利用 java 的序列化和反序列化实现记录的持久化；

- ✓ 实现对记录的增加、删除、修改、查询；
- ✓ 支持五种数据类型：`Int`，`Long`，`Float`，`Double`，`String`。

进阶要求：

- ☐ 实现高效的文件存储格式，如页式存储等

元数据管理模块

元数据管理主要体现在数据库对表的管理和 `Manager` 对数据库的管理

表的创建、删除（`Database`类相关接口）

- `void create(String name, Column[] columns)`
创建含有指定列的表
- `void delete(String name)`
删除指定表
- `void persist()`
持久化所有表及元数据
- `void drop()`
删除所有表
- `void recover()`
根据持久化的元数据恢复所有表

数据库的创建、删除、切换（`Manager`类相关接口）

- `void createDatabaseIfNotExists(String databaseName)`
创建指定数据库
- `void deleteDatabase(String databaseName)`
删除指定数据库
- `void switchDatabase(String databaseName)`
切换当前用户的数据库
- `void persist()`
持久化所有数据库及元数据

元数据持久化设计

所有元数据均存储在 `data` 目录下，`manager.meta` 会存储所有 `database` 的名字，每个 `database` 含有的所有表名字会存储在 `name.meta` 中，其中 `name` 为 `database` 的名字，数据库中的表的元信息（所有列的信息）会存储在 `databaseName/tableName.meta` 中，持久化方式均采用 `json` 格式序列化进行存储。

基础要求：

- ✓ 实现表的创建、删除；
- ✓ 实现数据库的创建、删除、切换；
- ✓ 实现表和数据库的元数据的持久化。

- ✓ 重启数据库时从持久化的元数据中恢复系统信息。

进阶要求

- ✓ 实现表的修改。

查询模块

查询模块的主要功能主要有两点，一是通过对 sql 语句的解析，得到输入文本中对应的 sql 语义和操作。二是根据我们解析得到的 sql 语句，执行对应的操作，与服务端其它部分对接，返回查询和操作结果。

语句解析

使用 antlr 生成 parser 后，使用 visitor 模式遍历 sql 语法树。对于每一条 sql 中合法的 statement，我们将 visitor 解析得到的元数据封装为一个 BaseStatement 的子类，例如一个 SelectStatement 对象封装了一条具体的查询语句，其包括查询的列名，where 条件，涉及的表等内容。为了更好地结构化 statement 的数据，我们还设计了一些辅助类。Comparer 代表一个布尔表达式中的元素，可能是常数，也可能是一个列名。Condition 代表一个具体的条件，可能对应 where 子句，也可能对应 join condition。TableQuery 代表一个 from 子句的子项，例如单独的一张表或是多张表的连接。ValueEntry 代表 insert 操作中的一组值。

测试

在 EvaluatorTest 文件中对上述功能进行了测试。

语句执行

在语句执行上，为了解耦各个模块的功能，我们在查询模块并不直接负责语句的执行，而是通过 Statement 对象封装执行该语句所需的所有数据，并提供 exec 接口，该接口负责执行当前语句，并返回一个 SQLEvalResult。这一设计下，我们的语句执行可以与之后的事务并发和控制相结合，实现更多样的执行过程。

基础要求：

- ✓ 创建表【CREATE TABLE tableName(attrName1 Type1, attrName2 Type2,..., 3attrNameN TypeN NOT NULL, PRIMARY KEY(attrName1))】
- ✓ 删除表【DROP TABLE tableName】
- ✓ 插入数据【INSERT INTO [tableName(attrName1, attrName2,..., attrNameN)] VALUES (attrValue1, attrValue2,..., attrValueN)】
- ✓ 删除数据【DELETE FROM tableName WHERE attrName = attrValue】
- ✓ 更新数据【UPDATE tableName SET attrName==attrValue WHERE attrName = attrValue】
- ✓ 选择数据【SELECT attrName1, attrName2, ..., attrNameN FROM tableName WHERE attrName1 = attrValue】
- ✓ 表连接选择数据【SELECT tableName1.AttrName1,tableName2.AttrName1,tableName2.AttrName2,... JOIN tableName2 ON tableName1.AttrName2...,FROM tableName1 tableName1.attrName1 = tableName2.attrName2 [WHERE attrName1 = attrValue]】

进阶要求：

- ☐ 应用课程中介绍的查询优化技术
- ☐ 支持多列主键；
- ☒ where 条件支持逻辑运算符（and/or）；
- ☒ 实现三张表以上的 join；
- ☒ 实现 outer join 等其他类型的 join；
- ☒ 其他标准 SQL 支持的查询语法。

进阶要求实现思路：

1. 对于 where 件逻辑运算符，我们通过 Condition 类，用二叉树组织一条语句的所有条件，一个 Condition 对象的左右子节点分别代表两条子条件，并根据该对象的 logic_op，即逻辑运算符类型，在查询时，递归地求左右子节点的逻辑值，并进行合并。
2. 对于三张表以上的 join，我们同样通过 TableQuery 类，用二叉树结构组织待连接的表，其两个子节点分别代表左表和右表(或是连接的中间体)，通过不同的 join 类型，实现不同的连接操作
3. 对于不同类型的 join, 我们通过 TableQuery 类的递归执行流程，通过 join type 的判断和分支，可以很自然的根据其原理实现。目前我们支持 natural, left/right/full outer, inner 这几种 join 类型。

事务模块

在事务模块的实现上，实现了 read commit 的隔离级别。我们实现了 TransactionManager 类，一个 TransactionManager 对象对应一个数据库，负责该数据库上所有事务的并发执行和记录。其内部维护多张列表，包括目前正在并发的会话列表，每个会话持有的写锁和读锁列表。在每一个 Statement 需要执行时，其对应的会话调用 TransactionManager 的 exec 接口，TransactionManager 根据 Statement 的类型，进行不同的加锁操作，并执行该操作，执行结束后将其写入日志。在需要回滚时，则根据目前 Logger（见下文介绍）的 undo_list 对操作逐一进行回滚操作。

WAL 模块

在 WAL 模块上，我们为每个数据库设立一个 Logger 类对象，其负责将针对于该数据库的操作记录下来并读写对应的文件。在其内部，它分别维护 undo_list 和 redo_list 两个列表，分别记录可能需要 redo 和 undo 的 Statement。每当一个 Statement 在事务中被执行时，我们便将该 Statement 和撤销该操作需要的 Statement 记录下来。在需要保存时(checkpoint 或数据库 shutdown 时)，将其以 java 序列化的格式写入磁盘。在恢复时，数据库根据自身 Logger 对象对应的文件，读取序列化的数据，进行数据的恢复。数据恢复过程中先进行 redo 的操作，再进行 undo 的操作。

基础要求：

- ☐ 实现 begin transaction 和 commit；采用普通锁协议，实现 read committed 的隔离级别即可；
- ☐ 实现单一事务的 WAL 机制，要求实现写 log 和读 log，在重启时能够恢复记录的数据即可。

进阶要求：

- ☐ 实现多事务的并发和恢复机制；
- ☐ 实现更高级的隔离级别（repeatable read/serializable）；
- ☐ 实现 rollback、savepoint 等功能。
- ☐ 实现 2PL 或 MVCC 协议。

