设计文档

功能实现

通信模块

Connect、Disconnect、ExecuteStatement。

存储模块

- 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。

元数据管理模块

- 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。

查询模块

- `CREATE TABLE tableName(attrName1 Type1, attrName2 Type2,…, attrNameN TypeN NOT NULL, PRIMARY KEY(attrName1))`
- `DROP TABLE tableName`
- `INSERT INTO [tableName(attrName1, attrName2,…, attrNameN)] VALUES (attrValue1, attrValue2,…, attrValueN)`
- `DELETE FROM tableName WHERE attrName = attValue`
- `UPDATE tableName SET attrName = attrValue WHERE attrName = attrValue`
- `SELECT attrName1, attrName2, … attrNameN FROM tableName [ WHERE attrName1 = attrValue ]`
- `SELECT tableName1.AttrName1, tableName1.AttrName2…, tableName2.AttrName1, tableName2.AttrName2,… FROM tableName1 JOIN tableName2 ON tableName1.attrName1 = tableName2.attrName2 [ WHERE attrName1 = attrValue ]`

事务与恢复模块

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

附加功能

- 元数据管理模块：实现表的修改；
- 查询模块：`SHOW TABLES`、`SHOW TABLE tableName`、`SHOW DATABASES` 命令支持；
- 查询模块：where 条件支持逻辑运算符（and/or）；
- 事务与恢复模块：2PL 锁协议。

代码结构

`client`：客户端模块

- `main`：接受命令参数，对 SQL 进行简单的预处理，包括去除空格、多个语句分割，并处理一些简单的 SQL 语句（`disconnect`、`help`、`quit`、`show_time`）
- `getTime`：返回客户端时间
- `connect`：利用用户名和密码连接服务端
- `disconnect`：断开与服务端的连接
- `executeStatement`：将 SQL 传送到服务端，并打印执行结果
- `showHelp`：显示帮助文本

`exception`：异常处理模块

实现了以下自定义异常类：数据库已存在、数据库不存在、键值重复、空值、键值不存在、锁等待超时、没有选定数据库、读取文件出错、表已存在、表不存在、未知数据类型、写入文件出错。

`index`：索引模块

- `BPlusTree`：索引 B+ 树
- `BPlusTreeInternalNode`：内部结点，继承自 `BPlusTreeNode`
- `BPlusTreeIterator`：结点迭代器
- `BPlusTreeLeafNode`：叶子结点，继承自 `BPlusTreeNode`
- `BPlusTreeNode`：树结点抽象类

`parser`：解析模块

- `SQLVisitorImple`：继承自 ANTLR 根据 `SQL.g4` 生成的 `SQLBaseVisitor`，实现了对 SQL 语句的解析，并实现或调用响应的执行方法
- `statement`
  - `Condition`：用于存储一个比较关系；
  - `Expression`：用于存储单个表达式（最多包含一个加减乘除运算符）
  - `ColumnFullName`：用于存储表名 + 列名；
  - `LiteralValue`：用于存储一个值；
  - `TableQuery`：用于存储两个表的比较关系。

`query`：查询模块

- `MetaInfo`：存储一个表的名字和所有列
- `QueryResult`：选择运算中选择某些列或全部列的结果
- `QueryTable`：选择运算中选择表的方法类

`rpc.thrift`：RPC 调用模块

根据 `rpc.thrift` 生成的请求和回应类，包括：获取时间、连接、断开连接、执行语句。

`schema`：存储模块

- `Column`：存储一列的名称、类型、是否为主键、能否为空、最大长度，及其序列化方法
- `Database`：存储一个数据库的名称及其中的表
  - `persist`：序列化方法
  - `recover`：反序列化方法
  - `create`：创建一个新表

    - `drop`：删除一个表
    - `getTable`：根据名称获取一个表
    - `quit`：将当前数据库的所有数据序列化
    - `getTableNameList`：获取表的名称列表
    - `getMetaPersistFile`：获取数据库元数据存储路径
    - `hasTable`：判断是否存在某个表
- `Entry`：存储一个记录的一个属性值
- `Manager`：数据库管理类
    - 成员变量：
        - `databases`：数据库列表
        - `transactionSessions`：处于事务状态中的连接列表
        - `logger`：日志管理类
    - `Manager`：构造时从序列化文件和 redo 日志中恢复数据
    - `createDatabaseIfNotExists`：建立一个数据库，如果已存在，则什么也不做
    - `deleteDatabase`：删除数据库
    - `recover`：反序列化方法
    - `persist`：序列化方法
    - `hasDatabase`：判断是否存在某个数据库
    - `getMetaPersistFile`：获取所有数据库的元数据文件路径
    - `getDatabaseNameList`：获取数据库名称列表
    - `isTransaction`：判断连接是否正处于事务状态
    - `addTransaction`：将连接启动事务状态
    - `commitTransaction`：将连接的事务进行提交
    - `Logger`：redo 日志管理类
        - 成员变量:
            - `logCnt` 记录当前日志条数，如果条数超出阈值则对日志内容进行redo并清空
        - `logDatabaseStmt`：对数据库操作(创建、删除)进行记录
        - `logTableStmt`：对表操作(创建、删除)进行记录
        - `logRowStmt`：对行操作(插入、删除、更新)进行记录
        - `commitLog`：将当前Session的日志记录写入到日志文件。此方法在事务正式 `commit` 前调用，确保在事务开始执行 `commit` 时包含的操作都已经在日志中
        - `redoLog`：读取日志文件，对其中的操作逐行进行 Redo
        - `redoDatabaseStmt`：对日志中的数据库操作(创建、删除)进行 Redo
        - `redoTableStmt`：对日志中的表操作(创建、删除)进行 Redo
        - `redoRowStmt`：对日志中的行操作(插入、删除、更新)进行 Redo
- `Row`：存储一行记录
- `Table`：存储一张表的信息
    - 成员变量：
        - `columns`：所有的列信息
        - `index` 表的索引信息
        - `databaseName`：所在数据库的名称

        - `tableName`：表的名称
        - `primaryIndex`：主键的序号
        - `xLockOwner`：拥有当前表的排他锁的连接的 sessionId
        - `sLockOwner`：拥有当前表的共享锁的所有连接的 sessionId
    - `removeXLock`：移除连接的排他锁
    - `removeSLock`：移除连接的共享锁
    - `getXLock`：尝试获取排他锁，成功返回 true，否则返回 false
    - `getXLockWithWait`：以超时等待的方法尝试获取排他锁，多次失败抛出异常
    - `getSLock`：尝试获取共享锁，成功返回 true，否则返回 false
    - `getSLockWithWait`：以超时等待的方法尝试获取共享锁，多次失败抛出异常
    - `persist`：表的内容进行序列化
    - `recover`：索引进行反序列化
    - `insert`：插入一行记录
    - `delete`：删除一行记录
    - `clear`：删除所有记录
    - `update`：更新一行记录
    - `addColumn`：增加一列
    - `dropColumn`：删除一列
    - `alterColumn`：更改一列的信息
    - `indexOfColumn`：列的顺序序号
    - `getPersistDir`：获取表的序列化文件夹
    - `getRowsPersistFile`：获取存储表的内容的文件路径

`server`：服务端启动模块

- `ThssDB`：为每个连接分配 `handler` 和 `processor`

`service`：服务端执行模块

- `IServiceHandler`：继承自 thrift 生成的 `IService.Iface`
    - `nextSessionId`：分配给下一个连接的 sessionId
    - `databaseManager`：负责数据库的管理操作
    - `abortSessions`：记录失效的 sessionId
    - `sessions`：记录键值对 （sessionId, session）
    - `getTime`：回复获取时间请求
    - `connect`：执行并回复连接请求
    - `disconnect`：执行并回复断开连接请求
    - `executeStatement`：执行并回复 SQL 命令请求
    - `isValidAccount`：检查连接的用户名和密码是否合法
    - `isValidSessionId`：检查 sessionId 是否合法
- `Session`：记录一个连接的信息
    - 成员变量：
        - `sessionId`：连接的标识
        - `currentDatabase`：当前选定的数据库

        - `currentDatabaseName`：当前选定的数据库名称
        - `xTables`：当前连接拥有的排他锁列表
        - `sTables`：当前连接拥有的共享锁列表
    - `releaseLocks`：释放所有锁

`type`：自定义类型

- `AlignType`：对齐类型，包括左、中、右对齐。
- `ColumnType`：元组类型，包括整数、长整数、单精度浮点、双精度浮点、字符串。
- `ConstraintType`：约束类型，包括非空和主键。

`utils`：封装工具类

- `Cell`：字符串对齐打印的封装类。
- `Global`：配置定义。
- `pair`：键值对模板类。
- `StringHelper`：字符串工具类，目前封装了：字符串数组查找函数。

模块设计

通信模块

通过 thirft 协议进行 RPC 调用，客户端发出请求，客户端返回状态码和可选的数据或报错信息。

客户端连接服务端时，服务端收到请求后返回一个 sessionId 作为连接的标识，并将该 sessionId + 1 作为下一个新连接的标识，客户端存储该 sessionId 并作为后续请求数据的一部分。客户端断开连接时，服务端收到请求后，进行相应的数据保存等工作，然后将该 sessionId 加入到废弃列表。下次重新连接会分配一个新的 sessionId 。

异常处理模块

首先定义了继承自 `RuntimeException` 的抽象类 `MyException` ，其中存储一个可选的字符串，其他异常类均继承 `MyException` 。子类覆写了 `getMessage` 方法，返回一个包含报错信息的字符串。

存储模块

所有数据库的所有信息都存储在 /data 文件夹中，需要存储的类实现了 `Serializable` 接口，并实现了序列化和反序列化方法。

- 所有数据库的元信息存储在 `/data/databases.meta` 文件中，由 `Manager` 负责序列化和反序列化。
- 每张表的元信息存储在 `/data/{table_name}/{table_name}.meta` 文件中，由 `Database` 负责序列化和反序列化。
- 每张表的数据存储在 `/data/{table_name}/{table_name}.data` 文件中，由 `Table` 负责序列化和反序列化。

查询模块

使用 B+ 树构建索引，其中键为主键，值为 Row 对象。

`Delete` 操作、`Update` 操作以及 `Select` 操作均需要单表查询。单表查询的方法为，根据语法解析结果构造 `Condition` 类，用需要查询的表构造 `QueryResult` ，将 `condition` 传入 `QueryResult` 可以筛选出符合要求的列索引，从而筛选出指定的 `Row` 。得到符合要求的 `Row` 之后，在此基础上完成 `Delete` 操作、`Update` 操作以及 `Select` 操作。

`Select` 操作有时需要通过 `join` 实现多表的查询。多表查询需要额外两个工作：

- 在通过两个表构造 `QueryResult` 时，需要通过 `on` 的连接条件，检查两对 `row` 之间是否可以拼接。如果可以，再将拼接后的 `row` 存储
- 需要将两个表的所有属性重命名，在属性前加上表名

对于多条件的查询，通过 `multiple_condition` 获取多条 `condition`，再通过 `multiple_condition` 中的操作符判断时 `AND` 还是 `OR` 操作，再选取 `row` 时应用所有判断条件。

事务模块

在 `SQL.g4` 中增加了事务开始和提交的命令： `begin transaction` 和 `commit` 。

为了防止访问冲突，每个表都拥有共享锁（读锁）和排他锁（写锁）， `Session` 中的 `sTables` 和 `xTables` 记录了一个连接获取的所有锁，以方便在提交事务时释放所有锁。每个 `Table` 中的 `xLockOwner` 和 `sLockOwner` 记录了当前表的锁的拥有者。当一个连接试图获取一个锁时，会每个若干秒进行一次获取尝试，若尝试失败则等待一定间隔后继续尝试，多次失败后抛出异常 `LockWaitTimeoutException` 。

实现了 2PL 锁协议，只有在 commit 时释放所有锁，中间过程只能加锁不能解锁。为了防止死锁，一旦多次获取锁失败，事务宣告失败，并释放所有锁。因此达到了 read committed 隔离级别。

恢复模块

在 `Manager` 中实现了 redo 日志的管理类 `Logger` ，提供了记录操作日志、将日志写入文件、从日志文件恢复操作的功能。数据库、表、行操作均会在一个内存中的 `logList` 里留下记录(维护在Session内)，当事务的 `commit` 被触发时，内存中的 `logList` 首先被写入文件，`log` 写入完毕后才会真正执行 `commit` 操作。数据库启动时，会读取之前的日志记录并重做相关操作。