DataBase System

MiniSQL □□□□□□

□□□ □ □ □

□□ □□□□□□□

- □□□□□□□□□□□□□□□□□□□□□SQL□□MinSQL
- □□□□□□□□□SQL□□□□□□□□□□□□□□SQL□□□□□□□□□□□□□□□□□
- □□□□□□□□□□□□□□□□□□
- □□□□□□□ □□□□□Index □□□ Executor

□□ □□□□□□

- □□□□□□□□□□□□□
  - `CATALOG`□□□□□
    ```
    catalog
    ├── catalog.cpp
    ├── indexes.cpp
    └── table.cpp
    ```
  - `RECOVERY MANAGER`□□□□□
    ```
    recovery/
    ├── log.txt
    ├── log_manager.h
    ├── log_rec.h
    └── recovery_manager.h
    ```
  - □□□□□□□□□□□□bug□□

□□ □□□□□□

`Catalog Manager`□□□□□□

1□ □□□□

Catalog Manager □□□□□□□□□□□□□□□□□□□□□
● □□
● □□□□□□□□□□□□□□□□□□□□□□□□□
● □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□
□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Catalog Manager□□□□□□□□□□Executor□□□□□□□□□□□□□□□□□
□□□□□□□□

2□ □□□□□□□□□□□□□□□

IndexInfo::Init

1.□□□meta_data_
2.□□□key_schema_(□□□□□)
3.CreateIndex□□□□

GetSerializedSize()

□□SerializeTo,DeserializeFrom□□□□□□□□□□□(CatalogMeta,TableMetadata,IndexMetadata)
□CatalogMeta□□

```
void CatalogMeta::SerializeTo(char *buf) const {
  MACH_WRITE_UINT32(buf, CATALOG_METADATA_MAGIC_NUM);
  buf += 4;
  MACH_WRITE_UINT32(buf, table_meta_pages_.size());
  buf += 4;
  MACH_WRITE_UINT32(buf, index_meta_pages_.size());
  buf += 4;
  for (auto iter : table_meta_pages_) {
    MACH_WRITE_TO(table_id_t, buf, iter.first);
    buf += 4;
    MACH_WRITE_TO(page_id_t, buf, iter.second);
    buf += 4;
  }
  for (auto iter : index_meta_pages_) {
    MACH_WRITE_TO(index_id_t, buf, iter.first);
    buf += 4;
    MACH_WRITE_TO(page_id_t, buf, iter.second);
    buf += 4;
  }
}
```

□□□□CatalogMeta□□□□□□: 4 bytes magic num + 4 bytes table_page num + 4 bytes index_page num

- table_page nums * (4 bytes table id + 4 bytes page id)
- index_page nums * (4 bytes index id + 4 bytes page id)

```
uint32_t CatalogMeta::GetSerializedSize() const {
  return 4 + 4 + 4
  + table_meta_pages_.size() * (4 + 4)
  + index_meta_pages_.size() * (4 + 4);
}
```

Catalog.cpp

CatalogManager Ctor

1.□□catalog_meta_page
2.□□init□□□□□metadata
init == true
1)NewInstance()□□metadata
2)□□catalog_meta_page□Data□□□□
init == false
1)□□catalog_meta_page□Data□□□□□
2)GetTable/IndexMetaPages□□table□index metadata
3.set next page id
4.unpin fetched page

CreateTable

1.create table_heap using DEEPCOPY SCHEMA
2.create table metadata
3.serialize to meta page
4.update catalog meta
5.create tableinfo
6.update table_names,tables_ in catalog
7.create index for unique attributes(primary key)

CreateIndex

1.create index metadata
2.serialize to meta page
3.update catalog meta
4.create indexinfo
5.update index_names_,indexes_
6.build up the tree(□□table_heap□bptree□□□□□□□InsertEntry)

GetTable(s)/GetIndex/GetTableIndexes

□□table_name□□tables_,(□□)□□table_info
□□table_name□□index_names_,□□index_id,□□□index_name□□indexes_□□index_info
□□table_name□□index_names_,□□□□index_id,□□index_id□□indexes_□□□□index_info

DropTable/DropIndex

1.delete metadata page
2.delete table_heap

3.update catalog meta
4.erase table_names,tables_/index_names_,indexes_

FlushCatalogMetaPage

1.get catalogmetapage
2.serialize catalog_meta_
3.unpin page(set dirty)

LoadTable/LoadIndex

1.get table/index metadata page
2.deserialize table/index metadata 3.create table_heap/get table info
4.create table_info/index_info
5.update tables_,table_names_/index_names_,indexes_
6.unpin page(set not dirty)

Recovery Manager□□□□□□

1□ □□□□

Recovery Manager □□□□□□□□□□□□□□□□□□□□
- □□□□□□□
- □□□CheckPoint□□□
- □□Redo□Undo□□

□□□Recovery Manager□□□□□ □□

2□ □□□□□□□□□□□□□□□

```
    std::map<lsn_t, LogRecPtr> log_recs_{}; //□□□□
    lsn_t persist_lsn_{INVALID_LSN};        //□□□□□□□□□
    ATT active_txns_{};                     //□□□□□□
    KvDatabase data_{};                     //□□

struct CheckPoint { //CheckPoint□□□□□
    lsn_t checkpoint_lsn_{INVALID_LSN};       //checkpoint□□□□□□
    ATT active_txns_{}; // txn_id -> last_lsn //checkpoint□□□□□□□last_lsn_
    KvDatabase persist_data_{};               //checkpoint□□□□□□
};
```

□□recovery_manager_test.cpp□□□□□□□□□□□□□□□

```
0. <T0 Start>
1. <T0,A,2000,2050>
2. <T0,B,1000,->
3. <T1 Start>
3. <CheckPoint{T0,T1}> Redo
4. <T1,C,-,600>         |
5. <T1 Commit>          |
6. <T0,C,600,700>       |
                        |
7. <T0 Abort>           |   Undo(A=2000,B=1000,C=600,D=-)
8. <T2 Start>           |   ↑
9. <T2,D,-,30000>       |   |
10.<T2,C,600,800>       ↓   |
                   Undo list{T2}(A=2000,B=1000,C=800,D=30000)
11.<T2,C,800,600>
12.<T2,D,->
13.<T2 Abort>
```

1.□checkpoint□□redo□□□□□□□□□□□□□□□□□□□undo list

2.□□undo list□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

RedoPhase

Redo□checkpoint□□□□□□□□□□□persist_lsn_□□□□□□□□□□□
□□□□□□□□data_□□□□□□□
□Begin□□□□□□□□□□□□□□□□□(undo list)
□Commit□□□□□□□□□□□□□□□□
□Abort□□□□□□□□□□□□□□prev_lsn□prev_log□□Abort□□□□□□□□□□□□□□□□□□Begin
□□□persist_lsn_□□□□□□□□lsn

```
    void RedoPhase() {
        for(std::map<lsn_t, LogRecPtr>::iterator it =
log_recs_.upper_bound(persist_lsn_); it != log_recs_.end(); ++it)
        {
            LogRecPtr log = it->second;
            switch (log->type_) {
                case LogRecType::kInsert:
                    data_[log->kv_data.key] = log->kv_data.val;
                    break;
                case LogRecType::kDelete:
                    data_.erase(log->kv_data.key);
                    break;
                case LogRecType::kUpdate:
                    data_[log->update_data.new_key] = log-
>update_data.new_val;
                    break;
                case LogRecType::kBegin:
                    active_txns_[log->txn_id_] = log->lsn_;
                    break;
                case LogRecType::kCommit:
                    active_txns_.erase(log->txn_id_);
                    break;
                case LogRecType::kAbort:{
                    lsn_t prev_lsn = log->prev_lsn_;
                    LogRecPtr prev_log = log_recs_[prev_lsn];
                    while (prev_log->type_ != LogRecType::kBegin) {
                        if (prev_log->type_ == LogRecType::kInsert)
                            data_.erase(prev_log->kv_data.key);
                        else if (prev_log->type_ == LogRecType::kDelete)
                            data_[prev_log->kv_data.key] = prev_log-
>kv_data.val;
                        else if (prev_log->type_ == LogRecType::kUpdate)
                            data_[prev_log->update_data.old_key] =
prev_log->update_data.old_val;
                        prev_lsn = prev_log->prev_lsn_;
                        prev_log = log_recs_[prev_lsn];
                      }
                    active_txns_.erase(log->txn_id_);
                    break;
                    }
            }
            persist_lsn_ = log->lsn_;
```

```
        }
    }
```

UndoPhase

□□RedoPhase□□□Undo list(active_txns_),□□□□□□□□□□□□□□□□□□,□Begin□□□Abort,□□□AppendLogRec□□□□

```
    void UndoPhase() {
        for(lsn_t undo_lsn_ = persist_lsn_; !active_txns_.empty();
undo_lsn_--)
        {
            LogRecPtr log = log_recs_[undo_lsn_];
            if(active_txns_.find(log->txn_id_) != active_txns_.end())
            {
                switch(log->type_) {
                    case LogRecType::kInsert:
                        data_.erase(log->kv_data.key);
                        AppendLogRec(CreateDeleteLog(log->txn_id_, log-
>kv_data.key, log->kv_data.val));
                        break;
                    case LogRecType::kDelete:
                        data_[log->kv_data.key] = log->kv_data.val;
                        AppendLogRec(CreateInsertLog(log->txn_id_, log-
>kv_data.key, log->kv_data.val));
                        break;
                    case LogRecType::kUpdate:
                        data_[log->update_data.old_key] = log-
>update_data.old_val;
                        AppendLogRec(CreateUpdateLog(
                            log->txn_id_,
                            log->update_data.new_key,
                            log->update_data.new_val,
                            log->update_data.old_key,
                            log->update_data.old_val
                        ));
                        break;
                    case LogRecType::kBegin:
                        AppendLogRec(CreateAbortLog(log->txn_id_));
                        active_txns_.erase(log->txn_id_);
                        break;
                    case LogRecType::kCommit:
                        break;
                    case LogRecType::kAbort:
                        break;
                    case LogRecType::kInvalid:
                        break;
                    default:
                        break;
                }
            }
        }
    }
```

3□ □□□

□□□□□□□□□□□□□□□□□□Recovery Manager□□ Recovery Manager□□□□□□□□□□□□CheckPoint□□□□□□□□□□□

1.checkpoint□□□□□serialize□deserialize,□□LogManager□□CheckPoint□□

```
struct CheckPoint {
    lsn_t checkpoint_lsn_{INVALID_LSN};
    ATT active_txns_{}; // txn_id -> last_lsn
    KvDatabase persist_data_{};

    void SerializeTo(char *buf) const {
        // checkpoint_lsn_
        MACH_WRITE_TO(lsn_t, buf, checkpoint_lsn_);
        buf += sizeof(lsn_t);

        // active_txns_□□□
        uint32_t active_txns_size = static_cast<uint32_t>
(active_txns_.size());
        MACH_WRITE_UINT32(buf, active_txns_size);
        buf += 4;

        // active_txns_□□
        for (const auto& [txn_id, last_lsn] : active_txns_) {
            MACH_WRITE_TO(txn_id_t, buf, txn_id);
            buf += sizeof(txn_id_t);
            MACH_WRITE_TO(lsn_t, buf, last_lsn);
            buf += sizeof(lsn_t);
        }

        // persist_data_□□□
        uint32_t data_size = static_cast<uint32_t>(persist_data_.size());
        MACH_WRITE_UINT32(buf, data_size);
        buf += 4;

        // persist_data_□□
        for (const auto& [key, val] : persist_data_) {
            key.SerializeTo(buf);
            buf += key.GetSerializedSize();
            val.SerializeTo(buf);
            buf += val.GetSerializedSize();
        }
    }

    static CheckPoint DeserializeFrom(char *buf) {
        CheckPoint checkpoint;

        // □□checkpoint_lsn_
        checkpoint.checkpoint_lsn_ = MACH_READ_FROM(lsn_t, buf);
        buf += sizeof(lsn_t);
```

```
        // □□active_txns_□□□
        uint32_t active_txns_size = MACH_READ_UINT32(buf);
        buf += 4;

        // □□active_txns_□□
        for (uint32_t i = 0; i < active_txns_size; ++i) {
            txn_id_t txn_id = MACH_READ_FROM(txn_id_t, buf);
            buf += sizeof(txn_id_t);
            lsn_t last_lsn = MACH_READ_FROM(lsn_t, buf);
            buf += sizeof(lsn_t);
            checkpoint.active_txns_[txn_id] = last_lsn;
        }

        // □□persist_data_□□□
        uint32_t data_size = MACH_READ_UINT32(buf);
        buf += 4;

        // □□persist_data_□□
        for (uint32_t i = 0; i < data_size; ++i) {
            KeyType key = KeyType::DeserializeFrom(buf);
            buf += key.GetSerializedSize();
            ValType val = ValType::DeserializeFrom(buf);
            buf += val.GetSerializedSize();
            checkpoint.persist_data_.emplace(std::move(key),
std::move(val));
        }
        return checkpoint;
    }

    size_t GetSerializedSize() const {
        size_t size = sizeof(lsn_t) + 4;

        // □□□□□□□□□
        size += active_txns_.size() * (sizeof(txn_id_t) + sizeof(lsn_t));

        // □□□□□□□□
        size += 4; // data_size
        for (const auto& [key, val] : persist_data_) {
            size += key.GetSerializedSize() + val.GetSerializedSize();
        }
        return size;
    }
};

// □RecoveryManager□□□CreateNewCheckpoint()□□
lsn_t CreateNewCheckpoint() {
    // □□□□□□□□LSN
    lsn_t current_lsn = log_manager_->GetNextLSN();

    // □□□□Checkpoint
    CheckPoint checkpoint;
    checkpoint.checkpoint_lsn_ = current_lsn;
```

```
    // □□□□□□□□□□□□□□LSN
    auto active_txns = txn_manager_->GetActiveTransactions();
    for (auto txn : active_txns) {
        checkpoint.AddActiveTxn(txn->GetTransactionId(), txn-
>GetLastLSN());
    }

    // □□□□□□□□□□□□

    // □□□□□□□□□□
    size_t serialized_size = checkpoint.GetSerializedSize();
    char *buf = new char[serialized_size];
    checkpoint.SerializeTo(buf);
    disk_manager_->Write(buf, serialized_size);
    delete[] buf;

    return current_lsn;
}
```

2.LogManager□log buffer□□□timeout□□□□□□□□□□□□□□□□log buffer

```
public:
    void AppendLog(LogRecPtr log_rec) {
        std::lock_guard<std::mutex> lock(latch_);
        log_buffer_.push_back(log_rec);
        if (log_buffer_.size() >= log_buffer_size_ || timeout()) {
            FlushLogBuffer();
        }
    }

    void FlushLogBuffer() {
        // □log_buffer_□□□□□□□□□
        disk_manager.write(log_buffer_);
        log_buffer_.clear();
        // □□□□Checkpoint□□□□□□□□□
        recovery_manager.CreateNewCheckpoint();
    }

private:
    RecoveryManager recovery_manager;
    std::vector<LogRecPtr> log_buffer_;
    size_t log_buffer_size_;
    std::mutex latch_;
    DiskManager disk_manager;
    lsn_t persist_lsn;
};
```

3.□Executor□□□□□□□□□□□□□log buffer(□□□□□,WAL)

4.□□□□□□□□Executor□□□□Begin,Commit□Abort

Begin□□□□□□□□□□□□□□□□□□□□□□□□txn_manager□□□txn_manager□□□□

```
dberr_t ExecuteEngine::ExecuteTrxBegin(pSyntaxNode ast, ExecuteContext
*context) {
#ifdef ENABLE_EXECUTE_DEBUG
  LOG(INFO) << "ExecuteTrxBegin" << std::endl;
#endif
  // □□□□□□□□□□□□□□
  if (context->GetTransaction() != nullptr) {
    LOG(ERROR) << "Transaction already in progress" << endl;
    return DB_FAILED;
  }

  // □□□□□□
  txn_id_t txn_id = next_txn_id_++;
  Transaction *txn = new Transaction(txn_id);
  context->SetTransaction(txn);

  // □□□□□□□□□
  LogRecPtr begin_log = log_manager_->CreateBeginLog(txn_id);
  log_manager_->AppendLog(begin_log);

  // □□□□□□□□□□□
  txn_manager_->Begin(txn);

  LOG(INFO) << "Transaction " << txn_id << " started successfully" << endl;
  return DB_SUCCESS;
}
```

Commit□□□□□□□□□□□□□□□□lock_manager□□□□□txn_manager□□□□

```
dberr_t ExecuteEngine::ExecuteTrxCommit(pSyntaxNode ast, ExecuteContext
*context) {
#ifdef ENABLE_EXECUTE_DEBUG
  LOG(INFO) << "ExecuteTrxCommit" << std::endl;
#endif
  // □□□□□□□
  Transaction *txn = context->GetTransaction();
  if (txn == nullptr) {
    LOG(ERROR) << "No transaction to commit" << endl;
    return DB_FAILED;
  }

  // □□□□□□□□□
  LogRecPtr commit_log = log_manager_->CreateCommitLog(txn-
>GetTransactionId());
  log_manager_->AppendLog(commit_log);

  // □□□□□□□□□□□□
  log_manager_->FlushLogBuffer();

  // □□□□□□
```

```
  lock_manager_->UnlockAll(txn);

  // □□□□
  txn_manager_->Commit(txn);

  // □□□□□
  context->SetTransaction(nullptr);
  delete txn;

  LOG(INFO) << "Transaction " << txn->GetTransactionId() << " committed
successfully" << endl;
  return DB_SUCCESS;
}
```

Abort□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
dberr_t ExecuteEngine::ExecuteTrxRollback(pSyntaxNode ast, ExecuteContext
*context) {
#ifdef ENABLE_EXECUTE_DEBUG
  LOG(INFO) << "ExecuteTrxRollback" << std::endl;
#endif
  // □□□□□□
  Transaction *txn = context->GetTransaction();
  if (txn == nullptr) {
    LOG(ERROR) << "No transaction to rollback" << endl;
    return DB_FAILED;
  }

  // □□□□□□□□
  LogRecPtr abort_log = log_manager_->CreateAbortLog(txn-
>GetTransactionId());
  log_manager_->AppendLog(abort_log);

  // □□□□□□
  txn_manager_->Abort(txn);

  // □□□□□
  lock_manager_->UnlockAll(txn);

  // □□□□□
  context->SetTransaction(nullptr);
  delete txn;

  LOG(INFO) << "Transaction " << txn->GetTransactionId() << " rolled back
successfully" << endl;
  return DB_SUCCESS;
}
```

5.□quit□□□□□□□□□checkpoint□□□

```
```

```
dberr_t ExecuteEngine::ExecuteQuit(pSyntaxNode ast, ExecuteContext
*context) {
#ifdef ENABLE_EXECUTE_DEBUG
  LOG(INFO) << "ExecuteQuit" << std::endl;
#endif
  // □□□□□□□□□□
  if (context->GetTransaction() != nullptr) {
    LOG(WARNING) << "Active transaction found during quit, rolling back..."
<< endl;
    ExecuteTrxRollback(nullptr, context);  // □□□□□□
  }

  // □□□□□□□□□
  log_manager_->FlushLogBuffer();

  // □□□□checkpoint
  recovery_manager_->CreateNewCheckpoint();

  // □□checkpoint□□□□□□
  disk_manager_->Sync();

  LOG(INFO) << "Database shutdown." << endl;
  return DB_SUCCESS;
}
```

6.□□□□□□□□□□□□□□□Catalog□□□□□□□□□□□□□□□□□checkpoint□□redo□undo□□□□□□□□□□checkpoint

```
CatalogManager::CatalogManager(BufferPoolManager *buffer_pool_manager,
LockManager *lock_manager,LogManager *log_manager, bool init)
    : buffer_pool_manager_(buffer_pool_manager),
lock_manager_(lock_manager), log_manager_(log_manager) {

    // ...
    if(init == true)
        // ...
    else {
        //load existing metadata,table,index
        //...
        RecoveryManager recovery_manager = log_manager-
>GetRecoveryManager();
        recovery_manager.Init(last_checkpoint);
        recovery_manager.RedoPhase();
        recovery_manager.UndoPhase();
        last_checkpoint = recovery_manager.CreateNewCheckpoint();
        log_manager->FlushLogBuffer();
    }
}
```

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□redo□undo□□log buffer□□□□□□□□□□□□□□□□□□□□

□□ □□□□□□□

- □□minisql_test□

```
[----------] 3 tests from CatalogTest
[ RUN      ] CatalogTest.CatalogMetaTest
[       OK ] CatalogTest.CatalogMetaTest (1 ms)
[ RUN      ] CatalogTest.CatalogTableTest
[       OK ] CatalogTest.CatalogTableTest (290 ms)
[ RUN      ] CatalogTest.CatalogIndexTest
[       OK ] CatalogTest.CatalogIndexTest (297 ms)
[----------] 3 tests from CatalogTest (589 ms total)
```

```
[----------] 1 test from RecoveryManagerTest
[ RUN      ] RecoveryManagerTest.RecoveryTest
[       OK ] RecoveryManagerTest.RecoveryTest (0 ms)
[----------] 1 test from RecoveryManagerTest (0 ms total)
```

- □□main□□□□□□□□□□□□□□□□□□□□