



沈阳工业大学
SHENYANG UNIVERSITY OF TECHNOLOGY

数据库管理系统 设计赛

队 伍：DataDance

答辩人：吴奕民

2024.8.19




目录

CONTENTS

01. 初赛——功能实现

02. 决赛——性能优化



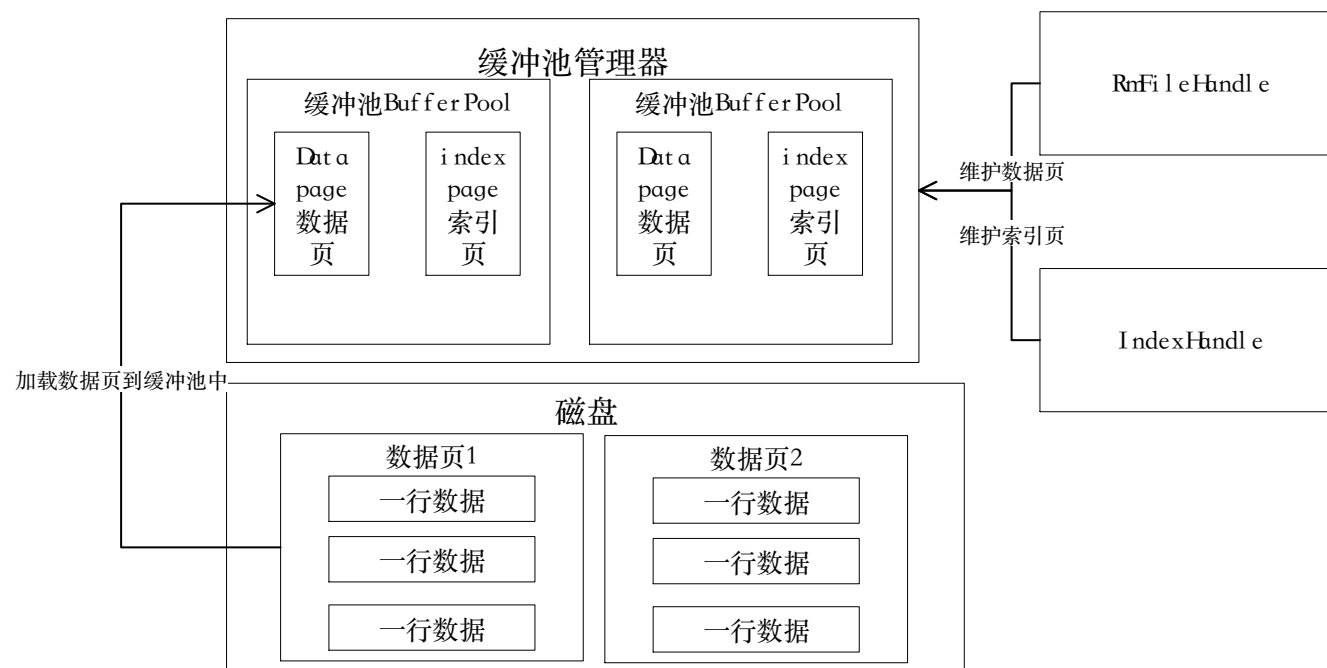
01 初赛——功能实现

- 缓冲池管理器
- B+树索引
- 基础算子
- 事务控制与日志恢复

缓冲池管理器

• 实现:

- 磁盘管理器负责读写页面
- 缓冲池管理器负责页面缓存到内存和淘汰页面
- RmFileHandle 和 IndexHandle 负责修改数据



• 优化

- 置换算法: LRU \rightarrow CLOCK
- 分片: 改进哈希算法, 实现16个缓冲池负载均衡, 减少大锁争用

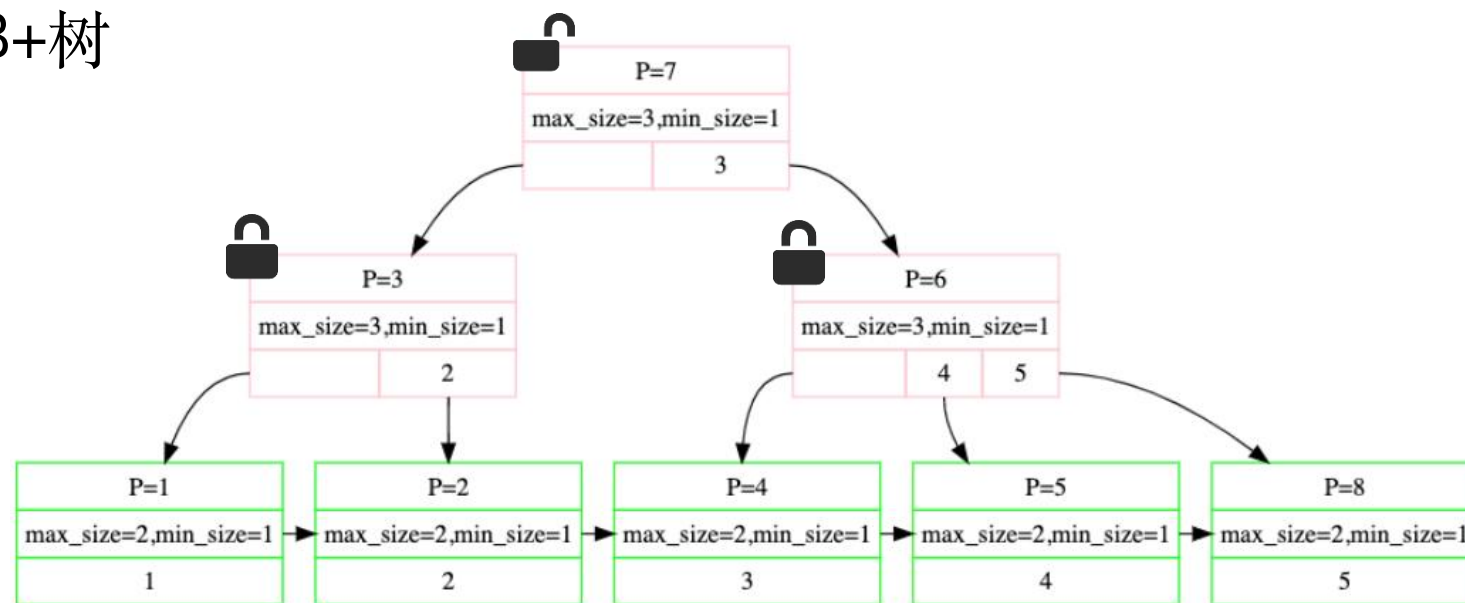
```
struct hash<PageId> {  
    size_t operator()(const PageId &obj) const {  
        // return (obj.fd << 16) | obj.page_no;  
        std::size_t h1 = std::hash<int>{}(obj.fd);  
        std::size_t h2 = std::hash<page_id_t>{}(obj.page_no);  
        return (h1 << 1) ^ (h2);  
    }  
};  
  
return hasher_(page_id) % BUFFER_POOL_INSTANCES;
```

• 实现

- 增、删、改、查接口来维护B+树
- 支持索引最左匹配规则
- 使用智能指针解决内存泄漏

• 并发

- 通过蟹形协议支持并发
- 支持后续间隙锁的并发控制



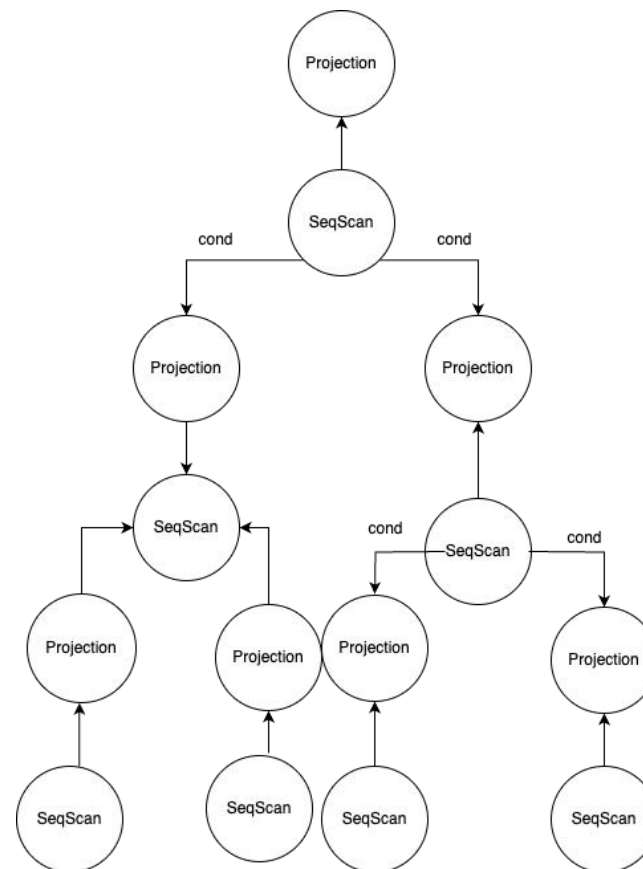
B+树的并发

• 实现

- Insert、Delete、Update、SeqScan 和 IndexScan 算子
- 连表算子：
 - 查询计划生成阶段
 - 根据表上是否有索引，是否有 sort 条件
 - 选择 Sort Merge 或者 Nested Loop 算子
- 聚合算子：
 - Group By 条件作为哈希表的键
 - 聚合值作为哈希表的值
 - Having 条件进一步过滤

• 不相关子查询

- 在 Condition 类支持查询计划，执行算子
- 在 Analyze 阶段递归 do_analyze



• 间隙锁

- PredicateManager 类：管理SQL谓词条件上的谓词
- Gap 类：基于谓词条件的间隙锁，提供间隙相交判定函数

• 两阶段锁

- 确保冲突调度的可串行化
- 保证事务的隔离性。

• Wait-Die


- 避免循环等待的发生
- 通过定义事务的优先级规则，确保系统运行时不会出现死锁情况。

• 并发控制

- 通过使用锁管理器控制对数据的并发访问
- 保证多个事务同时执行时的数据一致性和完整性。

• 日志恢复

- 通过 WAL 机制在写前生成日志
- 在事务提交或终止时刷盘
- 在接收到创建静态检查点命令时直接落盘，将缓冲池所有脏页刷盘
- 在系统启动时先 RedoLog 再 UndoLog



02 决赛——性能测试

- 分析
- 查询计划生成阶段优化
- 间隙锁的完善与事务控制优化
- 解析层优化
- 磁盘IO与STL容器优化
- 总结

- **W = 50**

- 数据量巨大，确保系统资源使用结束后正确释放

- **事务**

- 回滚会重做，new order 事务数量确定

- **tpmC**

- new orders / running time

- **优化目标：减少运行耗时**

- **运行耗时**

- 事务回滚重做的开销，降低事务冲突率
 - SQL执行过程的开销，优化查询计划
 - 磁盘 IO 开销，减少磁盘读写
 - 一些不必要的循环拷贝开销，冗余的函数调用

- **初分：20 tpmC/min**

- 逻辑优化

- 分析31条决赛示例SQL:

```
select min(max(asec on o)) as min_new_orders where  
w_id=1 and c_w_id=:w_id and c_d_id=2 and c_id=3;  
new orders and where no_id=:id order by new orders asc/desc limit 1;  
where w_id=1 and c_w_id=:w_id and c_d_id=2 and c_id=3;  
update order_line set ol_delivery_d=:datetime where  
ol_o_id=:no_o_id and ol_d_id=:d_id and ol_w_id=:w_id;
```

索引

修改列不在索引上，直接在行上更新，大表维护索引开销极大

- 结果: 3000 tpmC/min

- **完善**
 - 初赛测试较弱，在多线程大数据情况下依然锁不住数据
 - 解决：RmFileHandle支持并发
- **优化**
 - 观察 new order 事务涉及 5 到 10条 插入，如果频繁回滚开销极大
 - 解决：设置动态优先级，感知到当前事务进行了大量的插入则发生冲突时让其他更轻量的事务回滚。
- **结果：** 30000 tpmC/min

- **问题:**
 - 观察发现解析层占用了大量的时间，而 RucBase 框架并没有实现解析器的多线程支持，实际上16个线程共用一个SQL解析器。
- **解决:**
 - 开启Flex的可重入支持，修改 ast:TreeNode 为 thread local 变量。
- **结果:**

pass transaction_test/abort_test.sql.

pass transaction_test/commit_index_test.sql.

pass transaction_test/abort_index_test.sql.

You have passed all functional tests.

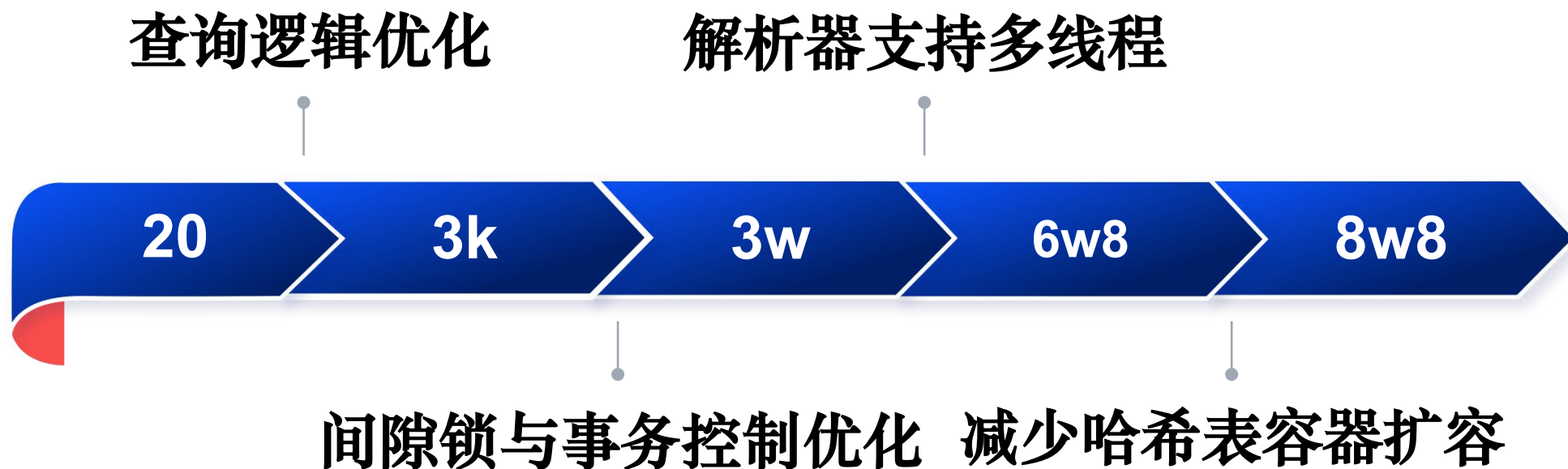
In performance test, run new order txn: 6964, running time: 6056950.0μs, tpmC: 68985.215331txns/min

- **问题:**
 - 观察发现日志在写盘、缓冲池读盘有较大开销，且页表因为扩容需要 rehash 产生了不少的开销，其他容器也是。
- **解决:**
 - 支持日志后台线程每隔 1s 刷盘
 - PAGE SIZE 设置为 16K，TPCC运行时基本没有读盘
 - 哈希表等容器在使用前预分配一定空间，减少扩容开销
- **结果:**

```
pass transaction_test/commit_test.sql.  
pass transaction_test/abort_test.sql.  
pass transaction_test/commit_index_test.sql.  
pass transaction_test/abort_index_test.sql.  
You have passed all functional tests.
```

```
In performance test, run new order txn: 6982, running time: 4751303.0μs, tpmC: 88169.497925txns/min
```

tmpC 时间线



谢谢！
敬请评价指正！

