学号	姓名	邮箱	完成题目
205220025	郑凯琳	205220025@smail.nju.edu.cn	t1

*由于时间问题，只尽力完成了基于理解的代码实现部分…… 来不及研究测试代码的部分…… 所以并不知道正确性…… T^T

（1）execution/executor_insert.cpp

实现细节：

Next：

1. **记录插入：** 遍历 inserts_ 中的每一条记录，调用 tbl_->InsertRecord(*record) 将记录插入目标表。
2. **索引更新：** 遍历索引列表 indexes_，为每一个索引执行 index->InsertRecord(*record)，将记录插入到相关索引中。
3. **更新插入计数：** 每插入一条记录，count 增加。

（2）execution/executor_seqscan.cpp

实现细节：

Init：设置初始的扫描状态，准备开始顺序扫描。

1. 初始化记录标识符： 调用 tab_->GetFirstRID() 获取数据表中的第一个记录标识符（RID）。rid_ 用于标识当前扫描的记录位置。
2. **获取首个记录：** 如果 rid_ 不等于 INVALID_RID，则通过 tab_->GetRecord(rid_) 获取对应的记录，将其存储到 record_ 中。否则，record_ 为 nullptr，表示没有可用记录。

Next：获取表中的下一条记录。

1. **检查当前记录：** 如果当前记录有效（即 record_ 不为 nullptr），则尝试获取下一条记录。
2. **更新 RID 和记录：** 调用 tab_->GetNextRID(rid_) 获取下一个记录的 rid_。然后，根据新的 rid_，通过 tab_->GetRecord(rid_) 获取新的记录。如果 rid_ 为 INVALID_RID，则表示扫描结束，record_ 被设为 nullptr。

IsEnd：检查当前扫描是否已经结束。

- 如果当前没有记录（即 record_ == nullptr），表示扫描已经结束，返回 true。
- 否则，返回 false，表示还有记录待扫描。

（3）execution/executor_limit.cpp

实现细节：

Init：

- 调用子执行器 child_ 的 Init() 方法，初始化子执行器。
- 将 count_ 设置为 0，表示尚未输出任何记录。
- 将 record_ 设置为 nullptr，等待获取子执行器的第一个记录。

若子执行器已结束（即 child_->IsEnd() 返回 true），则直接返回，不进行任何后续操作。

如果子执行器没有结束，获取子执行器的第一个记录并更新 count_，使其值为 1，表示已输出一条记录。

Next：

- 如果已经输出的记录数 count_ 大于或等于限制 limit_，或者子执行器已结束，则直接返回，不再执行后续操作。
- 否则，调用子执行器的 Next() 方法，获取下一条记录。
- 如果子执行器已经结束，则返回。
- 如果子执行器没有结束，则获取当前记录并更新 count_。

IsEnd：检查执行器是否已结束。

若 count_ 大于或等于 limit_，或子执行器已经结束，返回 true，表示查询已结束；否则，返回 false。

（4）execution/executor_projection.cpp

实现细节：

Init：

- 调用子执行器 child_ 的 Init() 方法，初始化子执行器。
- 将 count_ 设置为 0，表示尚未输出任何记录。
- 将 record_ 初始化为 nullptr，表示当前没有可用的记录。

若子执行器已结束（即 child_->IsEnd() 返回 true），则直接返回，不进行任何后续操作。

如果子执行器没有结束且当前记录不为空（即 child_->GetRecord() != nullptr），则根据子执行器的记录创建一个新的投影记录。新的投影记录通过 std::make_unique<Record>(out_schema_.get(), *child_->GetRecord()) 创建，并存储在 record_ 中。这样，输出的记录将遵循指定的投影模式。

Next：

- 每次调用 Next() 时，首先将 record_ 设为 nullptr，表示当前记录为空。
- 然后调用子执行器的 Next() 方法，获取下一条记录。
- 如果子执行器已经结束（即 child_->IsEnd() 返回 true），则直接返回，不进行后续操作。

如果子执行器没有结束且当前记录不为空（即 child_->GetRecord() != nullptr），则根据子执行器的记录创建一个新的投影记录，并将其存储在 record_ 中。

IsEnd：检查执行器是否已结束。

若子执行器已经结束（即 child_->IsEnd() 返回 true），或当前记录为空（即 record_ == nullptr），则返回 true，表示查询已结束；否则，返回 false。

（5）execution/executor_delete.cpp

实现细节：

Next：

1. 初始化子执行器（调用 child_->Init()）。
2. 遍历子执行器中的记录，获取每一条记录并删除：
    - 从表中删除当前记录。
    - 更新所有相关索引，删除索引中的记录。
    - 更新删除计数 count。

（6）execution/executor_update.cpp

实现细节：

Next：

1. 初始化子执行器（调用 child_->Init()）。
2. 遍历子执行器中的记录，获取每一条记录并执行更新：
    - 获取当前记录的模式（字段信息）。
    - 创建一个哈希表，将需要更新的字段与新值关联。
    - 对每个字段，若存在更新，则使用新值，否则保留原值。
    - 构造更新后的新记录，并更新表中的记录。
    - 更新所有相关索引中的记录。
    - 更新更新计数 count。

（7）execution/executor_filter.cpp

实现细节：

Init：初始化子执行器并开始过滤操作。

- 调用子执行器的 Init 函数进行初始化。
- 遍历子执行器的记录，直到找到符合过滤条件的记录，并将其存储在 record_ 中。如果找到了符合条件的记录，退出函数。

Next：获取下一条符合过滤条件的记录。

- 调用子执行器的 Next 函数，遍历子执行器中的记录。
- 若找到符合过滤条件的记录，则将其存储在 record_ 中，并返回。

- 若没有符合条件的记录，则将 record_ 设置为 nullptr。

IsEnd：判断是否已遍历完所有记录。

若 record_ 为 nullptr，则表示没有符合条件的记录，返回 true，否则返回 false。

（8）execution/executor_sort.cpp

实现细节：

SortBuffer：

Compare 函数用于比较两条记录的排序键，根据升序或降序排序规则返回比较结果。

Init：初始化排序执行器。

- 决定是否使用外部排序（归并排序），并根据子执行器的基数来判断。
- 如果使用内存排序（非归并排序），将子执行器中的记录读取到内存中，存储在 sort_buffer_ 中，并对记录进行排序。
- 排序后的记录存储在 sort_buffer_ 中，准备后续的访问。

Next：获取排序后的下一条记录。

- 如果使用归并排序，将通过归并算法加载下一条记录。
- 否则，依次从 sort_buffer_ 中返回记录，直到所有记录都被遍历完。

IsEnd：判断排序操作是否结束。

若当前记录为空或遍历完所有排序后的记录，则返回 true，表示排序结束。