Lab 3

1. 代码逻辑

Exercise 1

Exercise 1 要求我们实现 filter 和 join 两个操作，filter 即过滤掉不符合条件的元组，join 将两个 table 根据某一(些)字段进行拼接，即根据作为构造函数的一部分传入的 JoinPredicate 将来自其两个子节点的元组进行连接。

——fetchNext()函数实现了连接操作中获取下一个符合条件的元组。首先，检查左侧元组 left_tuple 是否为 null，并且子节点 child1 还有下一个元组。如果满足条件，则将下一个左侧元组赋值给 left_tuple。进入外层的 while 循环，只要左侧元组 left_tuple 不为 null，就会一直执行下去。在内层的 while 循环中，只要右侧子节点 child2 还有下一个元组，就会一直执行下去。该函数实现了对两个子节点的嵌套循环连接操作，并在满足过滤条件时创建新的连接元组返回，直到遍历完所有可能的连接组合。

Exercise 2

Exercise 2 要求我们实现 SQL 中的聚合操作，需要支持对单个字段进行分组，然后做聚合运算。且需要的实现的聚合运算有 COUNT、SUM、AVG、MIN、MAX。其中 count 即统计 tuple 的个数，SUM 即统计 tuple 相应字段的总和，AVG = SUM/COUNT，MIN 即计算 tuple 对应字段的最小值，MAX 即统计 tuple 对应字段的最大值。

——mergeTupleIntoGroup()函数根据给定的聚合操作类型，在聚合计算中将新元组合并到已有的结果中。首先，检查 gbfield 的值来确定是否进行分组。如果 gbfield 的值为 Aggregator.NO_GROUPING，表示不进行分组，将 groupfield 设置为 null；否则，根据元组 tup 和 gbfield 获取分组字段的值。接下来，代码根据聚合操作类型 aggreOp 进行不同的处理：(I)对于 MIN 操作，它比较旧的聚合值和当前值，并选择较小的值作为新的聚合值;(II)对于 MAX 操作，它比较旧的聚合值和当前值，并选择较大的值作为新的聚合值;(III)对于 COUNT 操作，它将旧的聚合值加 1 作为新的聚合值，用于计算元组的数量;(IV)对于 SUM 操作，它将旧的聚合值和当前值相加得到新的聚合值;(V)对于 AVG 操作，它将旧的聚合值和当前值相加得到新的聚合值，并更新计数器以便计算平均值。最后，将新的聚合值存储在 GroupByValue 映射表中，以分组字段的值作为键。在分组字段为 Aggregator.NO_GROUPING 时，结果将存储在单个元组中。

——StirngAggregator 只支持 COUNT 运算，否则会报错。

Exercise 3

Exercise 3 要求我们在 heap file 和 heappage 中实现元组的添加和删除操作。

——deleteTuple()函数实现了元组的删除。首先，获取要删除的元组的 RecordId。检查 recordId 是否为 null 或者其所属的页面与当前页面不匹配，如果不匹配则抛出 DbException 异常。然后获取元组在页面中的索引 tupleId。检查索引对应的槽位是否已被使用，如果未被使用则抛出 DbException 异常。最后将索引对应的槽位标记为未使用。

——insertTuple 函数实现了元组的添加。首先检查页面是否没有空槽位或者待插入元组的 TupleDesc 与页面的 td 不匹配，如果不匹配则抛出 DbException 异常。然后遍历页面的所有槽位，对于每个未使用的槽位，将其标记为已使用，为待插入的元组设置新的

RecordId（使用当前页面的 pid 和槽位索引），然后将元组存储到页面的对应槽位上，并结束循环。

Exercise 4

Exercise 4 要求我们实现了 Insert 和 Delete。对于实现 insert 和 delete 查询的计划，最顶层的运算符是一个特殊的 Insert 或 Delete 运算符，用于修改磁盘上的页面。这些运算符返回受影响的元组数量。这通过返回一个具有一个整数字段的单个元组来实现，该字段包含计数。这里的代码逻辑偏简单，不再赘述。

2. 实验心得

这次 lab 花了我 3 天时间，实现了多种运算操作，也是收获颇多。其中最大的困难是解决 lab1 lab2 遗留的不兼容性问题，比如在 query 实验时，我在 lab1 识别了.xxx 这样的辅助 field name 但是在 lab3 中会有异常，反倒识别不出真正的 filedname，这也花了我较久的时间进行 debug。