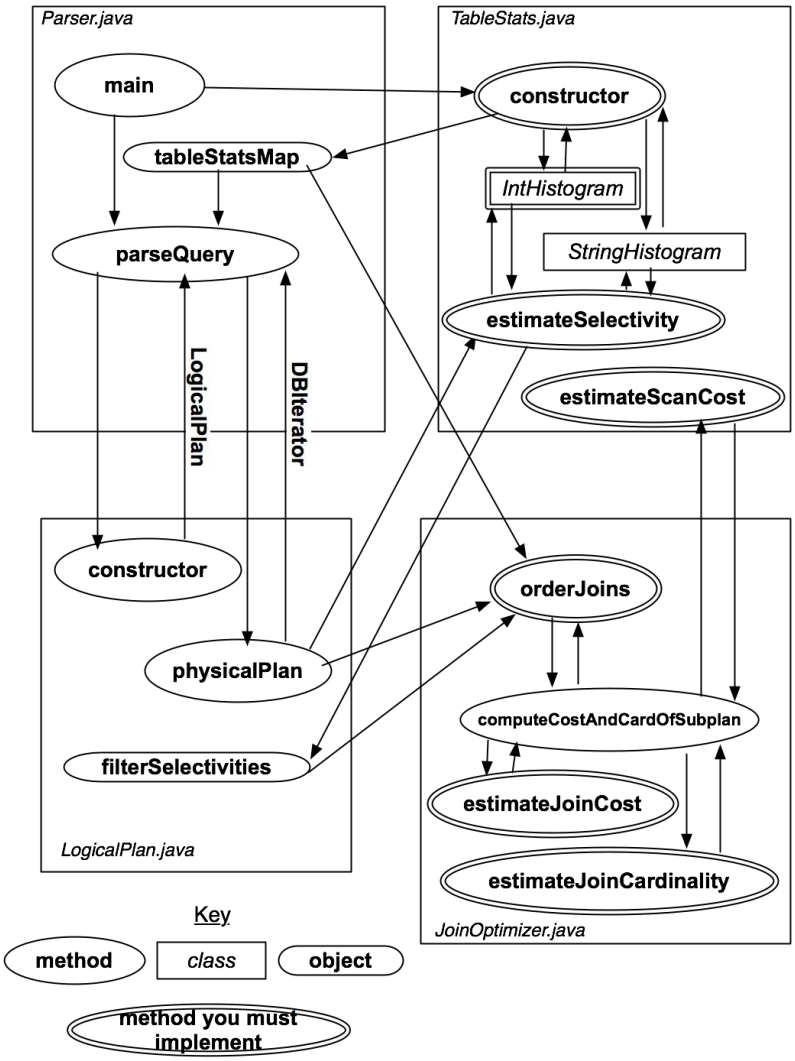
DataBase Lab4 Report

-521120910259
-郭培宇

1 Designing Part

基于 lab1 和 lab2、lab3 的数据库中组织方式、缓冲池规范、数据骨架、实现表修改（如插入和删除记录）、选择、连接和聚合等一系列数据库操作符等构建后，本次 lab4 主要需要我们实现查询优化器。

主要任务包括实施选择性估计框架和基于成本的优化器。



基本操作如下：

1.Parser.java 在启动时会构建一组表格统计信息（存储在 statsMap 容器中）。然后等待

输入查询，并调用该查询的 parseQuery 方法。

2.parseQuery 首先会构建一个逻辑计划（LogicalPlan）来表示解析后的查询。然后，parseQuery 在其构建的 LogicalPlan 实例上调用 physicalPlan 方法，physicalPlan 方法会返回一个 DBIterator 对象，可用于实际运行。

1.1 Exercise1

在练习一中我们需要某种方法来记录表格统计数据，以便进行选择性估算。原有代码已经提供了一个骨架类 IntHistogram，可以实现这一功能。因此我们需要使用上述基于桶的方法计算直方图。

我的想法是在原代码的基础上完善函数，比如定义一个 rectangle 的内部类，并加入 l,r,w,h 等变量，分别代表桶的左右边界，桶的宽度，桶的高度。加入传递给直方图的最小值、最大值以及直方图的桶数量等成员变量。具体来说补全了 addvalues 和 estimateSelectivity 等函数。使用内部类 Rectangle 来表示每个桶，通过 addvalues 来添加值，通过 estimateSelectivity 来估计选择性，并提供了掉使用的 tostring 等方法。最后顺利通过测试。

1.2 Exercise2

TableStats 类包含计算方法，这些方法可计算表中的图元数和页数，并估算谓词对表中字段的选择性。我们需要在 TableStats 中填写以下方法和类，我的具体方法是：

-执行 TableStats 构造函数：初始化如 tableid 等成员变量，用 try 结构获取对应表的 DBfile 扫描其元组以计算每个字段的最小值和最大值，根据最小值和最大值初始化直方图，再次扫描元组，将值添加到对应的直方图。

-实现 estimateSelectivity(int field, Predicate.Op op、字段常量）： 使用统计信息（如 IntHistogram 或 StringHistogram，取决于字段的类型），估计表中谓词字段 op 常量的选择性。

-实施 estimateScanCost()： 该方法估算顺序扫描文件的成本。成本为 costPerPageIO。可以假设没有寻道，且缓冲池中没有页面。此方法可以使用在构造函数中计算的成本或大小。

-实现 estimateTableCardinality(double 选择性因子）： 此方法会返回关系中的元组数量。的谓词时，该方法将返回关系中的图元数。该方法可以使用在构造函数中计算的成本或大小。

1.3 Exercise3+exercise4

因为 3 和 4 都在 joinOptimizer.java 文件中，因此我将其合并，在练习 3 中，我们将编写用于估算连接的选择性和成本的方法。具体来说

-实现 estimateJoinCost(LogicalJoinNode j, int card1, int card2, doublecost1, double cost2）：此方法估计 j 的成本。

-实现 estimateJoinCardinality（LogicalJoinNode j，int card1, int card2, 布尔值 t1pkey, 布尔值 t2pkey）： 此方法估计连接 j 输出的元组数，条件是左侧输入是大小为 card1 的数据，右侧输入是大小为 card2 的数据，以及 t1pkey 和 t2pkey 标志，这两个标志分别表示左侧和右侧字段是否唯一（分别为右（分别）字段是否唯一（主键）。

在练习 4 中，执行方法：

Vector<LogicalJoinNode> orderJoins(HashMap<String, TableStats> stats、 HashMap<String, Double> filterSelectivities、布尔解释）

此方法应在 joins 类成员上运行、返回一个新矢量，该矢量指定了连接的顺序。该向量中的第 0 项表示最左边的连接、最底层的连接。大致上，循环处理子集大小、子集和子集的子计划，调用 computeCostAndCardOfSubplan，并建立一个 PlanCache 对象，存储执行每个子集连接的最小成本的计划缓存对象。

最后，我补全后的这个 JoinOptimizer 类主要是为了优化 SQL 查询中的连接操作。它通过估算连接成本和基数来选择最优的连接顺序和连接方式。核心方法包括计算连接成本的 estimateJoinCost，计算连接基数的 estimateJoinCardinality，枚举子集的 enumerateSubsets，以及最终确定连接顺序的 orderJoins。辅助方法如 computeCostAndCardOfSubplan 则用于递归地计算和缓存子计划的成本和基数。

2 实验心得

此次 lab4 基于 lab1，2，3 的基础上继续给 SimpleDB 实现一套查询优化器。本次 lab 的难点可能是理解直方图相关以及与查询 cost 相关的背景知识，以及进一步熟悉 java 对于异常处理的代码规范及处理方法，并且因为此次实验基于整个 src 文件夹中各个文件的代码，需要我们对 lab1 和 lab2 中很多步骤非常熟悉，因此在思路构建方面就需要大量的复习之前代码或者看新代码的时间。同时此次编写操作符的一系列文件 debug 难度和复杂度也相较于之前有增加。

但是整体做完感觉收获还是非常多，我认为我对这 simpleDB 的底层架构到流程实现的认知又进一步清晰了。