实验说明文件

实验描述

在本实验中，需要为 SimpleDB 实现一个查询优化器，重点在于选择性估计和连接排序的实现。实验的主要任务包括：

1. 实现 `IntHistogram` 类，用于记录和估计整数类型字段的选择性。
2. 实现 `TableStats` 类，用于统计表的基本信息并提供选择性估计方法。
3. 实现 `JoinOptimizer` 类，用于确定连接的最佳顺序。

设计决策

1. **选择性估计方法：**
   - 我选择使用直方图（Histogram）来对字段的值分布进行建模。具体地，将字段值划分为固定数量的桶（buckets），并记录每个桶中的值数量。
   - 在实现 `IntHistogram` 类时，我定义了一个包含指定数量桶的数组，每个桶记录落在该范围内的值的数量。
   - 估计选择性时，根据谓词操作符（如 EQUALS、GREATER_THAN 等）和给定的值，计算该值在直方图中的分布情况，并返回相应的选择性。
2. **连接排序：**
   - 我使用基于成本的优化方法来确定连接的最佳顺序，类似 Selinger 优化器。该方法通过计算所有可能的连接顺序的成本，选择成本最低的顺序。
   - 在 `JoinOptimizer` 类中，我实现了一个递归方法来计算每个子连接计划的成本，并缓存这些中间结果以提高效率。
   - 成本计算基于嵌套循环连接（Nested Loops Join），包括 I/O 成本和 CPU 成本。

代码实现过程

1. **实现 `IntHistogram` 类：**
   - 定义直方图的基本结构，包括桶的数量、每个桶的宽度、最小值和最大值。
   - 实现 `addValue` 方法，用于将值添加到直方图中合适的桶。
   - 实现 `estimateSelectivity` 方法，根据给定的谓词操作符和值，估计相应的选择性。
   - 实现 `avgSelectivity` 方法，返回直方图的平均选择性。
2. **实现 `TableStats` 类：**
   - 初始化表统计信息，包括表的基本描述、页数和元组数。
   - 创建并初始化每个字段的直方图。
   - 扫描表数据，填充直方图和计算基本统计信息。
   - 实现 `estimateScanCost` 方法，估计顺序扫描表的成本。
   - 实现 `estimateTableCardinality` 方法，根据选择性因子估计表中的元组数。
   - 实现 `estimateSelectivity` 方法，根据字段、谓词操作符和常量值，估计选择性。
3. **实现 `JoinOptimizer` 类：**
   - 实现 `estimateJoinCost` 方法，估计连接的成本，包括 I/O 成本和 CPU 成本。

- 实现 `estimateJoinCardinality` 方法，估计连接的基数，根据是否有主键进行判断。
- 实现 `orderJoins` 方法，使用递归计算每个子连接计划的成本，并缓存结果以提高效率。最终返回最佳的连接顺序。

API 更改

在本实验中，我没有对原有的 API 进行更改，只是在实现过程中添加了一些辅助方法和类来完成任务。

缺失或不完整的部分

代码实现基本完整，但由于时间有限，某些优化方法（如更高级的连接算法）未能实现。这些部分在未来的改进中可以进一步优化和完善。

实验时间和难点

- 实验总共花费了大约 15 小时，包括设计、编码和调试。
- 主要困难在于如何高效地计算连接的最佳顺序，尤其是在处理多个连接时，递归计算和缓存中间结果的实现较为复杂。
- 另一个难点在于选择性估计，如何准确建模字段值的分布，并在各种谓词操作符下进行选择性估计，需要细致的逻辑和大量测试。

通过本实验，我深入理解了查询优化的原理和实现方法，掌握了选择性估计和连接排序的基本技术，为进一步研究和优化数据库查询打下了良好的基础。