

**

数据库系统

课程试验报告

|  |  |  |
| --- | --- | --- |
| **试验题目： MiniOB数据库管理系统功能的补充与完善** | | |
|  | 学生姓名： | 丁海桐 |
|  | 学生学号： | 202226010304 |
|  | 专业班级： | 软件2203班 |
|  | 开课时间： | 2023-2024-2学期 |
|  | 试验日期： | 2024年5月7日 |

|  |
| --- |
| 1. **试验任务：** 2. **实现SELECT语句中的ORDER BY功能；** 3. **实现SELECT语句中的GROUP BY功能；** |
| 1. **试验准备情况（对照任务，实验之前给出你的预案）：**   SQL层理解  当用户在 miniob 中输入 SQL 语句时，该请求以字符串形式存储：   1. Parser 阶段：将 SQL 字符串进行词法解析（lex\_sql.l）和语法解析（yacc\_sql.y）， 最终转换为 ParsedSqlNode（parse\_defs.h）。 2. Resolver 阶段：将 ParsedSqlNode 转换为 Stmt（Statement），进行语义解析。 3. Transformer 和 Optimizer 阶段：将 Stmt 转换为 LogicalOperator，并在优化后输 出 PhysicalOperator。这一阶段负责执行查询优化操作。 4. 对于命令执行类型的 SQL 请求，系统会创建相应的 CommandExecutor。 5. 执行阶段 Executor：将 PhysicalOperator（物理执行计划）转换为 SqlResult（执行 结果），或者通过 SqlResult 输出 CommandExecutor 执行后的结果。 |
| 1. **试验过程记录（对照任务，对试验方案和结果进行记录和分析）：** 2. **实现SELECT中的GROUP BY功能** 3. **Parser模块**   **lex\_sql.l:**  增加ORDER 的token    **yacc\_sql.l:**        在该SQL语句的解析规则中，关于`GROUP BY`部分的处理旨在将用户在查询中定义的分组逻辑整合进抽象语法树（AST）。如果查询中包含了`GROUP BY`子句（通过变量`$7`表示），解析器首先确认这个子句存在，然后将与之关联的表达式列表（即用于分组的列或表达式）转移至当前解析节点的`selection.groupby\_exprs`成员中，并删除原列表以释放资源。为了确保表达式顺序与SQL语句中的顺序一致（因为内部可能进行了逆序操作以适应解析过程），它还会对这些分组表达式进行一次逆序操作。这样，解析后的结构就准确地反映了原始SQL中`GROUP BY`子句的意图和内容，为后续的查询优化和执行奠定了基础。    在SQL解析的这一部分，`opt\_group\_by`规则负责处理查询中可选的`GROUP BY`子句。如果存在`GROUP BY`子句，它会捕获跟随其后的表达式列表（用于分组的列或表达式），并将这些表达式存储在`$$`中。为了匹配解析流程的需求， initially会将这些表达式的顺序反转，但这一操作不影响最终结果的正确性，因为在后续处理中，如上述`select\_stmt`规则所示，可能会有额外的调整来确保表达式最终按原SQL中的顺序呈现。如果查询中没有`GROUP BY`子句，此规则简单地返回`nullptr`，表示分组操作未被应用。总之，这段代码确保了无论SQL查询中是否包含`GROUP BY`，解析器都能妥善处理并准备相应的分组信息以供后续处理阶段使用。  **parse\_def.h**     1. **Resolve模块**   在src\observer\sql\stmt\orderby\_stmt.cpp添加orderby的stmt文件    **groupby.h**    **groupby.cpp**    这段代码定义了一个GroupByStmt类，用于表示SQL语句中的GROUP BY操作。此类继承自Stmt基类，并实现了特定于分组操作的功能。它管理三类表达式：分组依据表达式（groupby\_fields\_）、聚合函数表达式（agg\_exprs\_）以及非聚合字段表达式（field\_exprs\_）。  通过提供的公有接口，可以获取或设置这些表达式的集合。例如，get\_groupby\_fields允许访问用于分组的表达式列表。此外，create静态方法用于实例化GroupByStmt对象，它接收数据库上下文、表信息、表达式列表（用于GROUP BY子句）以及聚合函数和字段表达式的集合，然后初始化新创建的GroupByStmt实例，设置好所有相关的表达式集合，以便执行分组操作和其他后续处理。简而言之，这个类及其方法构成了处理SQL查询中GROUP BY逻辑的核心部分。   1. **Optimizer模块**   src\observer\sql\operator添加逻辑算子和物理算子文件    **groupby\_logical\_operator.h**    **groupby\_logical\_operator.cpp**    这段代码定义了一个名为GroupByLogicalOperator的类，它是用于数据库查询处理逻辑中的一个组件，专门负责实现SQL语句中的GROUP BY操作逻辑。此类继承自LogicalOperator基类，并实现了与分组操作相关的功能。  在构造函数中，该类接收三个参数：分组依据的表达式列表（groupby\_fields）、聚合函数表达式列表（agg\_exprs）以及非聚合字段表达式列表（field\_exprs）。这些表达式通过右值引用（std::move）进行传递，表明它们的 ownership 被转移到GroupByLogicalOperator对象中，从而该对象负责管理这些表达式的生命周期。  该类提供了访问这些内部表达式列表的接口，如groupby\_fields()、agg\_exprs()和field\_exprs()方法，允许外部调用者查询或进一步处理这些表达式，这对于执行实际的分组操作至关重要。简而言之，GroupByLogicalOperator类封装了SQL查询中GROUP BY操作的逻辑处理，包括确定分组的依据、执行聚合计算以及处理分组后的输出字段，是数据库查询引擎中处理复杂分组查询逻辑的核心组件之一。  **groupby\_physical\_operator.h**    **groupby\_physical\_operator.cpp**          这段代码定义了一个GroupByPhysicalOperator类，它是数据库查询执行计划中的物理运算符，负责实现SQL查询中的GROUP BY操作。该类继承自PhysicalOperator，主要任务是在记录集合上执行分组和聚合计算。  在构造函数中，它接收分组依据表达式、聚合函数表达式和非聚合字段表达式作为输入，初始化内部状态。类中还维护了几个关键的布尔标志（如is\_first\_、is\_new\_group\_、is\_record\_eof\_）以及用于存储前一次分组键值得pre\_values\_向量，这些用于跟踪处理过程中的分组状态。  open方法用于初始化操作，确保子运算符（即提供数据流的运算符）被正确打开，并准备好进行分组操作。next方法是核心逻辑所在，它逐条处理子运算符提供的记录，判断每条记录是否属于新的分组，如果是，则执行聚合函数的初始化或完成操作（通过do\_aggregate\_first和do\_aggregate\_done），同时更新分组键的前值；如果不是新分组，则继续累计聚合结果（通过do\_aggregate）。当所有记录处理完毕或遇到文件结束时，它会标记结束状态并完成最后一次聚合。  close方法用于关闭子运算符，释放资源。current\_tuple方法返回当前处理完成的分组合集结果，即聚合后的元组。  总的来说，GroupByPhysicalOperator类实现了对数据流的分组处理逻辑，根据指定的分组表达式对数据进行划分，并针对每个分组应用聚合函数进行计算，生成聚合后的结果集，是数据库管理系统中处理分组查询请求的关键组件。  **（二）实现SELECT中的ORDER BY功能**   1. **Parser模块**   **lex\_sql.l:**  增加ORDER 的token    **yacc\_sql.l:**        以上定义了如何解析SQL中的ORDER BY子句。sort\_unit定义了排序单元，可以是一个关系属性（列名）、聚合函数表达式或一般函数表达式，后面可跟比较运算符（如ASC、DESC）来指定排序方向。opt\_order\_by是非终结符，表示查询中可选的排序部分，它后面跟着一个或多个通过,分隔的sort\_unit构成的sort\_list，用来收集所有的排序条件。如果存在ORDER BY子句，解析器会创建相应的OrderBySqlNode对象并填充相关信息，包括排序表达式和排序方向（升序或降序），然后这些信息会被整合进最终的SQL解析结果中，用于后续的查询执行阶段对数据进行排序。    将ORDER BY表达式从解析上下文中提取出来，整合到SQL查询的抽象语法树中，并调整这些表达式的顺序以匹配SQL语句中的原始顺序，最后清理临时使用的内存。这样的处理保证了即使在解析阶段元素被逆序处理，最终的查询执行计划也能准确反映SQL语句的意图。    这部分代码定义了如何解析SQL查询中的ORDER BY子句，包括排序表达式、排序方向（升序或降序），构建排序列表，并在最终处理阶段将排序列表逆序，以便于后续执行时能正确应用排序规则。  **parse\_def.h**  这段代码定义了SQL解析器中用于表示ORDER BY子句的结构OrderBySqlNode，它包含排序依据的表达式和排序方向（升序或降序），用于在查询结果中进行排序。       1. **Resolve模块**   在src\observer\sql\stmt\orderby\_stmt.cpp添加orderby的stmt文件    **orderby.h**    **orderby.cpp**    这些代码主要用于解析和构建SQL查询中的ORDER BY子句，支持对查询结果进行排序，并为后续的查询执行提供必要的排序信息和表达式。   1. **Optimizer模块**   src\observer\sql\operator添加逻辑算子和物理算子文件    **orderby\_logical\_operator.h**    **orderby\_logical\_operator.cpp**    这段代码定义了一个名为`OrderByLogicalOperator`的类，它是`LogicalOperator`的子类，专门用于处理SQL查询中的排序逻辑（ORDER BY子句）。此类接收两组数据成员：一是按排序要求组织的`OrderByUnit`对象的集合，用来指示排序的列和排序方式（升序或降序）；二是与排序操作相关的表达式集合，可能包含非聚合表达式和聚合表达式。构造函数通过移动语句接受这些参数，以高效地管理内存和资源。简而言之，这个类设计用于构建数据库查询执行计划中的排序步骤，确保查询结果按照指定的列和顺序进行排序。  **orderby\_physical\_operator.h**    **orderby\_physical\_operator.cpp**            该代码段实现了一个OrderByPhysicalOperator类，它是数据库查询处理中用于实现ORDER BY功能的物理操作算子。该类接收排序依据（OrderByUnit对象组成的向量）和表达式（用于计算SELECT字段的Expression对象组成的向量），并在打开（open方法）时通过其子操作算子获取所有数据行，然后根据指定的排序条件对这些数据行进行排序。  具体过程包括：  在open方法中，首先检查是否有且仅有一个子操作算子，并确保其成功打开。随后调用fetch\_and\_sort\_tables方法准备排序。  fetch\_and\_sort\_tables方法逐行读取子操作算子提供的数据，收集每行中用于排序的值和SELECT字段的值，存储到相应的数据结构中。之后，使用这些值创建一个配对的排序表，并根据ORDER BY子句中指定的排序类型（升序或降序）进行排序。  排序完成后，存储每行数据在原始数据集中的索引，以便后续按排序顺序检索。  next方法用于按排序顺序提供下一行数据，通过之前记录的排序索引来定位并设置当前元组（tuple\_）的值。  close方法负责关闭子操作算子，释放资源。  整个类的设计旨在作为SQL查询执行计划的一部分，确保查询结果能够按照用户在ORDER BY子句中指定的规则正确排序。 |
| 1. **试验完成情况：** 2. **ORDER BY验证**      1. **GROUP BY验证** |
| 1. **试验总结：（遇到的问题及解决措施，对试验的评价，感想和认识）**   1. 加深了对SQL高级查询的理解：通过本次实验，我不仅掌握了`ORDER BY`和`GROUP BY`这两个关键子句的基本用法，还理解了它们在数据处理中的核心作用。`ORDER BY`让我能够按照指定列对查询结果进行排序，这对于数据分析和报告生成尤为重要；而`GROUP BY`则使我能够将数据分组，进行聚合操作（如计数、求和等），这对于数据汇总分析是必不可少的技能。  2. 提高了数据处理效率：在实验过程中，我意识到合理利用`ORDER BY`和`GROUP BY`可以显著提升数据处理的效率。比如，通过先分组再排序的策略，可以在大数据集上更快地得到想要的结果，减少了不必要的计算量。这让我学会了在面对大规模数据时，如何优化查询语句以达到更高的执行效率。  3. 增强了数据分析思维：实践`GROUP BY`时，我不得不思考如何根据业务需求对数据进行合理的分组，这一过程锻炼了我的逻辑思维和数据分析能力。我开始更加注重从数据中提取有价值的信息，而不仅仅是执行查询操作，这对于提升我的问题解决能力和决策支持能力大有裨益。  4. 体验了错误调试的过程：在实现这些功能的过程中，我也遇到了一些预料之外的错误，比如由于对`GROUP BY`与聚合函数搭配使用的规则不熟悉导致的查询错误。通过查阅文档、尝试不同的解决方案，我最终解决了这些问题，这个过程不仅巩固了我的SQL知识，也提高了我的问题排查和解决能力。  总结而言，本次实验不仅增强了我的SQL技能，更重要的是培养了我在数据分析领域的实战能力和问题解决思路，为未来处理复杂数据挑战打下了坚实的基础。 |