课程项目：DuckDB 并行分析性能评测与瓶颈分析

1. 项目目标

本项目旨在让研究生深入探索和评测 DuckDB 这一现代 "进程内" 分析型数据库的性能特性。学生将通过设计和执行一系列系统性实验，重点测试 DuckDB 在**单机环境**下处理**中大规模数据集**时的**并行数据读取**和 ** 联机分析处理（OLAP）** 性能表现。

学习目标

通过完成本项目，学生应能够：

1. **掌握 DuckDB 核心技术**：熟练使用 DuckDB 的 SQL 方言、配置参数和编程接口
2. **理解并行查询原理**：深入理解并行查询处理的基本概念及其对系统性能的影响机制
3. **掌握性能测试方法**：学会设计科学的数据库性能测试实验，合理控制变量并准确收集数据
4. **培养数据分析能力**：能够分析和解释性能数据，识别系统瓶颈（I/O 瓶颈、CPU 瓶颈等），并理解不同数据格式（CSV vs. Parquet）对分析性能的影响

2. 推荐数据集

数据集规模应适中："大到足以让并行处理产生明显优势，但又小到可以在单机环境（例如配备 4-16 核 CPU、16-64GB 内存的普通工作站）上完成实验"。

选择一：纽约市出租车行程数据（强烈推荐）

数据集简介：纽约市出租车和豪华轿车委员会（TLC）发布的行程数据，是大数据和数据库基准测试领域的经典中大规模数据集。

数据格式：官方提供 Parquet 和 CSV 两种格式。本项目**必须**使用 Parquet 格式来测试其高效的列式读取性能，并建议与 CSV 格式进行对比分析。

数据规模：建议学生下载 **1 到 3 年**的数据（例如 2019-2021 年）。单个 Parquet 文件（按月划分）大小约 50MB-150MB，总数据量可达 10GB-50GB。

下载地址：TLC Trip Record Data

选择二：TPC-H 基准测试数据（备选）

数据集简介：TPC-H 是数据库分析性能的标准基准测试套件，广泛用于评估 OLAP 系统的性能。

数据获取：学生可以使用 DuckDB 内置的 `dbgen` 扩展自行生成测试数据：

```
INSTALL 'dbgen';
LOAD 'dbgen';
CALL dbgen(sf=10); -- 生成 Scale Factor（SF）= 10（约 10GB）的数据
```

数据规模：建议使用 `SF=10`（10GB）或 `SF=100`（100GB），具体取决于学生的机器性能。

3. 实验设计

实验设计是本项目的核心环节。学生需要围绕 **"并行度"** 这一核心变量来设计系统性实验。

A. 核心控制变量

学生在 DuckDB 中主要通过 `PRAGMA` 语句来控制执行参数：

1. **线程数控制**：
   - 设置方式：`PRAGMA threads=N;` 或 `SET threads=N;`
   - 测试范围：建议测试一系列值，例如 `N = [1, 2, 4, 8, 16, ...]`，最高不超过机器的物理核心数
   - **基线设置**：`N=1` 作为基线（Baseline），用于对比并行执行（`N>1`）带来的加速比
1. **数据格式对比**：
   - 实验组 1：直接查询多个 Parquet 文件（DuckDB 支持 `FROM 'data/yellow_tripdata_*.parquet'` 这样的 glob 模式，自动并行扫描所有匹配文件）
   - 实验组 2：将 Parquet 转为 CSV 格式，执行完全相同的查询进行对比
1. **数据规模变化**：

- 测试查询 1 个月、12 个月（1 年）、36 个月（3 年）的数据时，性能如何随数据量扩展

B. 待测量指标

1. **查询执行时间**：最主要的性能指标。建议多次运行（例如 5 次）取平均值，以减少系统抖动带来的误差
2. **CPU 利用率（高级）**：在查询执行期间，使用系统工具（如 htop, top, perf）观察 CPU 是否被充分利用
3. **内存使用情况（高级）**：观察 DuckDB 的内存占用变化，分析内存使用模式

C. 实验环境清理

为保证 I/O 测试的公平性（避免第二次读取时数据已在操作系统文件缓存中），在每次运行不同实验（特别是对比不同格式时）前，应**清理操作系统的页缓存**：

- Linux/macOS 系统：sync; echo 3 > /proc/sys/vm/drop_caches（Linux 系统需要 root 权限）
- 替代方案：如果无法清理缓存，应在报告中明确说明，并注意区分第一次运行（冷缓存）和后续运行（热缓存）的时间差异

4. 任务步骤

第一阶段：环境搭建与数据准备

1. **安装 DuckDB**：选择合适的客户端（Python 客户端 duckdb、命令行工具 CLI 或 Java/C++ 接口等）
2. **获取数据集**：下载或生成所选数据集（例如 NYC Taxi 2019-2021 年的 Parquet 文件）
3. **数据预处理**：（可选）准备等效的 CSV 版本数据集用于格式对比实验

第二阶段：分析型查询设计

学生必须设计 **3-5 个**具有代表性的分析型（OLAP）查询。这些查询应涵盖不同的操作类型，以全面测试 DuckDB 引擎的各个组件。

Q1：简单聚合查询（测试 I/O 和并行扫描性能）

查询目的：测试 DuckDB 的基础数据读取和简单聚合能力

```
SELECT
    count(*),
    avg(total_amount)
FROM 'data/yellow_tripdata_*.parquet';
```

Q2：带过滤和分组的聚合查询（测试 Filter + GroupBy 性能）

查询目的：测试过滤条件和分组聚合的并行处理能力

```
SELECT
    passenger_count,
    avg(trip_distance) AS avg_dist
FROM 'data/yellow_tripdata_*.parquet'
WHERE trip_distance > 0 AND total_amount > 0
GROUP BY passenger_count
ORDER BY avg_dist DESC;
```

Q3：复杂聚合查询（测试高基数 GroupBy 和排序性能）

查询目的：测试高基数分组和排序操作的性能，这通常是 OLAP 查询中的性能瓶颈

```
SELECT
    PULocationID,
    DOLocationID,
    count(*) AS trip_count
```

```
FROM 'data/yellow_tripdata_*.parquet'
GROUP BY PULocationID, DOLocationID
ORDER BY trip_count DESC
LIMIT 10;
```

Q4：时间序列分析查询（可选）

查询目的：测试时间相关函数和多层级聚合的性能

```
SELECT
    hour(tpep_pickup_datetime) AS pickup_hour,
    dayofweek(tpep_pickup_datetime) AS day_of_week,
    avg(tip_amount)
FROM 'data/yellow_tripdata_*.parquet'
GROUP BY ALL
ORDER BY ALL;
```

第三阶段：实验执行

1. **编写自动化脚本**：使用 Python 或 Shell 脚本自动化实验流程，确保实验的可重复性
2. **实验一：并行度扩展性测试**
    - 固定条件：数据量（例如全部 3 年数据）和格式（Parquet）
    - 实验步骤：循环执行 Q1、Q2、Q3 查询
    - 变量控制：在每次循环开始前，设置不同的线程数：`SET threads = 1;`, `SET threads = 2;`, `SET threads = 4;`, ...
    - 数据记录：记录每个查询在不同线程数下的执行时间
1. **实验二：数据格式对比测试**
    - 固定条件：线程数（例如机器的最大物理核心数）
    - 实验步骤：分别对`*.parquet`文件和`*.csv`文件执行 Q1、Q2、Q3 查询

    - 数据记录：记录并对比两种格式下的查询执行时间

1. **实验三：数据规模扩展性测试（可选）**
    - 固定条件：线程数和格式（Parquet）
    - 实验步骤：分别对 1 个月、12 个月、36 个月的数据执行 Q1、Q2、Q3 查询
    - 数据记录：记录不同数据规模下的执行时间，分析性能扩展性

5. 交付成果

学生需要提交一份完整的项目报告，包含以下核心内容：

1. 项目概述

- **项目目标**：明确阐述本项目的学习目标和技术目标
- **数据集说明**：详细介绍所选数据集的特点、规模和获取方式
- **实验环境**：详细说明实验环境配置（CPU 型号、核心数、内存容量、磁盘类型 SSD/HDD 等）

2. 实验设计方案

- **查询设计**：详细说明设计的 3-5 个查询（Q1-Q3）及其测试目的
- **变量控制**：说明实验中控制的变量（线程数、数据格式、数据规模等）
- **实验流程**：描述完整的实验执行流程和数据收集方法

3. 实验结果与分析（核心部分）

图表展示

必须包含以下图表来可视化实验结果：

图 1：并行扩展性分析

- X 轴：线程数（Threads）
- Y 轴：查询时间（Time）或加速比（Speedup = Time (1 thread) / Time (N threads)）
- 为 Q1、Q2、Q3 分别绘制性能曲线

图 2：数据格式对比分析

- 使用柱状图对比 Parquet vs. CSV 在相同查询下的执行时间
- 清晰展示两种格式的性能差异

分析与讨论

必须回答以下关键问题：

1. **并行扩展性分析**：DuckDB 的性能是否随线程数增加而**线性**提升？如果不是，在哪个点开始出现收益递减？请结合阿姆达尔定律（Amdahl's Law）、I/O 瓶颈、任务调度开销等理论进行分析。
2. **查询类型差异**：对于 Q1（重 I/O 操作）和 Q3（重 CPU 操作），并行度提升的效果有何不同？为什么会出现这种差异？
3. **数据格式影响**：为什么 Parquet 格式比 CSV 格式查询速度快得多？请从列式存储、数据压缩、谓词下推、统计信息等方面进行深入分析。
4. **性能瓶颈识别**：在实验中观察到的主要性能瓶颈是 CPU 还是 I/O？例如，当 8 核 CPU 在 4 线程时已达到 100% 利用率，这说明了什么问题？

4. 结论与总结

- **主要发现**：总结实验中的关键发现和性能趋势
- **性能评价**：对 DuckDB 在并行分析方面的性能表现进行总体评价
- **经验总结**：分享在实验过程中的经验教训和技术心得

5. 附录

- **实验代码**：提供所有实验脚本的完整代码
- **原始数据**：包含实验过程中收集的原始性能数据
- **补充材料**：其他有助于理解实验的补充材料

6. 附加挑战（加分项）

技术对比分析

- **与 Polars 对比**：使用 Python Polars 库（另一个高性能内存查询引擎）执行相同操作，与 DuckDB 进行性能对比分析
- **与 Pandas 对比**：尝试使用 Pandas 读取和处理大规模数据（例如 10GB），观察其在内存使用和处理速度方面的局限性

高级性能优化

- **数据导入 vs 直接查询**：对比 "直接查询 Parquet 文件" 与 "先导入到 DuckDB 数据库文件（.duckdb）中再查询" 的性能差异
- **内存限制测试**：使用 PRAGMA memory_limit='XGB' 限制 DuckDB 的可用内存，观察其对大型 GroupBy 等操作性能的影响（是否会触发磁盘溢出 spill-to-disk）

深度性能分析

- **查询计划分析**：使用 DuckDB 的 EXPLAIN 命令分析不同查询的执行计划，理解查询优化器的工作原理
- **系统级性能监控**：使用更专业的性能监控工具（如 perf、systemtap 等）深入分析 DuckDB 的系统资源使用情况

7. 参考资料

DuckDB 官方资源

- DuckDB 官方文档
- DuckDB GitHub 仓库
- DuckDB Python API
-