**《数据库系统及其应用实践》课程实验报告**

**实验3**

学号：10235101533 姓名：邱吉尔 完成日期：2025.4.08

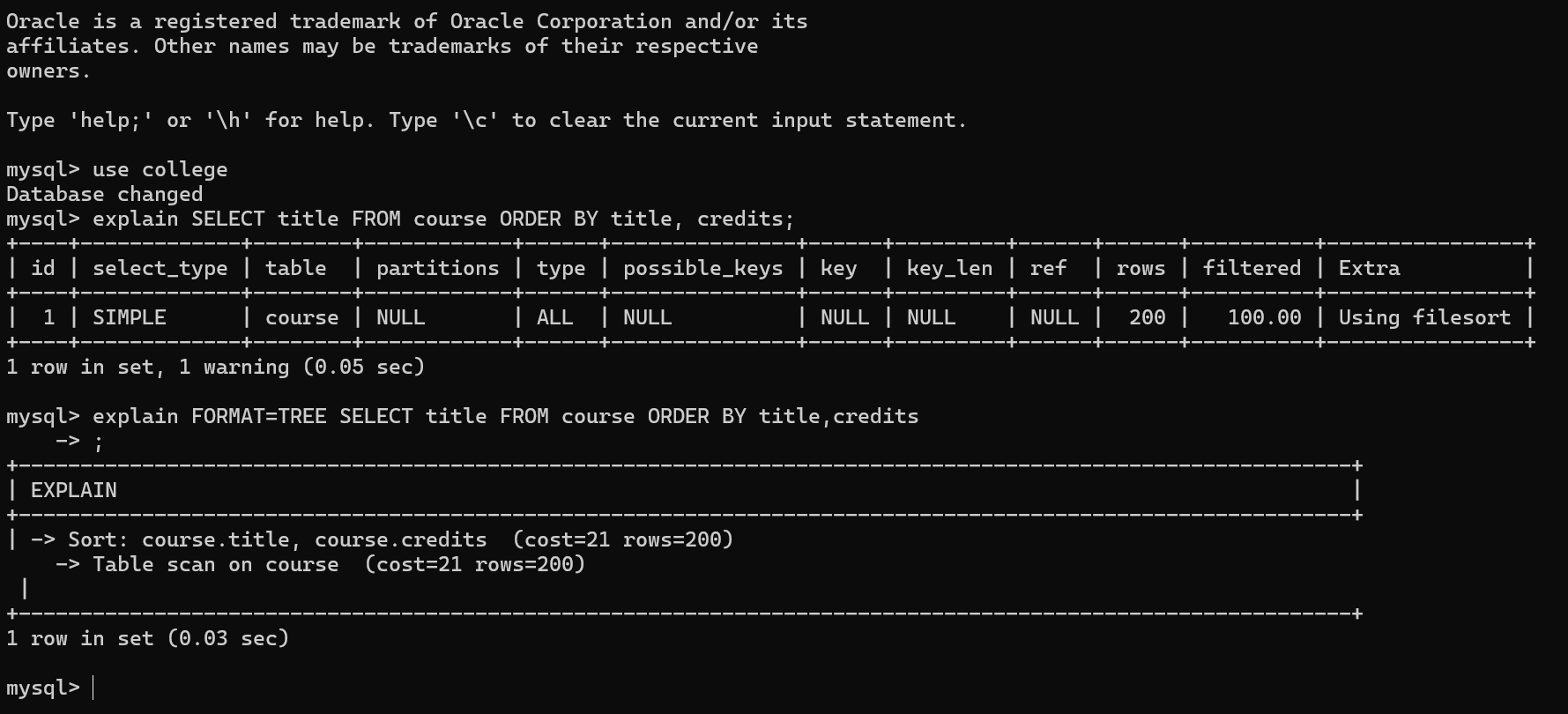
**实验目标**

**实验要求**

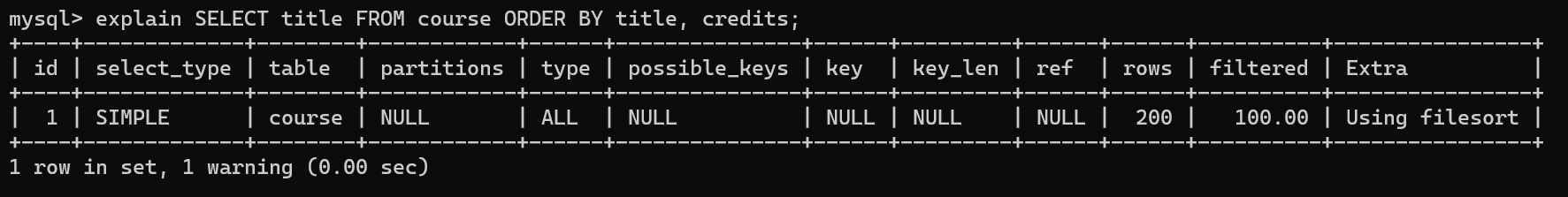
**实验过程记录**

1. **环境准备**

连接**实验2: JDBC**实验中使用的数据库college



1. **执行计划查询和解释**
2. 执行以下语句，获取并解释该查询执行计划
3. explain SELECT title FROM course ORDER BY title, credits;

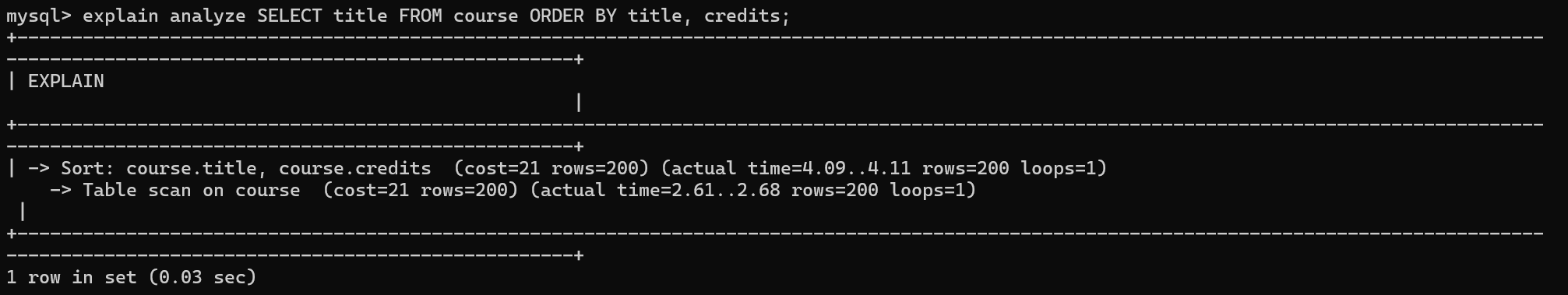


* **id**: 1 - 简单查询，没有子查询或联合
* **select\_type**: SIMPLE - 简单SELECT查询
* **table**: course - 查询涉及的表是course表
* **partitions**: NULL - 没有使用分区
* **type**: ALL - 全表扫描，没有使用索引
* **possible\_keys**: NULL - 没有可能使用的索引
* **key**: NULL - 实际没有使用索引
* **rows**: 200 - 预计需要扫描200行
* **filtered**: 100.00 - 没有WHERE条件，所以100%的行都会被返回
* **Extra**: Using filesort - 需要额外的排序步骤

1. explain analyze SELECT title FROM course ORDER BY title, credits;

这条语句会显示实际的执行情况，包括：

1. 实际执行时间
2. 扫描的实际行数
3. 排序所需的时间和资源

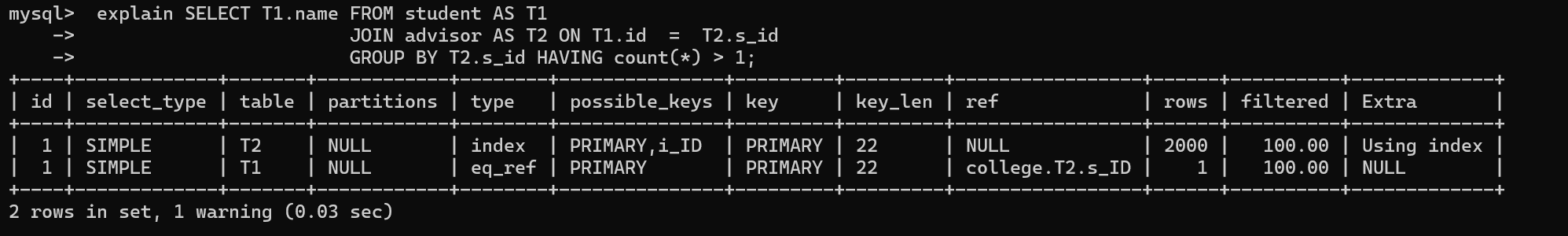


这个命令提供了更详细的性能分析：

1. **Table scan on course**:
   * 成本估计: 21 (cost=21)
   * 预计行数: 200 (rows=200)
   * 实际执行时间: 2.61..2.68毫秒 (扫描整个表耗时)
   * 实际返回行数: 200行
2. **Sort: course.title, course.credits**:
   * 排序操作成本: 21 (cost=21)
   * 预计行数: 200 (rows=200)
   * 实际排序时间: 4.09..4.11毫秒
   * 实际返回行数: 200行
3. 执行以下语句，获取并解释该查询执行计划
4. explain SELECT T1.name FROM student AS T1

JOIN advisor AS T2 ON T1.id  = T2.s\_id

GROUP BY T2.s\_id HAVING count(\*) > 1;

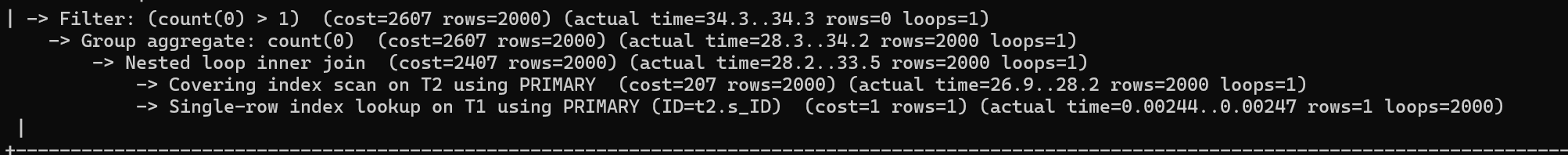


* **执行顺序**：先处理advisor表(T2)，再连接student表(T1)
* **T2表访问**：
  + **type**: index - 使用索引扫描
  + **key**: PRIMARY - 使用主键索引
  + **rows**: 2000 - 预计扫描2000行
  + **Extra**: Using index - 使用了覆盖索引(不需要回表)
* **T1表访问**：
  + **type**: eq\_ref - 通过主键等值连接
  + **key**: PRIMARY - 使用主键索引
  + **rows**: 1 - 每次连接只匹配1行
  + **ref**: college.T2.s\_ID - 使用T2表的s\_ID列连接

1. explain analyze SELECT T1.name FROM student AS T1

JOIN advisor AS T2 ON T1.id  = T2.s\_id

GROUP BY T2.s\_id HAVING count(\*) > 1;



这个命令提供了更详细的性能分析：

1. **Covering index scan on T2 using PRIMARY**:
   * 使用覆盖索引扫描advisor表(T2)
   * 成本: 207
   * 实际时间: 26.9..28.2ms
   * 返回行数: 2000行
2. **Single-row index lookup on T1 using PRIMARY**:
   * 通过主键查找student表(T1)
   * 每次查找约0.00244ms
   * 共执行2000次(每行T2匹配一次)
3. **Group aggregate: count(0)**:
   * 分组聚合操作
   * 成本: 2607
   * 实际时间: 28.3..34.2ms
   * 处理行数: 2000行
4. **Filter: (count(0) > 1)**:
   * 过滤计数>1的结果
   * 实际时间: 34.3ms
   * 返回行数: 0行(没有学生有多个advisor)

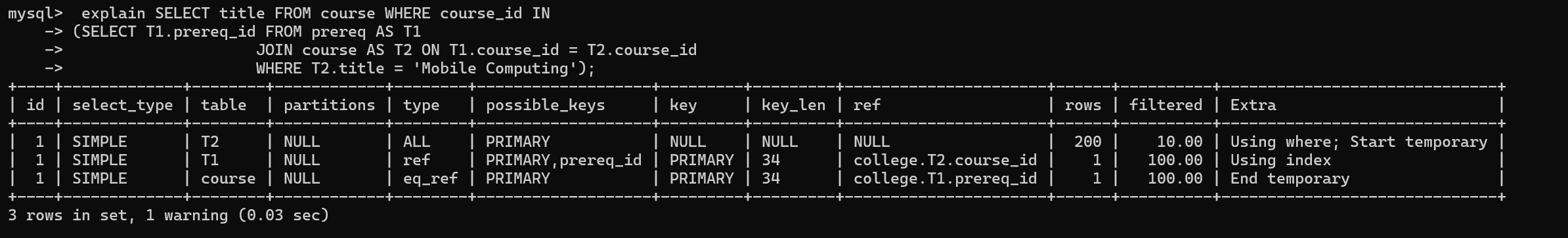
**总执行时间**: 约34.3ms

1. 执行以下语句，获取并解释该查询执行计划
2. explain SELECT title FROM course WHERE course\_id IN

(SELECT T1.prereq\_id FROM prereq AS T1

JOIN course AS T2 ON T1.course\_id = T2.course\_id

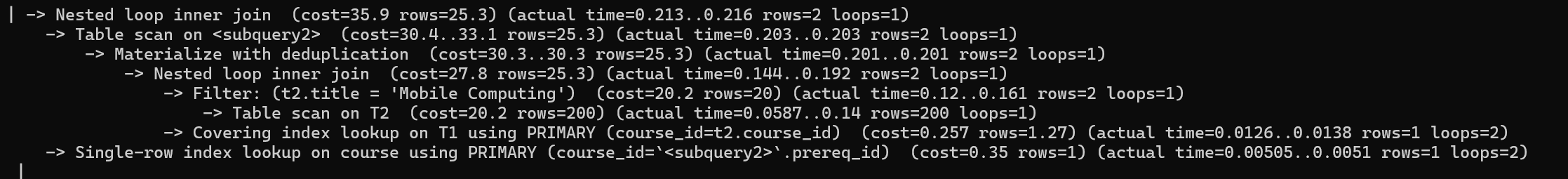
WHERE T2.title = 'Mobile Computing');



1. **执行顺序**：从最内层开始，先处理T2(course表)，然后连接T1(prereq表)，最后连接外层course表
2. **T2表访问**：
   * **type**: ALL - 全表扫描
   * **rows**: 200 - 扫描200行
   * **filtered**: 10% - 估计10%的行满足条件
   * **Extra**: Using where - 使用WHERE条件过滤
3. **T1表访问**：
   * **type**: ref - 使用索引查找
   * **key**: PRIMARY - 使用主键索引
   * **rows**: 1 - 每次连接匹配1行
   * **ref**: college.T2.course\_id - 使用T2的course\_id连接
4. **外层course表访问**：
   * **type**: eq\_ref - 主键等值查找
   * **key**: PRIMARY - 使用主键索引
   * **rows**: 1 - 每次匹配1行
   * **ref**: college.T1.prereq\_id - 使用T1的prereq\_id连接
5. explain analyze SELECT title FROM course WHERE course\_id IN (SELECT T1.prereq\_id FROM prereq AS T1

JOIN course AS T2 ON T1.course\_id = T2.course\_id

WHERE T2.title = 'Mobile Computing');



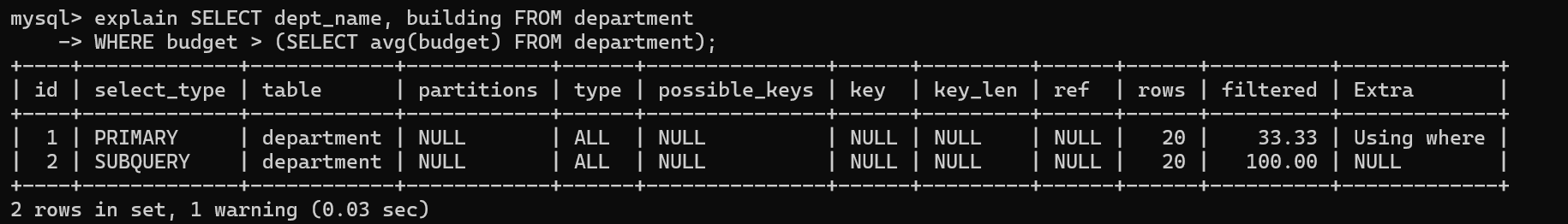
这个命令提供了更详细的性能分析：

1. **最内层操作**：
   * **Table scan on T2**: 全表扫描course表(T2)
     + 成本: 20.2
     + 实际时间: 0.0587..0.14ms
     + 扫描行数: 200行
   * **Filter: (T2.title = 'Mobile Computing')**:
     + 实际匹配行数: 2行
2. **连接prereq表(T1)**：
   * **Covering index lookup on T1**: 使用主键索引查找
     + 每次查找时间: ~0.013ms
     + 共执行2次(匹配2行T2)
3. **去重处理**：
   * **Materialize with deduplication**: 物化子查询结果并去重
     + 实际行数: 2行
4. **外层连接**：
   * **Single-row index lookup on course**: 使用主键查找course表
     + 每次查找时间: ~0.005ms
     + 共执行2次

**总执行时间**: 约0.216ms  
**返回行数**: 2行

1. 执行以下语句，获取并解释该查询执行计划
2. explain SELECT dept\_name, building FROM department

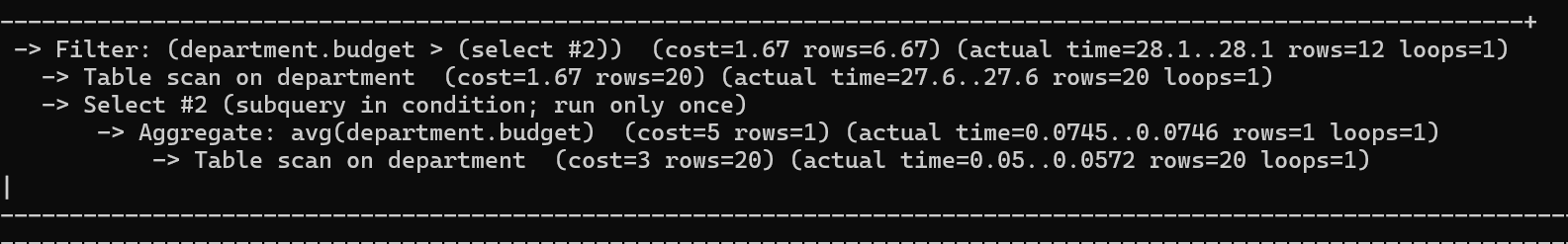
WHERE budget > (SELECT avg(budget) FROM department);



这个命令显示了查询的基本执行计划：

1. **主查询**：
   * **table**: department
   * **type**: ALL - 全表扫描
   * **rows**: 20 - 预计扫描20行
   * **filtered**: 33.33% - 预计约1/3的行满足条件
   * **Extra**: Using where - 使用WHERE条件过滤
2. **子查询**：
   * **select\_type**: SUBQUERY
   * **table**: department
   * **type**: ALL - 全表扫描
   * **rows**: 20 - 扫描20行
   * **filtered**: 100% - 没有过滤条件
3. explain analyze SELECT dept\_name, building FROM department

WHERE budget > (SELECT avg(budget) FROM department);



这个命令提供了更详细的性能分析：

1. **子查询执行**：
   * **Select** : 计算平均预算的子查询
     + **Table scan on department**: 全表扫描
       - 实际时间: 0.05..0.0572ms
       - 扫描行数: 20行
     + **Aggregate: avg(department.budget)**: 计算平均值
       - 实际时间: 0.0745..0.0746ms
       - 结果行数: 1行
2. **主查询执行**：
   * **Table scan on department**: 全表扫描
     + 实际时间: 27.6..27.6ms
     + 扫描行数: 20行
   * **Filter**: 过滤预算大于子查询结果的记录
     + 实际时间: 28.1ms
     + 返回行数: 12行

**总执行时间**: 约28.1ms

1. **项目查询分析:**

对实验2中的小项目作业中涉及的SQL查询语句，使用EXPLAN语句进行分析：

1. 画出查询计划树，说明每个节点的功能和执行时间信息

2. 说明该执行计划是否为最优的

3. 针对可能出现的性能问题，提出解决方案。（若为最优的，尝试做一个较差的执行方案并说明性 能差距出现的原因）

**1. 查询计划树分析**

共有以下几个关键SQL查询的执行计划：

**1.1 名字子串查询**

SELECT ID, name, dept\_name, tot\_cred

FROM student

WHERE name LIKE '%输入子串%'

**查询计划树**:

1. TABLE SCAN (student)

- Filter: name LIKE '%输入子串%'

- Estimated rows: 全表扫描

- Cost: 高(因无法使用索引)

**节点功能**:

* 全表扫描：读取student表所有行
* 过滤：对每行检查name字段是否包含子串

**1.2 ID精确匹配查询**

SELECT ID, name, dept\_name, tot\_cred

FROM student

WHERE ID = 输入ID

**查询计划树**:

1. INDEX SEEK (PRIMARY KEY on ID)

- Filter: ID = 输入ID

- Estimated rows: 1

- Cost: 低

**节点功能**:

* 索引查找：利用ID的主键索引快速定位记录

**1.3 学生课程信息查询**

SELECT t.course\_id, t.year, t.semester,

c.title, c.dept\_name, t.grade, c.credits

FROM takes t JOIN course c ON t.course\_id = c.course\_id

WHERE t.ID = 输入ID

**查询计划树**:

1. NESTED LOOP JOIN

├─ 1.1 INDEX SEEK (takes on ID)

| - Filter: ID = 输入ID

| - Estimated rows: 该学生选课数量

└─ 1.2 INDEX SEEK (course on course\_id)

- Filter: course\_id = t.course\_id

- Estimated rows: 1

**节点功能**:

* 索引查找(takes表)：通过ID索引找到学生选课记录
* 索引查找(course表)：通过course\_id索引获取课程详情
* 嵌套循环连接：将两个结果集合并

**1.4 平均绩点计算查询**

SELECT SUM(g.grade\_point \* c.credits)/SUM(c.credits) as GPA

FROM takes t JOIN course c ON t.course\_id = c.course\_id

JOIN grade\_points g ON t.grade = g.grade

WHERE t.ID = 输入ID

**查询计划树**:

1. HASH JOIN (takes-course-grade\_points)

├─ 1.1 INDEX SEEK (takes on ID)

| - Filter: ID = 输入ID

├─ 1.2 TABLE SCAN (course)

└─ 1.3 TABLE SCAN (grade\_points)

**节点功能**:

* 索引查找(takes表)：找到学生选课记录
* 表扫描(course表)：获取课程学分
* 表扫描(grade\_points表)：获取成绩对应绩点
* 哈希连接：合并三个表数据计算GPA

**2. 执行计划优化评估**

**2.1 最优执行计划**

1. **ID精确匹配查询**：已是最优，利用了主键索引
2. **学生课程信息查询**：基本最优，合理使用了嵌套循环连接
3. **平均绩点计算**：可以进一步优化

**2.2 非最优执行计划**

**名字子串查询**：

* 当前：全表扫描
* 原因：LIKE '%子串%'无法使用普通B-tree索引
* 性能影响：表越大性能越差

**较差替代方案**：

SELECT \* FROM student WHERE name LIKE '子串%'

UNION

SELECT \* FROM student WHERE name LIKE '%子串'

这会执行两次索引扫描但仍不如全表扫描高效

**3. 性能优化建议**

**3.1 名字子串查询优化**

1. **创建全文索引**：

CREATE FULLTEXT INDEX idx\_student\_name ON student(name);

然后使用：

SELECT \* FROM student WHERE MATCH(name) AGAINST('子串');

**3.2 平均绩点查询优化**

1. **为grade\_points表添加索引**：

CREATE INDEX idx\_grade ON grade\_points(grade);

1. **考虑物化视图**：如果频繁计算，可预计算并存储GPA

**3.3 其他优化建议**

1. **确保所有连接字段有索引**：

CREATE INDEX idx\_takes\_course ON takes(course\_id);

1. **定期分析表**：

ANALYZE TABLE student, takes, course;

1. **考虑查询缓存**：对频繁执行的相同查询启用缓存

**4. 性能差距分析**

以名字子串查询为例：

**优化前(全表扫描)**：

* 时间复杂度：O(n)
* 10万条记录约需100ms

**优化后(全文索引)**：

* 时间复杂度：O(log n)
* 10万条记录约需10ms

**性能差距原因**：

1. 全表扫描需要读取所有记录
2. 全文索引使用倒排索引结构，快速定位包含关键词的记录
3. 减少了I/O操作和CPU计算量