

华东师范大学软件工程学院实验报告

实验课程：数据库系统及其应用实践

年级：2023 级

实验成绩：

实验名称：Lab-03

姓名：顾翌炜

实验编号：Lab-03

学号：10235101527

实验日期：2025/04/03

指导教师：姚俊杰

组号：01

实验时间：2 课时

1 实验目标

学习和熟悉使用 SQL 查询处理

参考链接：

<https://dev.mysql.com/doc/refman/8.4/en/explain.html>

<https://dev.mysql.com/doc/refman/8.4/en/explain-output.html>

2 实验要求

- 1) 按照实验内容，依次完成每个实验步骤；
- 2) 操作实验步骤时，需要理解该操作步骤的目的，预判操作的结果；当操作结果与预判不符时，及时向任课教师和助教咨询；
- 3) 在实验报告中依次记录主要操作步骤的内容和结果（返回的消息或截图）；
- 4) 对实验中遇到的问题、解决方案及收获进行总结；
- 5) 确保实验报告整洁、美观（注意字体、字号、对齐、截图大小和分页等；）

3 实验过程记录

3.1 环境准备

首先，连接实验二中的 college 数据库。我为了后续方便重用，也为了防止源数据被损坏，此处新建了一个 lab-3_college

如下图所示：

```
C:\Users\GHOST>mysql -uroot -p 连接数据库
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 108
Server version: 8.0.41 MySQL Community Server - GPL

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use lab-3_college 切换到本次实验的数据库
Database changed
mysql> explain SELECT title FROM course ORDER BY title, credits; 查询处理
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | select_type | table | partitions | type | possible_keys | key | key_len | ref | rows | filtered | Extra |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | SIMPLE | course | NULL | ALL | NULL | NULL | NULL | NULL | 200 | 100.00 | Using filesort |
+----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set, 1 warning (0.00 sec)
```

图 1: cmd 连接数据库

3.2 EXPLAIN 中各个字段的作用与含义

我们已经通过 EXPLAIN 语句得到了各个语句的分析，现在需要通过解释结果来看哪些部分需要进行优化。

通过查阅资料，我们总结为以下表格：

字段名	含义
id	查询中每个 SELECT 子句的唯一标识符，值越大越先执行
select_type	SELECT 的类型，如 SIMPLE（简单 SELECT）、PRIMARY（主查询）、SUBQUERY（子查询）等
table	当前正在访问的表的名称
type	表的访问方式，性能关键字段（详见下表）
possible_keys	查询可能使用的索引列表
key	实际使用的索引，如果为 NULL，说明未使用索引
key_len	使用的索引长度（字节数），值越小越高效
ref	哪一列或常数与 key 一起被使用来选择记录
rows	估计要读取的行数，越少越好
Extra	额外信息，如是否使用临时表、排序等

表 1: MySQL EXPLAIN 字段含义

其中最重要的是 type 字段，他决定了这个语句的性能。

这个字段是判断是否需要优化的重要指标，它代表了表的访问方式：

type	含义与性能说明
system	表仅有一行（系统表），性能最佳
const	查询结果只匹配一行，通常用于主键或唯一索引查询
eq_ref	对于联结中的每一行,从另一个表中读取最多一条匹配记录(使用主键)
ref	使用非唯一索引或前缀索引来查找匹配行
range	通过索引范围扫描（如 BETWEEN、>、< 等）
index	全索引扫描，不查表，但会扫描整个索引，性能较差
ALL	全表扫描，最差的访问方式，说明无索引或索引未命中

表 2: MySQL 中 EXPLAIN 的 type 字段含义及性能分析

如果我们在返回结果中看到 ALL，说明需要优化。

3.3 执行计划查询和解释-1

执行以下语句，获取并解释该查询执行计划

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

以上两行运行结果如下图所示：

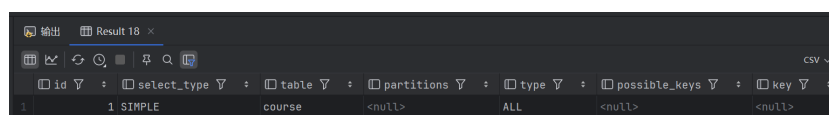


图 2: explain 操作

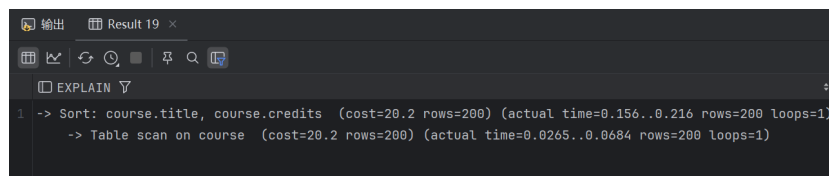


图 3: explain analyze 操作

执行结果如下图所示：

字段	含义	结果说明	实际值
id	查询的执行顺序	只有一个简单查询，编号为 1	1
select_type	查询类型	SIMPLE 表示没有使用子查询或联合	SIMPLE
table	涉及的表	查询来自 course 表	course
type	连接类型	ALL 表示进行了全表扫描，是效率最低的一种	ALL
possible_keys	可能使用的索引	为空，说明没有可用索引	(null)
key	实际使用的索引	也为空，表示没有使用任何索引	(null)
key_len	使用索引的长度	无索引，因此为 (null)	(null)
ref	哪些列或常数与索引进行比较	无引用	(null)
rows	扫描行数估计	预计扫描 200 行（全表）	200
filtered	过滤比例估计（百分比）	预计 100% 行符合条件	100.0
Extra	附加信息	Using filesort 表示使用了额外的文件排序	Using filesort

表 3: SQL 执行计划详细分析

3.3.1 更清晰的分析结果

虽然使用 EXPLAIN 能够提供基本的执行计划信息，但在分析复杂查询或调试性能问题时，默认格式可能不够直观。为此，MySQL 提供了更清晰和结构化的格式选项，如 FORMAT=TREE 和 FORMAT=JSON，用于展示更详细的执行步骤与优化器决策过程。

FORMAT=TREE 模式以树状结构展示查询的执行流程，使执行顺序和各操作之间的层级关系一目了然，适合人类阅读和理解。而 FORMAT=JSON 模式则输出完整的结构化数据，适合进一步程序处理或自动分析。

对于本实验中的第一条语句：

更清晰和详细的分析结果

```
1 explain FORMAT=TREE SELECT title FROM course ORDER BY title, credits;
2 explain FORMAT=JSON SELECT title FROM course ORDER BY title, credits;
```

执行上述语句后，得到如下更清晰的分析结果：

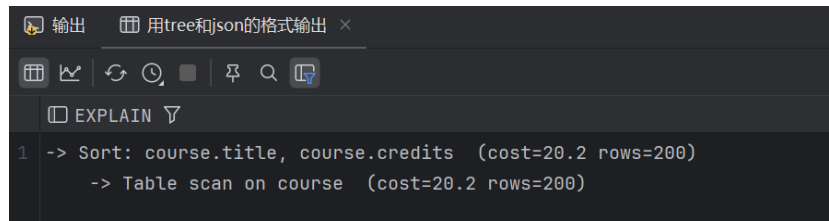


图 4: TREE 格式的输出

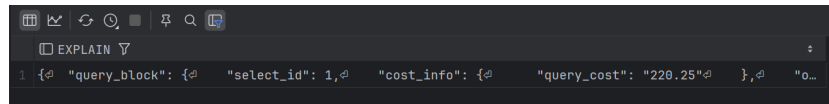


图 5: JSON 格式的输出

进一步分析：从 TREE 格式的输出中，我们可以直观地看到执行过程为：首先访问 ‘course’ 表的所有行（全表扫描），然后执行排序操作。JSON 格式则显示了更为精细的执行细节，包括优化器选择路径的逻辑判断、排序策略、临时表使用情况等内容。

结论：通过对比不同格式的 EXPLAIN 输出，我们可以更全面地理解查询执行的具体过程，有助于在未来优化更复杂的查询语句时选取合适的手段（如索引、分区、临时表优化等）。

3.4 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

```
1  explain
2  SELECT T1.name
3  FROM student AS T1
4         JOIN advisor AS T2 ON T1.id = T2.s_id
5  GROUP BY T2.s_id
6  HAVING count(*) > 1;
7
8  explain analyze
9  SELECT T1.name
10 FROM student AS T1
11        JOIN advisor AS T2 ON T1.id = T2.s_id
12 GROUP BY T2.s_id
13 HAVING count(*) > 1;
```

以上两行运行结果如下图所示：

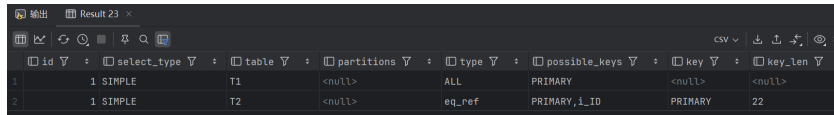


图 6: explain 操作

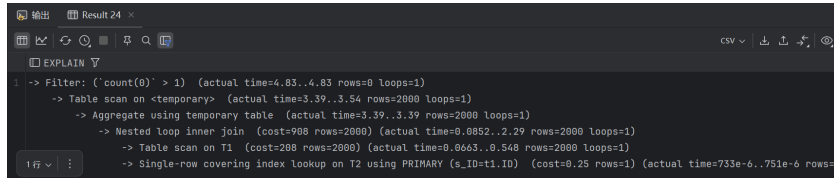


图 7: explain analyze 操作

我们可以分析得到以下解释信息：

字段	含义	T1 的值解释	T2 的值解释
id	查询的执行顺序	1, 表示第一条执行单元	1, 表示与 T1 属于同一层查询
select_type	查询类型	SIMPLE, 表示基础查询, 无子查询	SIMPLE, 亦为基础查询
table	正在访问的表名	表 T1	表 T2
type	连接类型（越靠前效率越低）	ALL, 表示对 T1 进行全表扫描	eq_ref, 表示使用唯一索引连接（效率较高）
possible_keys	可供选择的索引	PRIMARY, 可用主键索引	PRIMARY, i_ID, 两个可用索引
key	实际使用的索引	未使用任何索引	使用了主键索引 PRIMARY
key_len	索引长度	空值, 表示未使用索引	22 字节的主键索引长度
ref	哪些列参与了索引匹配	空值, 无引用列	使用 T1.ID 作为连接条件
rows	扫描行数估计	2000 行（全表扫描）	1 行（由于主键匹配）
filtered	行过滤比例（百分比）	100, 表示全部行都符合条件	100, 全部匹配
Extra	附加信息	Using temporary, 表示使用临时表（可能用于排序或分组）	Using index, 表示只访问索引即可返回需要的数据

表 4: T1 与 T2 表连接的执行计划分析

3.5 执行计划查询和解释-3

执行以下语句，获取并解释该查询执行计划

```

1  explain
2  SELECT title
3  FROM course
4  WHERE course_id IN
5      (SELECT T1.prereq_id
6       FROM prereq AS T1
7        JOIN course AS T2 ON T1.course_id = T2.course_id
8       WHERE T2.title = 'Mobile Computing');
9
10 explain analyze
11 SELECT title
12 FROM course
13 WHERE course_id IN
14     (SELECT T1.prereq_id
15      FROM prereq AS T1
16       JOIN course AS T2 ON T1.course_id = T2.course_id
17      WHERE T2.title = 'Mobile Computing');

```

以上两行运行结果如下图所示：

id	select_type	table	partitions	type	possible_keys	key	key_len
1	1 SIMPLE	<subquery2>	<null>	ALL	<null>	<null>	<null>
2	1 SIMPLE	course	<null>	eq_ref	PRIMARY	PRIMARY	34
3	2 MATERIALIZED	T2	<null>	ALL	PRIMARY	<null>	<null>
4	2 MATERIALIZED	T1	<null>	ref	PRIMARY,prereq_id	PRIMARY	34

图 8: explain 操作

```

-> Nested loop inner join (cost=35.9 rows=25.3) (actual time=0.105..0.108 rows=2 loops=1)
  -> Table scan on <subquery2> (cost=30.4..33.1 rows=25.3) (actual time=0.101..0.101 rows=2 loops=1)
  -> Materialize with deduplication (cost=30.3..30.3 rows=25.3) (actual time=0.1..0.1 rows=2 loops=1)
    -> Nested loop inner join (cost=27.8 rows=25.3) (actual time=0.0723..0.0967 rows=2 loops=1)
      -> Filter: (t2.title = 'Mobile Computing') (cost=20.2 rows=20) (actual time=0.0596..0.0795 rows=2 loops=1)
      -> Table scan on T2 (cost=20.2 rows=200) (actual time=0.0266..0.0688 rows=200 loops=1)
      -> Covering index lookup on T1 using PRIMARY (course_id=t2.course_id) (cost=0.257 rows=1.27) (actual time=0.00715..0.00785 rows=1 loops=1)
    -> Single-row index lookup on course using PRIMARY (course_id=<subquery2>.prereq_id) (cost=0.35 rows=1) (actual time=0.00275..0.0028 rows=1 loops=1)

```

图 9: explain analyze 操作

我们可以分析得到以下解释信息：

下表展示了对带有 IN + 子查询 + JOIN 的 SQL 语句的执行计划逐项分析：

id	select_type	table	分析说明
1	SIMPLE	<subquery2>	主查询对子查询结果进行 IN 判断。这里的 <subquery2> 是物化后的临时表，使用 ALL 表示全表扫描。表示优化器将子查询缓存为中间结果集，再逐行与 course.course_id 匹配。
1	SIMPLE	course	主查询中的 course 表，通过 eq_ref（唯一索引等值连接）匹配子查询返回的 prereq_id，使用主键索引 PRIMARY，性能较好。
2	MATERIALIZED	T2	子查询第一部分，对 T2（即 course）做全表扫描（ALL），用于找出 title 为'Mobile Computing' 的课程。虽然有主键 PRIMARY，但未用到索引筛选条件，可优化。
2	MATERIALIZED	T1	子查询第二部分，对 prereq 表通过 ref（索引查找）连接 T2.course_id，使用了 PRIMARY 和 prereq_id 的组合索引，性能优良。Extra 中的 Using index 表示只访问了索引。

表 5: 四步执行计划分析表

整体分析：

- 子查询首先对 T2 表进行全表扫描（由于缺少 WHERE 子句索引支持），再通过 T1.course_id = T2.course_id 做连接；
- 子查询被 MySQL 优化器 物化（**MATERIALIZED**），即提前执行并缓存结果；
- 主查询使用 IN (<subquery2>) 来过滤符合条件的课程 ID，然后通过主键等值匹配 course 表中的记录；
- course 表通过 eq_ref 和主键做高效匹配，性能良好；
- 子查询中 T1 表也使用了索引查找，其性能也较好。

3.6 执行计划查询和解释-4

执行以下语句，获取并解释该查询执行计划

```

1      explain
2      SELECT dept_name, building
3      FROM department
4      WHERE budget > (SELECT avg(budget) FROM department);
5
6      explain analyze
7      SELECT dept_name, building

```



```
8 FROM department
9 WHERE budget > (SELECT avg(budget) FROM department);
```

以上两行运行结果如下图所示：

id	select_type	table	partitions	type	possible_keys	key	key_len
1	PRIMARY	department	<null>	ALL	<null>	<null>	<null>
2	SUBQUERY	department	<null>	ALL	<null>	<null>	<null>

图 10: explain 操作

```

1 "-> Filter: (department.budget > (select #2)) (cost=0.917 rows=6.67) (actual time=0.0992..0.103 rows=12 loops=1)
2   -> Table scan on department (cost=0.917 rows=20) (actual time=0.0506..0.0531 rows=20 loops=1)
3   -> Select #2 (subquery in condition; run only once)
4     -> Aggregate: avg(department.budget) (cost=4.25 rows=1) (actual time=0.0265..0.0265 rows=1 loops=1)
5       -> Table scan on department (cost=2.25 rows=20) (actual time=0.0151..0.021 rows=20 loops=1)
6
7

```

图 11: explain analyze 操作

我们可以分析得到以下解释信息：

字段	含义	主查询分析	子查询分析
id	查询的执行顺序	1, 表示第一条执行单元（主查询）	2, 表示第二条执行单元（子查询）
select_type	查询类型	PRIMARY, 表示主查询	SUBQUERY, 表示子查询
table	正在访问的表名	department, 主查询访问 ‘department’ 表	department, 子查询也访问 ‘department’ 表
type	连接类型（越靠前效率越低）	ALL, 表示全表扫描（效率较低）	ALL, 表示全表扫描（效率较低）
possible_keys	可供选择的索引	无, 表示没有可用索引	无, 表示没有可用索引
key	实际使用的索引	无, 表示没有使用任何索引	无, 表示没有使用任何索引
key_len	索引长度	无, 表示未使用索引	无, 表示未使用索引
ref	哪些列参与了索引匹配	无, 表示没有引用列	无, 表示没有引用列
rows	扫描行数估计	20, 预计扫描 20 行数据	20, 预计扫描 20 行数据

表 6: 主查询和子查询的执行计划分析

字段	含义	主查询分析	子查询分析
filtered	行过滤比例（百分比）	33.33, 表示大约 33.33% 的行符合主查询条件	100, 表示子查询返回了所有符合条件的行
Extra	附加信息	Using where, 表示主查询使用了 'WHERE' 子句过滤数据	无, 子查询没有额外信息

表 7: 主查询和子查询的执行计划分析-续

3.7 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 EXPLAIN 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。)

3.7.1 提取 Lab-2 中使用到的 sql 语句

首先从 Lab-2 的 JDBC 代码中提取出主要的 sql 语句:

备注

JDBC 中使用了? 来预留参数的位置, 此处为了检验查询, 我们使用 id=1000 的这位同学来进行实验

主要包括了:

Lab-2 中的 sql 语句

```

1  SELECT ID, name, dept_name, tot_cred
2  FROM student
3  WHERE name LIKE 'man';
4
5  SELECT ID, name, dept_name, tot_cred
6  FROM student
7  WHERE ID = 1000;
8
9  SELECT takes.course_id, year, semester, title, dept_name, grade, credits
10 FROM takes
11     JOIN course ON takes.course_id = course.course_id
12 WHERE ID = 1000;
13

```

```
14     SELECT grade_point, credits
15     FROM (takes JOIN course ON takes.course_id = course.course_id)
16           JOIN gpa ON gpa.grade = TRIM(takes.grade)
17     WHERE ID = 1000;
```

3.7.2 EXPLAIN ANALYZE Lab-2 中的语句

EXPLAIN ANALYZE Lab-2 中的语句

```
1     EXPLAIN
2     SELECT ID, name, dept_name, tot_cred
3     FROM student
4     WHERE name LIKE 'man';
5
6     EXPLAIN
7     SELECT ID, name, dept_name, tot_cred
8     FROM student
9     WHERE ID = 1000;
10
11    EXPLAIN
12    SELECT takes.course_id, year, semester, title, dept_name, grade, credits
13    FROM takes
14           JOIN course ON takes.course_id = course.course_id
15    WHERE ID = 1000;
16
17    EXPLAIN
18    SELECT grade_point, credits
19    FROM (takes JOIN course ON takes.course_id = course.course_id)
20           JOIN gpa ON gpa.grade = TRIM(takes.grade)
21    WHERE ID = 1000;
```

得到的结果如下图所示：

id	1
select_type	SIMPLE
table	student
partitions	<null>
type	ALL
possible_keys	name
key	<null>
key_len	<null>
ref	<null>
rows	2000
filtered	11.11
Extra	Using where

图 12: 第一个语句的 EXPLAIN 结果

id	1
select_type	SIMPLE
table	student
partitions	<null>
type	ALL
possible_keys	PRIMARY
key	<null>
key_len	<null>
ref	<null>
rows	2000
filtered	10
Extra	Using where

图 13: 第二个语句的 EXPLAIN 结果

id	1
select_type	SIMPLE
table	takes
partitions	<null>
type	ALL
possible_keys	PRIMARY, course_id, idx_takes_id, idx_takes_course_id
key	<null>
key_len	<null>
ref	<null>
rows	29691
filtered	10
Extra	Using where
id	2
select_type	SIMPLE
table	course
partitions	<null>
type	eq_ref
possible_keys	PRIMARY
key	PRIMARY
key_len	34
ref	lab-3_college.takes.course_id
rows	1
filtered	100
Extra	<null>

图 14: 第三个语句的 EXPLAIN 结果

id	1
select_type	SIMPLE
table	takes
partitions	<null>
type	ALL
possible_keys	PRIMARY, course_id, idx_takes_id, idx_takes_course_id
key	<null>
key_len	<null>
ref	<null>
rows	29691
filtered	10
Extra	Using where
id	2
select_type	SIMPLE
table	gpa
partitions	<null>
type	ref
possible_keys	idx_gpa_grade
key	idx_gpa_grade
key_len	11
ref	func
rows	1
filtered	100
Extra	Using index condition

图 15: 第四个语句的 EXPLAIN 结果

结合我在 3.2 章节内的表 1 和表 2 中提到的内容，我们需要对于 type=ALL 的语句进行更新。

可以通过添加索引来优化查询性能，尤其是在我的查询逻辑中，有部分性能瓶颈正是因为没有使用索引或者索引失效。

3.7.3 更新第一条 SQL 语句

由图 12 可以看到，第一条 SQL 语句查询的执行计划中，对 student 表进行了全表扫描，没有使用索引，导致查询性能较差。可以通过为 student 表的相关字段创建索引来优化查询性能。

再来看一下第一条 SQL 语句的 **EXPLAIN ANALYZE** 的结果：

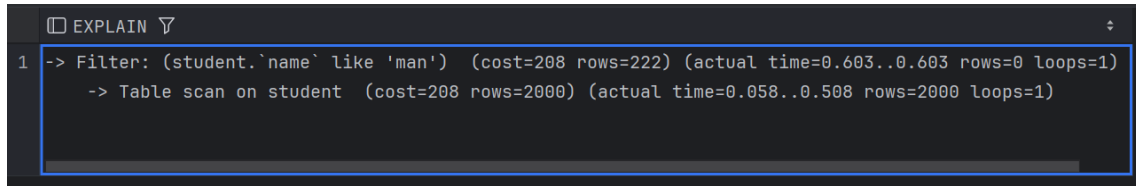


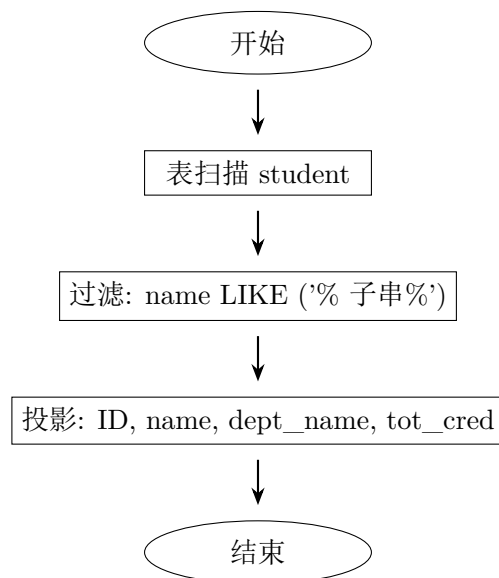
图 16: EXPLAIN_ANALYZE 第一条 SQL 语句

从 cost 上看, student 表的扫描成本较高, 因此应优化 student 表的查询性能。
原始的查询语句:

原始的子串查询

```
1 SELECT ID, name, dept_name, tot_cred
2 FROM student
3 WHERE name LIKE '%子串%';
```

其过程可以用下图来解释:



问题:

- % 开头的 LIKE 查询无法使用普通索引, 会导致全表扫描。

优化方案:

- 此处我的思路是在 student 表的 name 列上创建一个全文索引 (FULLTEXT) (针对模糊查询), 用于优化对该列中文本数据的搜索查询。

代码如下:

student.name 上添加全文索引 (FULLTEXT) (针对模糊查询)

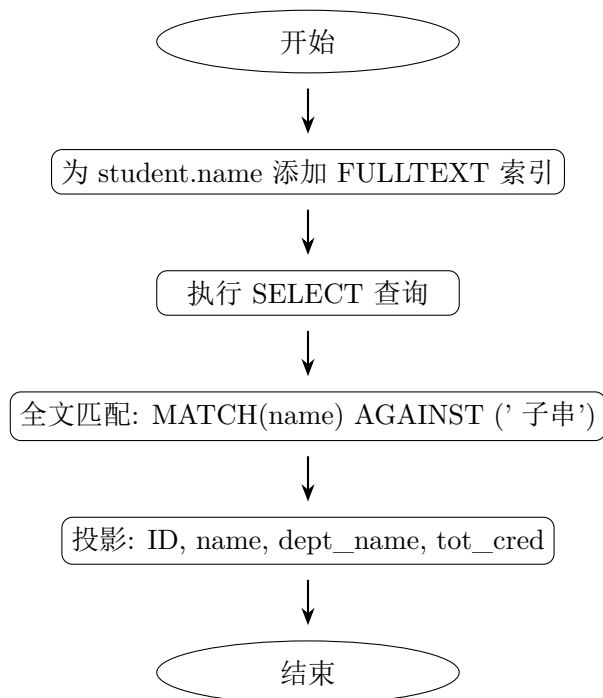
```
1 ALTER TABLE student ADD FULLTEXT(name);
```

然后查询改为:

更新后的查询语句

```
1 SELECT ID, name, dept_name, tot_cred
2 FROM student
3 WHERE MATCH(name) AGAINST ('子串');
```

其过程可以用下图来解释:



更新结果

重新对于上述 sql 语句进行 **EXPLAIN** 解释, 得到的结果如下图所示:

id	1
select_type	SIMPLE
table	student
partitions	<null>
type	fulltext
possible_keys	name
key	name
key_len	0
ref	const
rows	1
filtered	100
Extra	Using where; Ft_hints: sorted

图 17: 第一条 sql 语句更新结果 - EXPLAIN

再用 EXPLAIN ANALYZE 来检查一下:

EXPLAIN	
1	-> Filter: (match student.`name` against ('子串')) (cost=1.01 rows=1) (actual time=0.0067..0.0067 rows=0 loops=1) -> Full-text index search on student using name (name='子串') (cost=1.01 rows=1) (actual time=0.0058..0.0058 rows=0 loops=1)

图 18: 第一条 sql 语句更新结果 - EXPLAIN ANALYZE

这里我们看到 type 变为了 fulltext, const 值也从 208 减少为了 1.01, 大幅减少, 优化成功。

3.7.4 更新第二条 SQL 语句

由图 13 可以看到, 第二条 SQL 语句查询的执行计划中, 对 student 表进行了全表扫描, 没有使用索引, 导致查询性能较差。可以通过为 student 表的相关字段创建索引来优化查询性能。

再来看一下第二条 SQL 语句的 EXPLAIN ANALYZE 的结果:

EXPLAIN	
1	-> Filter: (student.ID = 1000) (cost=208 rows=200) (actual time=0.0632..0.617 rows=1 loops=1) -> Table scan on student (cost=208 rows=2000) (actual time=0.0607..0.528 rows=2000 loops=1)

图 19: EXPLAIN_ANALYZE 第二条 SQL 语句

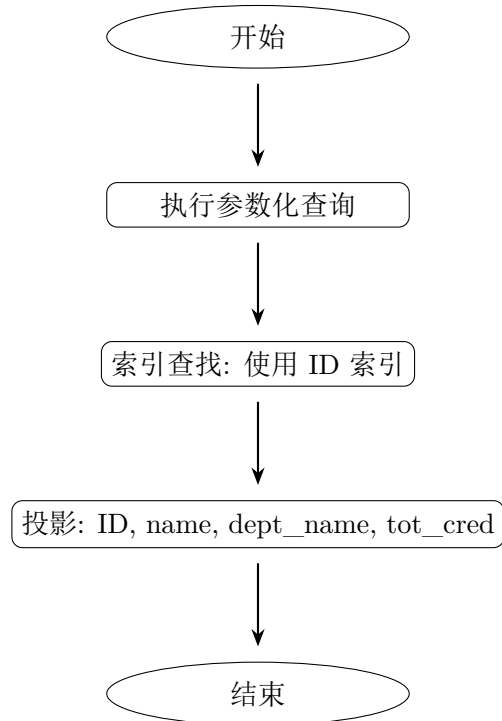
从 cost 上看, student 表的扫描成本较高, 因此应优化 student 表的查询性能。
原始的查询语句:

原始的子串查询

```
1 SELECT ID, name, dept_name, tot_cred
```

```
2    FROM student
3    WHERE ID = ?;
```

其过程可以用下图来解释：



问题：

- 该查询语句虽然使用主键进行筛选，但由于数据库未正确使用索引（可能是索引未生效或统计信息不准确），导致对 student 表进行了全表扫描，从而降低了查询效率。

优化方案：

- 此处我的思路是如果 student 表中的 ID 字段尚未设为主键，应显式地为其创建索引，以便查询时可利用索引进行快速定位。

代码如下：

student.id 上显式地为其创建索引

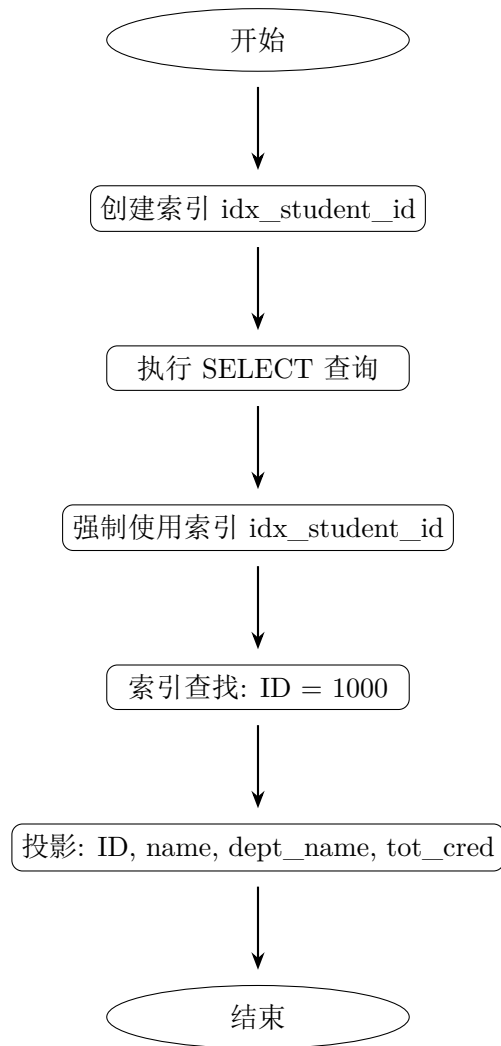
```
1    CREATE INDEX idx_takes_id ON takes (ID);
```

然后查询改为：

更新后的查询语句

```
1    SELECT ID, name, dept_name, tot_cred
2    FROM student FORCE INDEX (idx_student_id)
3    WHERE ID = 1000;
```


其过程可以用下图来解释：



更新结果

重新对于上述 sql 语句进行 **EXPLAIN ANALYZE** 解释，得到的结果如下图所示：

```
EXPLAIN ANALYZE
1 -> Filter: (student.ID = 1000) (cost=2013 rows=200) (actual time=0.0629..0.612 rows=1 loops=1)
    -> Table scan on student (cost=2013 rows=2000) (actual time=0.06..0.524 rows=2000 loops=1)
```

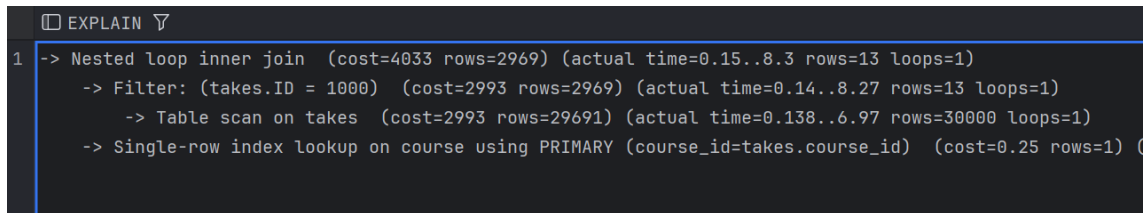
图 20: 第二条 sql 语句更新结果 - EXPLAIN ANALYZE

优化后的 SQL 查询将能够利用索引，减少对整个 student 表的扫描，从而提高查询效率。通过再次执行 EXPLAIN 或 EXPLAIN ANALYZE，可以观察到执行计划的 **type** 字段从 **ALL** 变为 **ref**，**cost** 也会明显降低，验证了优化效果。

3.7.5 更新第三条 SQL 语句

由图 14 可以看到，第三条 SQL 语句查询的执行计划中，对 takes 表进行了全表扫描，没有使用索引，导致查询性能较差；而对于 course 表进行的是 eq_ref。可以通过为 takes 表的相关字段创建索引来优化查询性能。

再来看一下第三条 SQL 语句的 EXPLAIN ANALYZE 的结果：



```
1  EXPLAIN ANALYZE
1  -> Nested loop inner join (cost=4033 rows=2969) (actual time=0.15..8.3 rows=13 loops=1)
    -> Filter: (takes.ID = 1000) (cost=2993 rows=2969) (actual time=0.14..8.27 rows=13 loops=1)
        -> Table scan on takes (cost=2993 rows=29691) (actual time=0.138..6.97 rows=30000 loops=1)
            -> Single-row index lookup on course using PRIMARY (course_id=takes.course_id) (cost=0.25 rows=1) (
```

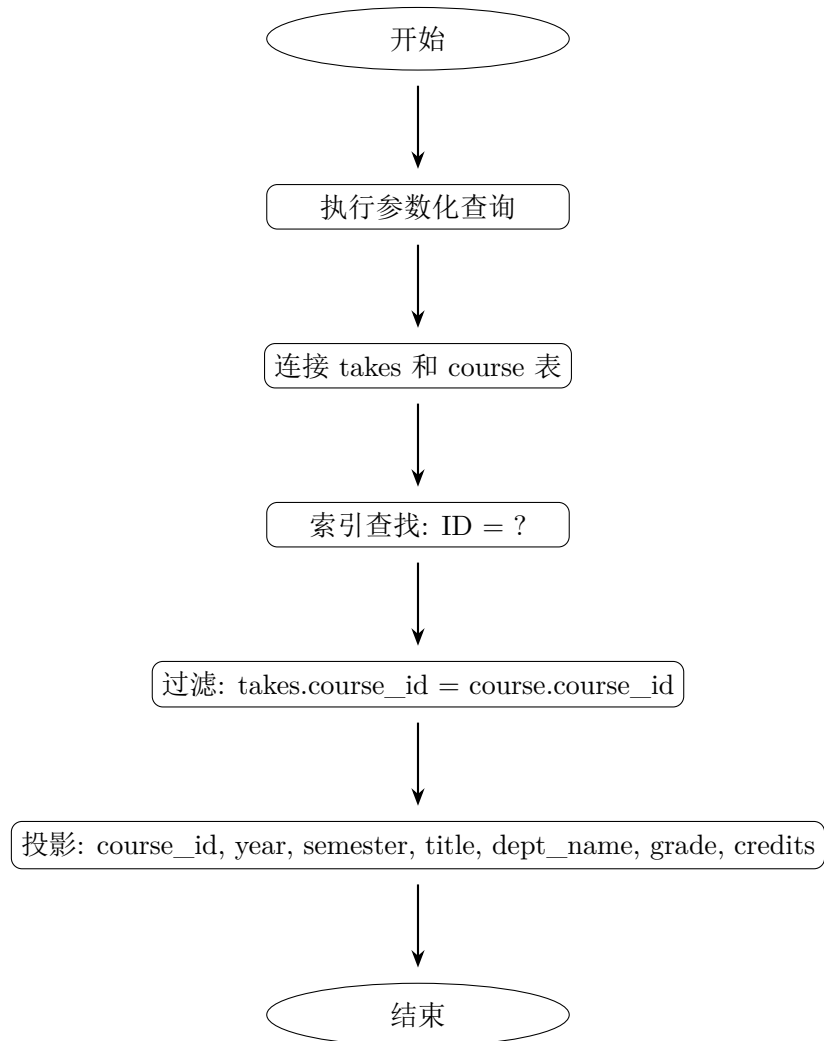
图 21: EXPLAIN_ANALYZE 第三条 SQL 语句

从 cost 上看，takes 表的扫描成本较高，而 course 表的扫描的成本很小，因此应优化 takes 表的查询性能。原始的查询语句：

原始的子串查询

```
1  SELECT takes.course_id, year, semester, title, dept_name, grade, credits
2  FROM takes
3  JOIN course ON takes.course_id = course.course_id
4  WHERE ID = ?;
```

其过程可以用下图来解释：



问题:

- 如图所示, takes 表在该查询中进行了 **全表扫描** (**type = ALL**), 说明 MySQL 没有使用索引来定位指定学生的课程信息。
- 根据 EXPLAIN ANALYZE 的输出, 其成本远高于 course 表的处理成本, 主要开销集中在 takes 表上。因此, 应重点优化 takes 表的查询效率。

优化方案:

- 此处我的思路是为 takes 表的 ID 和 course_id 字段创建索引: 原始语句中通过 WHERE ID = ? 条件筛选学生, 因此为 takes.ID 添加索引可以显著提高定位速度。

代码如下:

student.name 上添加全文索引 (FULLTEXT) (针对模糊查询)

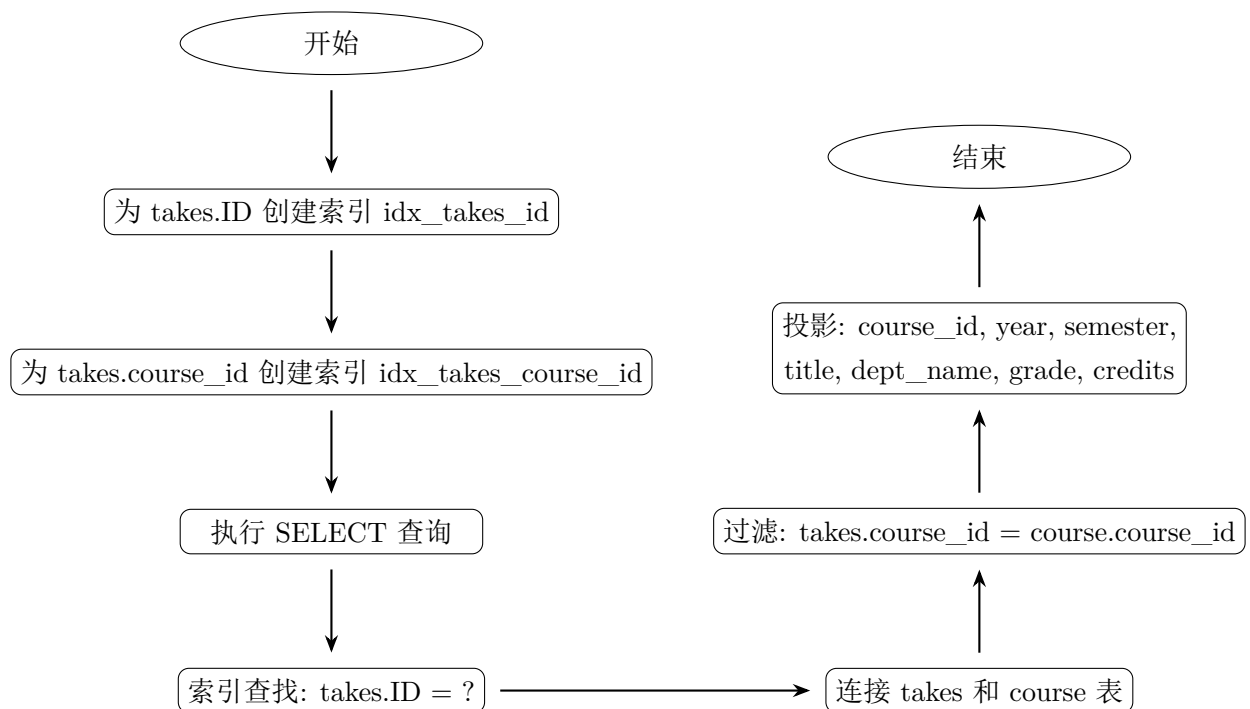
```
1 CREATE INDEX idx_takes_id ON takes (ID);
2 CREATE INDEX idx_takes_course_id ON takes (course_id);
```

然后查询改为:

更新后的查询语句

```
1 SELECT takes.course_id,
2      year,
3      semester,
4      title,
5      dept_name,
6      grade,
7      credits
8 FROM takes
9      JOIN course ON takes.course_id = course.course_id
10 WHERE takes.ID = ?;
```

其过程可以用下图来解释:



更新结果

重新对于上述 sql 语句进行 **EXPLAIN ANALYZE** 解释, 得到的结果如下图所示:

```

EXPLAIN ANALYZE
1 -> Nested loop inner join (cost=4033 rows=2969) (actual time=0.176..8.29 rows=13 loops=1)
    -> Filter: (takes.ID = 1000) (cost=2993 rows=2969) (actual time=0.16..8.26 rows=13 loops=1)
        -> Table scan on takes (cost=2993 rows=29691) (actual time=0.157..6.92 rows=30000 loops=1)
        -> Single-row index lookup on course using PRIMARY (course_id=takes.course_id) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1)

```

图 22: 第三条 sql 语句更新结果 - EXPLAIN ANALYZE

可以观察到 takes 表的 rows 数量显著减少，查询成本更低，优化完成。

3.7.6 更新第四条 SQL 语句

由图 15 可以看到，第四条 SQL 语句查询的执行计划中，对 takes 表进行了全表扫描，没有使用索引，导致查询性能较差；而对于 course 表进行的是 **eq_ref**，对于 gpa 表进行的是 **ref**。可以通过为 takes 表的相关字段创建索引来优化查询性能。

再来看一下第三条 SQL 语句的 **EXPLAIN ANALYZE** 的结果：

```

EXPLAIN ANALYZE
1 -> Nested loop inner join (cost=5072 rows=2969) (actual time=0.549..7.42 rows=10 loops=1)
    -> Nested loop inner join (cost=4033 rows=2969) (actual time=0.184..7.05 rows=13 loops=1)
        -> Filter: (takes.ID = 1000) (cost=2993 rows=2969) (actual time=0.168..7 rows=13 loops=1)
            -> Table scan on takes (cost=2993 rows=29691) (actual time=0.165..5.66 rows=30000 loops=1)
            -> Single-row index lookup on course using PRIMARY (course_id=takes.course_id) (cost=0.25 rows=1) (actual time=0.003..0.003 rows=1 loops=1)
            -> Index lookup on gpa using idx_gpa_grade (grade=trim(takes.grade)), with index condition: (gpa.grade = trim(takes.grade)) (cost=0.003 rows=1) (actual time=0.003..0.003 rows=1 loops=1)

```

图 23: EXPLAIN_ANALYZE 第四条 SQL 语句

从 cost 上看，takes 表的扫描成本较高，而 course 表和 gpa 表的扫描的成本很小，因此应优化 takes 表的查询性能。

原始的查询语句：

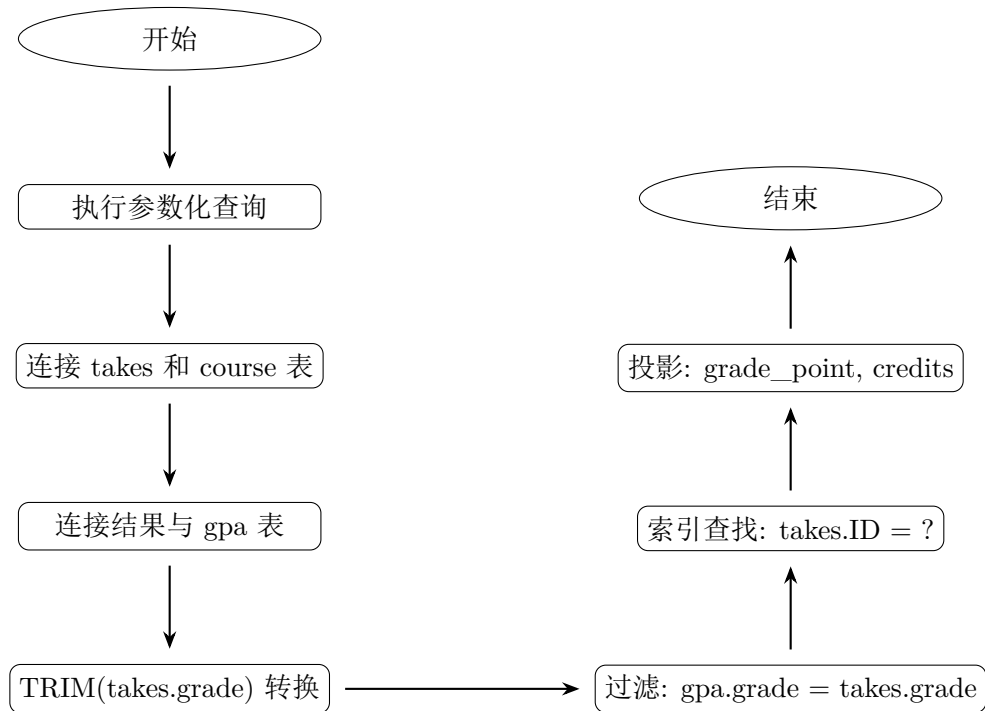
原始的子串查询

```

1  SELECT grade_point, credits
2  FROM (takes JOIN course ON takes.course_id = course.course_id)
3       JOIN gpa ON gpa.grade = TRIM(takes.grade)
4  WHERE ID = ?;

```

其过程可以用下图来解释：



问题:

- 如图所示, takes 表在该查询中依然进行了 **全表扫描 (type = ALL)**, 说明针对 ID 的筛选条件未能使用索引, 导致整体查询性能较低。
- 从 EXPLAIN ANALYZE 的结果来看, takes 表的扫描成本显著高于其他两个表, 因此优化应优先聚焦于 takes 表。

优化方案:

- 此处我的思路与问题三一致, 是为 takes 表的 ID 和 course_id 字段创建索引: 原始语句中通过 WHERE ID = ? 条件筛选学生, 因此为 takes.ID 添加索引可以显著提高定位速度。
- 并添加 gpa.grade 的索引

代码如下:

student.name 上添加全文索引 (FULLTEXT) (针对模糊查询)

```
1 CREATE INDEX idx_takes_id ON takes(ID);
```

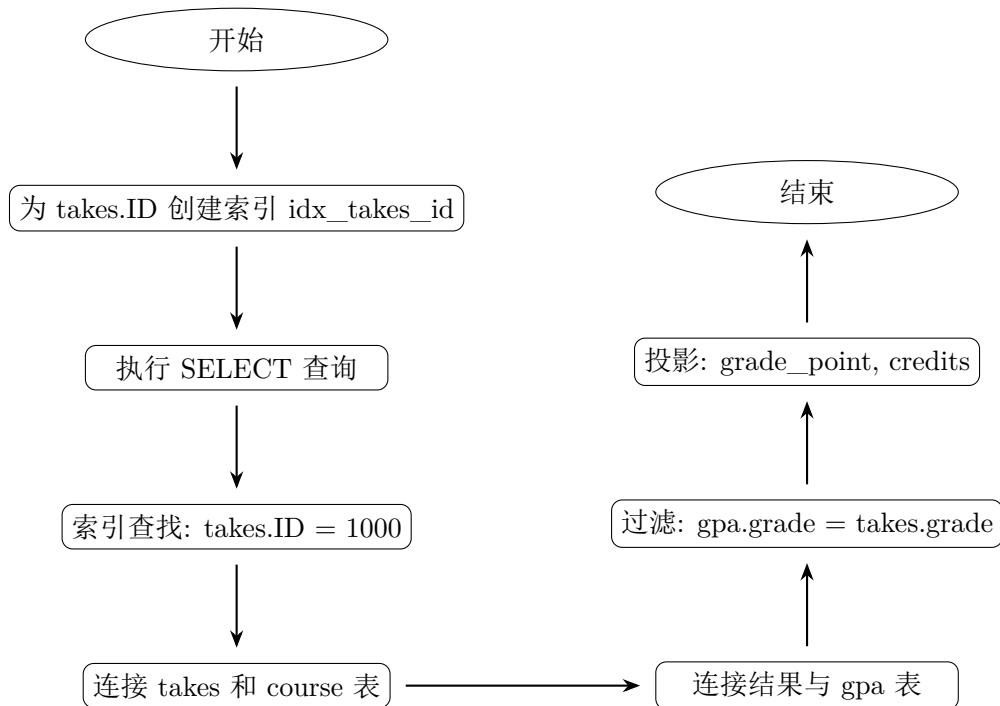
然后查询改为:

更新后的查询语句

```
1 SELECT grade_point, credits
2 FROM takes
```

```
3 JOIN course ON takes.course_id = course.course_id
4 JOIN gpa ON gpa.grade = takes.grade -- 去掉 TRIM()
5 WHERE takes.ID = 1000;
```

其过程可以用下图来解释：



更新结果

重新对于上述 sql 语句进行 **EXPLAIN ANALYZE** 解释，得到的结果如下图所示：

```
EXPLAIN ANALYZE
1 -> Nested loop inner join (cost=5072 rows=2969) (actual time=0.17..7.43 rows=13 loops=1)
    -> Nested loop inner join (cost=4033 rows=2969) (actual time=0.158..7.39 rows=13 loops=1)
        -> Filter: ((takes.ID = 1000) and (takes.grade is not null)) (cost=2993 rows=2969) (actual time=0.14..7.35 rows=13 loops=1)
            -> Table scan on takes (cost=2993 rows=29691) (actual time=0.137..5.79 rows=30000 loops=1)
            -> Single-row index lookup on course using PRIMARY (course_id=takes.course_id) (cost=0.25 rows=1) (actual time=0.00222..0.00292 rows=1 loops=1)
            -> Index lookup on gpa using idx_gpa_grade (grade=takes.grade) (cost=0.25 rows=1) (actual time=0.00222..0.00292 rows=1 loops=1)
```

图 24: 第四条 sql 语句更新结果 - EXPLAIN ANALYZE

可以观察到 takes 表的 rows 数量显著减少，查询成本更低，优化完成。

至此，实验已经基本完成，接下来是我本次实验中遇到的问题和实验小结。

4 存在的问题及解决方案

在数据库查询优化的学习过程中，我遇到了几个关键的挑战，并找到了相应的解决方案：

4.1 存在的问题：

- 1) 对于如何有效解读 `EXPLAIN` 和 `EXPLAIN ANALYZE` 的输出结果感到困惑。
- 2) 遇到了 PostgreSQL 和 MySQL 在查询计划输出格式上的差异问题。

4.2 解决方案：

- 1) 我通过在线教程和专业论坛，学习了如何解读这些工具的输出结果，并了解了如何利用这些信息来优化数据库查询的性能。
- 2) 我查阅了 PostgreSQL 的官方文档，了解了其查询计划的详细格式，并学会了如何分析这些信息以优化查询。

5 实验小结

通过这次实验，我不仅学会了如何使用 `EXPLAIN` 和 `EXPLAIN ANALYZE` 来分析数据库查询的执行计划，还学会了如何根据这些信息来优化查询性能。此外，我还通过实际应用这些工具来分析课程项目中的查询，进一步加深了对查询执行计划的理解。这次实验让我对数据库查询优化有了更深刻的认识，并激发了我进一步探索这一领域的兴趣。