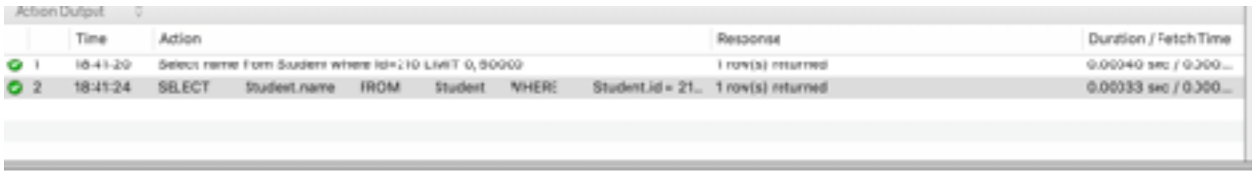
1. List the name of the student with id equal to v1 (id) - Sahil Patel

Description:

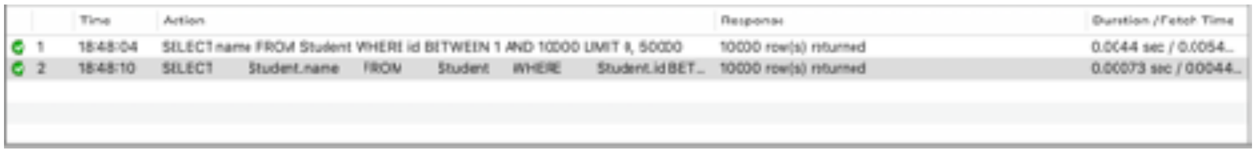


```
Action Output
     Time      Action                                                   Response              Duration / Fetch Time
●  1 18:41:20  Select name Form Student where id=210 LIMIT 0, 50000     1 row(s) returned     0.00040 sec / 0.000...
●  2 18:41:24  SELECT   Student.name   FROM   Student   WHERE   Student.id = 21...  1 row(s) returned     0.00033 sec / 0.000...
```

Since you cant have a graph with just two data pieces, I have attached a screenshot of the output duration and fetch time to the queries before tuning and after tuning. As for question one, it is a simple select statement. However, as you can see, there is still a difference in duration/efficiency of the two queries by 0.00007 seconds. This is because my sql syntax was a little off as I did not do "Student.name" instead I just did name. This, according to the MySQL guide is not too worrisome. However, when dealing with more complex sql queries, it is best to use the correct syntax for efficiency and easy readability.

2. List the names of students with id in the range of v2 (id) to v3 (inclusive) - Sahil Patel

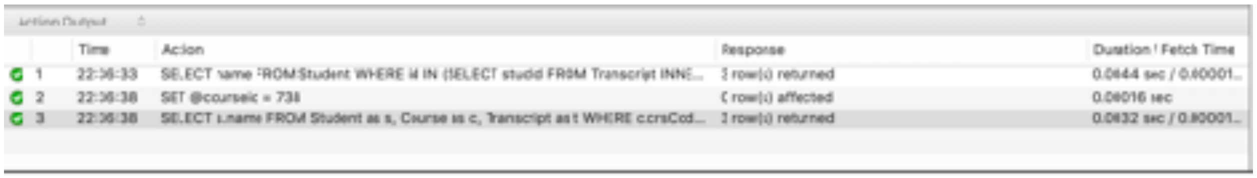
Description:



```
     Time      Action                                                          Response                 Duration / Fetch Time
●  1 18:48:04  SELECT name FROM Student WHERE id BETWEEN 1 AND 10000 LIMIT 0, 50000   10000 row(s) returned   0.0044 sec / 0.0054...
●  2 18:48:10  SELECT   Student.name   FROM   Student   WHERE   Student.id BET...      10000 row(s) returned   0.00073 sec / 0.0044...
```

As for question two asking us to list the names of students with id in the range of two values, my approach was to use the BETWEEN keyword which gets the data between two values. However, as stated above for question one, syntax plays an important role in efficiency of queries. You can see here that getting the range of data between 1-10000 has a drastic effect on duration of the query as query 1 (untuned) duration is 0.0044 seconds while query 2's duration is 0.00073 (tuned). This difference can have a major impact when dealing with live data with over 10s of thousands of data being handled. I realized that there is a major difference between "Student.name" and name. In other words, adding the table name can increase the efficiency.

3. List the names of students who have taken course v4 (crsCode) - Sahil Patel

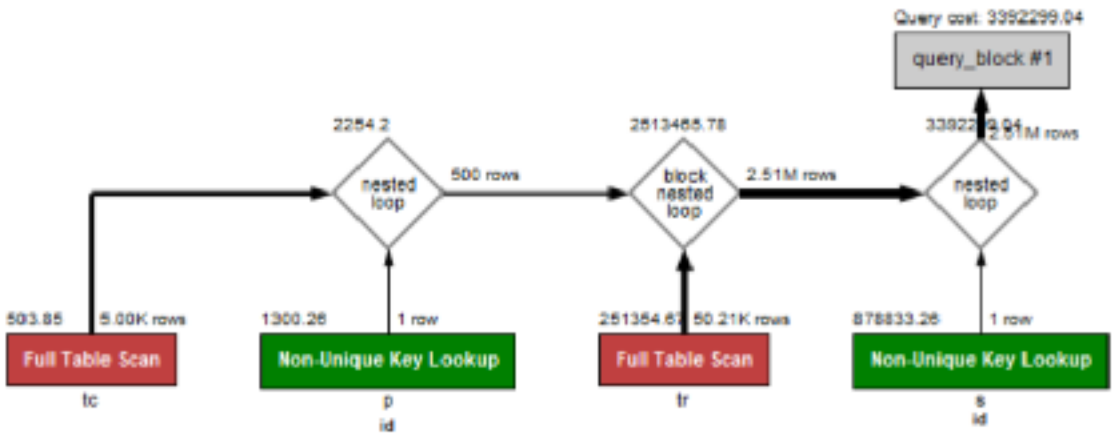


Action Output

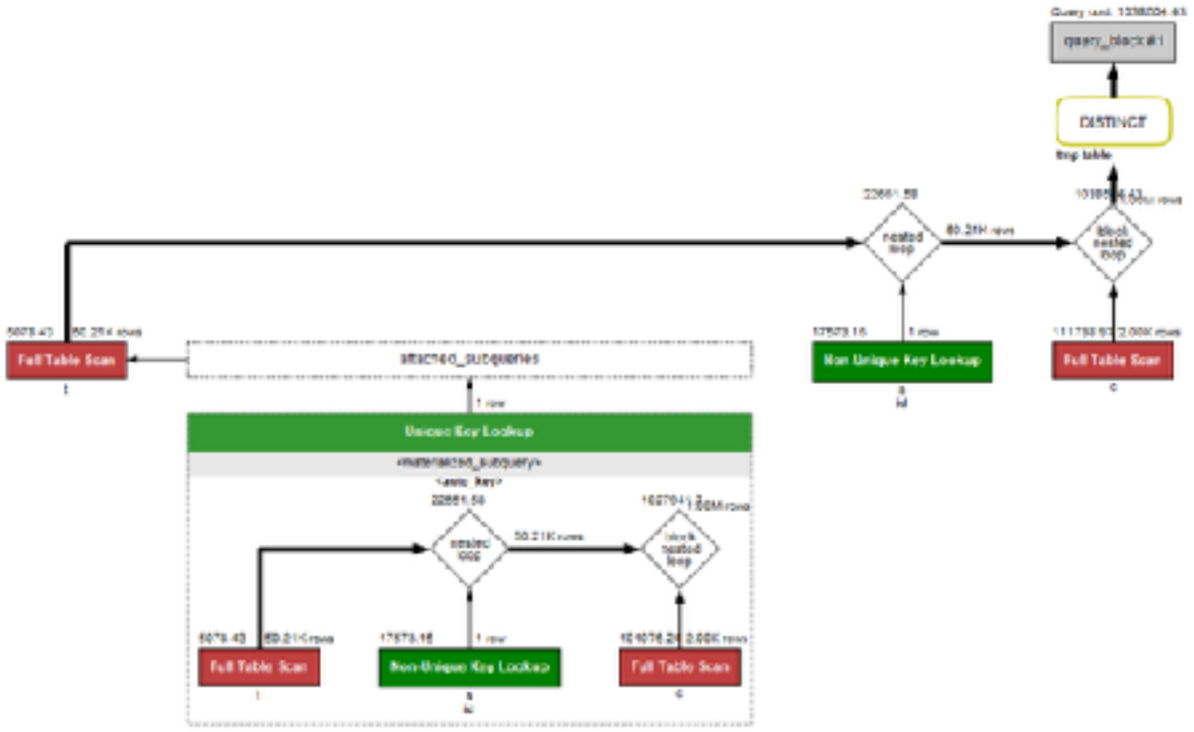
		Time	Action	Response	Duration / Fetch Time
✓	1	22:36:33	SELECT name FROM Student WHERE id IN (SELECT studId FROM Transcript INNE...	3 row(s) returned	0.0644 sec / 0.00001...
✓	2	22:36:38	SET @courseid = 738	0 row(s) affected	0.00016 sec
✓	3	22:36:38	SELECT s.name FROM Student as s, Course as c, Transcript as t WHERE c.crsCod...	3 row(s) returned	0.0632 sec / 0.00001...

Description: For this query, my approach was much different than the tuned query. My approach was to use a subquery and use the EXISTS. I used EXISTS instead of IN because I was taught that it makes the query more efficient. I used a subquery because it was easier for me to use. However, after this question, I realized that subqueries are not efficient at all. They take longer because there are more queries to run. After doing some research, I realized that setting variables (course id) makes not only the code more efficient (in this case), however even easier to read. The Query was more efficient because there were no subqueries involved and just a time join.

4. Query 4 - Alec



Query 5


Query cost: 1338004.43
query_block #1
DISTINCT
tmp table
1018554.43
1.00M rows
22661.58
nested loop
60.21K rows
block nested loop
5078.43
60.21K rows
Full Table Scan
t
attached_subqueries
17573.16
1 row
Non-Unique Key Lookup
s
id
1117938.93
2.00K rows
Full Table Scan
c
1 row
Unique Key Lookup
<materialized_subquery>
<auto_key>
22661.58
1027041.2
1.00M rows
nested loop
60.21K rows
block nested loop
5078.43
60.21K rows
Full Table Scan
t
17573.16
1 row
Non-Unique Key Lookup
s
id
1019076.28
2.00K rows
Full Table Scan
c