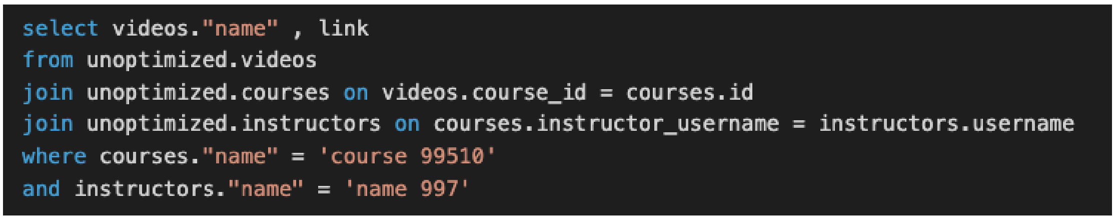
1.Query Statistics

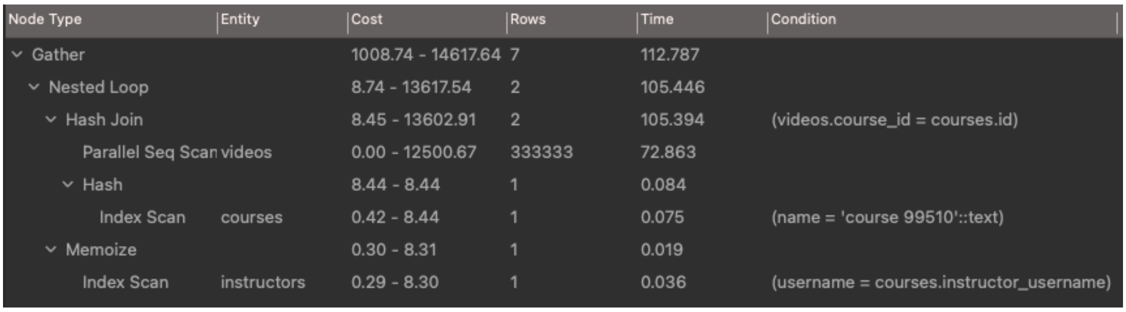
● Query 1

Select videos names and links for a specific course and instructor

○ Before

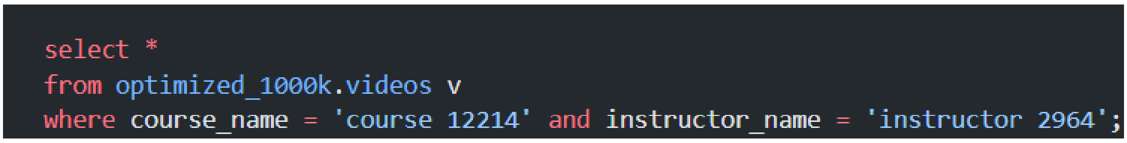


```sql
select videos."name" , link
from unoptimized.videos
join unoptimized.courses on videos.course_id = courses.id
join unoptimized.instructors on courses.instructor_username = instructors.username
where courses."name" = 'course 99510'
and instructors."name" = 'name 997'
```

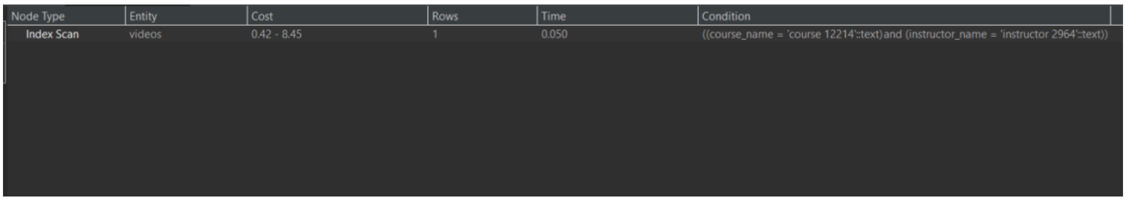


Node Type	Entity	Cost	Rows	Time	Condition
⌄ Gather		1008.74 - 14617.64	7	112.787	
⌄ Nested Loop		8.74 - 13617.54	2	105.446	
⌄ Hash Join		8.45 - 13602.91	2	105.394	(videos.course_id = courses.id)
Parallel Seq Scan	videos	0.00 - 12500.67	333333	72.863	
⌄ Hash		8.44 - 8.44	1	0.084	
Index Scan	courses	0.42 - 8.44	1	0.075	(name = 'course 99510'::text)
⌄ Memoize		0.30 - 8.31	1	0.019	
Index Scan	instructors	0.29 - 8.30	1	0.036	(username = courses.instructor_username)

○ After



```sql
select *
from optimized_1000k.videos v
where course_name = 'course 12214' and instructor_name = 'instructor 2964';
```

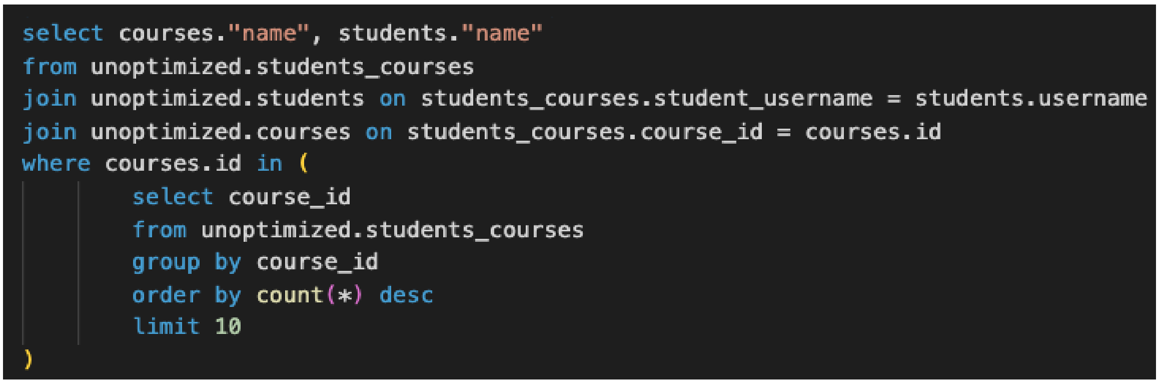


Node Type	Entity	Cost	Rows	Time	Condition
Index Scan	videos	0.42 - 8.45	1	0.050	((course_name = 'course 12214'::text)and (instructor_name = 'instructor 2964'::text))

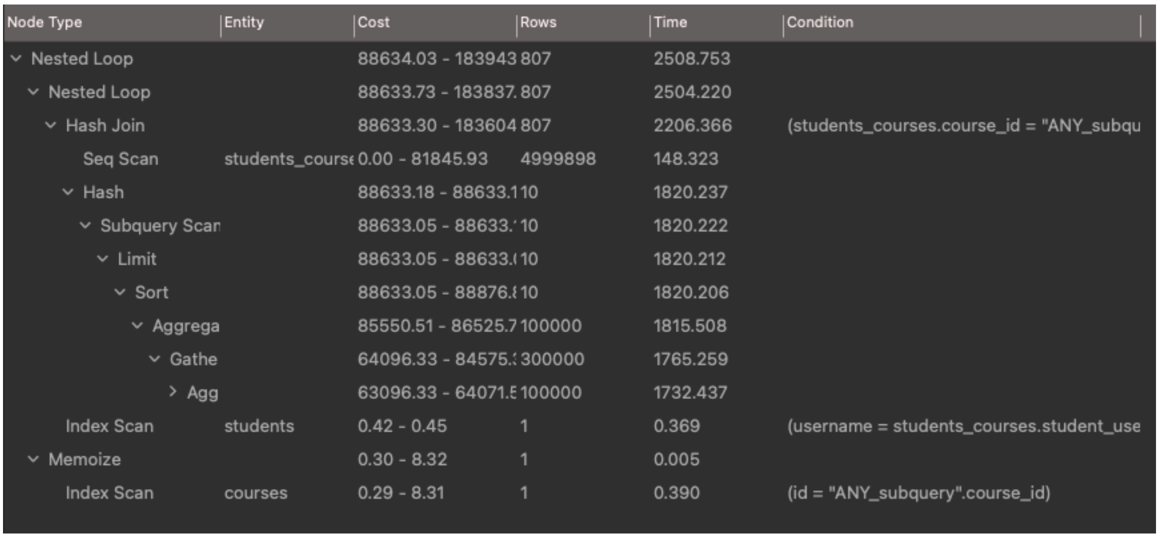
- **Query 2**

Select the names of the most 10 popular courses (the courses that have the most students enrolled in them) and the names of the students who enrolled in them.

- **Before**

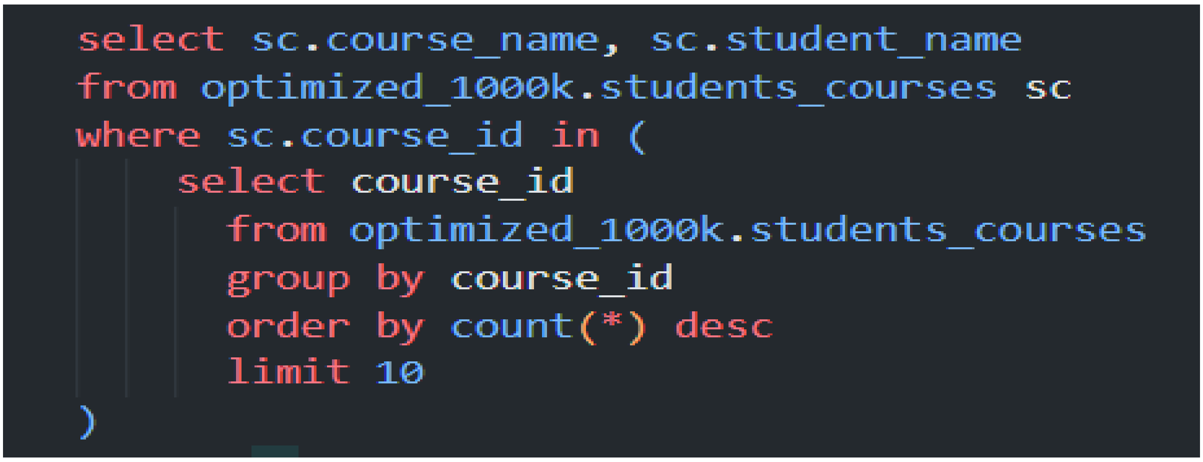


```
select courses."name", students."name"
from unoptimized.students_courses
join unoptimized.students on students_courses.student_username = students.username
join unoptimized.courses on students_courses.course_id = courses.id
where courses.id in (
        select course_id
        from unoptimized.students_courses
        group by course_id
        order by count(*) desc
        limit 10
)
```

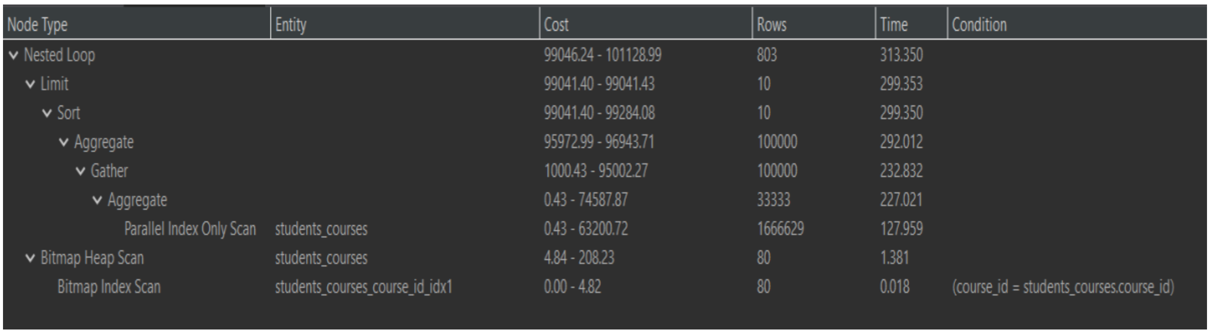


Node Type	Entity	Cost	Rows	Time	Condition
Nested Loop		88634.03 - 183943	807	2508.753	
Nested Loop		88633.73 - 183837.	807	2504.220	
Hash Join		88633.30 - 183604	807	2206.366	(students_courses.course_id = "ANY_subqu
Seq Scan	students_course	0.00 - 81845.93	4999898	148.323	
Hash		88633.18 - 88633.1	10	1820.237	
Subquery Scan		88633.05 - 88633.	10	1820.222	
Limit		88633.05 - 88633.	10	1820.212	
Sort		88633.05 - 88876.	10	1820.206	
Aggrega		85550.51 - 86525.7	100000	1815.508	
Gathe		64096.33 - 84575.	300000	1765.259	
Agg		63096.33 - 64071.5	100000	1732.437	
Index Scan	students	0.42 - 0.45	1	0.369	(username = students_courses.student_use
Memoize		0.30 - 8.32	1	0.005	
Index Scan	courses	0.29 - 8.31	1	0.390	(id = "ANY_subquery".course_id)

- **After**



```
select sc.course_name, sc.student_name
from optimized_1000k.students_courses sc
where sc.course_id in (
    select course_id
      from optimized_1000k.students_courses
      group by course_id
      order by count(*) desc
      limit 10
)
```

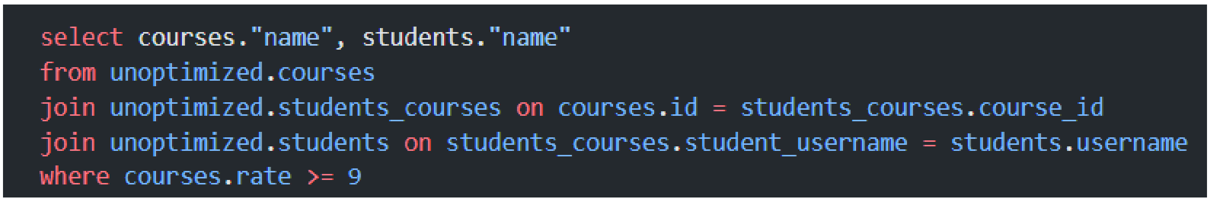


Node Type	Entity	Cost	Rows	Time	Condition
⌄ Nested Loop		99046.24 - 101128.99	803	313.350	
⌄ Limit		99041.40 - 99041.43	10	299.353	
⌄ Sort		99041.40 - 99284.08	10	299.350	
⌄ Aggregate		95972.99 - 96943.71	100000	292.012	
⌄ Gather		1000.43 - 95002.27	100000	232.832	
⌄ Aggregate		0.43 - 74587.87	33333	227.021	
Parallel Index Only Scan	students_courses	0.43 - 63200.72	1666629	127.959	
⌄ Bitmap Heap Scan	students_courses	4.84 - 208.23	80	1.381	
Bitmap Index Scan	students_courses_course_id_idx1	0.00 - 4.82	80	0.018	(course_id = students_courses.course_id)

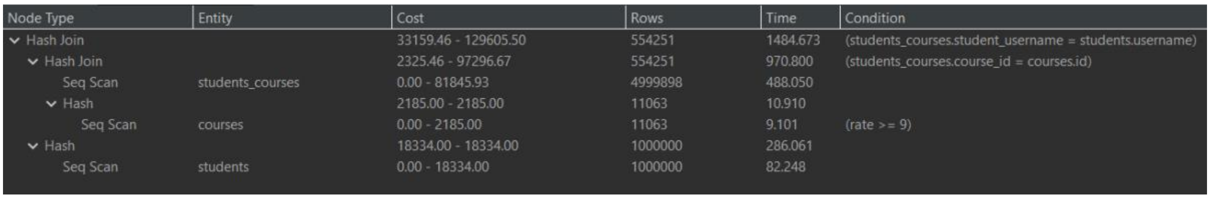
- ## Query 3

Select all the courses that have a rate >= 9 and the names of the students who enrolled in them

- ○ **Before**

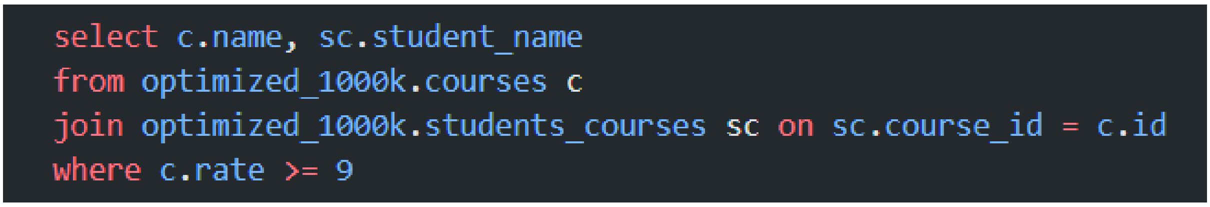


```
select courses."name", students."name"
from unoptimized.courses
join unoptimized.students_courses on courses.id = students_courses.course_id
join unoptimized.students on students_courses.student_username = students.username
where courses.rate >= 9
```

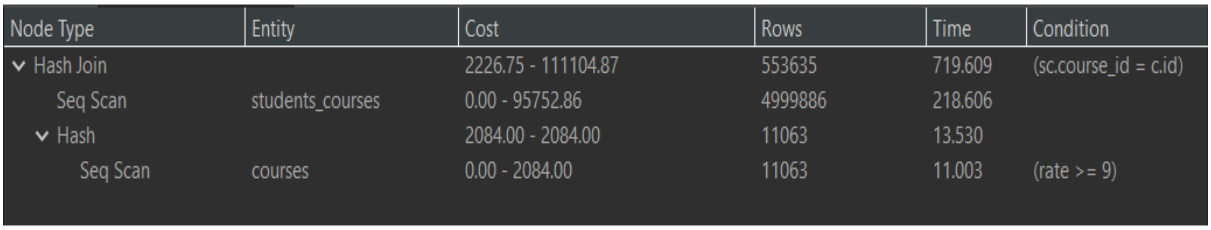


Node Type	Entity	Cost	Rows	Time	Condition
⌄ Hash Join		33159.46 - 129605.50	554251	1484.673	(students_courses.student_username = students.username)
⌄ Hash Join		2325.46 - 97296.67	554251	970.800	(students_courses.course_id = courses.id)
Seq Scan	students_courses	0.00 - 81845.93	4999898	488.050	
⌄ Hash		2185.00 - 2185.00	11063	10.910	
Seq Scan	courses	0.00 - 2185.00	11063	9.101	(rate >= 9)
⌄ Hash		18334.00 - 18334.00	1000000	286.061	
Seq Scan	students	0.00 - 18334.00	1000000	82.248	

- After



```
select c.name, sc.student_name
from optimized_1000k.courses c
join optimized_1000k.students_courses sc on sc.course_id = c.id
where c.rate >= 9
```

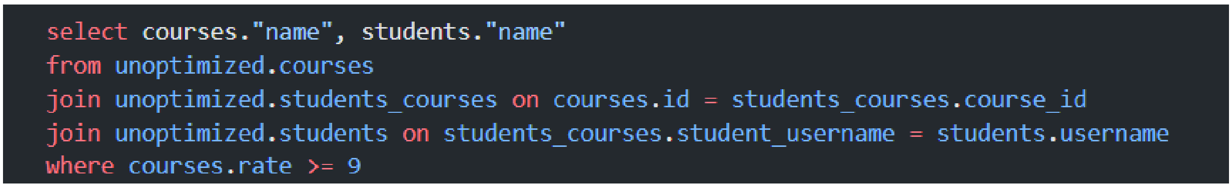


Node Type	Entity	Cost	Rows	Time	Condition
Hash Join		2226.75 - 111104.87	553635	719.609	(sc.course_id = c.id)
Seq Scan	students_courses	0.00 - 95752.86	4999886	218.606	
Hash		2084.00 - 2084.00	11063	13.530	
Seq Scan	courses	0.00 - 2084.00	11063	11.003	(rate >= 9)

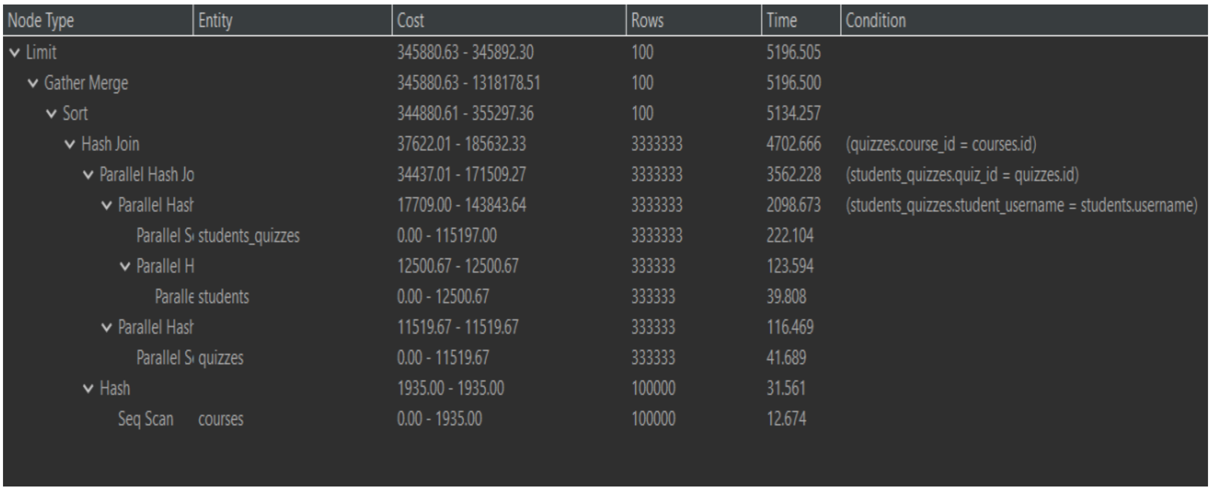
- ## Query 4

Select the names of the top 100 students in quizzes score and the courses they got the highest score in

- Before

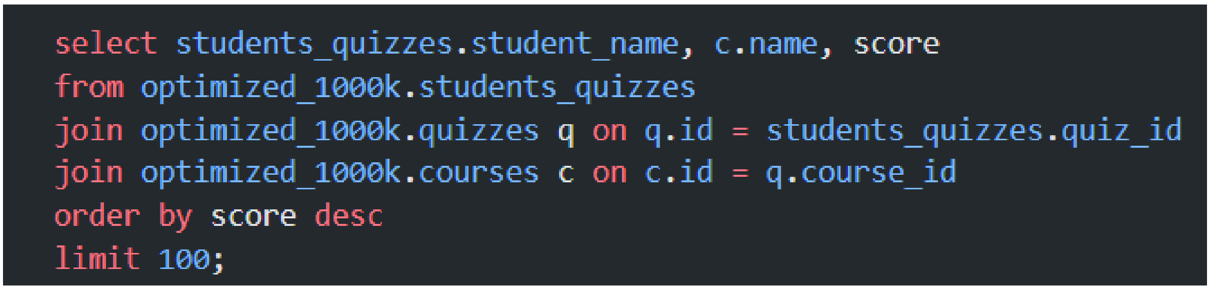


```
select courses."name", students."name"
from unoptimized.courses
join unoptimized.students_courses on courses.id = students_courses.course_id
join unoptimized.students on students_courses.student_username = students.username
where courses.rate >= 9
```

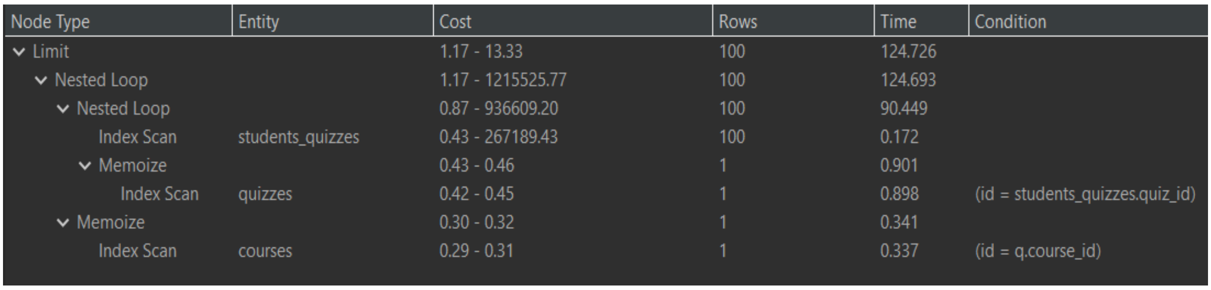


Node Type	Entity	Cost	Rows	Time	Condition
Limit		345880.63 - 345892.30	100	5196.505	
Gather Merge		345880.63 - 1318178.51	100	5196.500	
Sort		344880.61 - 355297.36	100	5134.257	
Hash Join		37622.01 - 185632.33	3333333	4702.666	(quizzes.course_id = courses.id)
Parallel Hash Jo		34437.01 - 171509.27	3333333	3562.228	(students_quizzes.quiz_id = quizzes.id)
Parallel Has		17709.00 - 143843.64	3333333	2098.673	(students_quizzes.student_username = students.username)
Parallel S	students_quizzes	0.00 - 115197.00	3333333	222.104	
Parallel H		12500.67 - 12500.67	333333	123.594	
Paralle	students	0.00 - 12500.67	333333	39.808	
Parallel Has		11519.67 - 11519.67	333333	116.469	
Parallel S	quizzes	0.00 - 11519.67	333333	41.689	
Hash		1935.00 - 1935.00	100000	31.561	
Seq Scan	courses	0.00 - 1935.00	100000	12.674	

- **After**



```sql
select students_quizzes.student_name, c.name, score
from optimized_1000k.students_quizzes
join optimized_1000k.quizzes q on q.id = students_quizzes.quiz_id
join optimized_1000k.courses c on c.id = q.course_id
order by score desc
limit 100;
```

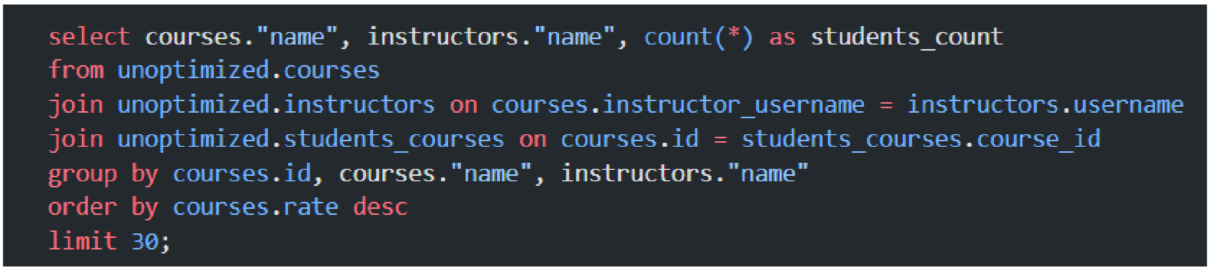


Node Type	Entity	Cost	Rows	Time	Condition
Limit		1.17 - 13.33	100	124.726	
Nested Loop		1.17 - 1215525.77	100	124.693	
Nested Loop		0.87 - 936609.20	100	90.449	
Index Scan	students_quizzes	0.43 - 267189.43	100	0.172	
Memoize		0.43 - 0.46	1	0.901	
Index Scan	quizzes	0.42 - 0.45	1	0.898	(id = students_quizzes.quiz_id)
Memoize		0.30 - 0.32	1	0.341	
Index Scan	courses	0.29 - 0.31	1	0.337	(id = q.course_id)

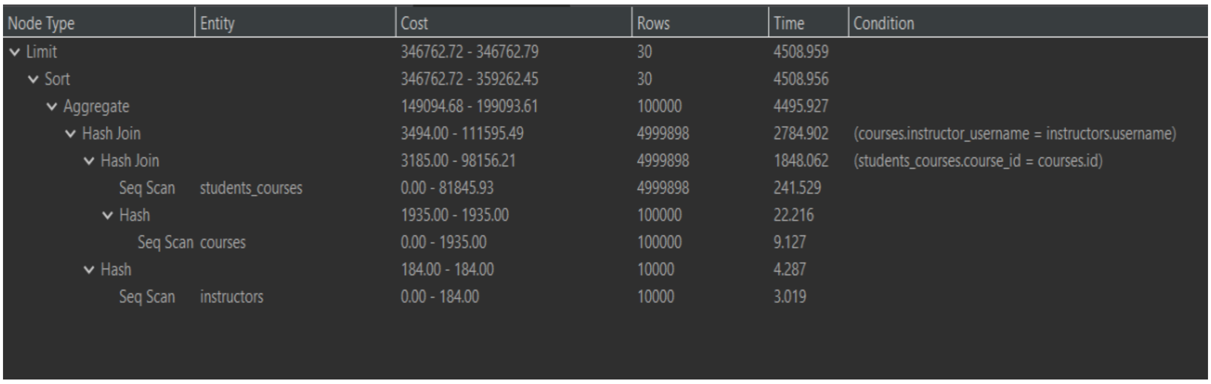
Query 5

Select the top rated 30 course with their instructors names and the number of students enrolled in them

- **Before**

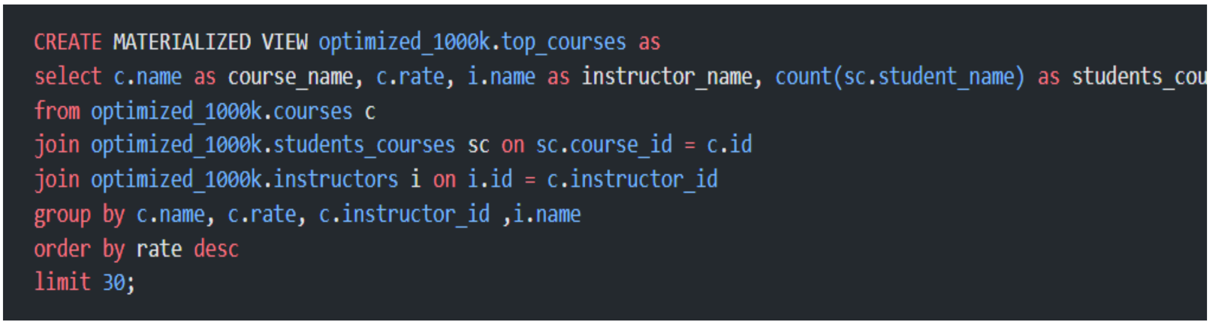


```sql
select courses."name", instructors."name", count(*) as students_count
from unoptimized.courses
join unoptimized.instructors on courses.instructor_username = instructors.username
join unoptimized.students_courses on courses.id = students_courses.course_id
group by courses.id, courses."name", instructors."name"
order by courses.rate desc
limit 30;
```

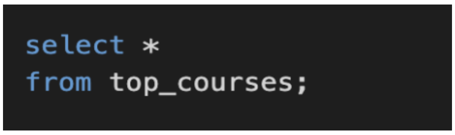


Node Type	Entity	Cost	Rows	Time	Condition
Limit		346762.72 - 346762.79	30	4508.959	
Sort		346762.72 - 359262.45	30	4508.956	
Aggregate		149094.68 - 199093.61	100000	4495.927	
Hash Join		3494.00 - 111595.49	4999898	2784.902	(courses.instructor_username = instructors.username)
Hash Join		3185.00 - 98156.21	4999898	1848.062	(students_courses.course_id = courses.id)
Seq Scan	students_courses	0.00 - 81845.93	4999898	241.529	
Hash		1935.00 - 1935.00	100000	22.216	
Seq Scan	courses	0.00 - 1935.00	100000	9.127	
Hash		184.00 - 184.00	10000	4.287	
Seq Scan	instructors	0.00 - 184.00	10000	3.019	

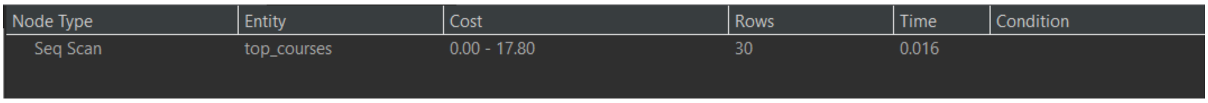
- **After**



```
CREATE MATERIALIZED VIEW optimized_1000k.top_courses as
select c.name as course_name, c.rate, i.name as instructor_name, count(sc.student_name) as students_cou
from optimized_1000k.courses c
join optimized_1000k.students_courses sc on sc.course_id = c.id
join optimized_1000k.instructors i on i.id = c.instructor_id
group by c.name, c.rate, c.instructor_id ,i.name
order by rate desc
limit 30;
```



```
select *
from top_courses;
```

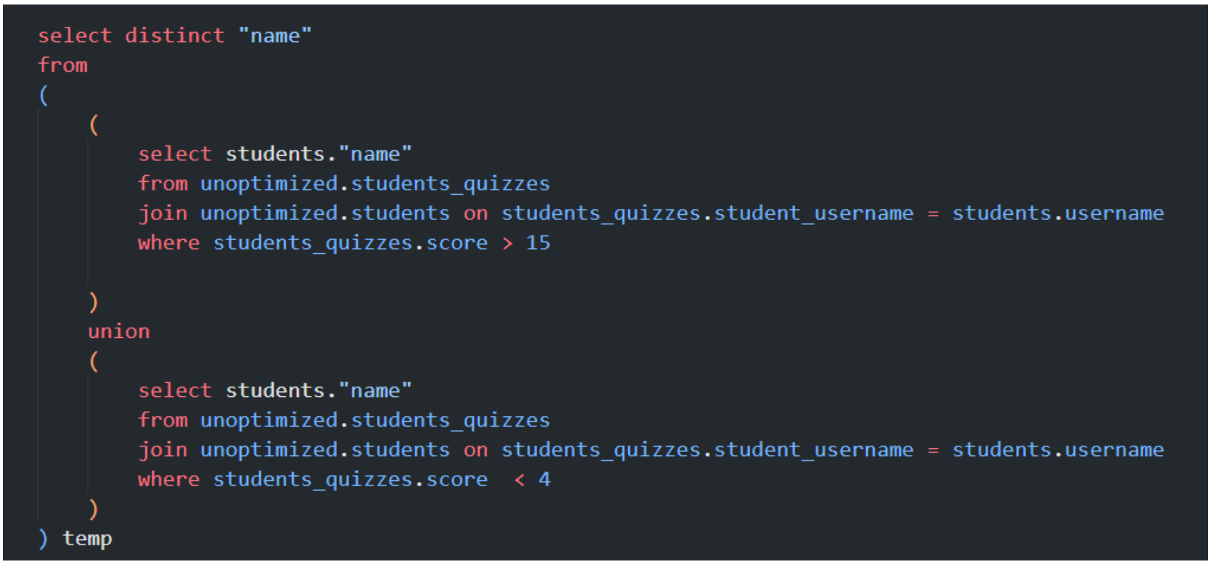


Node Type	Entity	Cost	Rows	Time	Condition
Seq Scan	top_courses	0.00 - 17.80	30	0.016	

- ## Query 6

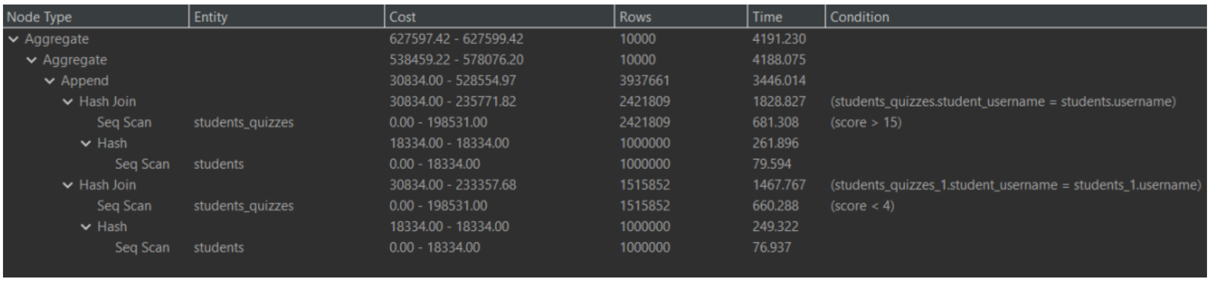
Select student names who got scores in quizzes > 15 or < 5

- **Before**



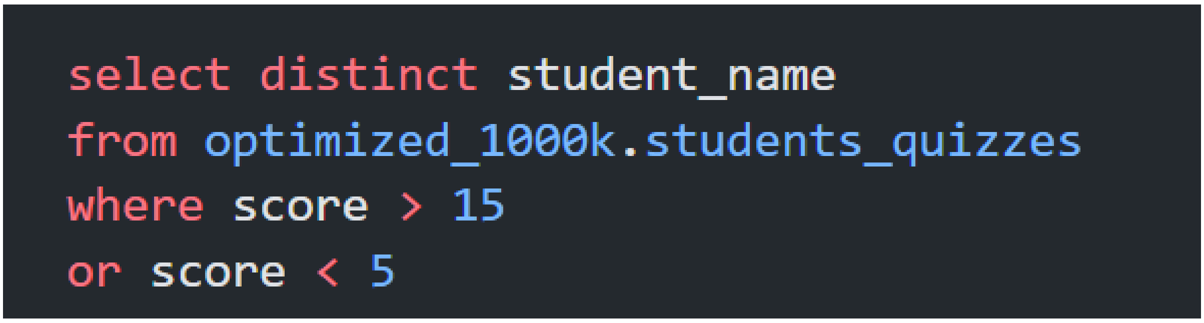
```
select distinct "name"
from
(
    (
        select students."name"
        from unoptimized.students_quizzes
        join unoptimized.students on students_quizzes.student_username = students.username
        where students_quizzes.score > 15

    )
    union
    (
        select students."name"
        from unoptimized.students_quizzes
        join unoptimized.students on students_quizzes.student_username = students.username
        where students_quizzes.score  < 4
    )
) temp
```

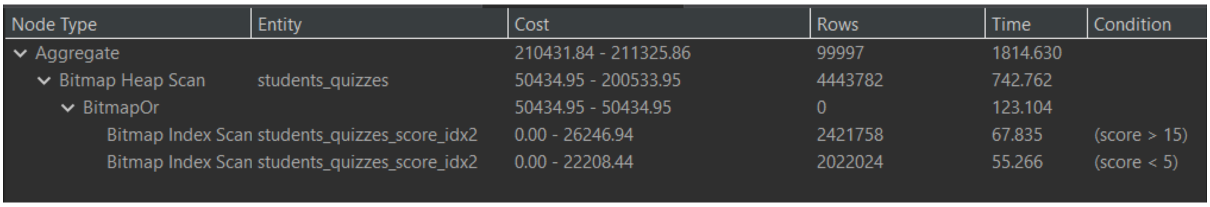


Node Type	Entity	Cost	Rows	Time	Condition
⌄ Aggregate		627597.42 - 627599.42	10000	4191.230	
⌄ Aggregate		538459.22 - 578076.20	10000	4188.075	
⌄ Append		30834.00 - 528554.97	3937661	3446.014	
⌄ Hash Join		30834.00 - 235771.82	2421809	1828.827	(students_quizzes.student_username = students.username)
Seq Scan	students_quizzes	0.00 - 198531.00	2421809	681.308	(score > 15)
⌄ Hash		18334.00 - 18334.00	1000000	261.896	
Seq Scan	students	0.00 - 18334.00	1000000	79.594	
⌄ Hash Join		30834.00 - 233357.68	1515852	1467.767	(students_quizzes_1.student_username = students_1.username)
Seq Scan	students_quizzes	0.00 - 198531.00	1515852	660.288	(score < 4)
⌄ Hash		18334.00 - 18334.00	1000000	249.322	
Seq Scan	students	0.00 - 18334.00	1000000	76.937	

- After1

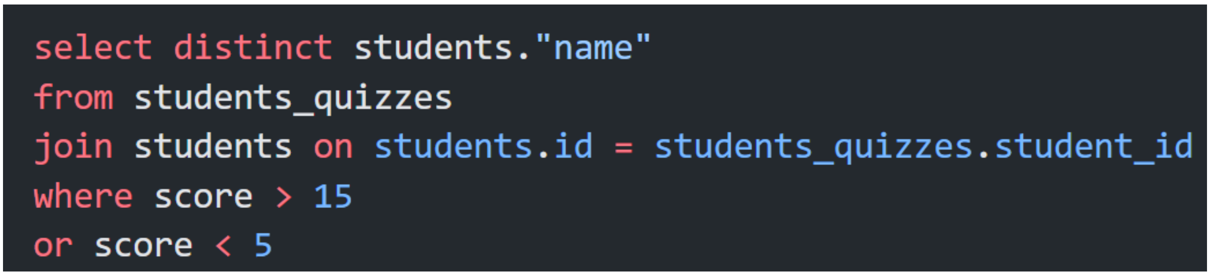


```sql
select distinct student_name
from optimized_1000k.students_quizzes
where score > 15
or score < 5
```

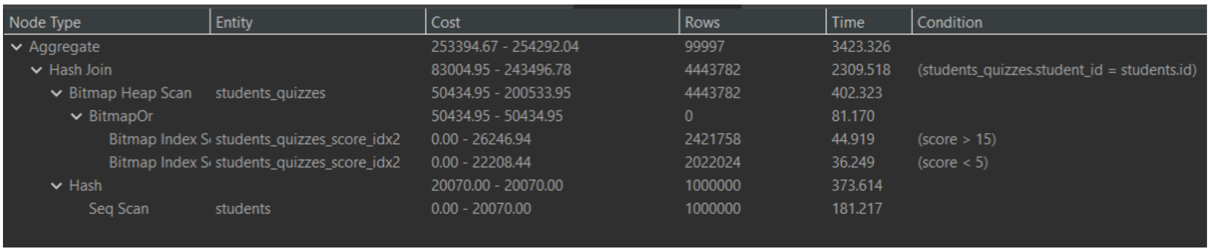


Node Type	Entity	Cost	Rows	Time	Condition
Aggregate		210431.84 - 211325.86	99997	1814.630	
Bitmap Heap Scan	students_quizzes	50434.95 - 200533.95	4443782	742.762	
BitmapOr		50434.95 - 50434.95	0	123.104	
Bitmap Index Scan	students_quizzes_score_idx2	0.00 - 26246.94	2421758	67.835	(score > 15)
Bitmap Index Scan	students_quizzes_score_idx2	0.00 - 22208.44	2022024	55.266	(score < 5)

- After 2



```sql
select distinct students."name"
from students_quizzes
join students on students.id = students_quizzes.student_id
where score > 15
or score < 5
```



Node Type	Entity	Cost	Rows	Time	Condition
Aggregate		253394.67 - 254292.04	99997	3423.326	
Hash Join		83004.95 - 243496.78	4443782	2309.518	(students_quizzes.student_id = students.id)
Bitmap Heap Scan	students_quizzes	50434.95 - 200533.95	4443782	402.323	
BitmapOr		50434.95 - 50434.95	0	81.170	
Bitmap Index S	students_quizzes_score_idx2	0.00 - 26246.94	2421758	44.919	(score > 15)
Bitmap Index S	students_quizzes_score_idx2	0.00 - 22208.44	2022024	36.249	(score < 5)
Hash		20070.00 - 20070.00	1000000	373.614	
Seq Scan	students	0.00 - 20070.00	1000000	181.217	

2.Optimizations Details

A. Database Statistics

- **Before Optimizations**

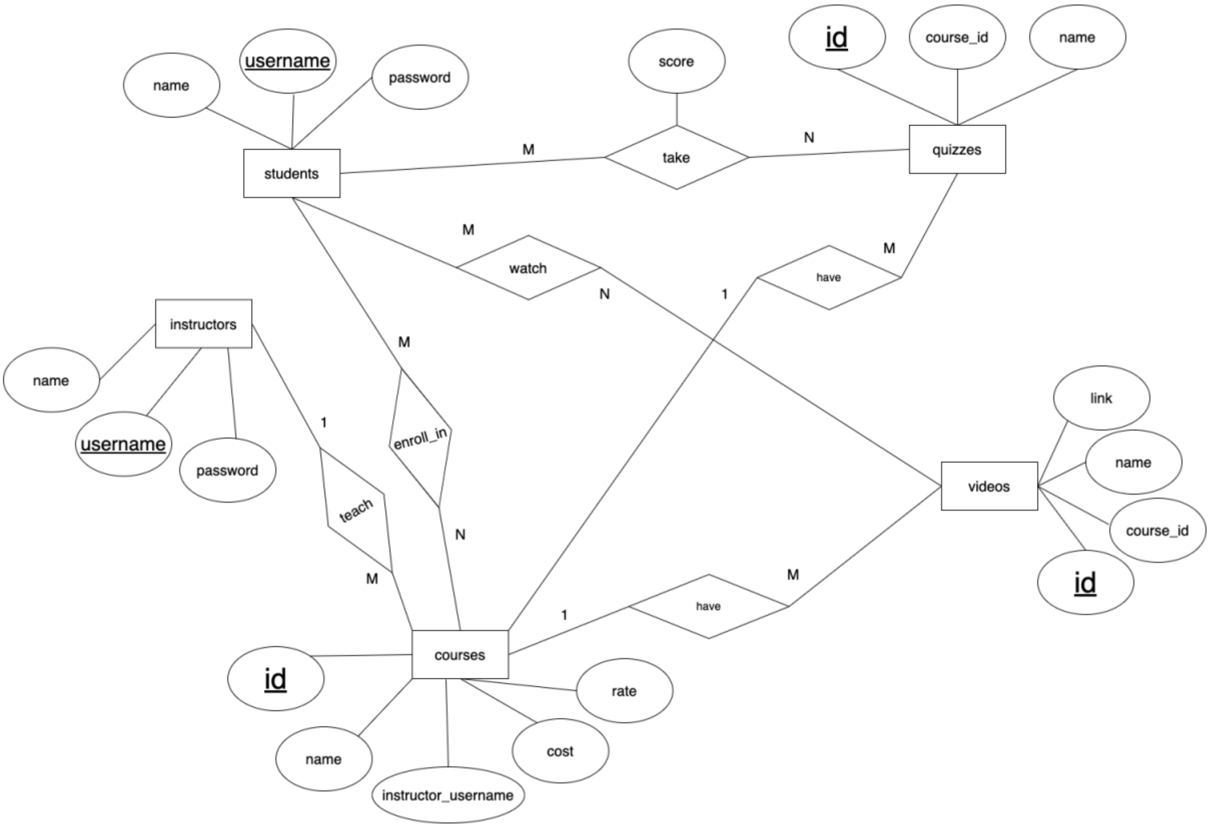
Table Name	Row Count	Max Row Size (Bytes)	Table Size (MB)	Number of Indexes	Number of Foreign Keys
students	1,000,000	40	116	1	0
instructors	10,000	36	1.2	1	0
courses	100,000	43	14	2	1
quizzes	1,000,000	39	86	1	1
videos	1,000,000	52	153	2	1
students_courses	5,000,000	24	504	1	2
students_videos	7,000,000	24	692	1	2
students_quizzes	10,000,000	28	1100	1	2

- **After Optimizations**

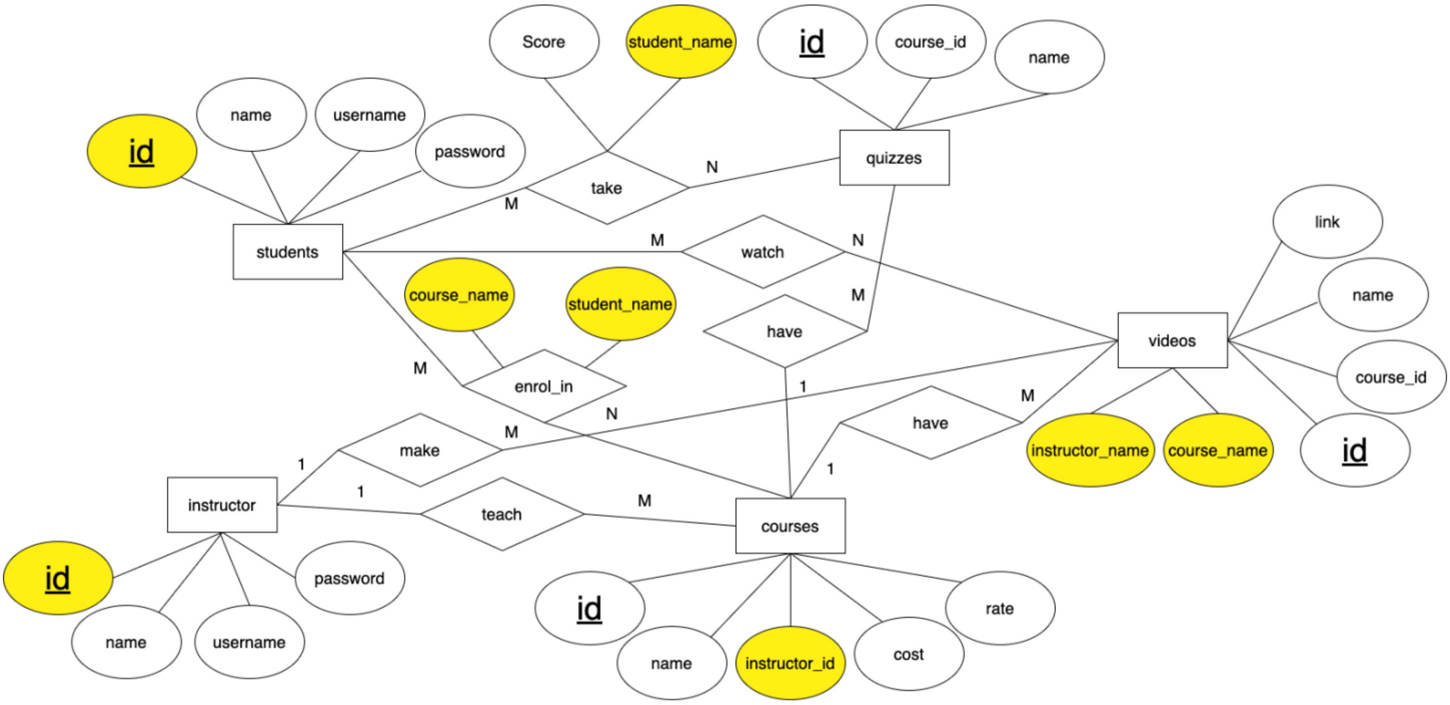
Table Name	Row Count	Max Row Size (Bytes)	Table Size (MB)	Number of Indexes	Number of Foreign Keys
students	1,000,000	45	146	2	0
instructors	10,000	40	1.4	2	0
courses	100,000	25	13	2	1
quizzes	1,000,000	19	71	1	1
videos	1,000,000	61	153	3	1
students_courses	5,000,000	32	1200	2	2
students_videos	7,000,000	8	433	1	2
students_quizzes	10,000,000	18	849	2	2

B. Schema Optimization

- **Schema Before Optimization**



- **Schema After Optimization**



- **Changes Explanation**

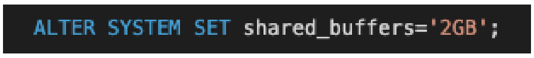
There are five types of optimizations done in the schema to enhance the performance of the database:

1. Change the primary key of students' and instructors' relations from username to auto increment ID. This will help in joins as joins on numbers are faster than on strings. Also, this will help reduce the foreign key's size when referencing those relations.
2. Change the type of IDs from big-int to int to reduce the size of the relation.
3. Add redundant data to relations to avoid using unnecessary joins. It is usual to fetch the video with the course name and the instructor's name, so adding those fields to the video's relation will enhance the performance as it is frequent to fetch the video with the course name or with the instructor's name.
4. Add redundant data to the many-to-many relations to avoid using unnecessary joins. This happens in two cases in our schema. Add course name and student name to the students_courses relation and student_name to the students_quizzes relation.
5. Add materialized view to store the top-rated courses, getting top-rated courses is a frequent query as it is usually on the home page of the website. The materialized view can be refreshed one time every hour to get updates. In this case, it is not important to make the data real-time, and top-rated courses don't change results a lot.

C. Memory and Cache Optimization

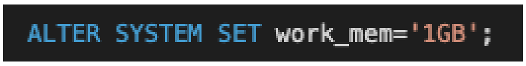
There are two changes done to optimize the database performance using memory

1. Increase the size of the shared buffers from 16MB (default) to 2GB. This determines how much memory is dedicated to the server for caching data.



```
ALTER SYSTEM SET shared_buffers='2GB';
```

2. Increase the size of the working memory from 4MB (default) to 1GB. This parameter shows how much memory is to be used by internal sort operations and hash tables before writing to temporary disk files. Sort operations are used for order by, distinct, and merge join operations. Hash tables are used in hash joins and hash-based aggregation.



```
ALTER SYSTEM SET work_mem='1GB';
```

D. Indexes Optimization

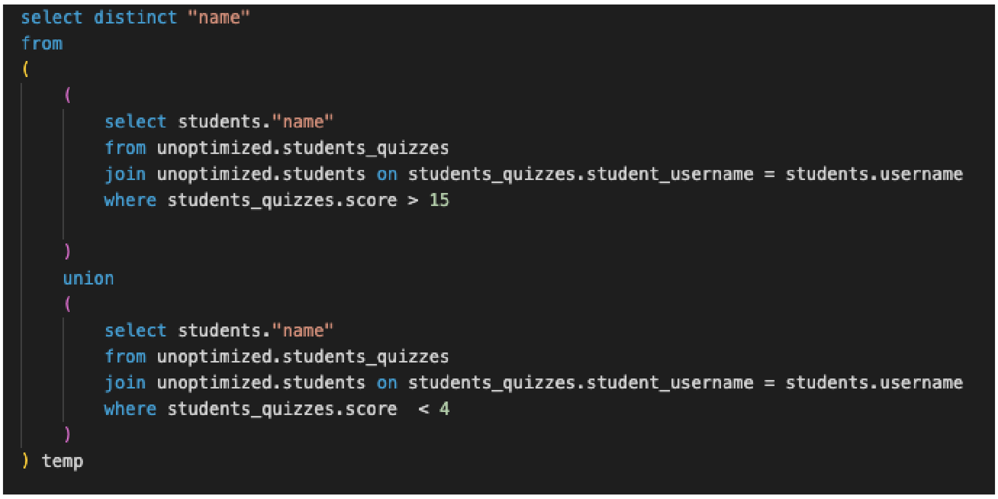
We have created three indexes to enhance the database performance.

1. Create a compound index on videos (course_name , instructor_name) to make frequent queries on videos related to some instructor in a specific course faster.
2. Create an index on students_quizzes (score). This helps make queries on score go to the scan index instead of scanning the large many-to-many relation.
3. Create an index on students_courses(course_id). This makes lookups and joins on course_id much faster.

E. Query Optimization

In this category of optimization, there is only one optimization which we did on query 6.

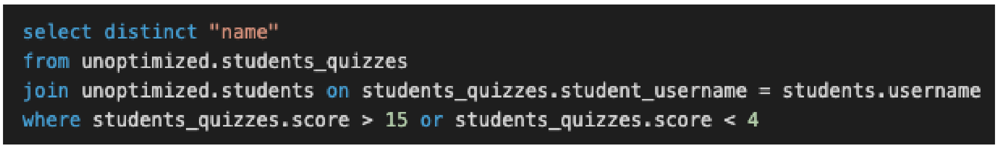
- **Before Optimization**



```
select distinct "name"
from
(
    (
        select students."name"
        from unoptimized.students_quizzes
        join unoptimized.students on students_quizzes.student_username = students.username
        where students_quizzes.score > 15

    )
    union
    (
        select students."name"
        from unoptimized.students_quizzes
        join unoptimized.students on students_quizzes.student_username = students.username
        where students_quizzes.score  < 4
    )
) temp
```

- **After Optimization**



```
select distinct "name"
from unoptimized.students_quizzes
join unoptimized.students on students_quizzes.student_username = students.username
where students_quizzes.score > 15 or students_quizzes.score < 4
```

- **Changes Explanation**

In the unoptimized query read the two relations in the first part of the union and read the two relations again in the second part of the union. This is a high cost compared to the optimized query which read the two relations once. Also, the cost of the join operation is higher in the case of the unoptimized query as there is less filtration applied to students_quizzes relation but in the optimized, it filters out a big portion of the relation before joining. Also, in the unoptimized query, there is an extra append operation due to join.

3. Validation Details

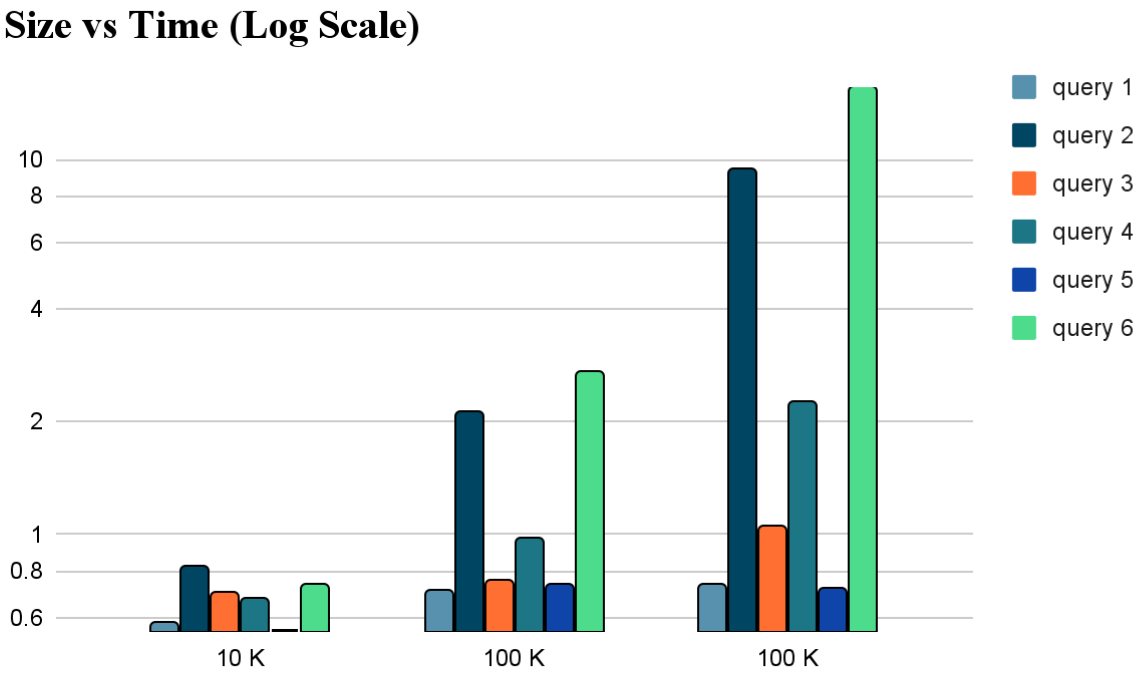
A. Enhancements

	Before (ms)	Memory Optimization (ms)	Query Optimization (ms)	Schema Optimization (ms)	Index Optimization (ms)
Query 1	204	124	124	109	65
Query 2	1460	906	906	1005	335
Query 3	670	527	527	83	76
Query 4	6709	6500	6500	5535	61
Query 5	4244	2817	2817	57	57
Query 6	3999	3008	2204	1331	798

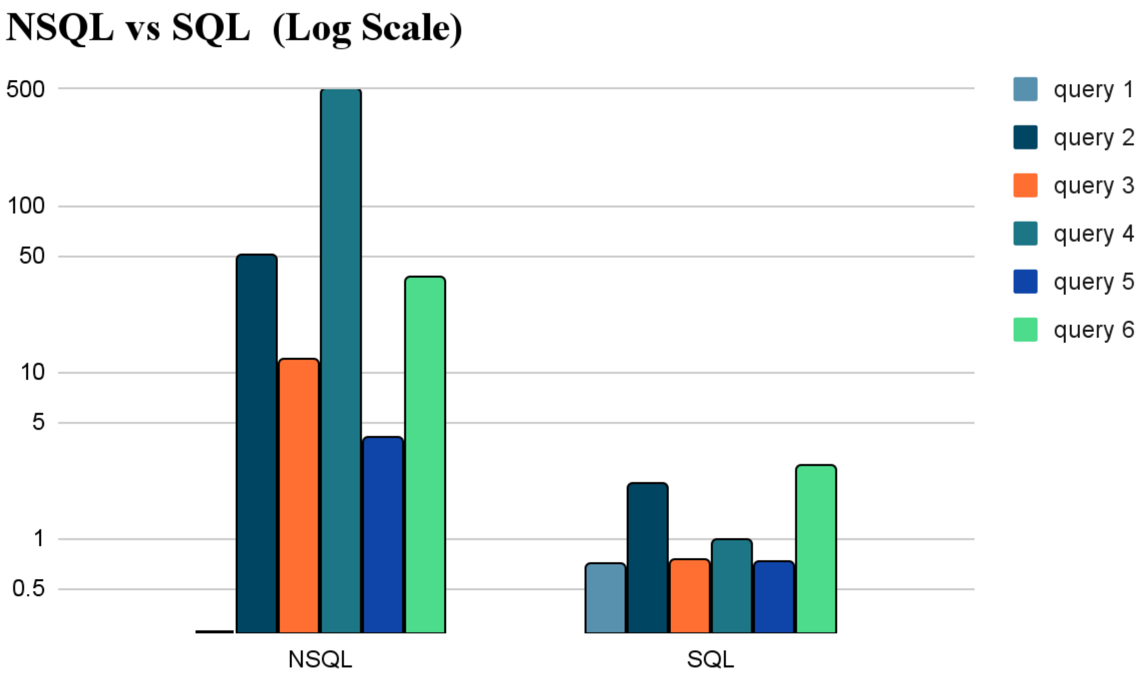
	Memory Optimization	Query Optimization	Schema Optimization	Index Optimization	Overall Optimization
Query 1	64.5 %	-	13.76 %	67 %	213 %
Query 2	61.1 %	-	- 10 %	200 %	335 %
Query 3	27 %	-	535 %	9.2 %	781 %

Query 4	3.2 %	-	17.4 %	8973.77 %	10898 %
Query 5	50.66 %	-	4842 %	-	7345 %
Query 6	32.9 %	36.4 %	65.589 %	66.7 %	401 %

B. Effect of Database Size on Performance



C. SQL vs NOSQL



D. Comments and Recommendations

- Using indexes increases the performance of the queries significantly but the problem is updating the index.

- Schema optimizations are an essential factor in any optimization process.

- SQL is better when the queries contain complex joins and when the size of the database becomes large.

- Cache and memory optimization is an essential factor in the optimization process and depends on the space available on the machine.

- We recommend using SSD with big RAM on the machine for high performance.