Advanced Databases Lab

Assignment 6

2019BTECS00058
Devang Kamble

Do the performance tuning for Assignment No.4 & 5

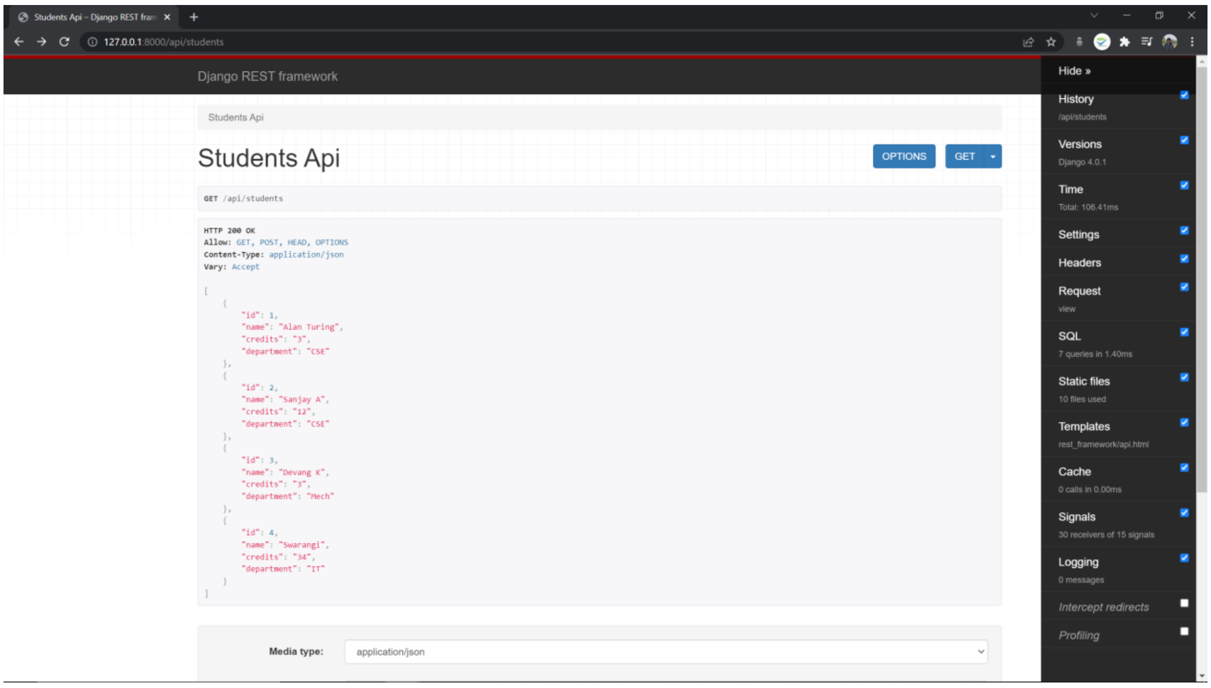
Hints:

1. Use the standard performance metrics and tabulate the results
2. Use any open-source tools / Oracle Explain Plan etc.
3. Prepare the benchmark report

Since we used Django ORM for querying with the database, we would have to optimize from Django itself.

We shall make use of ‘Django Debug Toolbar’ to analyze our queries.

Let’s set it up in our code of Assignment 4:



Students Api

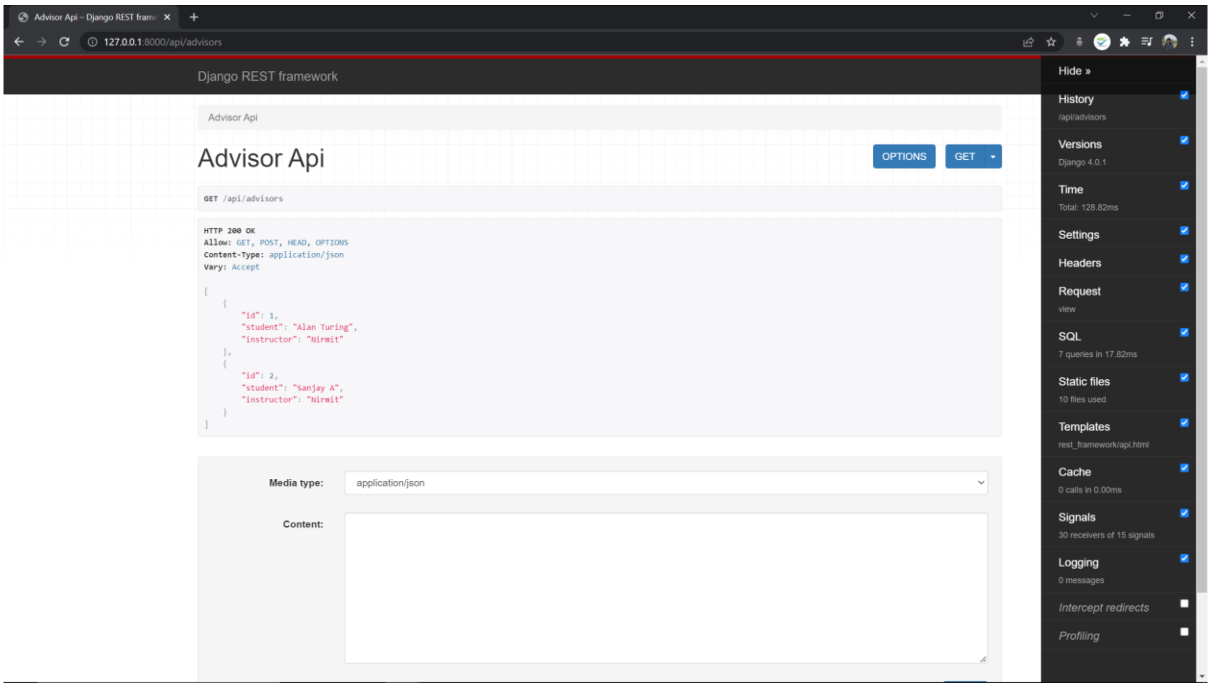
```
GET /api/students
```

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "name": "Alan Turing",
        "credits": "3",
        "department": "CSE"
    },
    {
        "id": 2,
        "name": "Sanjay A",
        "credits": "12",
        "department": "CSE"
    },
    {
        "id": 3,
        "name": "Devang K",
        "credits": "3",
        "department": "Mech"
    },
    {
        "id": 4,
        "name": "Swarangi",
        "credits": "34",
        "department": "IT"
    }
]
```

Media type: application/json

Hide »
History ✓ /api/students
Versions ✓ Django 4.0.1
Time ✓ Total: 106.41ms
Settings ✓
Headers ✓
Request ✓ view
SQL ✓ 7 queries in 1.40ms
Static files ✓ 10 files used
Templates ✓ rest_framework/api.html
Cache ✓ 0 calls in 0.00ms
Signals ✓ 30 receivers of 15 signals
Logging ✓ 0 messages
Intercept redirects ☐
Profiling ☐

Let’s take example of Advisor API



Advisor Api

```
GET /api/advisors
```

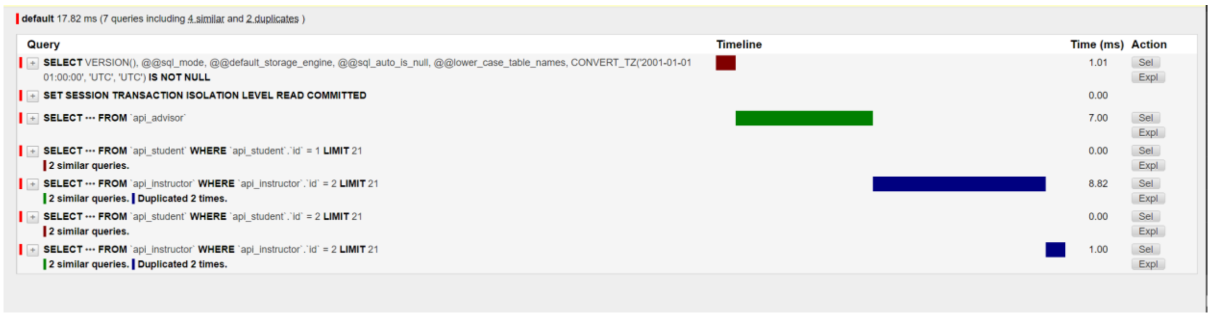
```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "student": "Alan Turing",
        "instructor": "Nirmit"
    },
    {
        "id": 2,
        "student": "Sanjay A",
        "instructor": "Nirmit"
    }
]
```

Media type: application/json

Content:

Hide »
History ✓ /api/advisors
Versions ✓ Django 4.0.1
Time ✓ Total: 128.82ms
Settings ✓
Headers ✓
Request ✓ view
SQL ✓ 7 queries in 17.82ms
Static files ✓ 10 files used
Templates ✓ rest_framework/api.html
Cache ✓ 0 calls in 0.00ms
Signals ✓ 30 receivers of 15 signals
Logging ✓ 0 messages
Intercept redirects ☐
Profiling ☐

To fetch all students, we have to run 7 queries in 17.82ms


default 17.82 ms (7 queries including 4 similar and 2 duplicates)

Query	Timeline	Time (ms)	Action
SELECT VERSION(), @@sql_mode, @@default_storage_engine, @@sql_auto_is_null, @@lower_case_table_names, CONVERT_TZ('2001-01-01 01:00:00', 'UTC', 'UTC') IS NOT NULL	🟥	1.01	Sel Expl
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED		0.00	
SELECT ••• FROM `api_advisor`	🟩	7.00	Sel Expl
SELECT ••• FROM `api_student` WHERE `api_student`.`id` = 1 LIMIT 21 2 similar queries.		0.00	Sel Expl
SELECT ••• FROM `api_instructor` WHERE `api_instructor`.`id` = 2 LIMIT 21 2 similar queries. Duplicated 2 times.	🟦	8.82	Sel Expl
SELECT ••• FROM `api_student` WHERE `api_student`.`id` = 2 LIMIT 21 2 similar queries.		0.00	Sel Expl
SELECT ••• FROM `api_instructor` WHERE `api_instructor`.`id` = 2 LIMIT 21 2 similar queries. Duplicated 2 times.	🟦	1.00	Sel Expl

Our goal is to optimise by reducing the number of queries and avoiding duplicates. Also keeping in mind that we have to reduce the time.

Currently time to load was 128.82ms

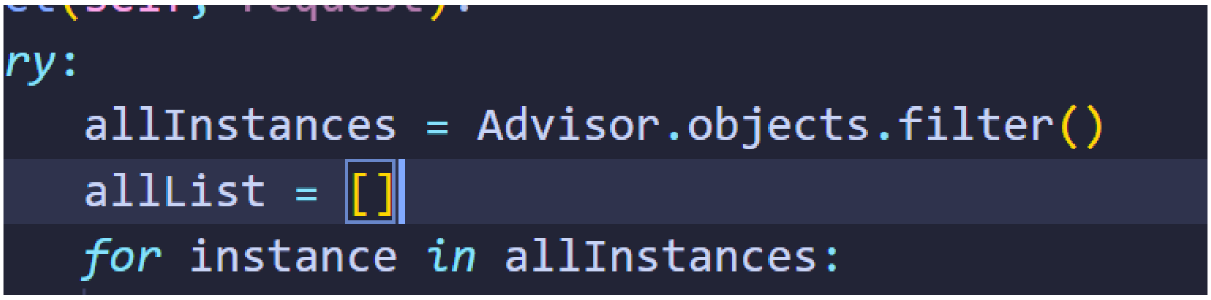
Similarly, for /sections:

Time – 153.30ms
8 queries in 36.34ms

We shall be making use of 'prefetch' and 'select_related' to lessen the bulk of queries and reduce the time during the API call.

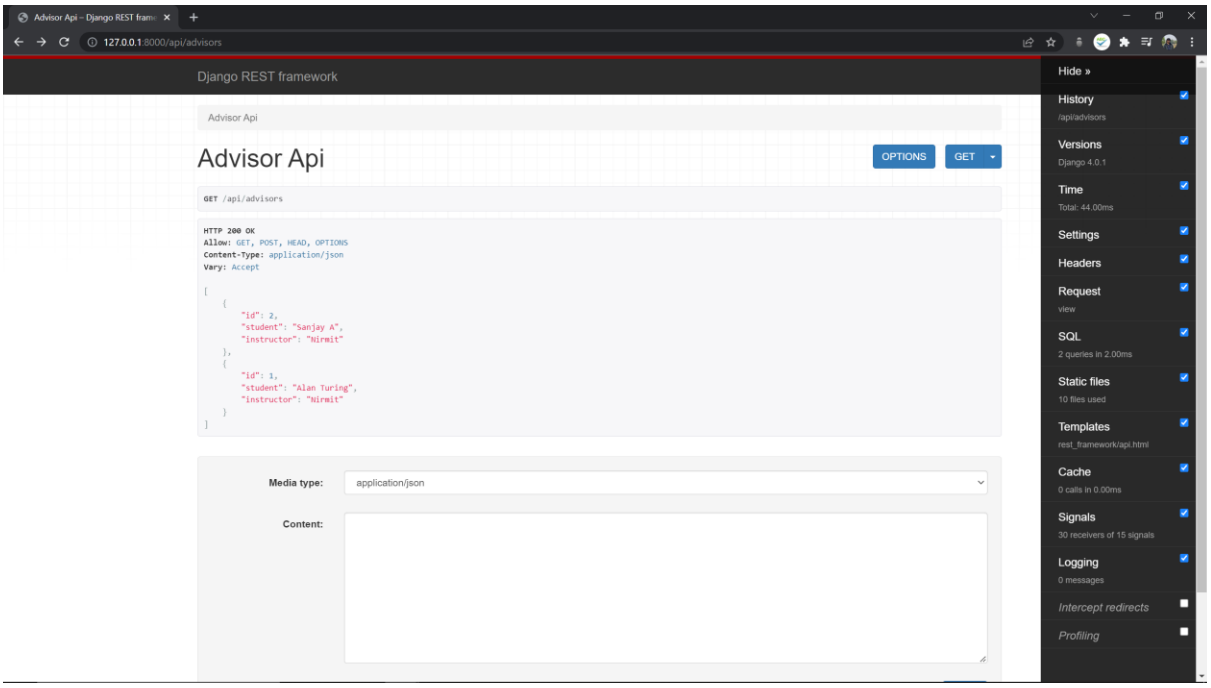
Every FK tag gets into the select_related tuple and a ManyToMany relation into a prefetch (called and stored all instances in buffer to get value again from).

Let's test this on /advisors



```
et(self, request):
ry:
   allInstances = Advisor.objects.filter()
   allList = []
   for instance in allInstances:
```

Instead of a blank filter call, we shall use the selected_related and all()



127.0.0.1:8000/api/advisors

Django REST framework

Advisor Api

```
GET /api/advisors
```

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 2,
        "student": "Sanjay A",
        "instructor": "Nirmit"
    },
    {
        "id": 1,
        "student": "Alan Turing",
        "instructor": "Nirmit"
    }
]
```

Media type: application/json

Content:

History ✓
/api/advisors

Versions ✓
Django 4.0.1

Time ✓
Total: 44.00ms

Settings ✓

Headers ✓

Request ✓
view

SQL ✓
2 queries in 2.00ms

Static files ✓
10 files used

Templates ✓
rest_framework/api.html

Cache ✓
0 calls in 0.00ms

Signals ✓
30 receivers of 15 signals

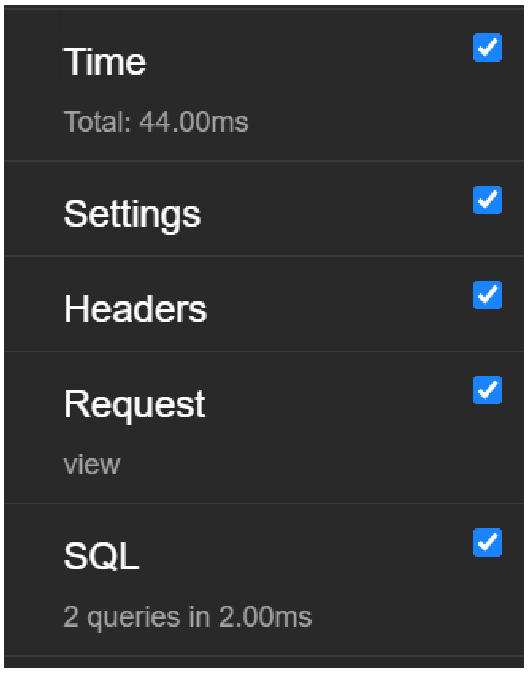
Logging ✓
0 messages

Intercept redirects ☐

Profiling ☐

Doing this, we see remarkable difference in our processing time and query count.

After optimisation:



Time ✓
Total: 44.00ms

Settings ✓

Headers ✓

Request ✓
view

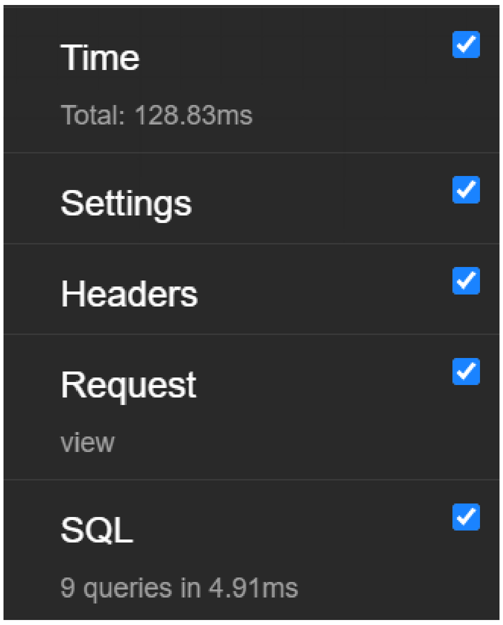
SQL ✓
2 queries in 2.00ms

Load time – 44ms
SQL queries – 2

Thus, we tremendously optimised our API and SQL query instance.

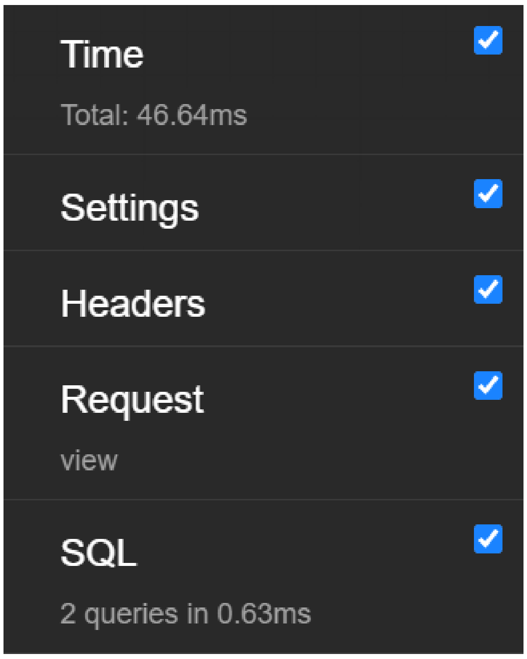
Similarly, we perform this for /sections

Before Optimisation:



Time ✓
Total: 128.83ms
Settings ✓
Headers ✓
Request ✓
view
SQL ✓
9 queries in 4.91ms

After Optimisation:



Time ✓
Total: 46.64ms
Settings ✓
Headers ✓
Request ✓
view
SQL ✓
2 queries in 0.63ms

This way, we could optimise the Django code while querying.

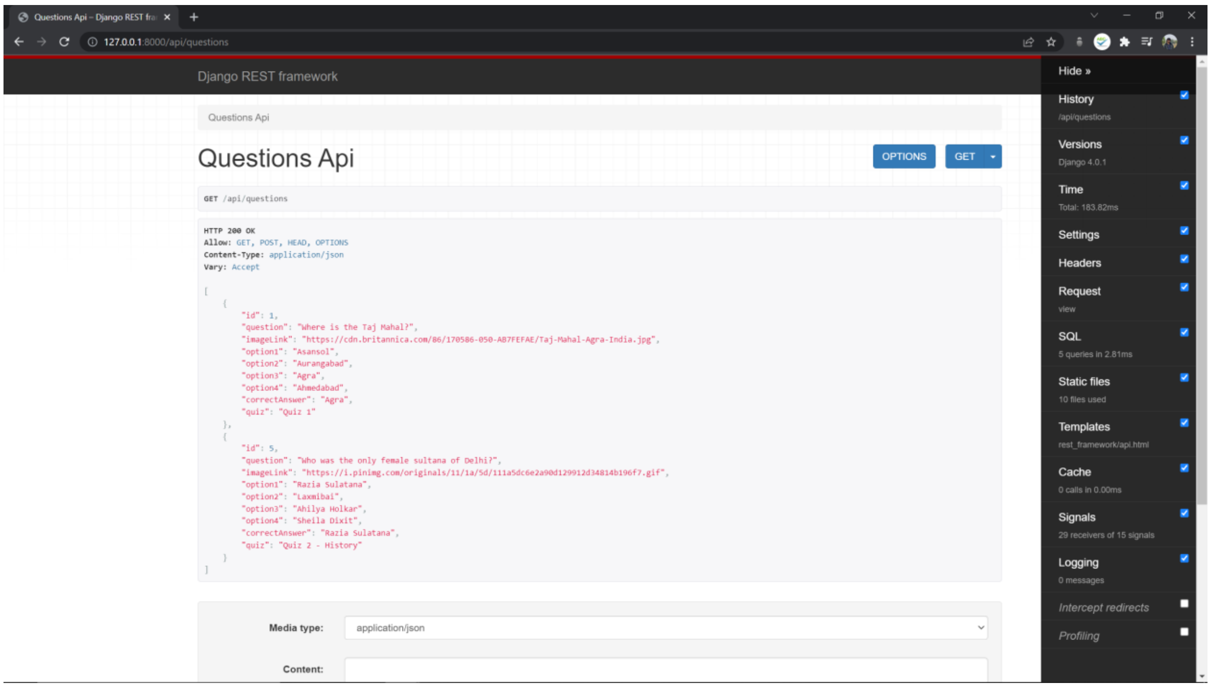
PFA – Optimised Backend Code.

Let's move on to Assignment 5

For the purposes of testing, am taking our the JWT Validations so we can test on the Browser itself.

Further we install Django debugger in this project as well.

We look at /questions – to fetch all questions of every quiz



Questions Api

```
GET /api/questions
```

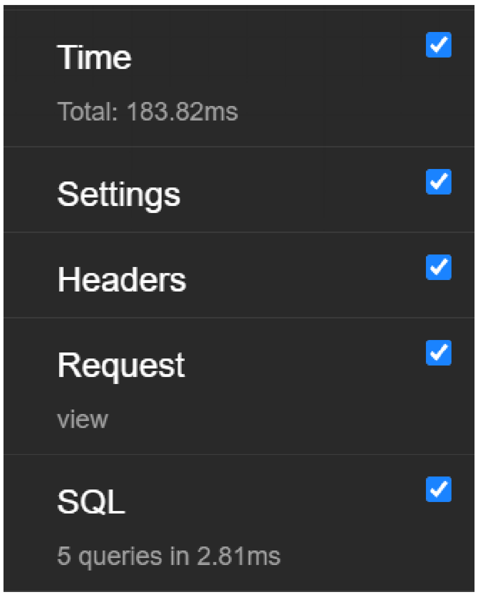
```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "question": "Where is the Taj Mahal?",
        "imageLink": "https://cdn.britannica.com/86/170586-050-AB7FEFAE/Taj-Mahal-Agra-India.jpg",
        "option1": "Asansol",
        "option2": "Aurangabad",
        "option3": "Agra",
        "option4": "Ahmedabad",
        "correctAnswer": "Agra",
        "quiz": "Quiz 1"
    },
    {
        "id": 5,
        "question": "Who was the only female sultana of Delhi?",
        "imageLink": "https://i.pinimg.com/originals/11/1a/5d/111a5dc6e2a90d129912d34814b196f7.gif",
        "option1": "Razia Sulatana",
        "option2": "Laxmibai",
        "option3": "Ahilya Holkar",
        "option4": "Sheila Dixit",
        "correctAnswer": "Razia Sulatana",
        "quiz": "Quiz 2 - History"
    }
]
```

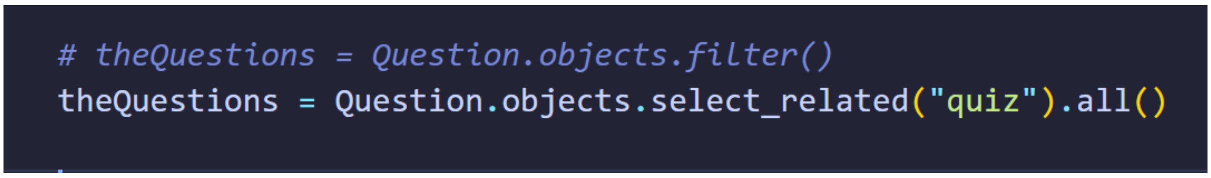
Media type: application/json

Content:

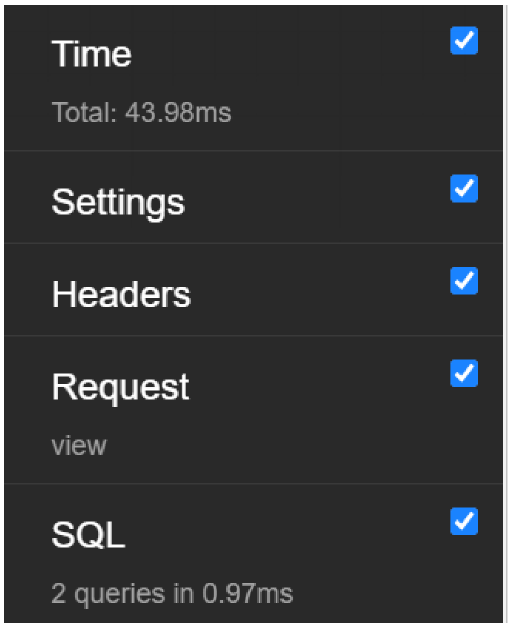
We shall be applying similar optimisation query to this.

Before Optimisation:


Time
Total: 183.82ms
Settings
Headers
Request
view
SQL
5 queries in 2.81ms


```
# theQuestions = Question.objects.filter()
theQuestions = Question.objects.select_related("quiz").all()
```

After Optimisation:


Time
Total: 43.98ms
Settings
Headers
Request
view
SQL
2 queries in 0.97ms

We thus optimised our SQL queries for higher performance.

Tabulated Results:

Time to execute:

Endpoint	Before Optimization (ms)	After Optimization (ms)
/advisors	128	44
/sections	153	46
/questions	183	43

SQL queries to execute:

Endpoint	Before Optimization	After Optimization
/advisors	8	2
/sections	9	2
/questions	5	2

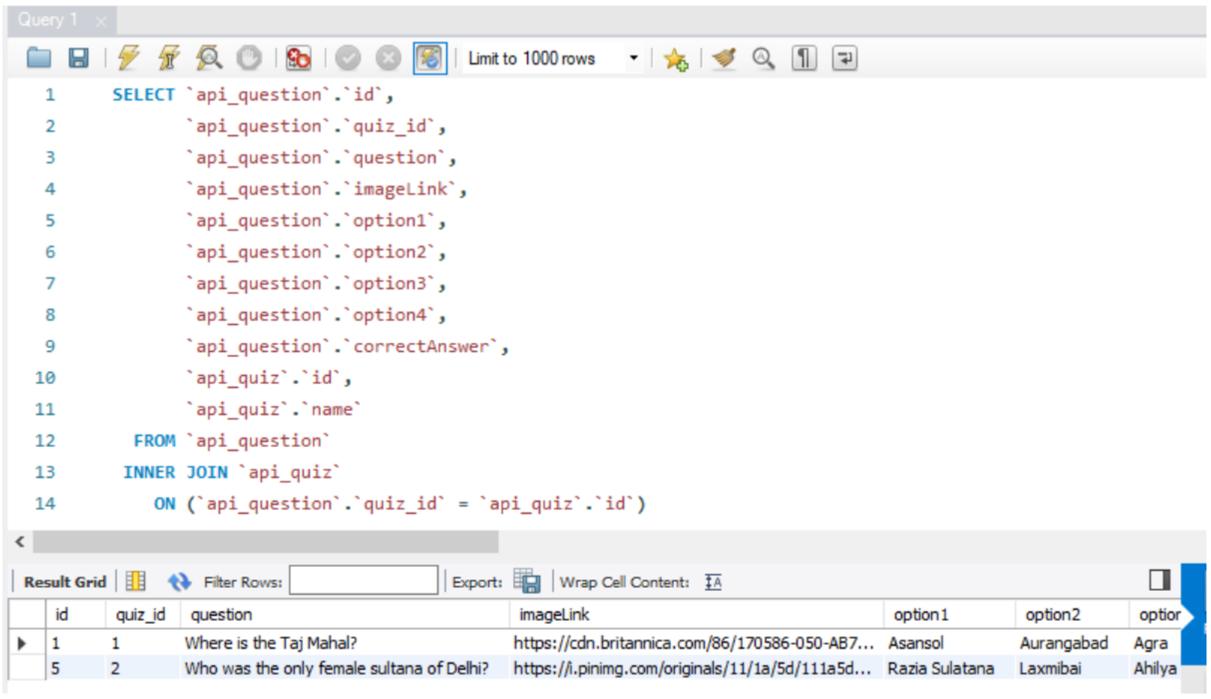
We now generate Database reports:

We shall use MySQL Explain Analyse commands

Assignment 5

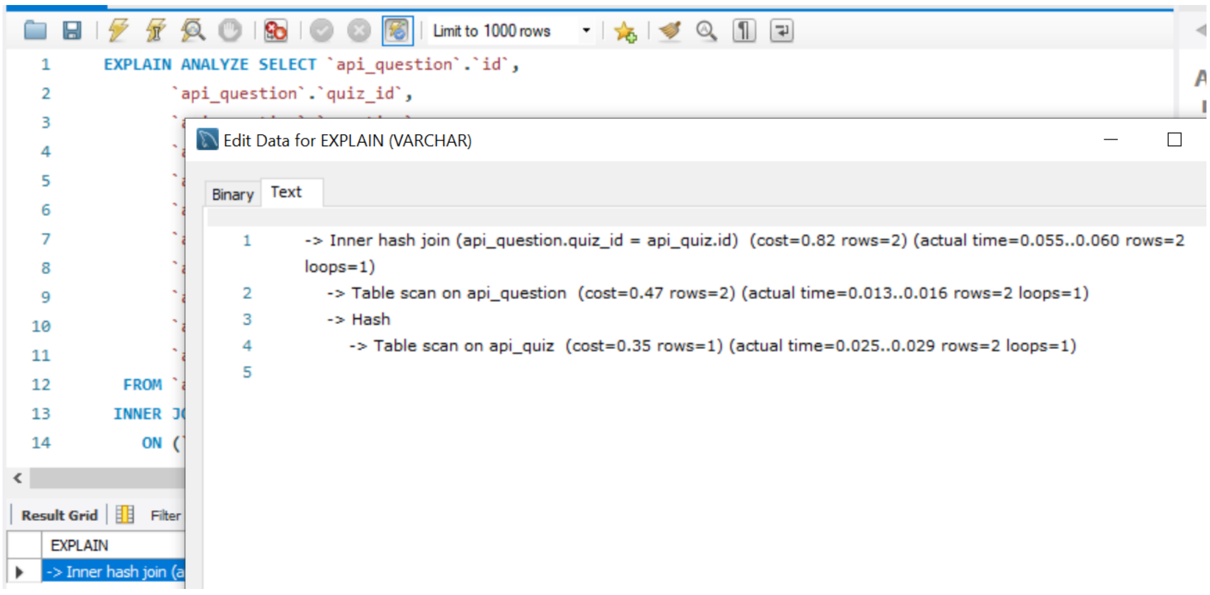
We analyse the SQL command that fetches all the questions:

```
SELECT `api_question`.`id`,
       `api_question`.`quiz_id`,
       `api_question`.`question`,
       `api_question`.`imageLink`,
       `api_question`.`option1`,
       `api_question`.`option2`,
       `api_question`.`option3`,
       `api_question`.`option4`,
       `api_question`.`correctAnswer`,
       `api_quiz`.`id`,
       `api_quiz`.`name`
  FROM `api_question`
 INNER JOIN `api_quiz`
    ON (`api_question`.`quiz_id` = `api_quiz`.`id`)
```



```
SELECT `api_question`.`id`,
       `api_question`.`quiz_id`,
       `api_question`.`question`,
       `api_question`.`imageLink`,
       `api_question`.`option1`,
       `api_question`.`option2`,
       `api_question`.`option3`,
       `api_question`.`option4`,
       `api_question`.`correctAnswer`,
       `api_quiz`.`id`,
       `api_quiz`.`name`
  FROM `api_question`
 INNER JOIN `api_quiz`
    ON (`api_question`.`quiz_id` = `api_quiz`.`id`)
```

id	quiz_id	question	imageLink	option1	option2	optior
1	1	Where is the Taj Mahal?	https://cdn.britannica.com/86/170586-050-AB7...	Asansol	Aurangabad	Agra
5	2	Who was the only female sultana of Delhi?	https://i.pinimg.com/originals/11/1a/5d/111a5d...	Razia Sulatana	Laxmibai	Ahilya



```
EXPLAIN ANALYZE SELECT `api_question`.`id`,
        `api_question`.`quiz_id`,
```

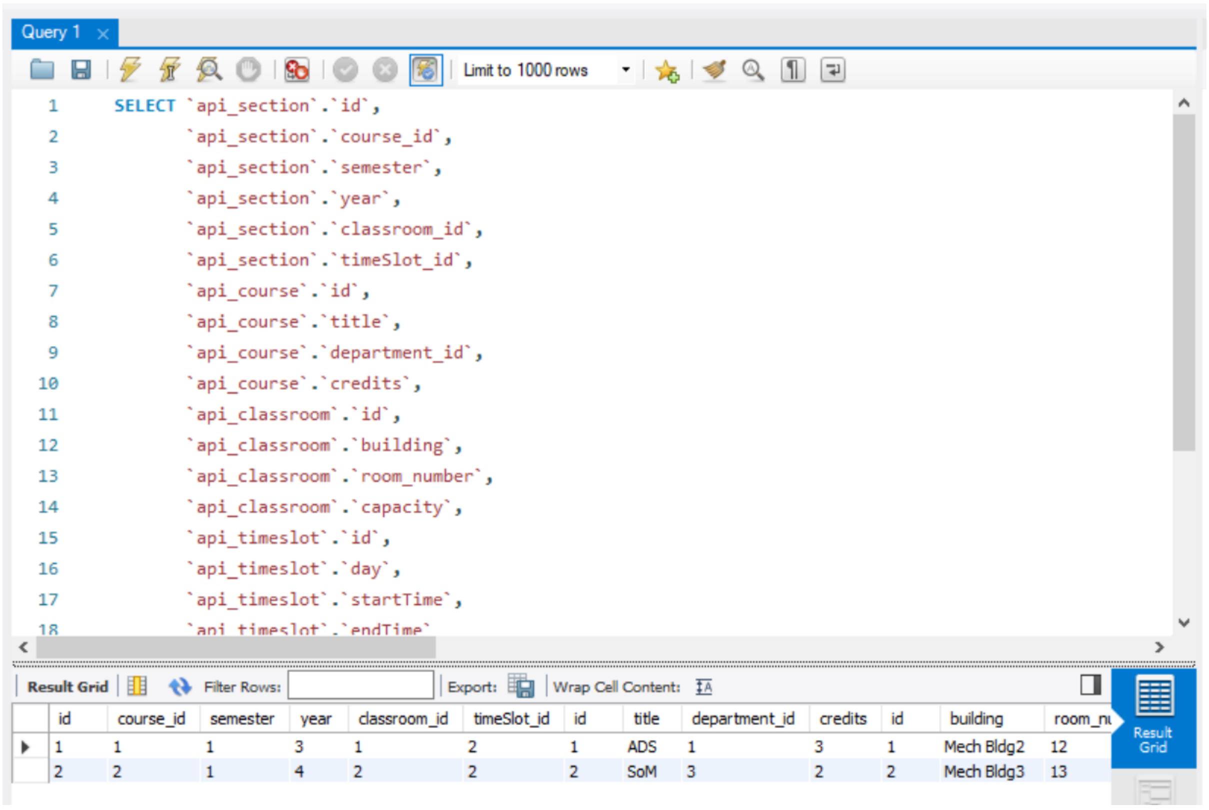
Edit Data for EXPLAIN (VARCHAR)

```
-> Inner hash join (api_question.quiz_id = api_quiz.id)  (cost=0.82 rows=2) (actual time=0.055..0.060 rows=2 loops=1)
    -> Table scan on api_question  (cost=0.47 rows=2) (actual time=0.013..0.016 rows=2 loops=1)
    -> Hash
        -> Table scan on api_quiz  (cost=0.35 rows=1) (actual time=0.025..0.029 rows=2 loops=1)
```

Assignment 4:

We analyse the query of /sections:

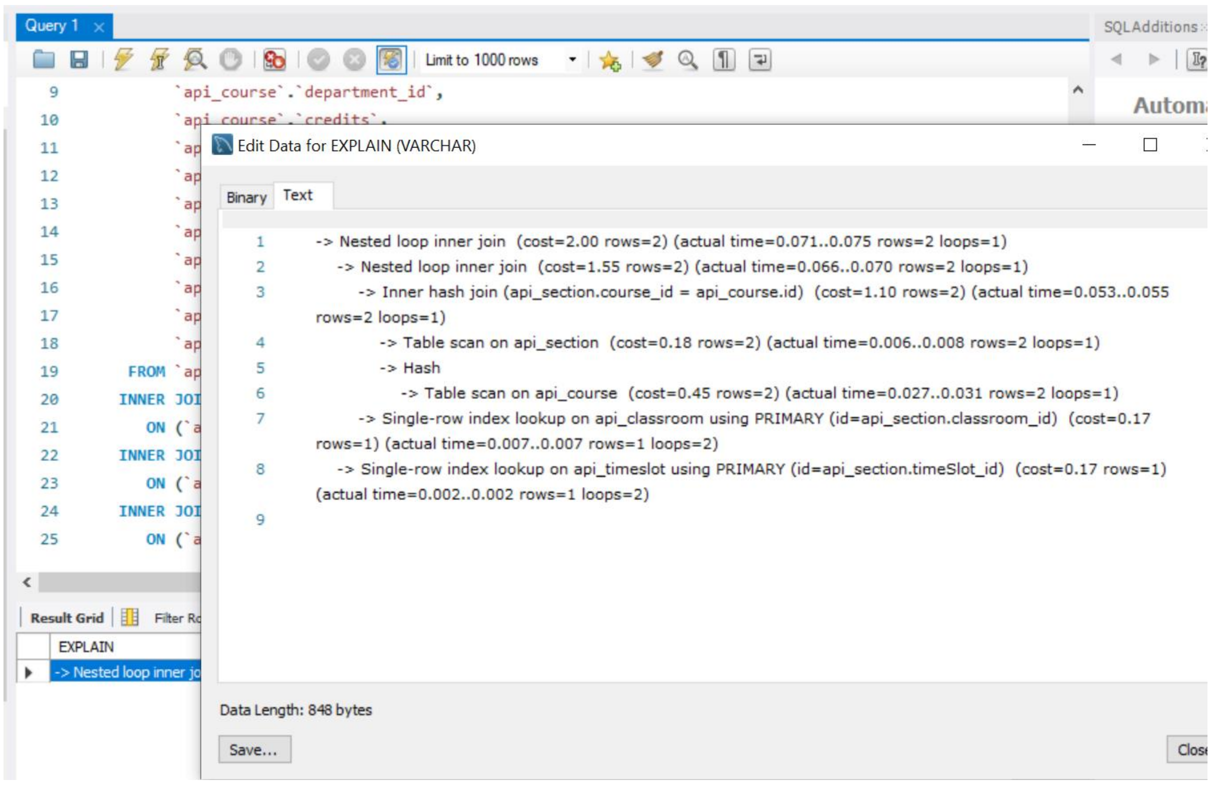
```
SELECT `api_section`.`id`,
       `api_section`.`course_id`,
       `api_section`.`semester`,
       `api_section`.`year`,
       `api_section`.`classroom_id`,
       `api_section`.`timeSlot_id`,
       `api_course`.`id`,
       `api_course`.`title`,
       `api_course`.`department_id`,
       `api_course`.`credits`,
       `api_classroom`.`id`,
       `api_classroom`.`building`,
       `api_classroom`.`room_number`,
       `api_classroom`.`capacity`,
       `api_timeslot`.`id`,
       `api_timeslot`.`day`,
       `api_timeslot`.`startTime`,
       `api_timeslot`.`endTime`
 FROM `api_section`
INNER JOIN `api_course`
  ON (`api_section`.`course_id` = `api_course`.`id`)
INNER JOIN `api_classroom`
  ON (`api_section`.`classroom_id` = `api_classroom`.`id`)
INNER JOIN `api_timeslot`
  ON (`api_section`.`timeSlot_id` = `api_timeslot`.`id`)
```



```
SELECT `api_section`.`id`,
       `api_section`.`course_id`,
       `api_section`.`semester`,
       `api_section`.`year`,
       `api_section`.`classroom_id`,
       `api_section`.`timeSlot_id`,
       `api_course`.`id`,
       `api_course`.`title`,
       `api_course`.`department_id`,
       `api_course`.`credits`,
       `api_classroom`.`id`,
       `api_classroom`.`building`,
       `api_classroom`.`room_number`,
       `api_classroom`.`capacity`,
       `api_timeslot`.`id`,
       `api_timeslot`.`day`,
       `api_timeslot`.`startTime`,
       `api_timeslot`.`endTime`
```

id	course_id	semester	year	classroom_id	timeSlot_id	id	title	department_id	credits	id	building	room_nu
1	1	1	3	1	2	1	ADS	1	3	1	Mech Bldg2	12
2	2	1	4	2	2	2	SoM	3	2	2	Mech Bldg3	13

Using Explain Analysis:



```
            `api_course`.`department_id`,
            `api_course`.`credits`,
```

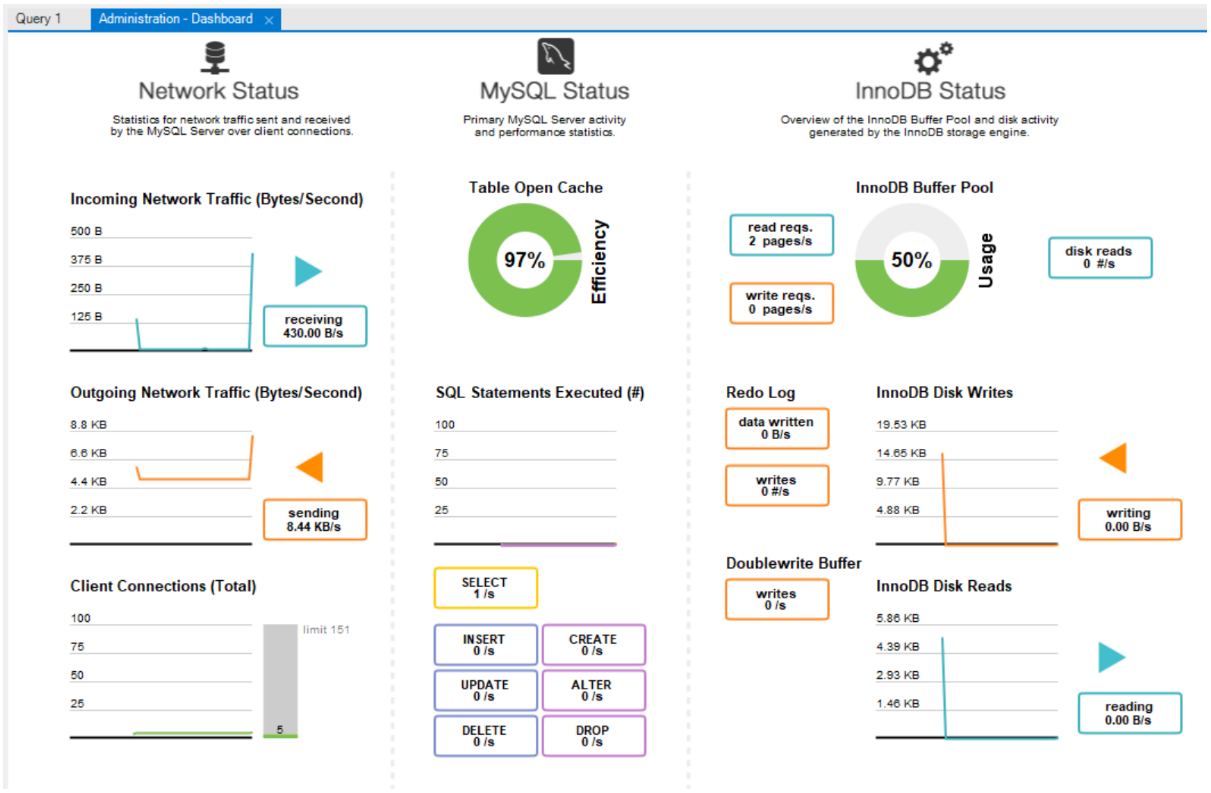
Edit Data for EXPLAIN (VARCHAR)

```
-> Nested loop inner join  (cost=2.00 rows=2) (actual time=0.071..0.075 rows=2 loops=1)
   -> Nested loop inner join  (cost=1.55 rows=2) (actual time=0.066..0.070 rows=2 loops=1)
      -> Inner hash join (api_section.course_id = api_course.id)  (cost=1.10 rows=2) (actual time=0.053..0.055
rows=2 loops=1)
         -> Table scan on api_section  (cost=0.18 rows=2) (actual time=0.006..0.008 rows=2 loops=1)
         -> Hash
            -> Table scan on api_course  (cost=0.45 rows=2) (actual time=0.027..0.031 rows=2 loops=1)
      -> Single-row index lookup on api_classroom using PRIMARY (id=api_section.classroom_id)  (cost=0.17
rows=1) (actual time=0.007..0.007 rows=1 loops=2)
   -> Single-row index lookup on api_timeslot using PRIMARY (id=api_section.timeSlot_id)  (cost=0.17 rows=1)
(actual time=0.002..0.002 rows=1 loops=2)
```

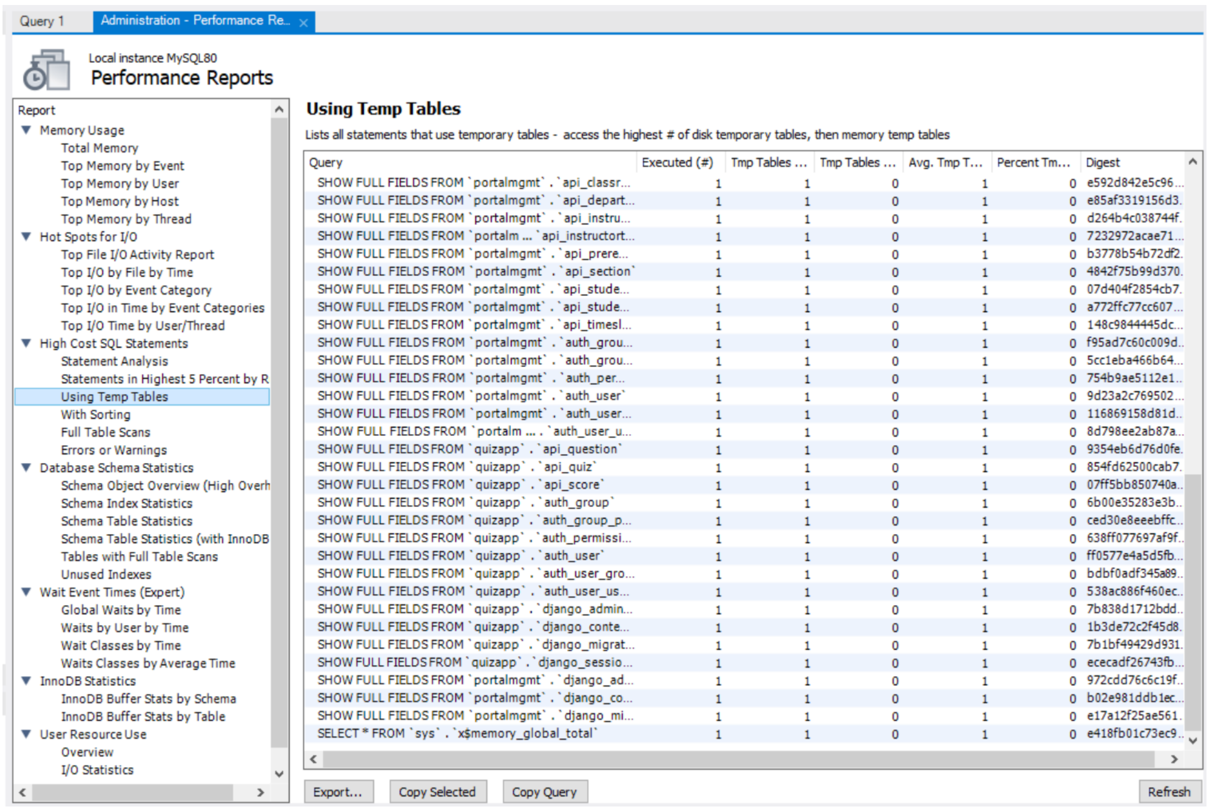
Data Length: 848 bytes

Moving on to benchmarking:

We execute a query and find Dashboard in Performances tab



We can then generate a performance report of our system.



Using Temp Tables

Lists all statements that use temporary tables - access the highest # of disk temporary tables, then memory temp tables

Query	Executed (#)	Tmp Tables ...	Tmp Tables ...	Avg. Tmp T...	Percent Tm...	Digest
SHOW FULL FIELDS FROM `portalmgmt` . `api_classr...	1	1	0	1	0	e592d842e5c96...
SHOW FULL FIELDS FROM `portalmgmt` . `api_depart...	1	1	0	1	0	e85af3319156d3.
SHOW FULL FIELDS FROM `portalmgmt` . `api_instru...	1	1	0	1	0	d264b4c038744f.
SHOW FULL FIELDS FROM `portalm ... `api_instructort...	1	1	0	1	0	7232972acae71...
SHOW FULL FIELDS FROM `portalmgmt` . `api_prere...	1	1	0	1	0	b3778b54b72df2.
SHOW FULL FIELDS FROM `portalmgmt` . `api_section`	1	1	0	1	0	4842f75b99d370.
SHOW FULL FIELDS FROM `portalmgmt` . `api_stude...	1	1	0	1	0	07d404f2854cb7.
SHOW FULL FIELDS FROM `portalmgmt` . `api_stude...	1	1	0	1	0	a772ffc77cc607...
SHOW FULL FIELDS FROM `portalmgmt` . `api_timesl...	1	1	0	1	0	148c9844445dc...
SHOW FULL FIELDS FROM `portalmgmt` . `auth_grou...	1	1	0	1	0	f95ad7c60c009d..
SHOW FULL FIELDS FROM `portalmgmt` . `auth_grou...	1	1	0	1	0	5cc1eba466b64...
SHOW FULL FIELDS FROM `portalmgmt` . `auth_per...	1	1	0	1	0	754b9ae5112e1..
SHOW FULL FIELDS FROM `portalmgmt` . `auth_user`	1	1	0	1	0	9d23a2c769502...
SHOW FULL FIELDS FROM `portalmgmt` . `auth_user...	1	1	0	1	0	116869158d81d..
SHOW FULL FIELDS FROM `portalm ... . `auth_user_u...	1	1	0	1	0	8d798ee2ab87a...
SHOW FULL FIELDS FROM `quizapp` . `api_question`	1	1	0	1	0	9354eb6d76d0fe.
SHOW FULL FIELDS FROM `quizapp` . `api_quiz`	1	1	0	1	0	854fd62500cab7.
SHOW FULL FIELDS FROM `quizapp` . `api_score`	1	1	0	1	0	07ff5bb850740a..
SHOW FULL FIELDS FROM `quizapp` . `auth_group`	1	1	0	1	0	6b00e35283e3b..
SHOW FULL FIELDS FROM `quizapp` . `auth_group_p...	1	1	0	1	0	ced30e8eeebffc...
SHOW FULL FIELDS FROM `quizapp` . `auth_permissi...	1	1	0	1	0	638ff077697af9f..
SHOW FULL FIELDS FROM `quizapp` . `auth_user`	1	1	0	1	0	ff0577e4a5d5fb...
SHOW FULL FIELDS FROM `quizapp` . `auth_user_gro...	1	1	0	1	0	bdbf0adf345a89..
SHOW FULL FIELDS FROM `quizapp` . `auth_user_us...	1	1	0	1	0	538ac886f460ec..
SHOW FULL FIELDS FROM `quizapp` . `django_admin...	1	1	0	1	0	7b838d1712bdd..
SHOW FULL FIELDS FROM `quizapp` . `django_conte...	1	1	0	1	0	1b3de72c2f45d8.
SHOW FULL FIELDS FROM `quizapp` . `django_migrat...	1	1	0	1	0	7b1bf49429d931.
SHOW FULL FIELDS FROM `quizapp` . `django_sessio...	1	1	0	1	0	ececadf26743fb...
SHOW FULL FIELDS FROM `portalmgmt` . `django_ad...	1	1	0	1	0	972cdd76c6c19f..
SHOW FULL FIELDS FROM `portalmgmt` . `django_co...	1	1	0	1	0	b02e981ddb1ec...
SHOW FULL FIELDS FROM `portalmgmt` . `django_mi...	1	1	0	1	0	e17a12f25ae561.
SELECT * FROM `sys` . `x$memory_global_total`	1	1	0	1	0	e418fb01c73ec9...

Performance report recorded: Link

Thus, we performance tuned our database, optimised queries, analysed them and generated their report.