Cairo University - Faculty of Engineering
Computer Engineering Department
Advanced Database CMP401

Project Phase 2

Group 1

Name	sec	BN
Muhammad Alaa Abdelkhaleq	**2**	**22**
Mahmoud Amr Mohamed Refaat	**2**	**23**
Hossam Alaa Ramadan	**1**	**21**
Muhammad Ahmad Hesham	**2**	**16**

January 2022

System Specification 3
DB system 3
Hardware characteristics 3
Relational Database (SQL) 3
NOSQL DATABASE 3
Optimizations 4
Queries 4
Schema Optimization 4
Q1- Before Optimization 6
Q1- After Optimization 6
Q1- Percentage Enhancement 6
Memory and Cache Optimization 7
Q2- Before Optimization 7
Q2- After Optimization 7
Q2- Percentage Enhancement 7
Index Tuning 8
Q3- Before Optimization 8
Q3- After Optimization 8
Q3- Percentage Enhancement 8
Q4- Before Optimization 9
Q4- After Optimization 9
Q4- Percentage Enhancement 9
Q5- Before Optimization 10
Q5- After Optimization 10
Q5- Percentage Enhancement 10
Query Optimization 11
Q6- Before Optimization 11
Q6- After Optimization 11
Q6- Percentage Enhancement 11
MongoDB Queries 12
Q1 12
Q2 12
Q3 14
Q4 14
Q5 16
Q6 17
Query Execution Plans 18
Schema Optimization 18

Q1- Before Optimization 18
Q1- After Optimization 18
Memory and Cache Optimization 19
Q2- Before & After Optimization 19
Schema Optimization 20
Q3- Before Optimization 20
Q3- After Optimization 20
Q4- Before Optimization 21
Q4- After Optimization 21
Q5- Before Optimization 22
Q5- After Optimization 22
Query Optimization 23
Q6- Before Optimization 23
Q6- After Optimization 23
MySQL Statistics after optimizations 24
MongoDB Schema after optimizations 25
Validation Details 25
Optimizations Enhancements 26
Schema Optimization 26
Memory and Cache Optimization 26
Index Tuning 27
Query Optimization 30
Database Size Effect on Execution Time 32
MySQL Queries 32
MongoDB Queries 32
SQL vs NOSQL 33
Recommendations 33

System Specification

DB system

The system allows us to store data of a university like the students, courses, professors data, and store exam grades for each student.

Hardware characteristics

- **OS:** Windows 10 64-bits
- **CPU:** Intel® Core™ i7-8750 CPU @2.20GHz (12 CPUs)
- **RAM:** 16 GB
- **Hard disk:** 1 TB HDD

Relational Database (SQL)

- MySQL Ver 14.14 Distrib 5.7.24, for Win32 (AMD64)

NOSQL DATABASE

- MongoDB shell version v4.4.6

Optimizations

Queries

We ran the following queries on a ~900K total records MySQL database and a ~900K total records NOSQL database and tried to optimize the execution time:

Q1- Select ID, Name, and Gender of students who got an “A” in a course final exam.
Q2- Select all the research projects worked on by professors working in a certain department along with the professors’ names.
Q3- Select ID, Name, Gender, and Salary of all professors with a salary less than 1200.
Q4- Select ID, Name, Gender, and Salary of a certain professor using his name.
Q5- Select ID, Name, Gender, and Salary of all professors with Certain Name whose Salary is greater than 6000.
Q6- Select student ID, Name, and City of students with a specific name or who live in a specific city.

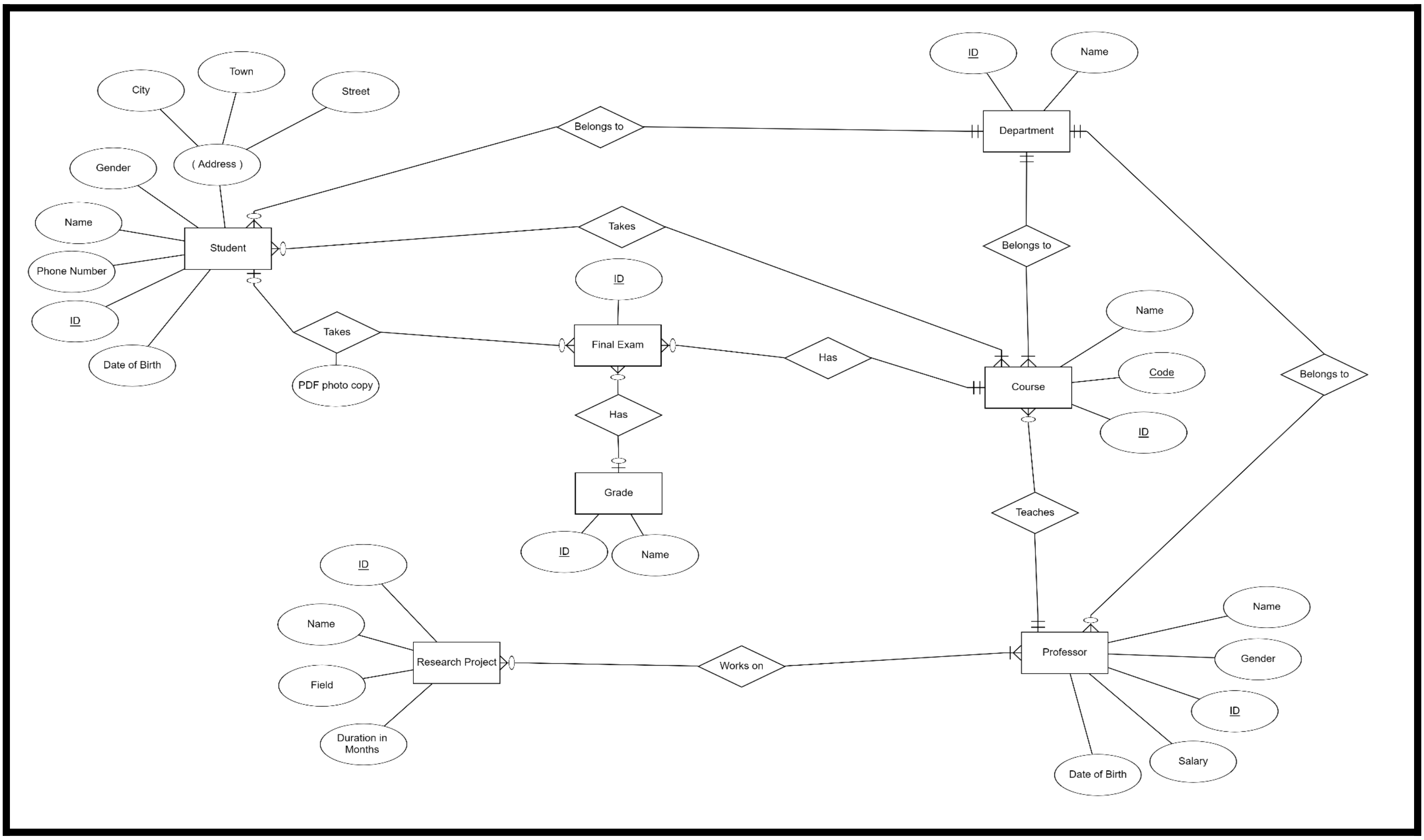
Schema Optimization

We applied one optimization on our original schema by removing unnecessary two relations: Final_Exam and Grade and placing their content on the many to many relation between students and the course. The previous design is a poor choice and a waste of space as It’s sufficient to have their data in the relation connecting the student and the course.

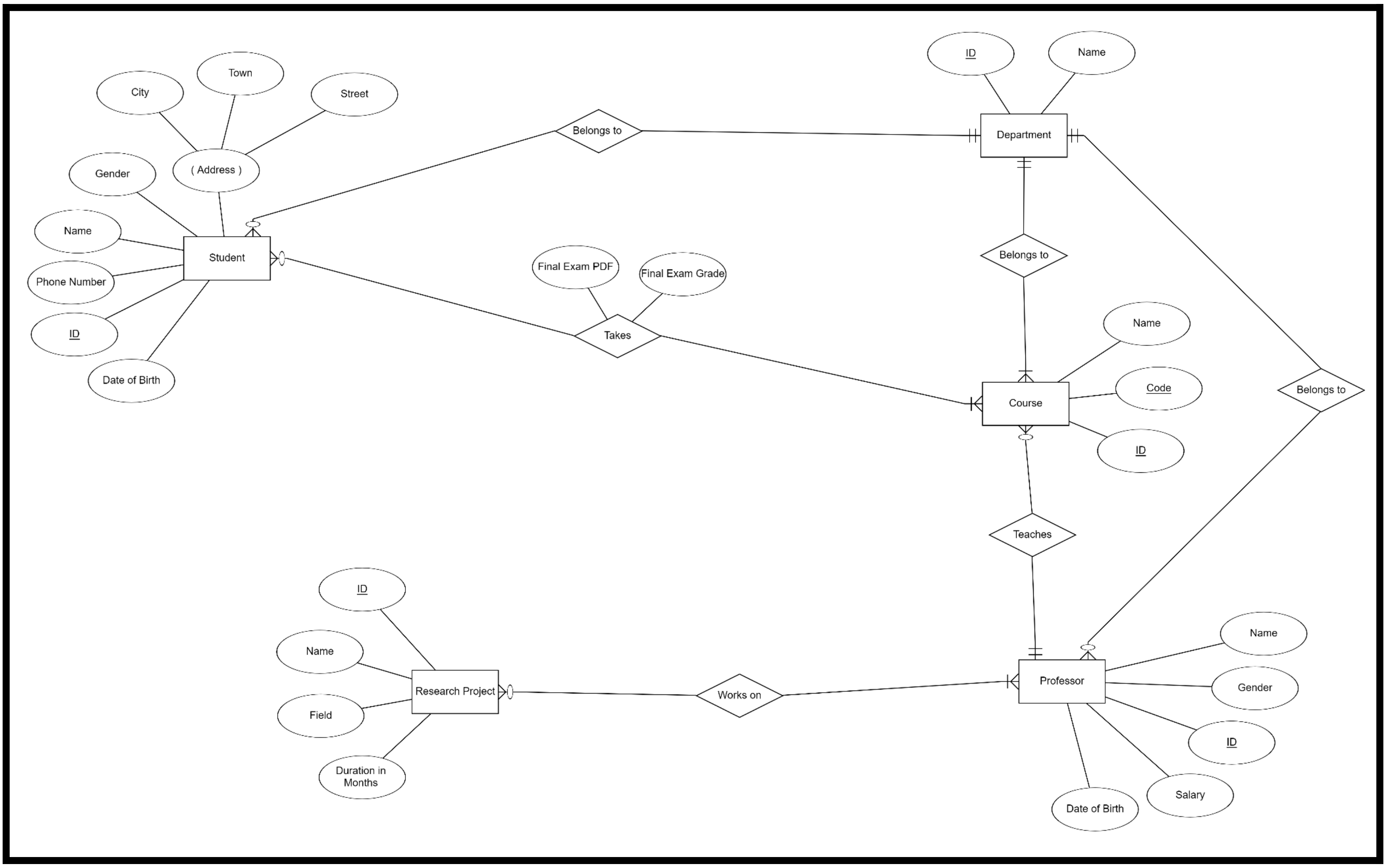
The removal of the unnecessary relations has an influence on for example when we had a database of size ~9M before and after optimization, the size decreased from 1.1GB before optimization to 866.6 MB a decrease in memory requirement of approx 27% Moreover, in many queries, the number of joined tables decreased so the execution time of these queries decreased also. which proves how important it is to take our time to make a good design for our schema and avoid unnecessary relations.

The following figures show the Entity Relation Diagram for our database before and after the schema optimization using the crow's foot notation.

Links for better quality: Optimized_ERD.png ERD.png



ID
Name
City
Town
Street
Belongs to
Department
(Address)
Gender
Name
Student
Takes
Belongs to
Phone Number
ID
Final Exam
Name
Date of Birth
Takes
Has
Code
PDF photo copy
Course
Belongs to
Has
ID
Grade
Teaches
ID
Name
ID
Name
Name
Professor
Gender
Research Project
Works on
Field
ID
Duration in Months
Salary
Date of Birth



ID
Name
Town
City
Street
Belongs to
Department
(Address)
Gender
Name
Student
Belongs to
Final Exam PDF
Final Exam Grade
Phone Number
Takes
Name
ID
Code
Date of Birth
Course
Belongs to
ID
Teaches
ID
Name
Name
Professor
Gender
Research Project
Works on
Field
ID
Duration in Months
Salary
Date of Birth

Q1- Before Optimization

```
select
    distinct s.ID, s.Name, s.Gender
from
    student s
join student_course sc on
    s.ID = sc.student_id
join course c on
    c.ID = sc.course_id
join final_exam fe on
    fe.course_id = c.ID
join grade g on
    g.ID = fe.grade_id
where
    g.Name = 'A';
```

Execution Time: `38.612 sec`

Q1- After Optimization

```
select
    distinct s.ID, s.Name, s.Gender
from
    student s
join student_course sc on
    s.ID = sc.student_id
join course c on
    c.ID = sc.course_id
where
    sc.Final_Exam_Grade = 'A';
```

Execution Time: `0.520333 sec`

Q1- Percentage Enhancement

We can see an Enhancement in the execution time of `7420.63%`

Memory and Cache Optimization

Q2- Before Optimization

```
SET SESSION query_cache_type = OFF;
select SQL_NO_CACHE
    distinct p.Name, rp.Name, rp.Field
from
    department d
join professor p on
    p.department_id = d.ID
join professor_research pr on
    pr.professor_id = p.ID
join research_project rp on
    rp.ID = pr.research_project_id
where d.Name = 'Mechanical Electrical & Process Engineering';
```

Execution Time: `0.032081 sec`

Q2- After Optimization

```
SET SESSION query_cache_type = ON;

select
    distinct p.Name, rp.Name, rp.Field
from
    department d
join professor p on
    p.department_id = d.ID
join professor_research pr on
    pr.professor_id = p.ID
join research_project rp on
    rp.ID = pr.research_project_id
where d.Name = 'Mechanical Electrical & Process Engineering';
```

Execution Time: `0.000147 sec`

Q2- Percentage Enhancement

We can see an Enhancement in the execution time of `21823.80%`

Index Tuning

Q3- Before Optimization

```
select
    *
from
    professor p
join department d on
    p.department_id = d.ID
where
    p.Salary < 1200;
```

Execution Time: `0.06407975 sec`

Q3- After Optimization

```
create index index_salary
    using BTREE on
professor(Salary);


select
    *
from
    professor p
join department d on
    p.department_id = d.ID
where
    p.Salary < 1200;
```

Execution Time: `0.016592 sec`

Q3- Percentage Enhancement

We can see an Enhancement in the execution time of `386.20%`

Q4- Before Optimization

```
select
    distinct p.ID,
    p.Name,
    p.Gender,
    p.Salary
from
    professor p
join department d on
    p.department_id = d.ID
where
    p.Name = 'Abel Warren';
```

Execution Time: `0.03356725 sec`

Q4- After Optimization

```
create index index_name
    using HASH on
professor(Name);


select
    distinct p.ID,
    p.Name,
    p.Gender,
    p.Salary
from
    professor p USE INDEX (index_name)
join department d on
    p.department_id = d.ID
where
    p.Name = 'Abel Warren';
```

Execution Time: `0.00032175 sec`

Q4- Percentage Enhancement

We can see an Enhancement in the execution time of `10432.71%`

Q5- Before Optimization

```
select
    distinct p.ID,
    p.Name,
    p.Gender,
    p.Salary
from
    professor p
join department d on
    p.department_id = d.ID
where
    p.Name = 'Abel Warren' and p.Salary > 6000;
```

Execution Time: `0.03569925 sec`

Q5- After Optimization

```
create index index_prof_name
    using HASH on
professor(Name);

create index index_salary
    using BTREE on
professor(Salary);

select
    distinct p.ID,
    p.Name,
    p.Gender,
    p.Salary
from
    professor p
join department d on
    p.department_id = d.ID
where
    p.Name = 'Abel Warren' and p.Salary > 6000;
```

Execution Time: `0.00030475 sec`

Q5- Percentage Enhancement

We can see an Enhancement in the execution time of `11714.27%`

Query Optimization

Q6- Before Optimization

```
select
    distinct s.ID, s.Name, s.City
from
(
  (
    select s2.ID, s2.Name, s2.City
    from student s2
    where s2.Name = "Demond Valentine"
  )
  union
  (
    select s3.ID, s3.Name, s3.City
    from student s3
    where s3.City = "Miami"
  )
) as s;
```

Execution Time: `0.9467585 sec`

Q6- After Optimization

```
select
    distinct s.ID,
    s.Name,
    s.City
from
    student s
where
    (s.Name = "Demond Valentine"
        or s.City = "Miami");
```

Execution Time: `0.111124 sec`

Q6- Percentage Enhancement

We can see an Enhancement in the execution time of `416.41%`

MongoDB Queries

Q1

```
db.getCollection("student_course").aggregate([
    { '$match': { 'Final_Exam_Grade': 'A' } },
    {
        $lookup: {
            from: 'course',
            localField: 'course_id',
            foreignField: '_id',
            as: 'course'
        }
    },
    {
        $unwind: {
            path: '$course',
            preserveNullAndEmptyArrays: false
        }
    },
    {
        $lookup: {
            from: 'student',
            localField: 'student_id',
            foreignField: '_id',
            as: 'student'
        }
    },
    {
        $unwind: {
            path: '$student',
            preserveNullAndEmptyArrays: false
        }
    },
    {
        $lookup: {
            from: 'final_exam',
            localField: 'student.student_id',
            foreignField: 'student_id',
            as: 'final_exam'
        }
    },
    {
        $unwind: {
            path: '$final_exam',
            preserveNullAndEmptyArrays: false
        }
    },
    {
        $lookup: {
            from: 'grade',
            localField: 'final_exam.grade_id',
            foreignField: '_id',
            as: 'grade'
        }
    },
    {
        $unwind: {
            path: '$grade',
            preserveNullAndEmptyArrays: false
        }
    },
    {
        '$group': {
            _id: null, students: {
                $addToSet: {
                    "stud_id": "$student._id",
                    "stud_name": "$student.Name",
                    "stud_gender": "$student.Gender"
                }
            }
        }
    },
    { '$unwind': '$students' },
    { '$set': { 'ID': '$students.stud_id' } },
    { '$set': { 'Name': '$students.stud_name' } },
    { '$set': { 'Gender': '$students.stud_gender' } },
    { '$project': { 'ID': 1, 'Name': 1, 'Gender': 1, _id: 0 }
})
```

Q2

```
db.getCollection("professor_research").aggregate([
    {
        $lookup: {
            from: 'professor',
            localField: 'professor_id',
            foreignField: '_id',
            as: 'professor'
        }
    },
    {
        $unwind: {
            path: '$professor',
            preserveNullAndEmptyArrays: false
        }
    },
    {
        $lookup: {
            from: 'department',
            localField: 'professor.department_id',
            foreignField: '_id',
            as: 'department'
        }
    },
    {
        $unwind: {
            path: '$department',
            preserveNullAndEmptyArrays: false
        }
    },
    { '$match': { 'department.Name': 'Mechanical Electrical & Process Engineering' } },
    {
        $lookup: {
            from: 'research_project',
            localField: 'research_project_id',
            foreignField: '_id',
            as: 'research_project'
        }
    },
    {
        $unwind: {
            path: '$research_project',
            preserveNullAndEmptyArrays: false
        }
    },
    {
        '$group': {
            _id: "professor._id", array: {
                $addToSet: {
                    "prof_name": "$professor.Name",
                    "research_project_name": "$research_project.Name",
                    "research_project_filed": "$research_project.Field"
                }
            }
        }
    },
    {
        $unwind: {
            path: '$array',
            preserveNullAndEmptyArrays: false
        }
    },
    { '$set': { 'Professor_Name': '$array.prof_name' } },
    { '$set': { 'Research_Project_Name': '$array.research_project_name' } },
    { '$set': { 'Research_Project_Field': '$array.research_project_filed' } },
    { '$project': { 'Professor_Name': 1, 'Research_Project_Name': 1,
'Research_Project_Field': 1, _id: 0 } },
])
```

Q3

```
db.getCollection("professor").createIndex(
    {
        "Salary": 1
    },
    {
        name: 'Salary_1',
        unique: false,
        sparse: false
    }
)
db.getCollection("professor").aggregate([
    { '$match': { Salary: { $lt: 1200 } } },
    {
        $lookup: {
            from: 'department',
            localField: 'department_id',
            foreignField: '_id',
            as: 'department'
        }
    },
    {
        $unwind: {
            path: '$department',
            preserveNullAndEmptyArrays: true
        }
    },
])
```

Q4

```
db.getCollection("professor").createIndex(
    {
        "Name": "hashed"
    },
    {
        name: 'Name_1',
        unique: false,
        sparse: false
    }
)
db.getCollection("professor").aggregate([
    { '$match': { Name: 'Abel Warren' } },
    {
        $lookup: {
            from: 'department',
            localField: 'department_id',
            foreignField: '_id',
            as: 'department'
        }
    },
    {
        $unwind: {
            path: '$department',
            preserveNullAndEmptyArrays: true
        }
    },
    { '$group': { _id: null, professor_id: { $addToSet: '$_id' } } },
    { '$unwind': '$professor_id' },
    {
        $lookup: {
            from: 'professor',
            localField: 'professor_id',
            foreignField: '_id',
            as: 'professor'
        }
    },
    {
        $unwind: {
            path: '$professor',
            preserveNullAndEmptyArrays: false
        }
    },
    { '$set': { 'ID': '$professor._id' } },
    { '$set': { 'Name': '$professor.Name' } },
    { '$set': { 'Gender': '$professor.Gender' } },
    { '$set': { 'Salary': '$professor.Salary' } },
    { '$project': { 'ID': 1, 'Name': 1, 'Gender': 1, 'Salary': 1, '_id': 0 }
})
```

Q5

```
db.getCollection("professor").createIndex(
    {
        "Name": "hashed"
    },
    {
        name: 'Name_1',
        unique: false,
        sparse: false
    }
)
db.getCollection("professor").createIndex(
    {
        "Salary": 1
    },
    {
        name: 'Salary_1',
        unique: false,
        sparse: false
    }
)
db.getCollection("professor").aggregate([
    { '$match': { $and: [{ Name: 'Abel Warren' }, { Salary: { $gt: 6000 } }] } },
    {
        $lookup: {
            from: 'department',
            localField: 'department_id',
            foreignField: '_id',
            as: 'department'
        }
    },
    {
        $unwind: {
            path: '$department',
            preserveNullAndEmptyArrays: true
        }
    },
    { '$group': { _id: null, professor_id: { $addToSet: '$_id' } } },
    { '$unwind': '$professor_id' },
    {
        $lookup: {
            from: 'professor',
            localField: 'professor_id',
            foreignField: '_id',
            as: 'professor'
        }
    },
    {
        $unwind: {
            path: '$professor',
            preserveNullAndEmptyArrays: false
        }
    },
    { '$set': { 'ID': '$professor._id' } },
    { '$set': { 'Name': '$professor.Name' } },
    { '$set': { 'Gender': '$professor.Gender' } },
    { '$set': { 'Salary': '$professor.Salary' } },
    { '$project': { 'ID': 1, 'Name': 1, 'Gender': 1, 'Salary': 1, '_id': 0 } },
])
```

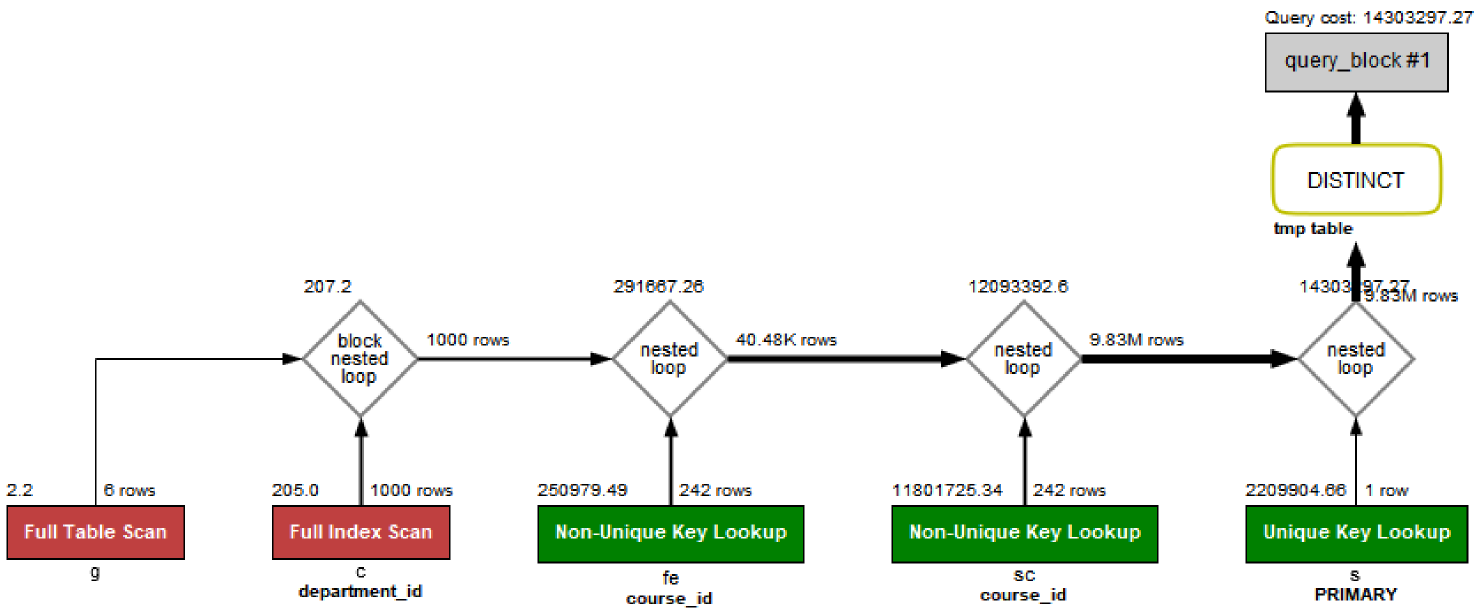
Q6

```
db.getCollection("student").find({
  $or: [
    { City: 'Miami' },
    { Name: 'Demond Valentine' }
  ]
})
```

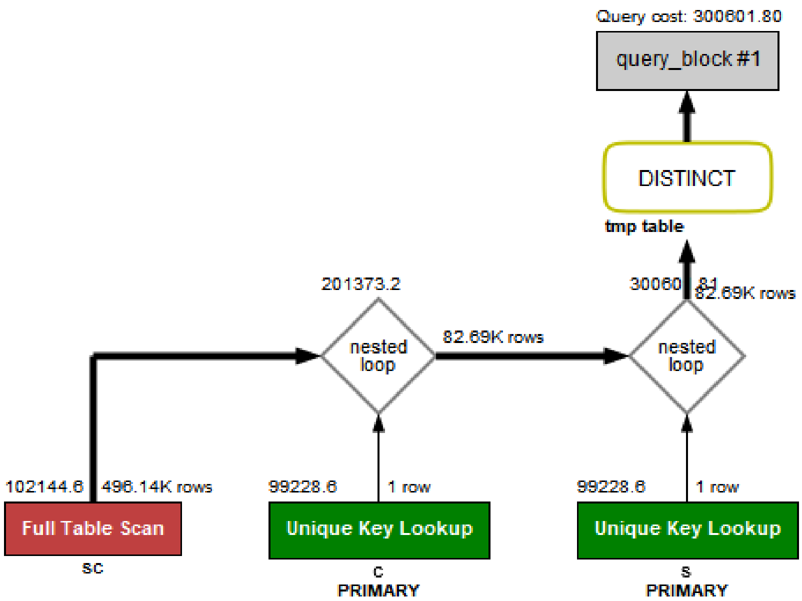
Query Execution Plans

Schema Optimization

Q1- Before Optimization

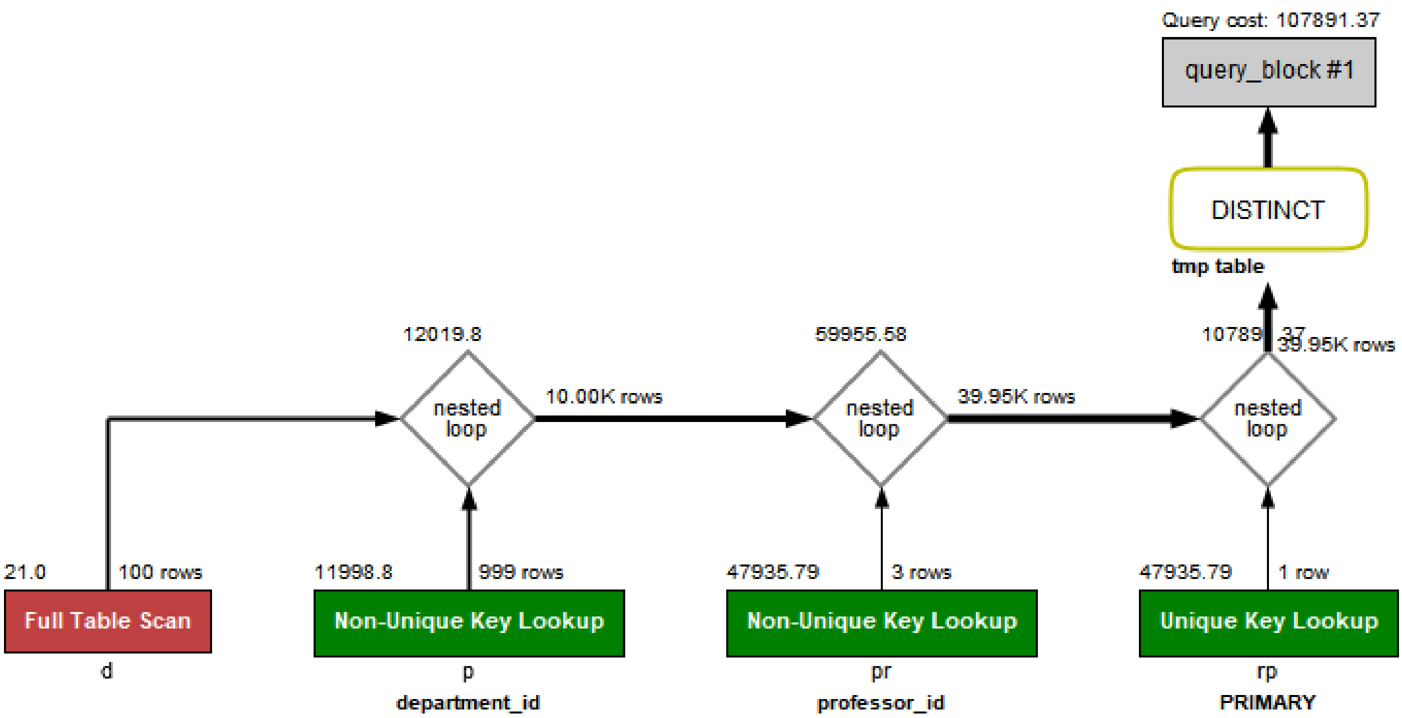


Q1- After Optimization



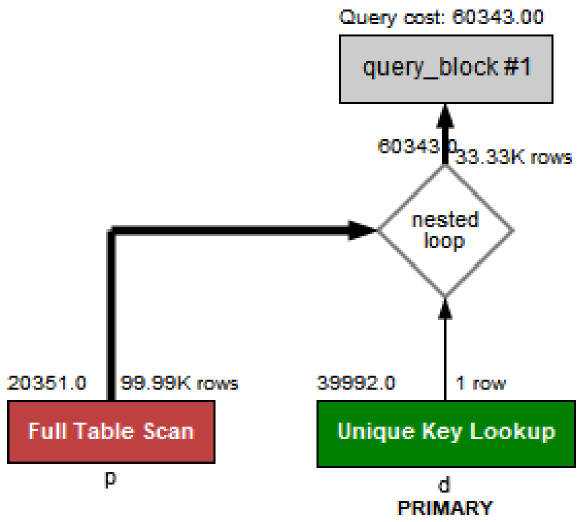
Memory and Cache Optimization

Q2- Before & After Optimization

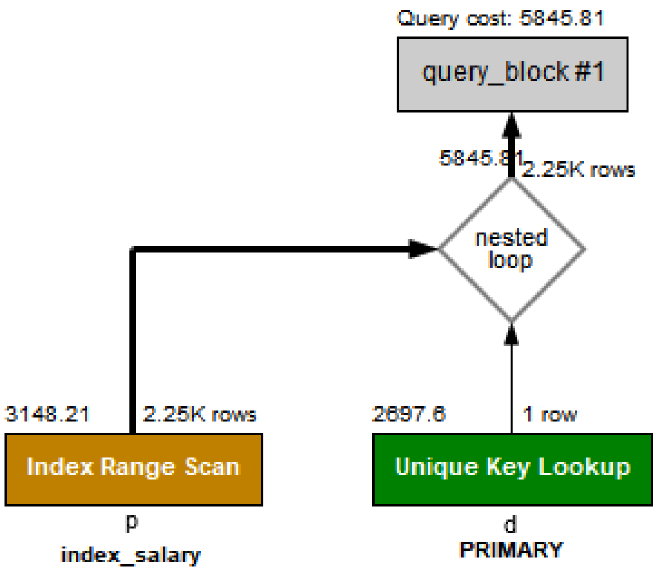


Schema Optimization

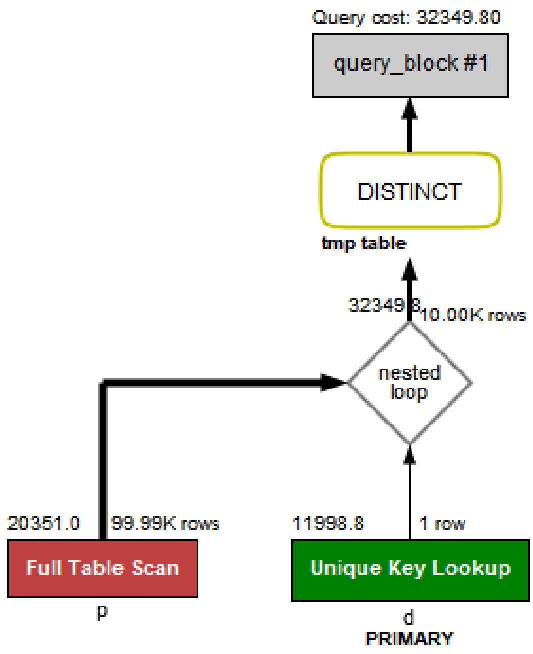
Q3- Before Optimization



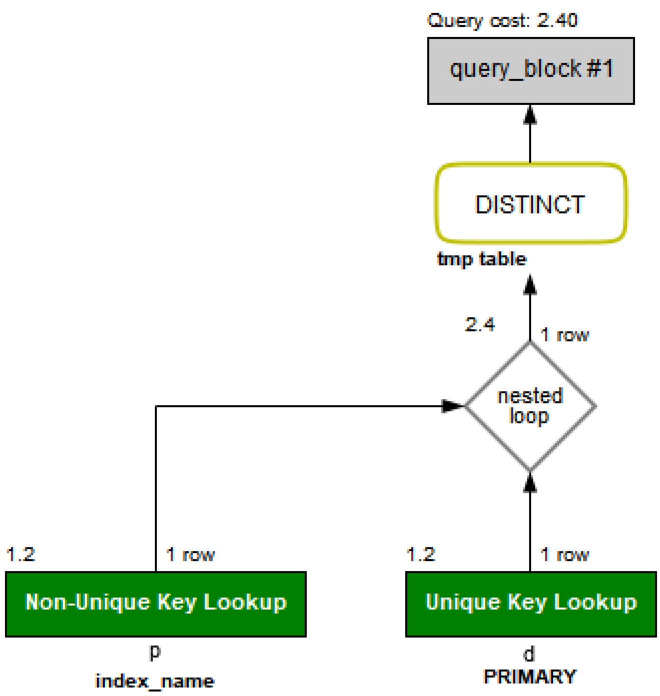
Q3- After Optimization



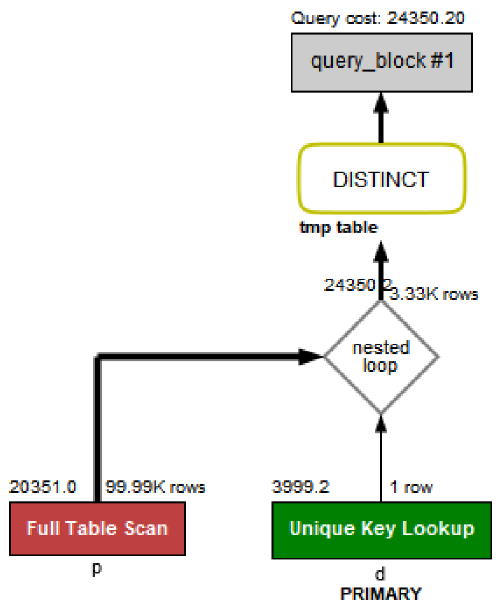
Q4- Before Optimization



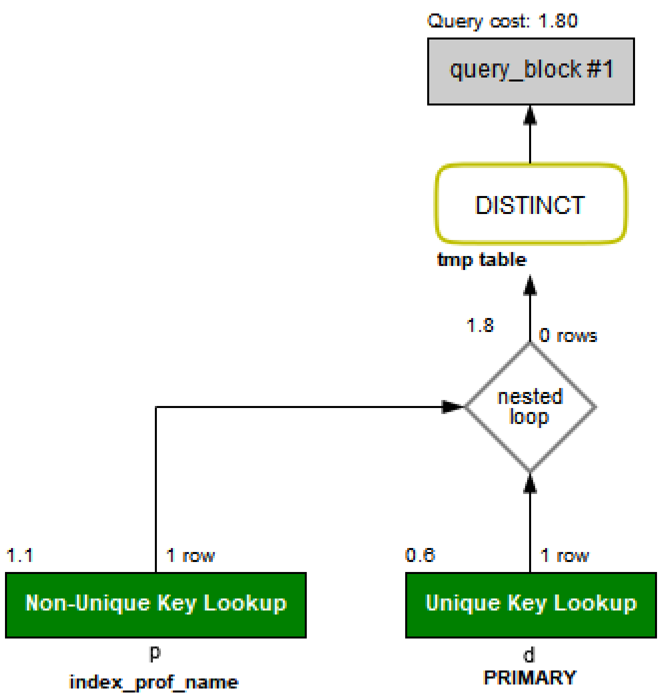
Q4- After Optimization



Q5- Before Optimization

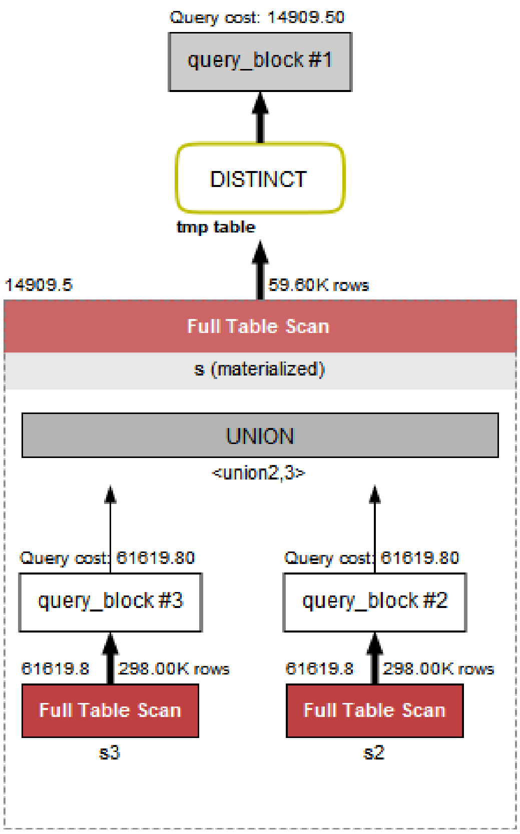


Q5- After Optimization

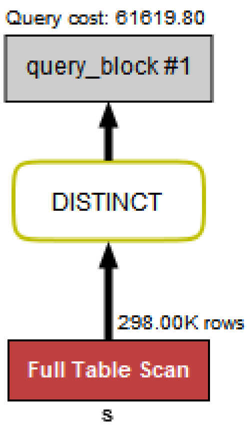


Query Optimization

Q6- Before Optimization



Q6- After Optimization



MySQL Statistics after optimizations

Relation Name	Row count	Main key	Index	FK	Identity column	Avg Row size in (bytes)
course	10,000	Yes	4	2	Yes	159
department	1,000	Yes	1	0	Yes	81
professor	1,000,000	Yes	2	1	Yes	53
professor_research	4,000,000	Yes	3	2	Yes	32
research_project	2,000,000	Yes	1	0	Yes	50
student	4,000,000	Yes	2	1	Yes	112
student_course	11,000,000	Yes	3	2	Yes	97

Note: MySQL creates a B-tree index for any primary key or secondary key

MongoDB Schema after optimizations

```json
{
  "course": {
    "_id": "int",
    "ID": "int",
    "Name": "string",
    "Code": "string",
    "department_id": "int",
    "professor_id": "int"
  },
  "department": {
    "_id": "int",
    "ID": "int",
    "Name": "string"
  },
  "professor_research": {
    "_id": "int",
    "ID": "int",
    "research_project_id": "int",
    "professor_id": "int"
  },
  "professor": {
    "_id": "int",
    "ID": "int",
    "Name": "string",
    "Gender": "string",
    "Salary": "int",
    "Date_of_Birth": "string",
    "department_id": "int"
  },
  "research_project": {
    "_id": "int",
    "ID": "int",
    "Name": "string",
    "Field": "string",
    "Duration_in_Months": "int"
  },
  "student_course": {
    "_id": "int",
    "ID": "int",
    "student_id": "int",
    "course_id": "int",
    "Final_Exam_Grade": "string",
    "Final_Exam_PDF": "string"
  },
  "student": {
    "_id": "int",
    "ID": "int",
    "Name": "string",
    "Phone_Number": "string",
    "Gender": "string",
    "Date_of_Birth": "string",
    "Street": "string",
    "City": "string",
    "Town": "string",
    "department_id": "int"
  }
}
```

Validation Details

Optimizations Enhancements

Schema Optimization

DataBase Size	Q1 Unoptimized Time in Sec	Q1 Optimized Time in Sec	Percentage time enhancement
~90K	3.55346	0.3324605	1068.84%
~900K	31.2632785	3.15877125	989.73%
~9M	446.6810555	59.81084525	746.82%
~20M	2359.50108875	61.87842125	3813.12%

DataBase Size	Q1 Unoptimized Memory Req.	Q1 Optimized Memory Req.	Memory enhancement
~90K	~10 MB	~11.5 MB	86.96%
~900K	~83 MB	~105 MB	79.05%
~9M	~751 MB	~963 MB	77.99%
~20M	~1.54 GB	~1.86 GB	82.80%

Comments:

- We see a significant decrease in time needed for the query after the schema optimization.

- At the first glance, you might think that we had a degradation in memory but that is not the case. For the sake of fair comparison in execution time. We had to generate the same size of data for both schemas so the removal of the 2 unnecessary relations enabled us to add more data in the student course relation (involved in our query), allowing much more (double to triple) students enrollment in courses which increased the size of the database slightly.

Memory and Cache Optimization

DataBase Size	Q2 Unoptimized Time in Sec	Q2 Optimized Time in Sec	Percentage Time Enhancement
~90K	0.48396325	0.0000995	486395.23%
~900K	1.18268525	0.00016575	713535.60%
~9M	6.04019	1.69611925	356.12%
~20M	22.2689345	12.38255825	179.84%

Comments:

- We notice a significant decrease in time needed when the cache is used with the frequent queries with a chosen Cache size of 10MB.

- We notice a decrease in the rate of percentage time enhancement when the database size increases due to the fact that the cache size is limited and can not contain all the results of the query.

- We may use other memory enhancements:
    a. increasing block size which will require rebuilding the database so we didn’t use it.
    b. using materialized views
    c. increasing the buffer pool size but we decided not to go further with the buffer pool size increase as it causes some issues with the operating system used for example during our experiments with the different buffer pool sizes all the programs running crashed so better safe than sorry.

Index Tuning

DataBase Size	Q3 Unoptimized Time in Sec	Q3 Optimized Time in Sec	Percentage Time Enhancement
~90K	0.01103675	0.00123075	896.75%
~900K	0.03759075	0.005891	638.10%
~9M	1.201752	0.07439075	1615.46%
~20M	1.27200575	0.09247875	1375.46%

DataBase Size	Q3 Unoptimized Memory Req.	Q3 Optimized Memory Req.	Memory Enhancement
~90K	~1.7 MB	~1.861 MB	91.35%
~900K	~8 MB	~9.5 MB	84.21%
~9M	~73.1 MB	~88.6 MB	82.51%
~20M	~73.1 MB	~88.6 MB	82.51%

Comments:

- We see a significant decrease in time needed for the query after the index tuning using btree index for range queries.

- Using indexes has the tradeoff of speed vs memory usage as it degrades the memory usage in exchange for speed data retrieval but also slows down insertions, updates, and deletes.

- We notice the memory for 9M and 20M in databases is the same which is due to the constant number of data in the professor table between the two database instances. As the increase in database size was chosen to be in the students' related tables.

DataBase Size	Q4 Unoptimized Time in Sec	Q4 Optimized Time in Sec	Percentage Time Enhancement
~90K	0.0891485	0.00031575	28233.89%
~900K	0.4455435	0.0003254	136921.79%
~9M	0.9921065	0.00044525	222820.10%
~20M	0.90361375	0.0004975	181630.91%

DataBase Size	Q4 Unoptimized Memory Req.	Q4 Optimized Memory Req.	Memory Enhancement
~90K	~1.7 MB	~2 MB	85.00%
~900K	~8 MB	~11.5 MB	69.57%
~9M	~73.1 MB	~99.7 MB	73.32%
~20M	~73.1 MB	~99.7 MB	73.32%

Comments:

- We see a significant decrease in time needed for the query after the index tuning using a hash index for equality queries.

- Using indexes has the tradeoff of speed vs memory usage as it degrades the memory usage in exchange for speed data retrieval but also slows down insertions, updates, and deletes.

- We notice the memory for 9M and 20M in databases is the same which is due to the constant number of data in the professor table between the two database instances. As the increase in database size was chosen to be in the students' related tables.

DataBase Size	Q5 Unoptimized Time in Sec	Q5 Optimized Time in Sec	Percentage Time Enhancement
~90K	0.4457895	0.00030425	146520.79%
~900K	0.45180375	0.00030525	148011.06%
~9M	1.2987235	0.00029775	436179.18%
~20M	0.8519075	0.00038075	223744.58%

DataBase Size	Q5 Unoptimized Memory Req.	Q5 Optimized Memory Req.	Memory Enhancement
~90K	~1.7 MB	~2.1 MB	80.95%
~900K	~8 MB	~13.1 MB	61.07%
~9M	~73.1 MB	~115.2 MB	63.45%
~20M	~73.1 MB	~115.2 MB	63.45%

Comments:

- We see a significant decrease in time needed for the query after the index tuning using multiple indexes i.e hash and btree indexes for queries with both equality and range conditions.

- Using indexes has the tradeoff of speed vs memory usage as it degrades the memory usage in exchange of speed data retrieval but also slows down insertions, updates, and deletes.

- We notice the memory for 9M and 20M in databases is the same which is due to the constant number of data in the professor table between the two database instances. As the increase in database size was chosen to be in the students' related tables.

Query Optimization

DataBase Size	Q6 Unoptimized Time in Sec	Q6 Optimized Time in Sec	Percentage Time Enhancement
~90K	0.500985	0.01107775	4522.44%
~900K	0.9467585	0.111124	851.98%
~9M	7.90295675	3.8419745	205.70%
~20M	9.18613725	4.267862	215.24%

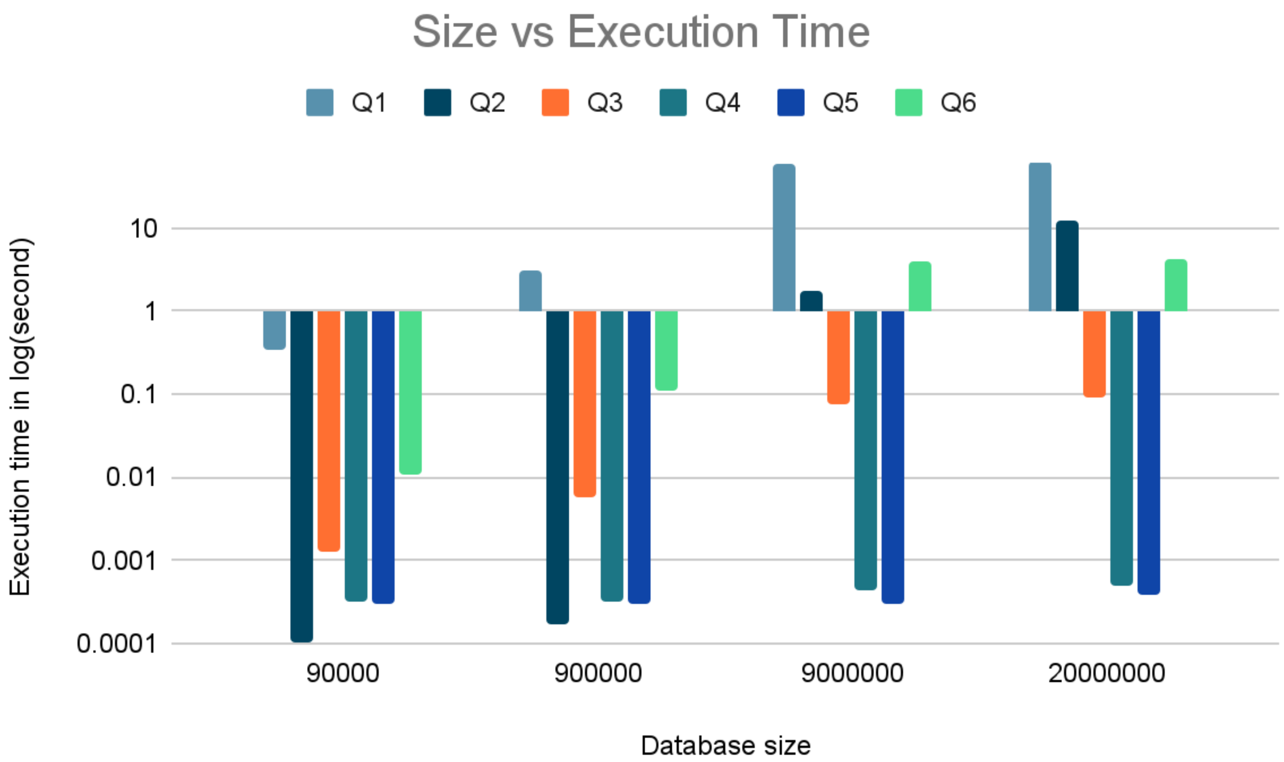
DataBase Size	Q6 Unoptimized Memory Req.	Q6 Optimized Memory Req.	Memory Enhancement
~90K	~4 MB	~4 MB	100%
~900K	~39.1 MB	~39.1 MB	100%
~9M	~349.4 MB	~349.4 MB	100%
~20M	~460.5 MB	~460.5 MB	100%

Comments:

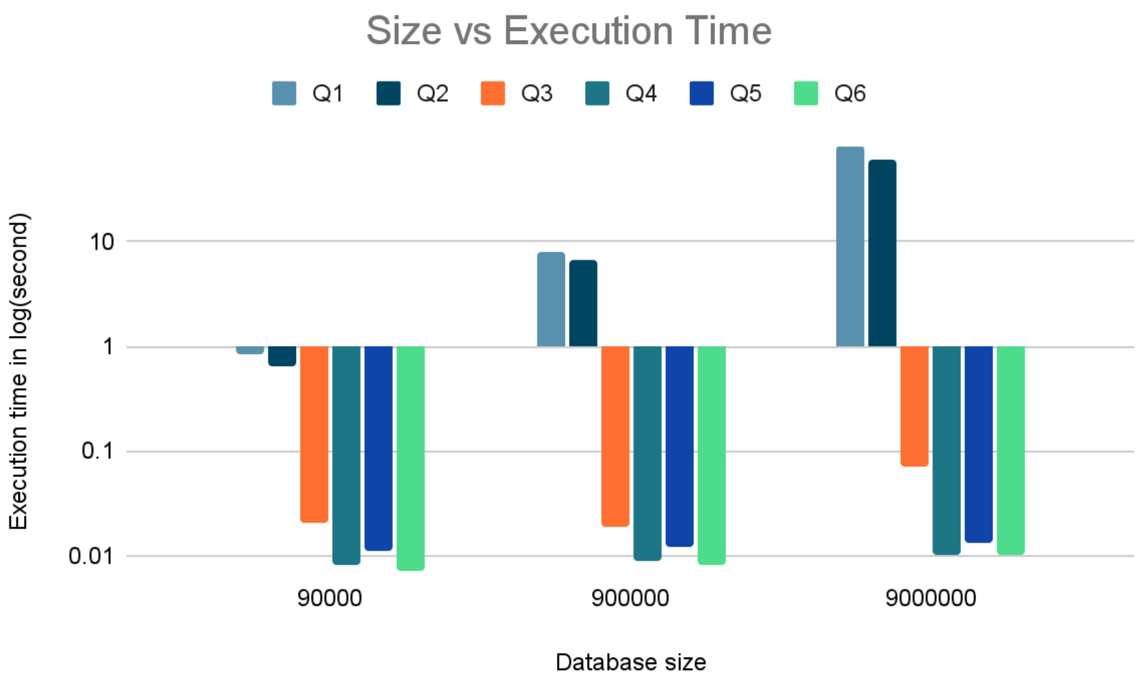
- We see a significant decrease in time needed for the query after the index tuning using btree index for range queries.
- We should not use Union unless it's absolutely necessary as it costs us a sort on the data to achieve that Union which is a very time-consuming process in general.

Database Size Effect on Execution Time

MySQL Queries



MongoDB Queries

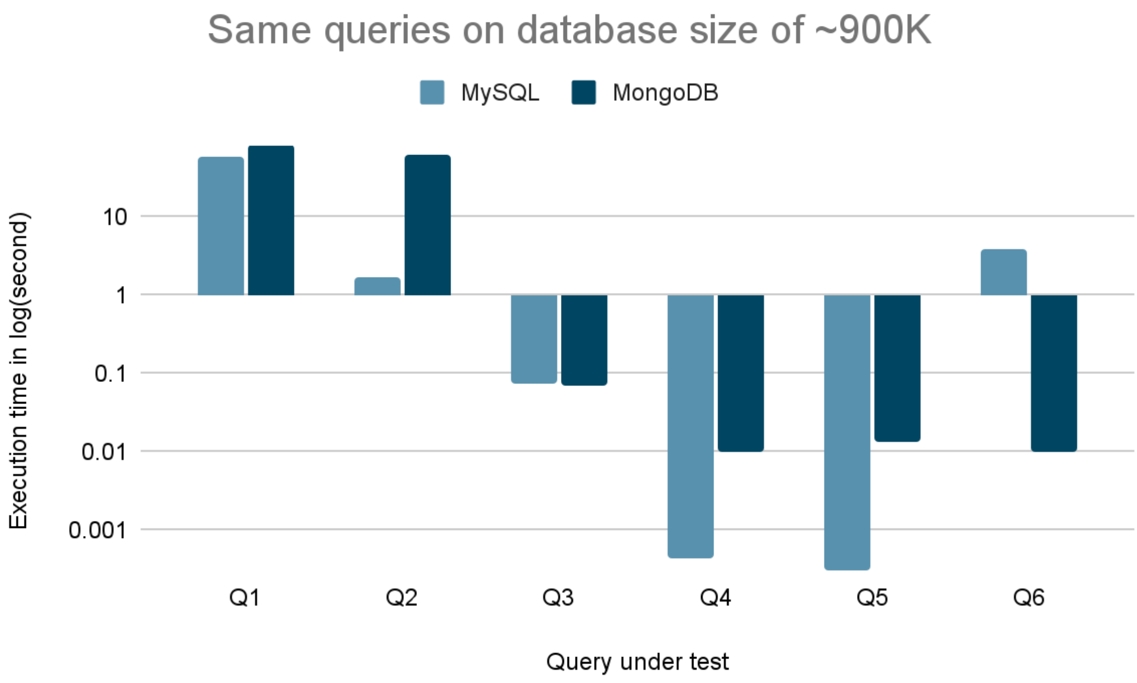


DataBase Size	Q1 Time in Sec	Q2 Time in Sec	Q3 Time in Sec	Q4 Time in Sec	Q5 Time in Sec	Q6 Time in Sec
~90K	0.854	0.656	0.021	0.008	0.011	0.007
~900K	8.029	6.718	0.019	0.009	0.012	0.008
~9M	82.105	61.185	0.072	0.010	0.013	0.010

Comments:

- It was very impractical to test the data on a database of order 10 Million as it takes too much already on the data in order of 1 Million so we skipped that size.

SQL vs NOSQL



Recommendations

- Using indexing in MySQL improves the queries execution time significantly on the cost of additional memory and slower updates.

- Taking your time to design the schema is a very important step in database design as we showed how schema optimization significantly improves the execution time of the queries involved in that optimization.

- Using SQL is better than NOSQL in complex queries that join multiple tables together, while NOSQL is better in retrieving a huge number of records from the same collection so when designing the NOSQL schema we should consider Embedding rather than referencing whenever possible.

- For the queries proposed in this document after the comparisons, we can confidently recommend using MySQL rather than MongoDB.

- Using caching significantly decreases the execution time of frequently used selection queries. So we recommend using caching whenever possible but note that when data is too large you need a very large cache to have that significant speedup.