CMPN-401

Advanced Database Systems

Cairo University

Faculty of Engineering

ADB Project report

Submitted by:

- Omar Mohamed Ahmed Abdellatif 1162003
- Mahmoud Atwa 1162014
- Ahmed Mohamed Ali Gad Hashish 1162017
- Ahmed Mohamed Ahmed Hassan 1162057

Submitted to:

- Dr. Michael Nawar
- Eng. Mostafa Mahmoud

Contents

Specs of the used device:

- Processor: Intel Core i7-9750H CPU @ 2.60GHz
- RAM: 16.0 GB
- OS: Windows 10 64-bit

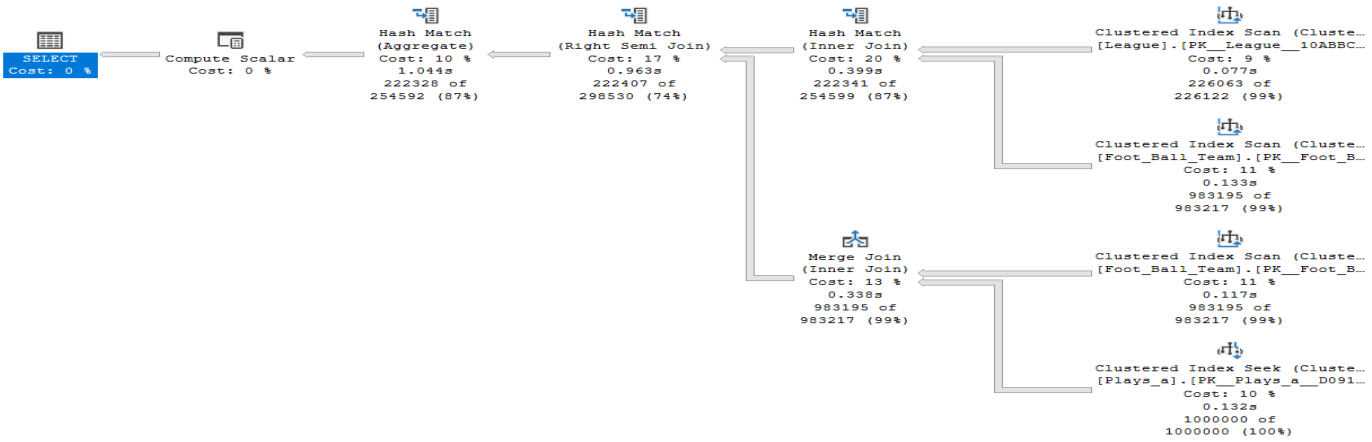
Before and After Optimization Query Statistics:

Query 1:

Before Optimization:

```
SELECT COUNT(*)
FROM dbo.Plays_a,dbo.Foot_Ball_Team,dbo.Game
WHERE dbo.Plays_a.TeamID = dbo.Foot_Ball_Team.TeamID AND dbo.Plays_a.GameID = dbo.Game.GameID
AND  Foot_Ball_Team.Goals IN
(SELECT dbo.Foot_Ball_Team.Goals
FROM dbo.Foot_Ball_Team, dbo.League
WHERE dbo.Foot_Ball_Team.LeagueID = dbo.League.LeagueID AND dbo.League.year > 2007
)
and Foot_Ball_Team.Goals > 15000000
GROUP BY dbo.Foot_Ball_Team.Goals
```

- Execution Plan:



- Theoretical parallel query processing:

  The following can be executed in parallel:

  - Clustered Index scan on League table
  - Two clustered Index scans on Foot_Ball_Team table
  - Clustered Index seek on Play_a table

  Also the following can be executed in parallel:

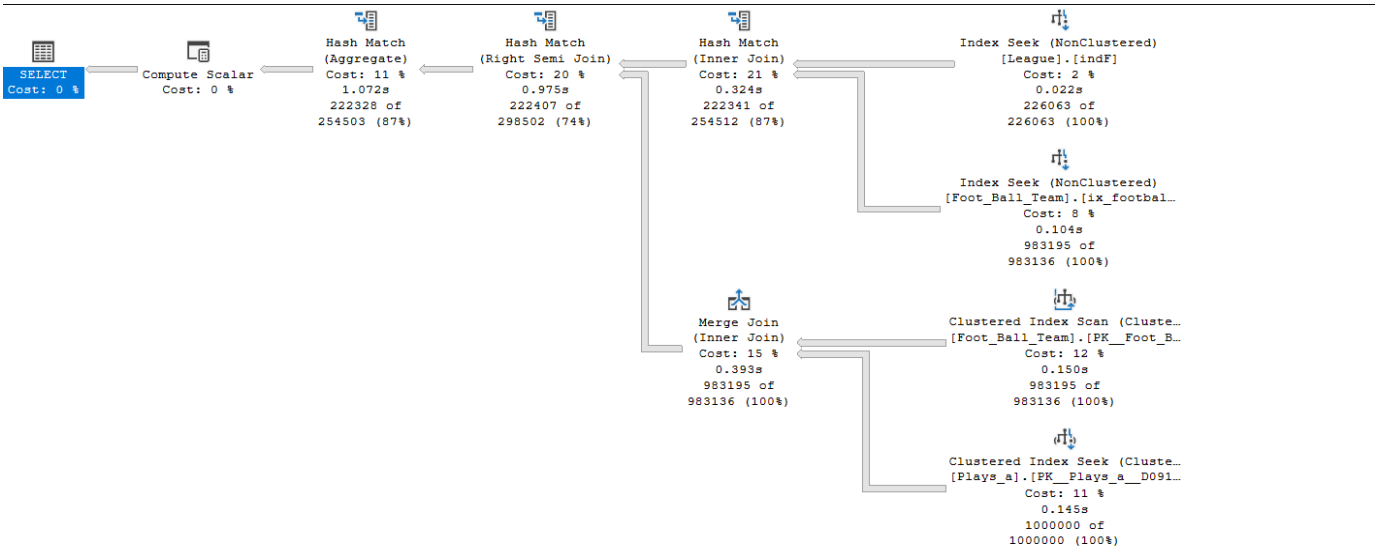
  - Hash Match
  - Merge Join

After Optimization:

- ### Optimization using Indexing:

```
CREATE INDEX indf ON dbo.League(year)
CREATE INDEX ix_footballA ON dbo.Foot_Ball_Team(Goals,LeagueID)
```

- ### Execution plan:



- ### Theoretical parallel query processing:

The following can be executed in parallel:

- Index seek on League table
- Index seek on Foot_ball_Team table
- Clustered Index scan on Foot_ball_Team table
- Clustered Index seek on Plays_a table

Also the following can be executed in parallel:

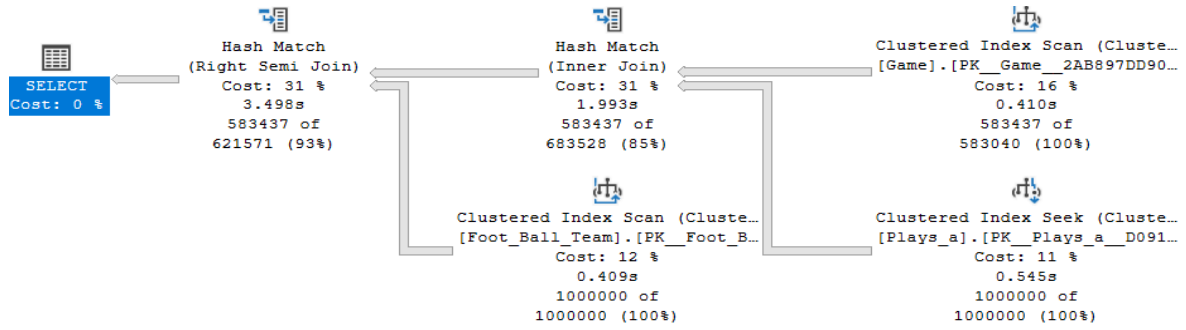
- Hash Match
- Merge Join

Query 2:

Before Optimization:

```
SELECT dbo.Foot_Ball_Team.Name
FROM dbo.Foot_Ball_Team
WHERE TeamID IN (SELECT dbo.Plays_a.TeamID
FROM dbo.Plays_a,dbo.Game
WHERE dbo.Plays_a.GameID = dbo.Game.GameID AND dbo.Game.Monthh > 5)
```

- ## Execution Plan:



- ## Theoretical parallel query processing:

  The following can be executed in parallel:

  - Clustered Index scan on Game table
  - Clustered Index seek on Plays_a table
  - Clustered Index scan on Foot_Ball_Team table

After Optimization:

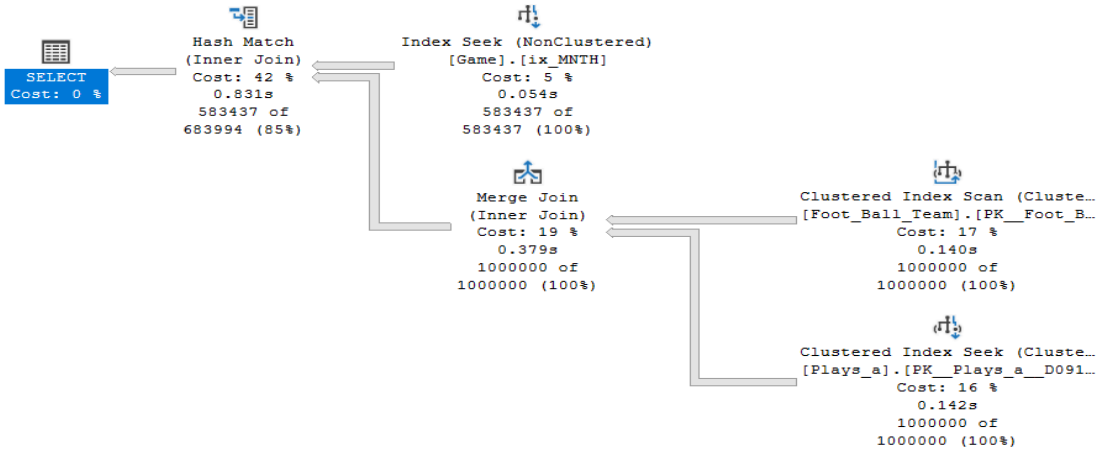
- ## Optimization using Indexing:

```
CREATE INDEX ix_MNTH ON dbo.Game(Monthh)
```

- ## Optimization using query re-writing:

```
SELECT dbo.Foot_Ball_Team.Name
FROM dbo.Foot_Ball_Team,dbo.Plays_a,dbo.Game
WHERE Foot_Ball_Team.TeamID = Plays_a.TeamID AND Plays_a.GameID = Game.GameID AND
dbo.Game.Monthh>5
```

- Execution plan:



- Theoretical parallel query processing:

The following can be executed in parallel:

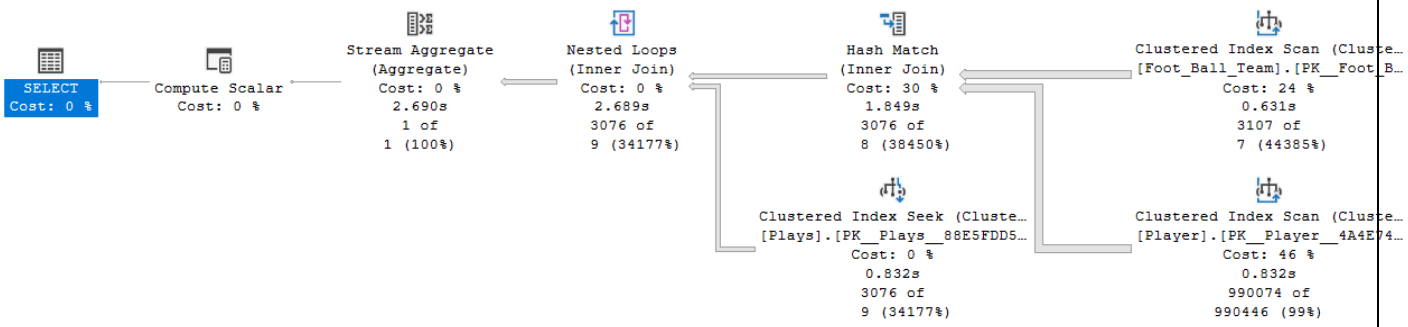
- Clustered Index scan on Foot_Ball_Team table
- Clustered Index seek on Plays_a table
- Index seek on Game table

Query 3:

Before Optimization:

```
SELECT COUNT(*)
FROM dbo.Plays
INNER JOIN dbo.Player ON Player.PlayerID = Plays.PlayerID
INNER JOIN dbo.Foot_Ball_Team ON Foot_Ball_Team.TeamID = Player.TeamID
WHERE SUBSTRING(dbo.Foot_Ball_Team.Name,1,2) = 'AB'
```

- ### Execution Plan:



- ### Theoretical parallel query processing:

  The following can be executed in parallel:

  - Clustered Index scan on Foot_Ball_Team table
  - Clustered Index scan on Player table
  - Clustered Index seek on Plays table

After Optimization:

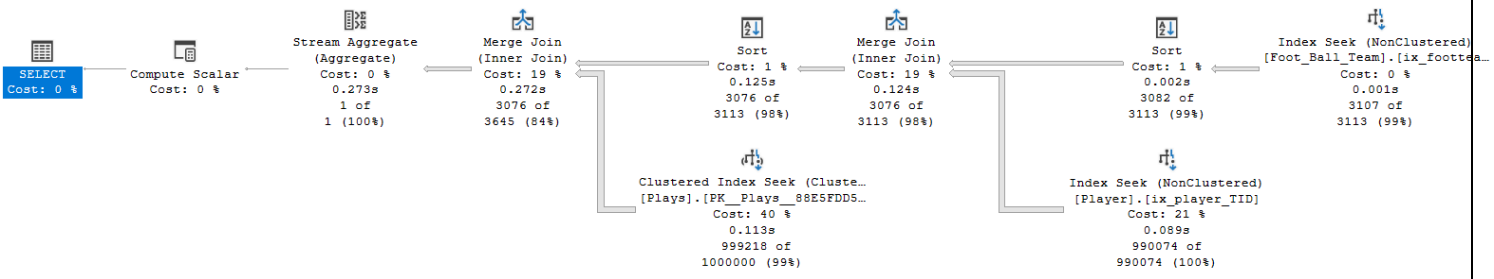
- ### Optimization using Indexing:

```
CREATE INDEX ix_player_TID ON dbo.Player(TeamID)
CREATE INDEX ix_footteam_team_name ON dbo.Foot_Ball_Team(Name)
```

- ### Optimization using query re-writing:

```
SELECT COUNT(*)
FROM dbo.Plays
INNER JOIN dbo.Player ON Player.PlayerID = Plays.PlayerID
INNER JOIN dbo.Foot_Ball_Team ON Foot_Ball_Team.TeamID = Player.TeamID
WHERE dbo.Foot_Ball_Team.Name LIKE 'AB%'
```

- Execution plan:



- Theoretical parallel query processing:

  The following can be executed in parallel:

  - Index seek on Player table
  - Index seek on Foot_Ball_Team table
  - Clustered Index seek on Plays table

Optimization Details:

The new database statistics after modification:

- Disk Usage by Partition:

Table Name	# Records	Reserved (KB)	Used (KB)
⊟ dbo.Foot_Ball_Team	1,000,000	104,344	75,144
⊞ Index (PK__Foot_Bal__123AE7B93A9770BB)	1,000,000	65,160	36,088
⊞ Index (ix_footballA)	1,000,000	17,928	17,896
⊞ Index (ix_footteam_name)	1,000,000	21,256	21,160
⊟ dbo.Game	1,000,000	79,696	70,056
⊞ Index (PK__Game__2AB897DD900A37E4)	1,000,000	65,736	56,160
⊞ Index (ix_MNTH)	1,000,000	13,960	13,896
⊟ dbo.League	1,000,000	78,864	42,016
⊞ Index (PK__League__10ABBC14D2017D91)	1,000,000	64,904	28,120
⊞ Index (indF)	1,000,000	13,960	13,896
⊟ dbo.Player	1,000,000	144,464	95,624
⊞ Index (PK__Player__4A4E74A83ADD5CF7)	1,000,000	130,504	81,720
⊞ Index (ix_player_TID)	1,000,000	13,960	13,904
⊟ dbo.Plays	1,000,000	65,544	36,776
⊞ Index (PK__Plays__88E5FDD575D92846)	1,000,000	65,544	36,776
⊟ dbo.Plays_a	1,000,000	65,288	31,696
⊞ Index (PK__Plays_a__D0916EC47299211C)	1,000,000	65,288	31,696
⊟ dbo.Purchase_Tickets_Entrance	1,000,000	65,288	31,696
⊞ Index (PK__Purchase__4999D91FE8CEFA4E)	1,000,000	65,288	31,696
⊟ dbo.Round	1,000,000	64,712	20,872
⊞ Index (PK__Round__94D84E1A6045E08A)	1,000,000	64,712	20,872
⊟ dbo.Userr	1,000,000	65,672	54,520
⊞ Index (PK__Userr__3214EC27F76887D0)	1,000,000	65,672	54,520

- Disk Usage by Table:

Table Name	# Records	Reserved (KB)	Data (KB)	Indexes (KB)	Unused (KB)
dbo.Foot_Ball_Team	1,000,000	104,344	35,944	39,200	29,200
dbo.Game	1,000,000	79,696	55,944	14,112	9,640
dbo.League	1,000,000	78,864	28,008	14,008	36,848
dbo.Player	1,000,000	144,464	81,408	14,216	48,840
dbo.Plays	1,000,000	65,544	36,592	184	28,768
dbo.Plays_a	1,000,000	65,288	31,536	160	33,592
dbo.Purchase_Tickets_Entrance	1,000,000	65,288	31,536	160	33,592
dbo.Round	1,000,000	64,712	20,784	88	43,840
dbo.Userr	1,000,000	65,672	54,312	208	11,152

The enhancements made:

The enhancements in the schema:

No enhancements were needed to be made in the schema

The enhancement in memory management:

We used stored procedures because they can be used as a modular programming technique, which means create once, store, and then use several times whenever required. This means faster execution and no need to write and store different .sql file types that might get lost.

We decreased the size of varchar attributes from 55 to 20, which impacted the size of the database. Additionally, any query that tried to access any varchar typed attribute is now executed much faster.

The modification in the indexes:

The following indices were made to enhance the query performance:

- `CREATE INDEX indf ON dbo.League(year)`
- `CREATE INDEX ix_footballA ON dbo.Foot_Ball_Team(Goals,LeagueID)`
- `CREATE INDEX ix_MNTH ON dbo.Game(Monthh)`
- `CREATE INDEX ix_footteam_name ON dbo.Foot_Ball_Team(Name)`
- `CREATE INDEX ix_player_TID ON dbo.Player(TeamID)`

Modifications on the query statements:

For query 2, the inner query was removed and replaced with join conditions and a condition on the “Monthh” field of the Game table. Note that SQL server implicitly converts the join conditions to inner join.

For query 3, the scalar-valued function “SUBSTRING” was replaced with the “LIKE” operator. Index seek operators cannot be used in general with scalar-valued functions, which is why they should be avoided.

Validation Details

Effect of modifications on query performance:

Queries	Non-Optimized	Query Optimization	Indexing	Stored Procedure	Reduce col. size
Query 1	CPU: 3782 Elapsed: 6694 Memory:88 Kb	No query optimization	CPU: 3297 Elapsed: 5677 [15.2%] Memory:96 Kb [-9%]	CPU: 2987 Elapsed: 5449[18.6%] Memory:96 Kb [-9%]	CPU: 2672 Elapsed: 5091[23.9%] Memory:96 Kb [-9%]
Query 2	CPU: 3453 Elapsed: 9362 Memory:64 Kb	CPU: 2312 Elapsed: 7781[16.8%] Memory:48 Kb [25%]	CPU: 2110 Elapsed: 7665[18.1%] Memory:48 Kb [25%]	CPU: 1937 Elapsed: 7173[23.38%] Memory:48 Kb [25%]	CPU: 1828 Elapsed: 6952[25.74%] Memory:48 Kb [25%]
Query 3	CPU: 1766 Elapsed: 2759 Memory:56 Kb	CPU: 1703 Elapsed: 1946[29.47%] Memory:40 Kb [29%]	CPU: 1579 Elapsed: 1753[36.46%] Memory:40 Kb [29%]	CPU: 1141 Elapsed: 1133[58.9%] Memory:40 Kb [29%]	CPU: 1000 Elapsed: 1012[63.3%] Memory:40 Kb [29%]

(Time is measured in ms)

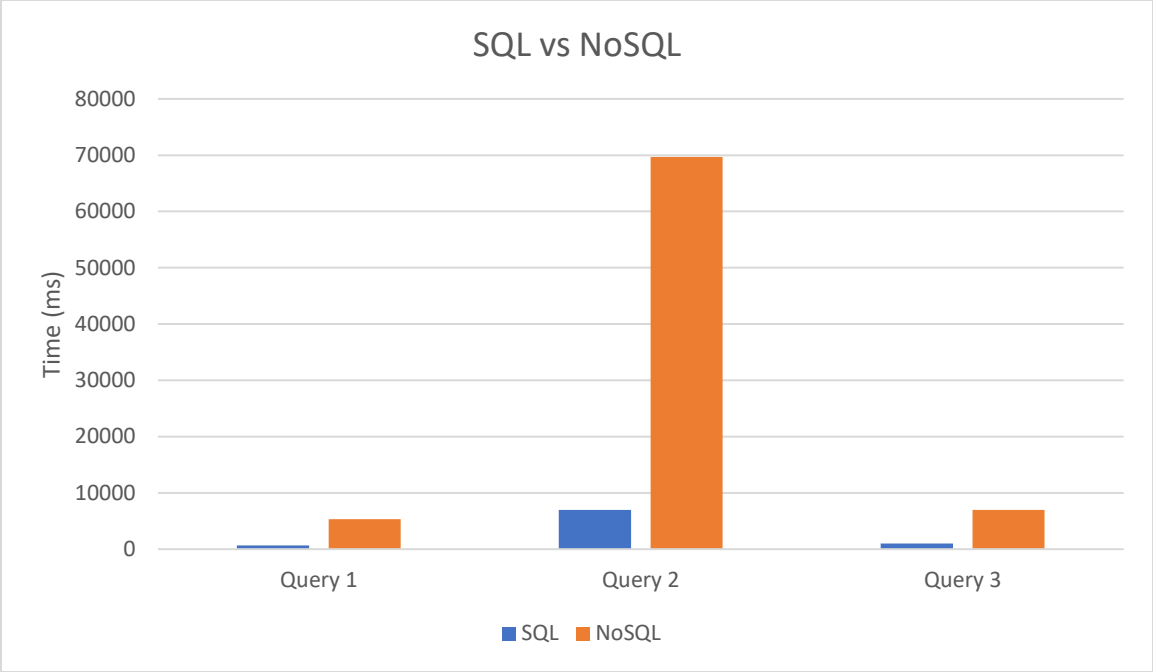
Effect of database size on query performance:

Queries	10,000	100,000	1,000,000	10,000,000
Query 1	CPU: 110 Elapsed: 226 Memory:96 Kb	CPU: 343 Elapsed: 687 Memory:96 Kb	CPU: 2672 Elapsed: 5091 Memory:96 Kb	CPU: 34812 Elapsed: 58147 Memory:96 Kb
Query 2	CPU: 47 Elapsed: 253 Memory:64 Kb	CPU: 234 Elapsed: 1019 Memory:48 Kb	CPU: 1828 Elapsed: 6952 Memory:48 Kb	CPU: 26296 Elapsed: 73637 Memory:48 Kb
Query 3	CPU: 32 Elapsed: 93 Memory:40 Kb	CPU: 141 Elapsed: 231 Memory:40 Kb	CPU: 1000 Elapsed: 1012 Memory:40 Kb	CPU: 11180 Elapsed: 13506 Memory:40 Kb

(Time is measured in ms)

SQL vs NoSQL:

Comparisons were made on a database of size 1,000,000 rows, however, for query 1, we had to use a database of size 100,000 because we got an error converting to BSON as the size of the object exceeded the 16 MBs limit.



Equivalent NoSQL queries used:

Query1:

```
db.dbo.Foot_Ball_Team.find({
  Goals: {$in:
    db.dbo.Foot_Ball_Team.find({
      LeagueID: {$in:
        db.dbo.League.find({
          year: {$gt: 2007}
        }).map(l => l.LeagueID)
      },
      Goals: {$gt: 15000000}
      }).map(t=>t.Goals)
  },
  TeamID: {$in:
    db.dbo.Plays_a.find({
      GameID: {$in:
        db.dbo.Game.find().map(g => g.GameID)
      }
    }).map(p => p.TeamID)
  }
```

```
}).map(x => x.Name).length
```

Query 2:

```
db.dbo.Foot_Ball_Team.find({
  TeamID: {$in:
  db.dbo.Plays_a.find({
  GameID: {$in:
           db.dbo.Game.find({
           Monthh: {$gt: 5}
                        }).map(g => g.GameID)
           }
         }).map(p => p.TeamID)
   }
}).map(t => t.Name)
```

Query 3:

```
db.dbo.Plays.find({
  PlayerID: {$in:
        db.dbo.Player.find({
          TeamID: {$in:
                db.dbo.Foot_Ball_Team.find({
                Name: /^AB.*/i
                }).map(t => t.TeamID)
          }
        }).map(p => p.PlayerID)
  }
  }).map(x=>x).length
```

Comments and Recommendations:

- We would recommend using SQL server as it does some pretty good optimizations to the queries implicitly, and it also suggests generating indices when applicable to enhance the query performance.
- We would also recommend trying to understand the execution plan and try to look for any index scans being performed and perhaps generating a non-clustered index to convert the scanning operations into seeking operations, which would enhance the query performance.