**Cairo University**

**Faculty of Engineering**

**Computer Department**

# Advanced Database Systems Project Phase 2

Team 11



| Name | Sec. | B.N. |
|---|---|---|
| Omar Mohammed | 2 | 08 |
| Nourhan Kamal | 2 | 27 |
| Walid Mohammed | 2 | 31 |
| Yahia Ali | 2 | 33 |

# Query Selection

First: To write queries, we tried a lot of query types with minimum depth of 3, but it was hard to find queries that takes enough time so we can notice the optimization's effect.

For example, we tried the 'equal' and 'AND' conditions. Both of them produce very small results (0 to 10,000 records) and run in less than a second. We also tried adding more conditions but this too decreases the results and time.

We decided to use these 3 queries, each of them are run on 1,000,000 records for each table.

| SQL Query | |
|---|---|
| #1 | `Select * from Member M where EXISTS ( select * from Travel T where M.MemberID=T.MEID and (T.Feedback>0 or EXISTS ( select * from Trip Tr where T.TRID=Tr.TripID and Tr.Cost > 1500)));` |
| #2 | `Select * from Member M where EXISTS ( select * from Participate P where M.MemberID=P.MEID and EXISTS ( select * from Team T where P.TEID=T.TeamID and exists ( select * from Compete C where C.TEID =T.TeamID and (C.Points>500 and (C.Standing<8000 or T.CoachID<100000)))));` |
| #3 | `Select * from Employee E where EXISTS ( select * from Trip T where E.EmployeeID=T.SupervisorID and EXISTS ( select * from Travel Tr where T.TripID=Tr.TRID and (Tr.Feedback > 0 or E.Salary>500)));` |

| Query | Time | #result row | depth | #conditions |
|---|---|---|---|---|
| #1 | 00:00:05.065 | 618,164 | 3 | 2 |
| #2 | 00:00:08.266 | 995,060 | 4 | 3 |
| #3 | 00:00:07.522 | 872,248 | 3 | 2 |

Note: The run time varies with each run. The results are the median of several runs.

# Results of using databases with different sizes

Running the queries on database instances with less than 100,000 records per table is too fast (less than one second) to compare, so we don't mention any of those results

| Query\ # of record | 100,000 | 1,000,000 | 10,000,000 | 10,000,000 (*) | |
|---|---|---|---|---|---|
| #1 | 00:00:00.714 | 00:00:05.065 | 00:00:52.965 | 00:00:25.517 | 622K row |
| #2 | 00:00:00.703 | 00:00:08.266 | 00:01:00.498 | 00:00:08.431 | 945K row |
| #3 | 00:00:00.995 | 00:00:07.522 | 00:01:11.878 | 00:00:08.956 | 880K row |

*edit the condition so it return the same number of row as the 1,000,000 for

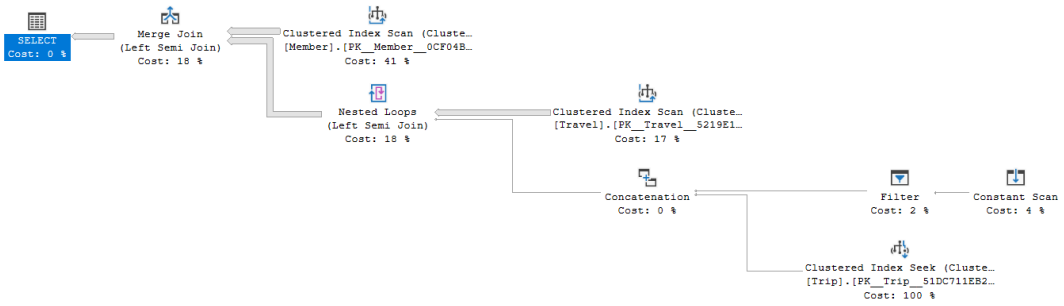
# Results of using different systems:

| System 1 | System 2 |
|---|---|
| CPU : intel i7 5700HQ 2.7GHZ<br>GPU: NVIDIA GeForce GTX 960M 2GB<br>RAM :16GB DDR3 1600mhz<br>OS: Windows 10 Enterprise<br>Storage: 500GB SSD Samsung 860 EVO | CPU: intel i5 7200U 2.5GHZ<br>GPU: Radeon R7 M440 2GB<br>RAM: 8GB DDR4<br>OS: Windows 10 Enterprise<br>Storage: 240GB SSD Kingston uv400 |

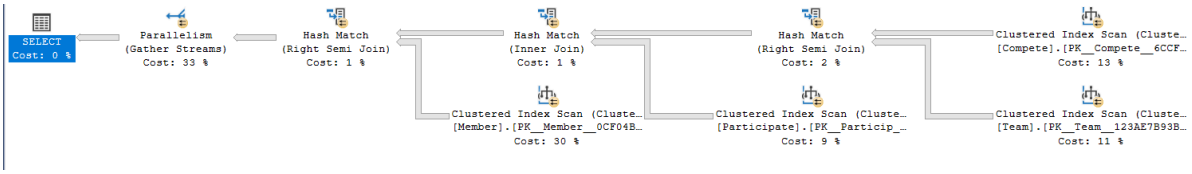
| Query\ system type | System 1 | System 2 |
|---|---|---|
| #1 | 00:00:05.065 | 00:00:11.244 |
| #2 | 00:00:08.266 | 00:00:16.486 |
| #3 | 00:00:07.522 | 00:00:15.806 |

# Execution plan before optimization

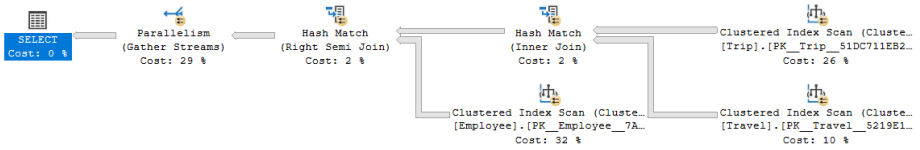
*Q1:*




Q2:



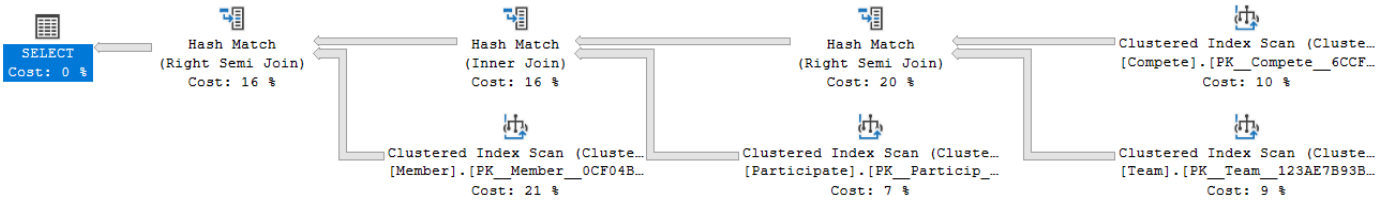

Q3:



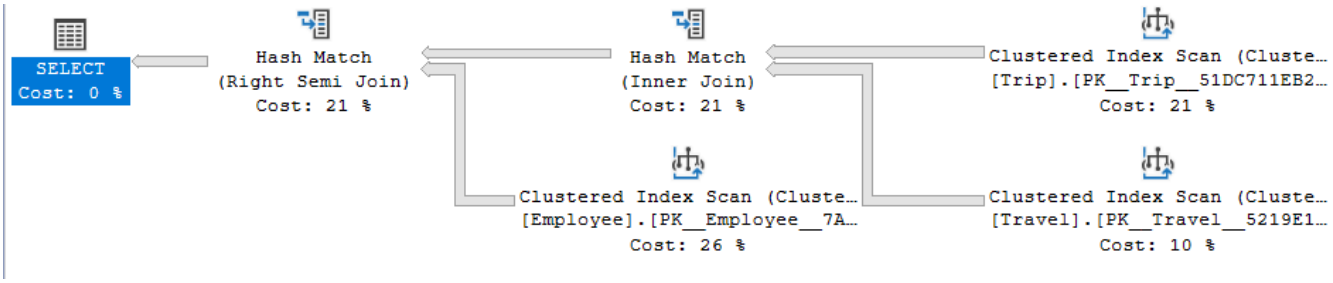

# With no Parallel

Q1: no difference

Q2: time: 00:00:07.666




Q3: time: 00:00:08.370



# Optimization Phase

Note: each Optimization applied individual first then all applied sequential

1. Schema Optimization:
   - Truncate some attributes to the logical maximum.
     - Ex: *City* attribute had limit 255 characters, so we limited it to 50.
   - Change Float to Integer for *Salary*, *Cost* and *Budget* attributes.

   The new schema is included in (New schema.txt)

   As the new schema is different we need to use different data so #of result row will be different, we change the condition so the # of row is close to the original one.

| Query | #1 | #2 | #3 |
|---|---|---|---|
| Time | 00:00:04.413 | 00:00:07.168 | 00:00:07.259 |
| #row | 618,091 | 947,033 | 872,045 |

2. Indexing Optimization: Based on the Execution Plan we added the following:
   a. For Q1, we added an index for Trip(Cost) and Travel (Feedback).
   b. For Q2, we added an index for Compete(Points, Standing) and Team(Coach ID).
   c. For Q3, we added an index for Travel(Feedback) and Employee(Salary, Supervisor).

| Query for insert index: |
|---|
| create index i1 on Trip(Cost) ;<br>create index i2 on Travel(Feedback) ;<br>create index i3 on Compete(Points) ;<br>create index i4 on Compete(Standing) ;<br>create index i5 on Team(CoachID) ;<br>create index i6 on Employee(Salary) ;<br>create index i7 on Employee(subervisor) ; |

| Query | #1 | #2 | #3 |
|---|---|---|---|
| Time | 00:00:04.610 | 00:00:07.728 | 00:00:07.225 |

3. Query Optimization:

We edited the queries such that they return only the important information.

## Results of the optimized queries in the old Schema without added index:

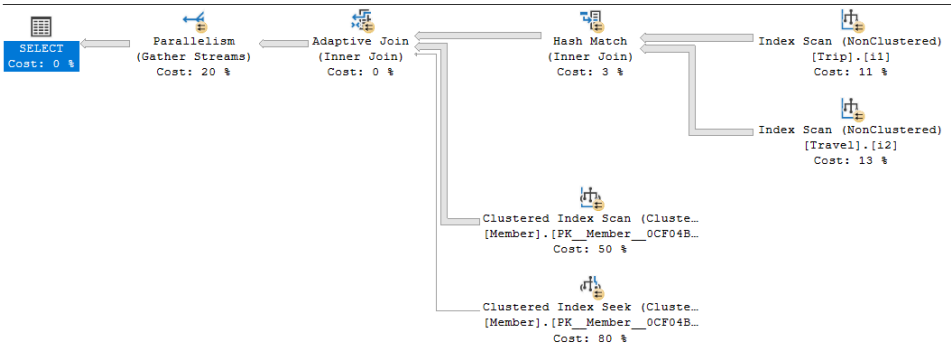
| SQL Query | | Time |
|---|---|---|
| #1 | `Select Name,MemberID,SSN,PhoneNumber from Member as M inner join Travel on M.MemberID = Travel.MEID inner join (select TripID , cost from Trip ) as T on Travel.TRID = T.TripID where T.cost>1500 or Travel.Feedback>0;` | 00:00:03.123 |
| #2 | `Select MemberID,SSN,PhoneNumber from Member as M inner join Participate on M.MemberID=Participate.MEID inner join Team on Participate.TEID=Team.TeamID inner join Compete on Compete.TEID=Team.TeamID where Compete.Points>500 and (Compete.Standing<8000 or Team.CoachID<100000);` | 00:00:03.913 |
| #3 | `Select EmployeeID,Name,Salary from Employee Inner join Trip on Employee.EmployeeID = Trip.SupervisorID inner join Travel on Trip.TripID=Travel.TRID where Travel.Feedback > 0 or Employee.Salary>500 ;` | 00:00:04.190 |

## Apply all optimization sequential

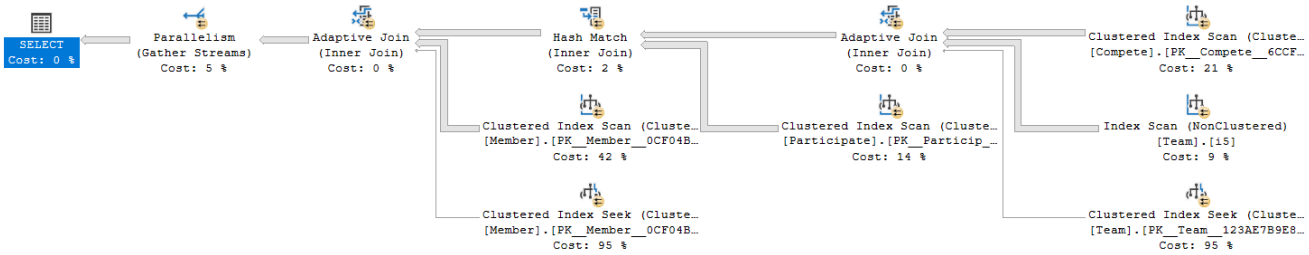
| | Without Optimization | Apply schema | | Apply index | | Apply Query | |
|---|---|---|---|---|---|---|---|
| #1 | 00:00:05.065 | 00:00:04.413 | 14.77% | 00:00:04.416 | 0% | 00:00:03.032 | 45.64% |
| #2 | 00:00:08.266 | 00:00:07.168 | 15.31% | 00:00:07.104 | 0.90% | 00:00:03.140 | 126.24% |
| #3 | 00:00:07.522 | 00:00:07.259 | 3.623% | 00:00:07.202 | 0.79% | 00:00:03.970 | 81.41% |

# Execution plan after optimization

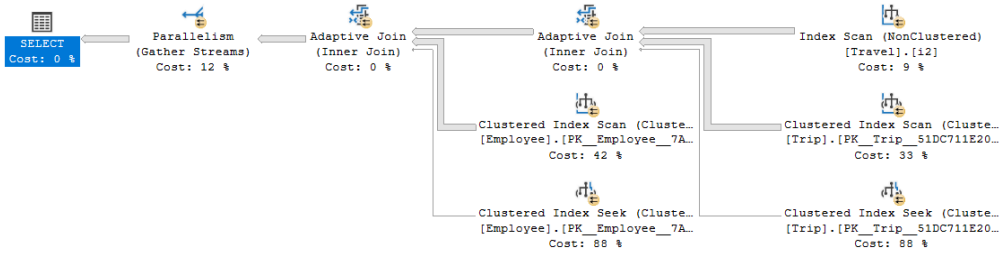
Q1:




*Q2:*




Q3:



# NOSQL

For NOSQL, we had two approaches:

- Merge all the Collections into one Collection so we can use a single query but we decided against this approach for two reasons:
    - The Database would be different from the original one (import from SQL server as had been mention in Lab 2)
    - Since we chose a Schema that has many Relations (many-to-many), the redundancy will be unacceptable as the size of data would dramatically increase.
- Use multiple collections and access each of them in a separate operation, we tried many things but only Python scripts return the exact expected values for the 3 queries.

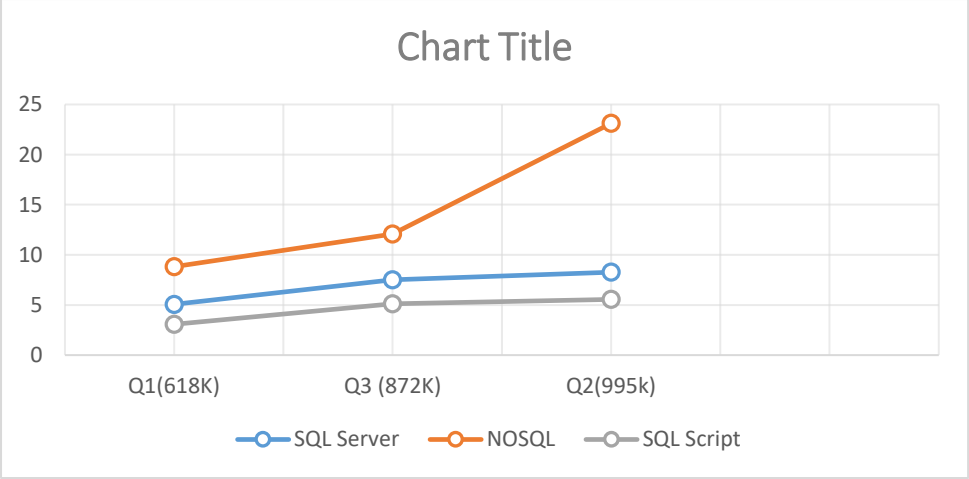
We decided to use a Python script for each query to access multiple collections. All Python scripts are included in separate files for each Query (Q1.py, Q2.py, Q3.py).

## Results

| Query | | NoSQL Script | SQL Script | SQL Server |
|---|---|---|---|---|
| #1 | Q1 | 8.8259 sec | 3.0722 sec | 5.065 sec |
| #2 | Q2 | 23.114 sec | 5.5559 sec | 8.266 sec |
| #3 | Q3 | 12.083 sec | 5.1080 sec | 7.522 sec |

As we notice NOSQL is the worst time as it was designed for schema with few relations

## Graph



# Final Results

All results comparison:

| | Before optimize SQL | After optimize SQL | NOSQL |
|---|---|---|---|
| #1 | 00:00:05.065 | 00:00:03.032 | 00:00:08.82 |
| #2 | 00:00:08.266 | 00:00:03.140 | 00:00:23.11 |
| #3 | 00:00:07.522 | 00:00:03.970 | 00:00:12.08 |

# Recommendations

## Hardware

We Recommended at least a quad Core CPU and 16GB RAM

## Database

For the Schema, we recommend the following:

- Reduce the relations and redundancy/null arguments as much as possible.
- Don’t add any complex types (ex: float) unless they’re absolutely necessary.
- Don’t give any attribute a non-logical size.

For indexing, it will depend on the type of queries needed and the nature of the system.

For queries, we recommend choosing only the exact information needed from a query instead of returning all attributes. The overhead of processing useless information is too great.