Executable and Parameterized SQL

ISM 6218

Due on October 8th

The Avengers Team

"We will avenge every problem on our way"

Aitemir Yeskenov (Team Lead)

Nagarjuna Kanneganti

Sai Suraj Argula

Vinay Kumar Reddy Baradi

Table of Contents

Business Process Supported

For this assignment, we decided to use the Employee database which contains two tables Employee Details and Employee Service where we created 5 columns to server the purpose of the assignment. We have used stored procedures, executing parameters and emphasized on performance of dynamic SQL methods. To solidify the previous statement, we tailored our methods to the business needs.

Requirements Described

1. Create a select statement
   - Must contain 1 join, 2 aggregators
   - Must have if/else
2. Create select with a parameter SPROC
3. Create select, Execute with EXEC (@SQL)
4. Create select as, Execute with sp_executeSQL @SQL
5. Create exec @SQL called within SPROC
6. Use execution plan and vertical panes to show differences in performance
   - Must use as least 3 parameters
   - Must support a user story and derived requirements
   - Must include a summary report and table of results comparison

User story

For this assignment, we have taken employee database to solve the business requirement specified.

User story: In our company, how many employees have average salary more than $10000

For this experiment, we have taken ‘count’ and ‘avg’ to find out average salary and years of service of employees who’s salary is more than $10000

```
declare @Salary money
set @Salary = 10000
if(@Salary < 12000)
select p.EmployeeID,count(q.YearsOfService) as YearsOfService,
avg(Salary) as AvgSalary from Other.EmployeeNew p join
Other.EmployeeService q on p.EmployeeID = q.EmployeeID
group by p.EmployeeID having sum(p.Salary) > @Salary
else
select p.EmployeeID,count(q.YearsOfService) as NoOfEmployees,
avg(Salary) as MinimumSalary from Other.EmployeeNew p join
Other.EmployeeService q on p.EmployeeID = q.EmployeeID
group by p.EmployeeID having sum(p.Salary) < @Salary
```

Dynamic SQL Methods

User story: In our company, how many employees have average salary more than $10000

Creating stored procedure with exec@sql:

```
create proc Other.DynamicSQLSP @Salary int
as
begin
declare @SqlCommand nvarchar(4000)
declare @ColumnList nvarchar(1000)
set @Salary = 10000
set @ColumnList  = 'p.EmployeeID,count(q.YearsOfService) as NoOfEmployees,
avg(Salary) AvgSalary'
set @SqlCommand = 'if(' + cast(@Salary as varchar) +' < 12000) ' +
'select ' + @ColumnList + ' from Other.EmployeeNew p join
Other.EmployeeService q on p.EmployeeID = q.EmployeeID
group by p.EmployeeID having sum(p.Salary) > '+ cast(@Salary as varchar) +
' else ' +
'select ' + @ColumnList + ' from Other.EmployeeNew p join
Other.EmployeeService q on p.EmployeeID = q.EmployeeID
group by p.EmployeeID having sum(p.Salary) < ' + cast(@Salary as varchar)
exec (@SqlCommand)
end

set statistics time on
exec Other.DynamicSQLSP @Salary = 10000
set statistics time off
```

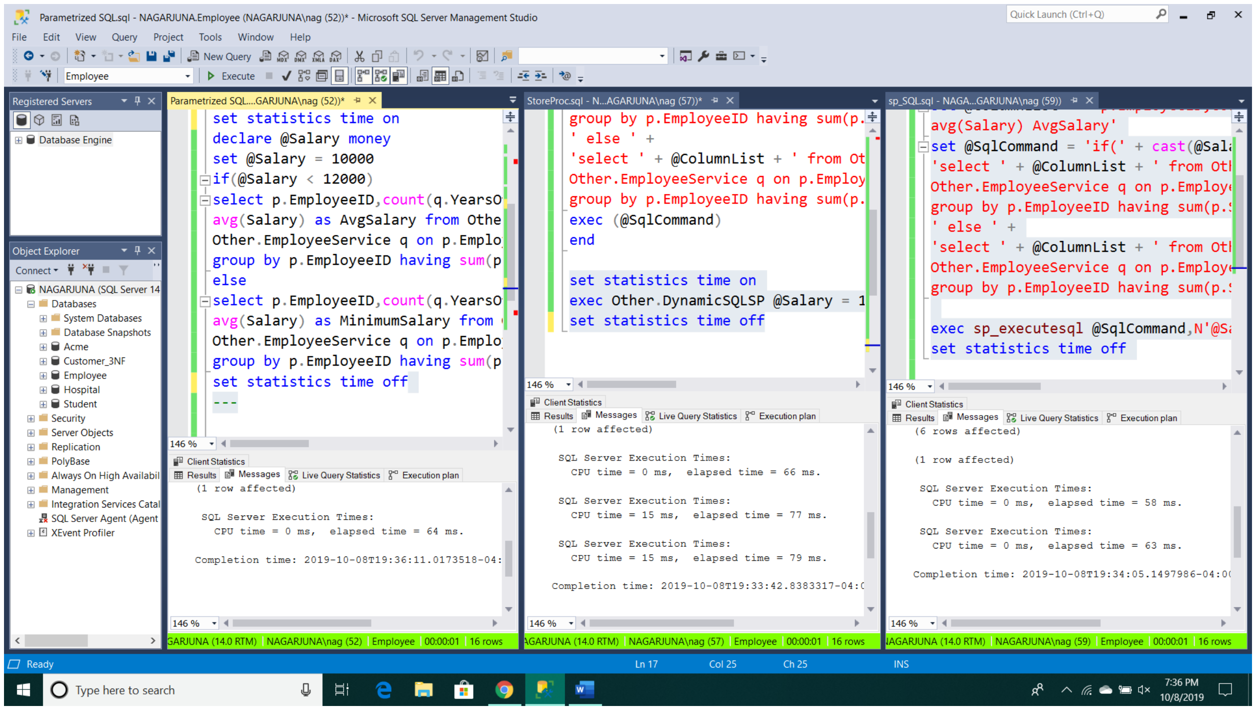
Creating with exec statement:

```
set statistics time on
declare @Salary int
declare @SqlCommand nvarchar(4000)
declare @ColumnList nvarchar(1000)
set @Salary = 10000
set @ColumnList  = 'p.EmployeeID,count(q.YearsOfService) as NoOfEmployees,
avg(Salary) AvgSalary'
set @SqlCommand = 'if(' + cast(@Salary as varchar) +' < 12000) ' +
'select ' + @ColumnList + ' from Other.EmployeeNew p join
Other.EmployeeService q on p.EmployeeID = q.EmployeeID
group by p.EmployeeID having sum(p.Salary) > '+ cast(@Salary as varchar) +
' else ' +
'select ' + @ColumnList + ' from Other.EmployeeNew p join
Other.EmployeeService q on p.EmployeeID = q.EmployeeID
group by p.EmployeeID having sum(p.Salary) < ' + cast(@Salary as varchar)

exec (@SqlCommand)

set statistics time off
```

Creating with system stored procedure

```
set statistics time on
declare @Salary int
declare @SqlCommand nvarchar(4000)
declare @ColumnList nvarchar(1000)
set @Salary = 10000
set @ColumnList  = 'p.EmployeeID,count(q.YearsOfService) as NoOfEmployees,
avg(Salary) AvgSalary'
set @SqlCommand = 'if(' + cast(@Salary as varchar) +' < 12000) ' +
'select ' + @ColumnList + ' from Other.EmployeeNew p join
Other.EmployeeService q on p.EmployeeID = q.EmployeeID
group by p.EmployeeID having sum(p.Salary) > '+ cast(@Salary as varchar) +
' else ' +
'select ' + @ColumnList + ' from Other.EmployeeNew p join
Other.EmployeeService q on p.EmployeeID = q.EmployeeID
group by p.EmployeeID having sum(p.Salary) < ' + cast(@Salary as varchar)

exec sp_executesql @SqlCommand,N'@Salary int', @Salary = @Salary
set statistics time off
```

Comparison between all the methods:



```
set statistics time on
declare @Salary money
set @Salary = 10000
if(@Salary < 12000)
select p.EmployeeID,count(q.YearsO
avg(Salary) as AvgSalary from Othe
Other.EmployeeService q on p.Emplo
group by p.EmployeeID having sum(p
else
select p.EmployeeID,count(q.YearsO
avg(Salary) as MinimumSalary from
Other.EmployeeService q on p.Emplo
group by p.EmployeeID having sum(p
set statistics time off
---
```

```
(1 row affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 64 ms.

Completion time: 2019-10-08T19:36:11.0173518-04:
```

GARJUNA (14.0 RTM) | NAGARJUNA\nag (52) | Employee | 00:00:01 | 16 rows

```
group by p.EmployeeID having sum(p.
' else ' +
'select ' + @ColumnList + ' from Ot
Other.EmployeeService q on p.Employ
group by p.EmployeeID having sum(p.
exec (@SqlCommand)
end

set statistics time on
exec Other.DynamicSQLSP @Salary = 1
set statistics time off
```

```
(1 row affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 66 ms.

 SQL Server Execution Times:
   CPU time = 15 ms,  elapsed time = 77 ms.

 SQL Server Execution Times:
   CPU time = 15 ms,  elapsed time = 79 ms.

Completion time: 2019-10-08T19:33:42.8383317-04:0
```

AGARJUNA (14.0 RTM) | NAGARJUNA\nag (57) | Employee | 00:00:01 | 16 rows

```
avg(Salary) AvgSalary'
set @SqlCommand = 'if(' + cast(@Sala
'select ' + @ColumnList + ' from Oth
Other.EmployeeService q on p.Employe
group by p.EmployeeID having sum(p.S
' else ' +
'select ' + @ColumnList + ' from Oth
Other.EmployeeService q on p.Employe
group by p.EmployeeID having sum(p.S

exec sp_executesql @SqlCommand,N'@Sa
set statistics time off
```

```
(6 rows affected)

(1 row affected)

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 58 ms.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 63 ms.

Completion time: 2019-10-08T19:34:05.1497986-04:0
```

NAGARJUNA (14.0 RTM) | NAGARJUNA\nag (59) | Employee | 00:00:01 | 16 rows

In the screenshot, we can see the execution time for sp_executesql method is 0ms and elapsed time is 63ms and it took less time to execute the whole query and also it prevents the SQL injection attack

Summary

From this experiment, we have depicted few observations which are shown as follows.

Dynamic SQL Method	CPU execution time	Elapsed time or time taken to execute the query
Stored procedure	0ms	64ms
Calling exec@SQL from stored procedure	15ms	77ms
exec@SQL	0ms	79ms
Sp_executesql	0ms	63ms

'ms' – milliseconds

As we can see from the comparisons table, CPU time for Stored procedure and Sp_executesql is 0ms and elapsed time are also almost equal. Dynamic SQL methods are created on the fly which also means these execution times depend on the cache memory. We have observed that Sp_executesql has better structure over exec@SQL because it will avoid the possibility of SQL injection and gives almost the same performance and sometimes better based on the cache memory.