(T41)討論效率，包括 QueryPlanCache 和 SqlInjection。討論 CrossApply 和 OuterApply。討論 Exec、sp_Executesql
CourseGUID: e48417fc-9db5-4e99-822c-706c5ccef6cc
==========
(T41)討論效率，包括 QueryPlanCache 和 SqlInjection。討論 CrossApply 和 OuterApply。討論 Exec、sp_Executesql
==========

==========

0. Summary

1.
INNER JOIN V.S. CROSS APPLY
```
--SELECT  t.TeamName ,
--       g.[Name] ,
--       g.Gender ,
--       g.GameScore
--FROM    dbo.Team t
--       CROSS APPLY fnGetGamerByTeamId(t.Id) g
--ORDER BY t.Id;
```
1.1.
```
--FROM    dbo.Team t
--       CROSS APPLY fnGetGamerByTeamId(t.Id) g
```
Pass each TeamId into fnGetGamerByTeamId()
This will return all the Gamers who has Team.
Thus, fnGetGamerByTeamId() CROSS APPLY Team
will return all the Gamers with their TeamName.

1.2.
--TableA INNER JOIN TableB
--ON TableA.ColumnAB = TableB.ColumnAB
INNER JOIN is for join 2 tables.
1.3.
--fnGetGamerByTeamId CROSS APPLY TableA
This will cause ERROR,
fnGetGamerByTeamId must be the right hand side of CROSS APPLY
1.4.
--TableA CROSS APPLY fnGetGamerByTeamId
fnGetGamerByTeamId must be the right hand side of CROSS APPLY
CROSS APPLY is similar to INNER JOIN
which retrieves only the matching rows.
However,
INNER JOIN is for join 2 tables.
CROSS APPLY joins 1 table(Left Hand Side)
and fnGetGamerByTeamId(Right Hand Side).

2.
LEFT JOIN V.S. OUTER APPLY
--SELECT t.TeamName ,
-- g.[Name] ,
-- g.Gender ,
-- g.GameScore
--FROM dbo.Team t
-- OUTER APPLY fnGetGamerByTeamId(t.Id) g
--ORDER BY t.Id;
2.1.
--FROM dbo.Team t
-- OUTER APPLY fnGetGamerByTeamId(t.Id) g
Pass each TeamId into fnGetGamerByTeamId()
This will return all the Gamers who has Team.
Team is in Left Hand Side of OUTER APPLY.
Thus, the query will return
all the Gamers with their TeamName
plus all Team name which has no Gamers.
2.2.
--TableA LEFT JOIN TableB
--ON TableA.ColumnAB = TableB.ColumnAB
LEFT JOIN is for join 2 tables.
2.3.
--fnGetGamerByTeamId OUTER APPLY TableA
This will cause ERROR,
fnGetGamerByTeamId must be the right hand side of OUTER APPLY
2.4.
--TableA OUTER APPLY fnGetGamerByTeamId
fnGetGamerByTeamId must be the right hand side of OUTER APPLY
OUTER APPLY is similar to LEFT JOIN
which retrieves only the matching rows + Left Hand Side un-matching rows
However,
LEFT JOIN is for join 2 tables.
OUTER APPLY is join 1 table(Left Hand Side)
and fnGetGamerByTeamId(Right Hand Side).

3.
--SELECT cp.usecounts ,
-- cp.cacheobjtype ,
-- cp.objtype ,
-- st.text ,
-- qp.query_plan
--FROM sys.dm_exec_cached_plans AS cp
-- CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
-- CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
--ORDER BY cp.usecounts DESC;
3.1.

When a queryA was run at the first time,
SQL server will generate a query plan in query plan cache.
When queryA was run again,
the same query plan will be re-used.
Reusing a query plan can increase the performance.
The more often the plan is reused the longer it stays in the plan cache.
To check the query plan cache, we need
--sys.dm_exec_cached_plans , sys.dm_exec_sql_text , sys.dm_exec_query_plan
---
3.2.
--FROM   sys.dm_exec_cached_plans AS cp
--   CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
--   CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
Reference:
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-cached-plans-transact-sql
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-sql-text-transact-sql
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-query-plan-transact-sql
--sys.dm_exec_cached_plans
is a dynamic management view which returns query plans from cache.
CROSS APPLY will pass each plan_handle value into 2 table value function,
sys.dm_exec_sql_text(plan_handle) and sys.dm_exec_query_plan(plan_handle).
--sys.dm_exec_sql_text(plan_handle)
is a table value function
which returns the text of the SQL batch that is identified by the specified sql_handle.
--sys.dm_exec_query_plan(plan_handle)
is a table value function
which returns the Showplan in XML format for the batch specified by the plan handle.

3.3.
--sys.dm_exec_cached_plans
is a dynamic management view which returns query plans from cache.
and it has the following columns we care.

3.3.1.
--usecounts
UseCounts tell us how many times the query plan in the query plan cache is reused.
3.3.2.
--CacheObjType
"CacheObjType" tell us the cached object type,
in this case, "CacheObjType" is a "complied plan"
which means the query plan in the query plan cache is a complied plan.
3.3.3.
--objtype
3.3.3.1.
"objtype" tell us the object type for this "complied plan"

3.3.2.2.1.
When "objtype" is "Adhoc" which means this is an ad hoc query
which is short lived and is created at runtime.
Reference:
https://www.techopedia.com/definition/30581/ad-hoc-query-sql-programming
ad hoc query is a loosely typed query which cannot be predetermined
Each time the command is executed,
the result is different.
-->
When a queryA was run at the first time,
SQL server will generate a query plan in query plan cache.
This query plan is normally a "complied plan" as its "CacheObjType",
and "Adhoc" as its "objtype".
When queryA was run again,
the same query plan will be re-used.
that menas "usecounts" value will be increased by 1 each time when the query plan is re-used.
Reusing a query plan can increase the performance.
The more often the plan is reused the longer it stays in the plan cache.

3.3.2.2.2.
When "objtype" is "Prepared" which means
this object is automatically created by SQL server in the background.
-->
SQL Server can detect parameter values
and automatically generate parameterised queries
in order to reuse its query plan,
Even if you don't explicitly declare them.
This kind of automatically generated parameterised queries
are "Prepared" queries which are created by SQL server in the background.

3.3.2.2.3.
When "objtype" is "Prague" which means
this "complied plan" is a query plan for stored procedure.

3.4.
--CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
is a table value function
which returns the text of the SQL batch that is identified by the specified sql_handle.
3.4.1.
--text
text column of this table value function sys.dm_exec_sql_text(plan_handle)
returns the text of the SQL batch.

3.5.
--CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
is a table value function
which returns the Showplan in XML format for the batch specified by the plan handle.
3.5.1.
--query_plan
query_plan column of this table value function sys.dm_exec_query_plan(plan_handle)
returns the Query execution plan in XML format
When you click the Query Plan Xml
You will see the query plan in graphical format.

4.
--DBCC FREEPROCCACHE;
FreeProcCache clear the query plans in the query plan cache.
Output as following
--DBCC execution completed. If DBCC printed error messages, contact your system administrator.

5.
Exec V.S. sp_Executesql
5.1.
using Exec() with QUOTENAME() function can prevent sql injection,
but still NOT recommend to use Exec() to run dynamic sql query.
5.2.
in order to reuse its query plan,
SQL Server can detect parameter values
and sometimes automatically generate parameterised queries,
Even if you don't explicitly declare them.
This kind of automatically generated parameterised queries
are "Prepared" queries which are created by SQL server in the background.
Thus, Cached query plan reusability is also not an issue while using Exec().
but still NOT recommend to use Exec() to run dynamic sql query.
5.3.
Using sp_executesql with parameters can always explicitly create parameterise queries.
We should NOT relying on sql server auto-parameterisation feature or
QUOTENAME() function to prevent SQL injection and increase reusability.

==========

1. CrossApply_OuterApply

```
--=============================================================================================
--T041_01_CrossApply_OuterApply
--=============================================================================================
```

1.1. Create Sample Data

```
--=============================================================================================
--T041_01_01
--Create Sample Data
--If Table exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Team' ) )
    BEGIN
        TRUNCATE TABLE dbo.Team;
        DROP TABLE Team;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Team
(
  Id INT PRIMARY KEY ,
  TeamName NVARCHAR(50)
);
GO -- Run the previous command and begins new batch
INSERT  INTO Team
VALUES  ( 1, 'Team01' );
INSERT  INTO Team
VALUES  ( 2, 'Team02' );
INSERT  INTO Team
VALUES  ( 3, 'Team03' );
INSERT  INTO Team
VALUES  ( 4, 'Team04' );
INSERT  INTO Team
VALUES  ( 5, 'Team05' );
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT PRIMARY KEY ,
  [Name] NVARCHAR(50) ,
  Gender NVARCHAR(10) ,
  GameScore MONEY ,
  TeamId INT FOREIGN KEY REFERENCES dbo.Team ( Id )
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
VALUES  ( 1, 'Name01', 'Male', 41000, 1 );
INSERT  INTO Gamer
VALUES  ( 2, 'Name02', 'Female', 75000, 3 );
```

```
INSERT  INTO Gamer
VALUES  ( 3, 'Name03', 'Female', 65000, 2 );
INSERT  INTO Gamer
VALUES  ( 4, 'Name04', 'Female', 44000, 3 );
INSERT  INTO Gamer
VALUES  ( 5, 'Name05', 'Male', 38000, 1 );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    dbo.Gamer;
SELECT  *
FROM    dbo.Team;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	TeamId
1	1	Name01	Male	41000.00	1
2	2	Name02	Female	75000.00	3
3	3	Name03	Female	65000.00	2
4	4	Name04	Female	44000.00	3
5	5	Name05	Male	38000.00	1

	Id	TeamName
1	1	Team01
2	2	Team02
3	3	Team03
4	4	Team04
5	5	Team05

1.2. Team INNER JOIN Gamer

```
--=============================================================================================
--T041_01_02
--Team INNER JOIN Gamer
SELECT  t.TeamName ,
        g.Name ,
        g.Gender ,
        g.GameScore
FROM    Team t
        INNER JOIN dbo.Gamer g ON g.TeamId = t.Id
ORDER BY t.Id;
GO -- Run the previous command and begins new batch
```

	TeamName	Name	Gender	GameScore
1	Team01	Name01	Male	41000.00
2	Team01	Name05	Male	38000.00
3	Team02	Name03	Female	65000.00
4	Team03	Name04	Female	44000.00
5	Team03	Name02	Female	75000.00

1.3. Team LEFT JOIN Gamer

```
--=============================================================================================
--T041_01_03
--Team LEFT JOIN Gamer
SELECT  t.TeamName ,
        g.Name ,
        g.Gender ,
        g.GameScore
FROM    Team t
        LEFT JOIN dbo.Gamer g ON g.TeamId = t.Id
ORDER BY t.Id;
GO -- Run the previous command and begins new batch
```

	TeamName	Name	Gender	GameScore
1	Team01	Name01	Male	41000.00
2	Team01	Name05	Male	38000.00
3	Team02	Name03	Female	65000.00
4	Team03	Name02	Female	75000.00
5	Team03	Name04	Female	44000.00
6	Team04	NULL	NULL	NULL
7	Team05	NULL	NULL	NULL

1.4. Create Table Value Function, fnGetGamerByTeamId

```
--=============================================================================================
--T041_01_04
--Create Table Value Function, fnGetGamerByTeamId
--If function exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.ROUTINES
              WHERE     ROUTINE_TYPE = 'FUNCTION'
                        AND LEFT(ROUTINE_NAME, 2) NOT IN ( '@@' )
                        AND SPECIFIC_NAME = 'fnGetGamerByTeamId' ) )
    BEGIN
        DROP FUNCTION fnGetGamerByTeamId;
    END;
GO -- Run the previous command and begins new batch
CREATE FUNCTION fnGetGamerByTeamId ( @TeamId int )
RETURNS TABLE
AS
RETURN
    ( SELECT    g.Id ,
                g.[Name] ,
                g.Gender ,
                g.GameScore ,
                g.TeamId
      FROM      dbo.Gamer g
      WHERE     TeamId = @TeamId
    );
GO -- Run the previous command and begins new batch
SELECT  *
FROM    fnGetGamerByTeamId(1);
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	TeamId
1	1	Name01	Male	41000.00	1
2	5	Name05	Male	38000.00	1

1.5. Table Value Function can not use INNER JOIN

```
--=============================================================================================
--T041_01_05
--Table Value Function can not use INNER JOIN
SELECT  t.TeamName ,
        g.[Name] ,
        g.Gender ,
        g.GameScore
FROM    Team t
        INNER JOIN fnGetGamerByTeamId(t.Id) g ON t.Id = g.TeamId;
GO -- Run the previous command and begins new batch
/*
Error Message
--Msg 4104, Level 16, State 1, Line 133
--The multi-part identifier "d.Id" could not be bound.
Table Value Function can not use INNER JOIN
*/
```

1.6. Table Value Function can not use INNER JOIN

```
--=============================================================================================
--T041_01_06
--Table Value Function must use CROSS APPLY
SELECT  t.TeamName ,
        g.[Name] ,
        g.Gender ,
        g.GameScore
FROM    dbo.Team t
        CROSS APPLY fnGetGamerByTeamId(t.Id) g
ORDER BY t.Id;
GO -- Run the previous command and begins new batch
/*
1.
--FROM    dbo.Team t
--        CROSS APPLY fnGetGamerByTeamId(t.Id) g
Pass each TeamId into fnGetGamerByTeamId()
This will return all the Gamers who has Team.
Thus, fnGetGamerByTeamId() CROSS APPLY Team
will return all the Gamers with their TeamName.
1.1.
--TableA INNER JOIN TableB
--ON TableA.ColumnAB = TableB.ColumnAB
INNER JOIN is for join 2 tables.
1.2.
--fnGetGamerByTeamId CROSS APPLY TableA
This will cause ERROR,
fnGetGamerByTeamId must be the right hand side of CROSS APPLY
1.3.
--TableA CROSS APPLY fnGetGamerByTeamId
fnGetGamerByTeamId must be the right hand side of CROSS APPLY
CROSS APPLY is similar to INNER JOIN
which retrieves only the matching rows.
However,
INNER JOIN is for join 2 tables.
CROSS APPLY joins 1 table(Left Hand Side)
and fnGetGamerByTeamId(Right Hand Side).
*/
```

	TeamName	Name	Gender	GameScore
1	Team01	Name01	Male	41000.00
2	Team01	Name05	Male	38000.00
3	Team02	Name03	Female	65000.00
4	Team03	Name04	Female	44000.00
5	Team03	Name02	Female	75000.00

1.7. ERROR: fnGetGamerByTeamId must be the right hand side of CROSS APPLY

```
--=============================================================================================
--T041_01_07
--ERROR: fnGetGamerByTeamId must be the right hand side of CROSS APPLY
SELECT  t.TeamName ,
        g.[Name] ,
        g.Gender ,
        g.GameScore
FROM    fnGetGamerByTeamId(t.Id) g
       CROSS APPLY dbo.Team t
ORDER BY t.Id;
GO -- Run the previous command and begins new batch
/*
1.
--fnGetGamerByTeamId CROSS APPLY TableA
This will cause ERROR,
fnGetGamerByTeamId must be the right hand side of CROSS APPLY
2.
Output
--Msg 4104, Level 16, State 1, Line 278
--The multi-part identifier "t.Id" could not be bound.
*/
```

1.8. OUTER APPLY

```
--=============================================================================================
--T041_01_08
--OUTER APPLY
SELECT  t.TeamName ,
        g.[Name] ,
        g.Gender ,
        g.GameScore
FROM    dbo.Team t
       OUTER APPLY fnGetGamerByTeamId(t.Id) g
ORDER BY t.Id;
GO -- Run the previous command and begins new batch
/*
1.
--FROM    dbo.Team t
--        OUTER APPLY fnGetGamerByTeamId(t.Id) g
Pass each TeamId into fnGetGamerByTeamId()
This will return all the Gamers who has Team.
Team is in Left Hand Side of OUTER APPLY.
Thus, the query will return
all the Gamers with their TeamName
plus all Team name which has no Gamers.
1.1.
--TableA LEFT JOIN TableB
--ON TableA.ColumnAB = TableB.ColumnAB
```

```
LEFT JOIN is for join 2 tables.
1.2.
--fnGetGamerByTeamId OUTER APPLY TableA
This will cause ERROR,
fnGetGamerByTeamId must be the right hand side of OUTER APPLY
1.3.
--TableA OUTER APPLY fnGetGamerByTeamId
fnGetGamerByTeamId must be the right hand side of OUTER APPLY
OUTER APPLY is similar to LEFT JOIN
which retrieves only the matching rows + Left Hand Side un-matching rows
However,
LEFT JOIN is for join 2 tables.
OUTER APPLY is join 1 table(Left Hand Side)
and fnGetGamerByTeamId(Right Hand Side).
*/
```

	TeamName	Name	Gender	GameScore
1	Team01	Name01	Male	41000.00
2	Team01	Name05	Male	38000.00
3	Team02	Name03	Female	65000.00
4	Team03	Name02	Female	75000.00
5	Team03	Name04	Female	44000.00
6	Team04	NULL	NULL	NULL
7	Team05	NULL	NULL	NULL

2. QueryPlanCache

```
--=============================================================================================
--T041_02_QueryPlanCache
--=============================================================================================
```

2.1. sys.dm_exec_cached_plans

```
--=============================================================================================
--T041_02_01
--sys.dm_exec_cached_plans
SELECT  *
FROM    sys.dm_exec_cached_plans;
GO -- Run the previous command and begins new batch
/*
1.
sys.dm_exec_cached_plans
Reference:
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-cached-plans-transact-sql
dynamic management view returns a row for each query plan that is cached by SQL Server for faster query execution.
*/
```

2.2. sys.dm_exec_cached_plans , sys.dm_exec_sql_text

```
--=============================================================================================
--T041_02_02
--sys.dm_exec_cached_plans , sys.dm_exec_sql_text
SELECT  *
FROM    sys.dm_exec_cached_plans AS cp
        CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
ORDER BY cp.usecounts DESC;
```

```
GO -- Run the previous command and begins new batch
/*
--CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
Reference:
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-sql-text-transact-sql
The table value function returns the text of the SQL batch that is identified by the specified
sql_handle.
This table-valued function replaces the system function fn_get_sql.
*/
```

2.3. sys.dm_exec_cached_plans , sys.dm_exec_sql_text , sys.dm_exec_query_plan

```
--============================================================================================
--T041_02_03
--sys.dm_exec_cached_plans , sys.dm_exec_sql_text , sys.dm_exec_query_plan
SELECT  *
FROM    sys.dm_exec_cached_plans AS cp
        CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
        CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
ORDER BY cp.usecounts DESC;
GO -- Run the previous command and begins new batch
/*
--CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
Reference:
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-query-plan-transact-sql
The table value function returns the Showplan in XML format for the batch specified by the plan handle.
The plan specified by the plan handle can either be cached or currently executing.
*/
```

2.4. See the query plan in the query plan cache.

```
--============================================================================================
--T041_02_04
--See the query plan in the query plan cache.
--sys.dm_exec_cached_plans , sys.dm_exec_sql_text , sys.dm_exec_query_plan
SELECT  cp.usecounts ,
        cp.cacheobjtype ,
        cp.objtype ,
        st.text ,
        qp.query_plan
FROM    sys.dm_exec_cached_plans AS cp
        CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
        CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
ORDER BY cp.usecounts DESC;
GO -- Run the previous command and begins new batch
```

	usecounts	cacheobjtype	objtype	text	query_plan
1	1	Compiled Plan	Adhoc	--==...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2	1	Compiled Plan	Adhoc	declare @BatchID uniqueidentifier ...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
3	1	Compiled Plan	Adhoc	declare @BatchID uniqueide...	<ShowPlanXML xmlns="http://schemas.microsoft.com...

```
/*
1.
--SELECT  cp.usecounts ,
--        cp.cacheobjtype ,
--        cp.objtype ,
--        st.text ,
--        qp.query_plan
--FROM    sys.dm_exec_cached_plans AS cp
```

```
--        CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
--        CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
--ORDER BY cp.usecounts DESC;
-------------------------------------------------
1.1.
When a queryA was run at the first time,
SQL server will generate a query plan in query plan cache.
When queryA was run again,
the same query plan will be re-used.
Reusing a query plan can increase the performance.
The more often the plan is reused the longer it stays in the plan cache.
To check the query plan cache, we need
--sys.dm_exec_cached_plans , sys.dm_exec_sql_text , sys.dm_exec_query_plan
-------------------------------------------------
1.2.
--FROM    sys.dm_exec_cached_plans AS cp
--        CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
--        CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
Reference:
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-cached-plans-transact-sql
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-sql-text-transact-sql
https://docs.microsoft.com/en-us/sql/relational-databases/system-dynamic-management-views/sys-dm-exec-query-plan-transact-sql
--sys.dm_exec_cached_plans
is a dynamic management view which returns query plans from cache.
CROSS APPLY will pass each plan_handle value into 2 table value function,
sys.dm_exec_sql_text(plan_handle) and sys.dm_exec_query_plan(plan_handle).
--sys.dm_exec_sql_text(plan_handle)
is a table value function
which returns the text of the SQL batch that is identified by the specified sql_handle.
--sys.dm_exec_query_plan(plan_handle)
is a table value function
which returns the Showplan in XML format for the batch specified by the plan handle.
-------------------------------------------------
1.3.
--sys.dm_exec_cached_plans
is a dynamic management view which returns query plans from cache.
and it has the following columns we care.
------------------------------
1.3.1.
--usecounts
UseCounts tell us how many times the query plan in the query plan cache is reused.
1.3.2.
--CacheObjType
"CacheObjType" tell us the cached object type,
in this case, "CacheObjType" is a "complied plan"
which means the query plan in the query plan cache is a complied plan.
1.3.3.
--objtype
1.3.3.1.
"objtype" tell us the object type for this "complied plan"
---------------
1.3.2.2.1.
When "objtype" is "Adhoc" which means this is an ad hoc query
which is short lived and is created at runtime.
Reference:
https://www.techopedia.com/definition/30581/ad-hoc-query-sql-programming
ad hoc query is a loosely typed query which cannot be predetermined
Each time the command is executed,
the result is different.
-->
When a queryA was run at the first time,
SQL server will generate a query plan in query plan cache.
This query plan is normally a "complied plan" as its "CacheObjType",
and "Adhoc" as its "objtype".
```

```
When queryA was run again,
the same query plan will be re-used.
that menas "usecounts" value will be increased by 1 each time when the query plan is re-used.
Reusing a query plan can increase the performance.
The more often the plan is reused the longer it stays in the plan cache.
---------------
1.3.2.2.2.
When "objtype" is "Prepared" which means
this object is automatically created by SQL server in the background.
-->
SQL Server can detect parameter values
and automatically generate parameterised queries
in order to reuse its query plan,
Even if you don't explicitly declare them.
This kind of automatically generated parameterised queries
are "Prepared" queries which are created by SQL server in the background.
---------------
1.3.2.2.3.
When "objtype" is "Prague" which means
this "complied plan" is a query plan for stored procedure.
-------------------------------------------------
1.4.
--CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
is a table value function
which returns the text of the SQL batch that is identified by the specified sql_handle.
1.4.1.
--text
text column of this table value function sys.dm_exec_sql_text(plan_handle)
returns the text of the SQL batch.
-------------------------------------------------
1.5.
--CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
is a table value function
which returns the Showplan in XML format for the batch specified by the plan handle.
1.5.1.
--query_plan
query_plan column of this table value function sys.dm_exec_query_plan(plan_handle)
returns the Query execution plan in XML format
When you click the Query Plan Xml
You will see the query plan in graphical format.
*/
```

2.5. FreeProcCache clear the query plans in the query plan cache.

```
--============================================================================================
--T041_02_05
--FreeProcCache clear the query plans in the query plan cache.
DBCC FREEPROCCACHE;
GO -- Run the previous command and begins new batch
/*
Output as following
--DBCC execution completed. If DBCC printed error messages, contact your system administrator.
*/
```

2.6. check usecounts, reuse query plans can increase the performance..

```
--============================================================================================
--T041_02_06
--check usecounts, reuse query plans can increase the performance.
--T041_02_06_01
--FreeProcCache clear the query plans in the query plan cache.
DBCC FREEPROCCACHE;
```

```
GO -- Run the previous command and begins new batch
```

Messages

```
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
```

```
--T041_02_06_02
--QueryA
SELECT  *
FROM    dbo.Gamer
WHERE   Name = 'Name03';
--WHERE   Name = 'Name04';
GO -- Run the previous command and begins new batch
--T041_02_06_03
--QueryB
--See the query plan in the query plan cache.
SELECT  cp.usecounts ,
        cp.cacheobjtype ,
        cp.objtype ,
        st.text ,
        qp.query_plan
FROM    sys.dm_exec_cached_plans AS cp
        CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
        CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
ORDER BY cp.usecounts DESC;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	TeamId
1	3	Name03	Female	65000.00	2

	usecounts	cacheobjtype	objtype	text	query_plan
1	1	Compiled Plan	Adhoc	--Ch143_06_03 --QueryB --See the query plan in the q...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2	1	Compiled Plan	Adhoc	--Ch143_06_02 --QueryA SELECT * FROM dbo.Ga...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
3	1	Compiled Plan	Prepared	(@1 varchar(8000))SELECT * FROM [dbo].[Gamer] WHE...	<ShowPlanXML xmlns="http://schemas.microsoft.com...

```
/*
1.
Highlight QueryA and QueryB.
When execute both queries at first time.
We can see the usecounts of query plan of QueryA is 1.
That means the query plan of QueryA is generated
when QueryA was run at first time.
2.
Highlight QueryA and QueryB.
When execute both queries at second time.
We can see the usecounts of query plan of QueryA is 2.
That means the query plan of QueryA is re-used.
when QueryA was run at first time.
3.
Change
--WHERE   Name = 'Name03';
to
--WHERE   Name = 'Name04';
Highlight QueryA and QueryB.
When execute both queries at first time.
We can see the usecounts of query plan of QueryA is 1.
We only changed the parameter value in where clause.
It actually creates another query plan.
Even if we add extra space, it will create another query plan.
This is bad for performance.
we need to find a way to reuse the query plan in order to increase the performance.
*/
```

2.7. check usecounts of automatically generated parameterised queries

```
--=============================================================================================
--T041_02_07
--check usecounts of automatically generated parameterised queries,
--reuse query plans can increase the performance.
--T041_02_07_01
--FreeProcCache clear the query plans in the query plan cache.
DBCC FREEPROCCACHE;
GO -- Run the previous command and begins new batch
```

Messages

```
DBCC execution completed. If DBCC printed error messages, contact your system administrator.
```

```
--T041_02_07_02
--QueryA
DECLARE @FN NVARCHAR(50)
SET @FN= 'Name03';
--SET @FN= 'Name04';
--SET @FN= 'Name05';
DECLARE @sql NVARCHAR(MAX)
= 'SELECT * FROM Gamer WHERE Name = ''' + @FN + '''';
EXEC(@sql);
/*
SQL Server can detect parameter values
and automatically generate parameterised queries
in order to reuse its query plan,
Even if you don't explicitly declare them.
This kind of automatically generated parameterised queries
are "Prepared" queries which are created by SQL server in the background.
*/
--T041_02_07_03
--See the query plan in the query plan cache.
SELECT  cp.usecounts ,
        cp.cacheobjtype ,
        cp.objtype ,
        st.text ,
        qp.query_plan
FROM    sys.dm_exec_cached_plans AS cp
        CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
        CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
ORDER BY cp.usecounts DESC;
GO -- Run the previous command and begins new batch
```

	Id	Name	Gender	GameScore	TeamId
1	3	Name03	Female	65000.00	2

	usecounts	cacheobjtype	objtype	text	query_plan
1	1	Compiled Plan	Adhoc	SELECT * FROM Gamer WHERE Name = 'Name03'	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2	1	Compiled Plan	Prepared	(@1 varchar(8000))SELECT * FROM [Gamer] WHERE [Na...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
3	1	Compiled Plan	Adhoc	--Ch143_07_02 --QueryA DECLARE @FN NVARCHA...	<ShowPlanXML xmlns="http://schemas.microsoft.com...

```
-------------------------------------------------
```

	Id	Name	Gender	GameScore	TeamId
1	4	Name04	Female	44000.00	3

	usecounts	cacheobjtype	objtype	text	query_plan
1	2	Compiled Plan	Adhoc	declare @BatchID uniqueidentifier ...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2	2	Compiled Plan	Adhoc	declare @BatchID uniqueide...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
3	1	Compiled Plan	Adhoc	SELECT * FROM Gamer WHERE Name = 'Name04'	<ShowPlanXML xmlns="http://schemas.microsoft.com...
4	1	Compiled Plan	Prepared	(@1 varchar(8000))SELECT * FROM [Gamer] WHERE [Na...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
5	1	Compiled Plan	Adhoc	--Ch143_07_02 --QueryA DECLARE @FN NVARCHAR(...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
6	1	Compiled Plan	Adhoc	SELECT dtb.name AS [Name], dtb.database_id AS [ID], C...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
7	1	Compiled Plan	Prepared	(@_msparam_0 nvarchar(4000))SELECT dtb.collation_nam...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
8	1	Compiled Plan	Adhoc	SELECT [session_name], [definition], [host_address], [own...	<ShowPlanXML xmlns="http://schemas.microsoft.com...

```
/*
1.
Highlight QueryA and QueryB.
When execute both queries at first time.
We can see the usecounts of query plan of QueryA is 1.
That means the query plan of QueryA is generated
when QueryA was run at first time.
2.
Highlight QueryA and QueryB.
When execute both queries at second time.
We can see the usecounts of query plan of QueryA is 2.
That means the query plan of QueryA is re-used.
when QueryA was run at first time.
3.
Change
SET @FN= 'Name03';
to
SET @FN= 'Name04';
Highlight QueryA and QueryB.
When execute both queries at first time.
We can see the usecounts of query plan of QueryA is 2.
Because the SQL server automatically generate parameterised queries in the background.
However, SQL server is not always
automatically generating parameterised queries.
In order to always create parameterised queries,
we need to use sp_executesql.
*/
```

2.8. Clean up

```
--============================================================================================
--T041_02_08
--Clean up
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Team' ) )
    BEGIN
        TRUNCATE TABLE dbo.Team;
        DROP TABLE Team;
    END;
GO -- Run the previous command and begins new batch
--If function exists then DROP it
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.ROUTINES
              WHERE     ROUTINE_TYPE = 'FUNCTION'
                        AND LEFT(ROUTINE_NAME, 2) NOT IN ( '@@' )
                        AND SPECIFIC_NAME = 'fnGetGamerAByTeamId' ) )
    BEGIN
        DROP FUNCTION fnGetGamerAByTeamId;
    END;
GO -- Run the previous command and begins new batch
```

3. Exec V.S. sp_Executesql

```
--=============================================================================================
--T041_03_Exec V.S. sp_Executesql
--Revise DynamicSQL_SearchWebPage
--=============================================================================================
/*
Exec V.S. sp_Executesql
1.
using Exec() with QUOTENAME() function can prevent sql injection,
but still NOT recommend to use Exec() to run dynamic sql query.
2.
in order to reuse its query plan,
SQL Server can detect parameter values
and sometimes automatically generate parameterised queries,
Even if you don't explicitly declare them.
This kind of automatically generated parameterised queries
are "Prepared" queries which are created by SQL server in the background.
Thus, Cached query plan reusability is also not an issue while using Exec().
but still NOT recommend to use Exec() to run dynamic sql query.
3.
Using sp_executesql with parameters can always explicitly create parameterise queries.
We should NOT relying on sql server auto-parameterisation feature or
QUOTENAME() function to prevent SQL injection and increase reusability.
*/
```

3.1. Create Sample Data

```
--=============================================================================================
--T041_03_01
--Create Sample Data
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Gamer
(
  Id INT IDENTITY(1, 1)
        PRIMARY KEY ,
  FirstName NVARCHAR(50) ,
  LastName NVARCHAR(50) ,
  Gender NVARCHAR(50) ,
  GameScore INT
);
GO -- Run the previous command and begins new batch
INSERT  INTO Gamer
VALUES  ( 'AFirst01', 'XLast01', 'Female', 3500 );
INSERT  INTO Gamer
VALUES  ( 'AFirst02', 'YLast02', 'Female', 4000 );
INSERT  INTO Gamer
VALUES  ( 'BFirst03', 'YLast03', 'Male', 4600 );
INSERT  INTO Gamer
```

```
VALUES  ( 'BFirst04', 'YLast04', 'Male', 5400 );
INSERT  INTO Gamer
VALUES  ( 'BFirst05', 'ZLast05', 'Female', 2000 );
INSERT  INTO Gamer
VALUES  ( 'CFirst06', 'YLast06', 'Male', 4320 );
INSERT  INTO Gamer
VALUES  ( 'CFirst07', 'YLast07', 'Male', 4400 );
GO -- Run the previous command and begins new batch

SELECT  *
FROM    Gamer;
GO -- Run the previous command and begins new batch
```

	Id	FirstName	LastName	Gender	GameScore
1	1	AFirst01	XLast01	Female	3500
2	2	AFirst02	YLast02	Female	4000
3	3	BFirst03	YLast03	Male	4600
4	4	BFirst04	YLast04	Male	5400
5	5	BFirst05	ZLast05	Female	2000
6	6	CFirst06	YLast06	Male	4320
7	7	CFirst07	YLast07	Male	4400

3.2. SQL Injection

```
--=========================================================================
--T041_03_02
--SQL Injection
-------------------------------------------------------------
--T041_03_02_01
--Create Sample data
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Table1' ) )
    BEGIN
        TRUNCATE TABLE dbo.Table1;
        DROP TABLE Table1;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Table1
(
  Id INT IDENTITY(1, 1)
         PRIMARY KEY ,
  [Name] NVARCHAR(50)
);
GO -- Run the previous command and begins new batch

-------------------------------------------------------------
--T041_03_02_02
--FreeProcCache clear the query plans in the query plan cache.
DBCC FREEPROCCACHE;
GO -- Run the previous command and begins new batch
```

Messages

DBCC execution completed. If DBCC printed error messages, contact your system administrator.

```
--QueryA
Declare @FN NVARCHAR(100)
SET @FN = 'AFirst02';
--SET @FN = 'AFirst01';
Declare @sql nvarchar(max) =
'SELECT * FROM Gamer where FirstName = ''' +  @FN + ''''
Exec(@sql)
SELECT  *
FROM    Table1;
GO -- Run the previous command and begins new batch
/*
Display the FirstName=N'AFirst02'
*/
--QueryB
SELECT  cp.usecounts ,
        cp.cacheobjtype ,
        cp.objtype ,
        st.text ,
        qp.query_plan
FROM    sys.dm_exec_cached_plans AS cp
        CROSS APPLY sys.dm_exec_sql_text(plan_handle) AS st
        CROSS APPLY sys.dm_exec_query_plan(plan_handle) AS qp
ORDER BY cp.usecounts DESC;
GO -- Run the previous command and begins new batch
/*
1.
Highlight QueryA and QueryB.
When execute both queries at first time.
We can see the usecounts of query plan of QueryA is 1.
That means the query plan of QueryA is generated
when QueryA was run at first time.
2.
Highlight QueryA and QueryB.
When execute both queries at second time.
We can see the usecounts of query plan of QueryA is 2.
That means the query plan of QueryA is re-used.
when QueryA was run at first time.
3.
Change
SET @FN = 'AFirst02';
to
SET @FN = 'AFirst01';
Highlight QueryA and QueryB.
When execute both queries at first time.
We can see the usecounts of query plan of QueryA is 2.
Because the SQL server automatically generate parameterised queries in the background.
However, SQL server is not always
automatically generating parameterised queries.
In order to always create parameterised queries,
we need to use sp_executesql.
*/
```

	Id	FirstName	LastName	Gender	GameScore
1	2	AFirst02	YLast02	Female	4000

	Id	Name

	usecounts	cacheobjtype	objtype	text	query_plan
1	1	Compiled Plan	Adhoc	/* Display the FirstName=N'AFirst02' */ --QueryB SEL...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2	1	Compiled Plan	Adhoc	SELECT * FROM Gamer where FirstName = 'AFirst02'	<ShowPlanXML xmlns="http://schemas.microsoft.com...
3	1	Compiled Plan	Prepared	(@1 varchar(8000))SELECT * FROM [Gamer] WHERE [...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
4	1	Compiled Plan	Adhoc	--QueryA Declare @FN NVARCHAR(100) SET @FN ...	<ShowPlanXML xmlns="http://schemas.microsoft.com...

Results Messages

	Id	FirstName	LastName	Gender	GameScore
1	1	AFirst01	XLast01	Female	3500

	Id	Name

	usecounts	cacheobjtype	objtype	text	query_plan
1	2	Compiled Plan	Adhoc	/* Display the FirstName=N'AFirst02' */ --QueryB SEL...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
2	2	Compiled Plan	Prepared	(@1 varchar(8000))SELECT * FROM [Gamer] WHERE [...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
3	2	Compiled Plan	Adhoc	declare @BatchID uniqueidentifi...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
4	2	Compiled Plan	Adhoc	declare @BatchID uniquei...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
5	1	Compiled Plan	Adhoc	--QueryA Declare @FN NVARCHAR(100) SET @FN ...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
6	1	Compiled Plan	Adhoc	SELECT * FROM Gamer where FirstName = 'AFirst02'	<ShowPlanXML xmlns="http://schemas.microsoft.com...
7	1	Compiled Plan	Proc	CREATE PROCEDURE [dbo].[GetMyRunningJobs] @...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
8	1	Compiled Plan	Proc	CREATE PROCEDURE [dbo].[GetDBVersion] @...	<ShowPlanXML xmlns="http://schemas.microsoft.com...
9	1	Compiled Plan	Adhoc	SELECT * FROM Gamer where FirstName = 'AFirst01'	<ShowPlanXML xmlns="http://schemas.microsoft.com...
10	1	Compiled Plan	Adhoc	--QueryA Declare @FN NVARCHAR(100) SET @FN ...	<ShowPlanXML xmlns="http://schemas.microsoft.com...

```
------------------------------------------------------------
--T041_03_02_03
Declare @FN NVARCHAR(100) = '''; DROP TABLE dbo.Table1; --''';
Declare @sql nvarchar(max) =
'SELECT * FROM Gamer where FirstName = ''' +  @FN + ''''
Exec(@sql)
SELECT  *
FROM    Table1;
GO -- Run the previous command and begins new batch
/*
1.
**SQL Injection
The Table1 will be dropped.
2.
In summary,
Never use Exec/Execute to run dynamic sql query,
it can cause sql injection.
In order to always create parameterised queries to avoid sql injection.
we need to use sp_executesql.
*/


------------------------------------------------------------
--T041_03_02_04
--Create Sample data
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Table1' ) )
    BEGIN
        TRUNCATE TABLE dbo.Table1;
        DROP TABLE Table1;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Table1
```

```
(
  Id INT IDENTITY(1, 1)
        PRIMARY KEY ,
  [Name] NVARCHAR(50)
);
GO -- Run the previous command and begins new batch
------------------------------------------------------------
--T041_03_02_05
Declare @FN NVARCHAR(100)
SET @FN = 'AFirst02';
--SET @FN = 'AFirst02';
Declare @sql nvarchar(max) =
'SELECT * FROM Gamer where FirstName = '+  +QUOTENAME(@FN,'''')
Exec(@sql)
SELECT  *
FROM    Table1;
GO -- Run the previous command and begins new batch
------------------------------------------------------------
--T041_03_02_06
Declare @FN NVARCHAR(100)
SET @FN = '''; DROP TABLE dbo.Table1; --''';
--SET @FN = 'AFirst02';
Declare @sql nvarchar(max) =
'SELECT * FROM Gamer where FirstName = '+  +QUOTENAME(@FN,'''')
Exec(@sql)
SELECT  *
FROM    Table1;
GO -- Run the previous command and begins new batch
/*
1.
The Table1 will NOT be dropped, because of QUOTENAME(@FN,'''').
QuoteName(str1, str2) will use str2 to wrap str1.
QUOTENAME('AA','''') will return 'AA'.
2.
In summary,
Even the QUOTENAME() function can prevent sql injection.
Still never use Exec/Execute to run dynamic sql query
In order to always create parameterised queries to avoid sql injection.
we better to use sp_executesql.
*/
------------------------------------------------------------
--T041_03_02_07
--Create Sample data
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Table1' ) )
    BEGIN
        TRUNCATE TABLE dbo.Table1;
        DROP TABLE Table1;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Table1
(
  Id INT IDENTITY(1, 1)
        PRIMARY KEY ,
  [Name] NVARCHAR(50)
);
GO -- Run the previous command and begins new batch
------------------------------------------------------------
```

```
--T041_03_02_08
--Bad dynamic sql queries.
--Building a dynamic sql queries by concatenating strings cause the vulnerability of SQL injection.
DECLARE @sql1 NVARCHAR(1000)
= 'SELECT *
FROM Gamer
WHERE FirstName LIKE ''%' + 'B' + '%'' AND ' + 'LastName LIKE ''%' + 'Y'
    + '%''';
EXECUTE sp_executesql @sql1;
GO -- Run the previous command and begins new batch
/*
Display the FirstName LIKE '%B%' AND LastName LIKE '%Y%'
*/
-----------------------------------------------------------
--T041_03_02_09
--Bad dynamic sql queries.
--Building a dynamic sql queries by concatenating strings cause the vulnerability of SQL injection.
DECLARE @sql1 NVARCHAR(1000)
= 'SELECT *
FROM Gamer
WHERE FirstName LIKE ''%' + N'''; DROP TABLE dbo.Table1; --' + '%'' AND ' + 'LastName LIKE ''%' + 'Y'
    + '%''';
EXECUTE sp_executesql @sql1;
SELECT  *
FROM    Table1;
GO -- Run the previous command and begins new batch
/*
**SQL Injection
The Table1 will be dropped.
*/
-----------------------------------------------------------
--T041_03_02_10
--Create Sample data
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Table1' ) )
    BEGIN
        TRUNCATE TABLE dbo.Table1;
        DROP TABLE Table1;
    END;
GO -- Run the previous command and begins new batch
CREATE TABLE Table1
(
  Id INT IDENTITY(1, 1)
        PRIMARY KEY ,
  [Name] NVARCHAR(50)
);
GO -- Run the previous command and begins new batch
-----------------------------------------------------------
--T041_03_02_11
--Good dynamic sql queries.
--Using sp_executesql parameters is always the best for dynamic sql queries.
DECLARE @sq2 NVARCHAR(1000)
= 'SELECT *
FROM Gamer
WHERE FirstName LIKE ''%''+@FirstName+''%''
AND LastName LIKE ''%''+@LastName+''%''';
DECLARE @params NVARCHAR(1000) = '@FirstName NVARCHAR(100), @LastName NVARCHAR(100)';
EXECUTE sp_executesql @sq2, @params, @FirstName = 'B', @LastName = 'Y';
```

```
GO -- Run the previous command and begins new batch
/*
Display the FirstName LIKE '%B%' AND LastName LIKE '%Y%'
*/
----------------------------------------------------------
--T041_03_02_12
--Good dynamic sql queries.
--Using sp_executesql parameters is always the best for dynamic sql queries.
DECLARE @sq2 NVARCHAR(1000)
= 'SELECT *
FROM Gamer
WHERE FirstName LIKE ''%''+@FirstName+''%''
AND LastName LIKE ''%''+@LastName+''%''';
DECLARE @params NVARCHAR(1000) = '@FirstName NVARCHAR(100), @LastName NVARCHAR(100)';
EXECUTE sp_executesql @sq2, @params, @FirstName =  N'''; DROP TABLE dbo.Table1; --', @LastName = 'Y';
SELECT  *
FROM    Table1;
GO -- Run the previous command and begins new batch
/*
**Prevent SQL Injection
The Table1 will NOT be dropped.
*/

==========================================================================
```

4. Clean up

```
--=============================================================================================
--T041_03_03
--Clean up
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Gamer' ) )
    BEGIN
        TRUNCATE TABLE dbo.Gamer;
        DROP TABLE Gamer;
    END;
GO -- Run the previous command and begins new batch
IF ( EXISTS ( SELECT    *
              FROM      INFORMATION_SCHEMA.TABLES
              WHERE     TABLE_NAME = 'Table1' ) )
    BEGIN
        TRUNCATE TABLE dbo.Table1;
        DROP TABLE Table1;
    END;
GO -- Run the previous command and begins new batch
```