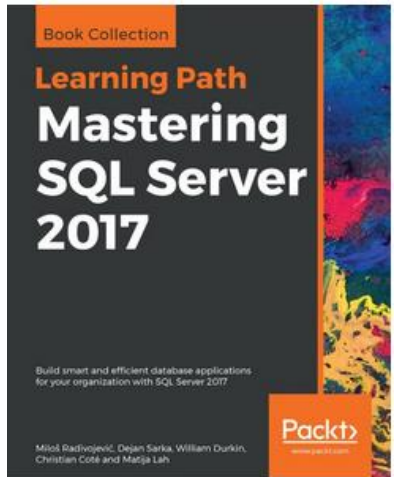
PARAMETER SNIFFING IN SQL SERVER STORED PROCEDURES

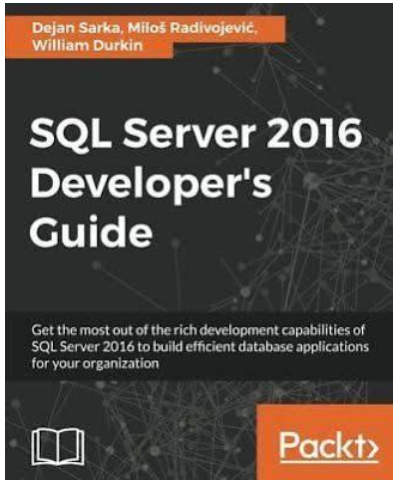
MILOŠ RADIVOJEVIĆ

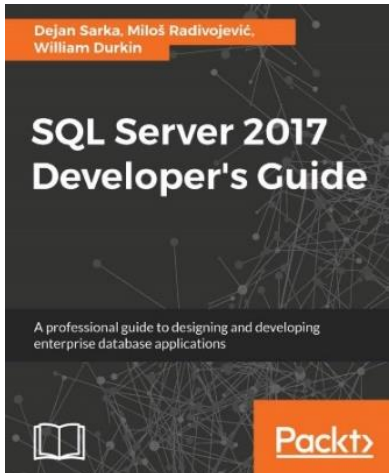
PRINCIPAL DATABASE CONSULTANT, BWIN GVC, AUSTRIA

MILOŠ RADIVOJEVIĆ

- Data Platform MVP
- Principal Database Consultant at bwin, Vienna, Austria
- Co-Founder: SQL Pass Austria
- Conference Speaker, Book Author







AGENDA

- What is Parameter Sniffing?
- Symptoms
- When and Why It is a Problem?
- Solutions

Demo – Sample Table

StudentExams

Column Name	Data Type	Allow Nulls
exam_number	int	☐
student_id	int	☐
exam_id	tinyint	☐
exam_note	tinyint	☐
exam_date	datetime	☐
exam_comment	char(500)	☑
		☐

```
SELECT TOP 10 * FROM dbo.StudentExams
```

exam_number	student_id	exam_id	exam_note	exam_date	exam_comment
1	136261	13	8	1998-07-23 00:00:00.000	test
2	447761	39	9	2003-04-22 00:00:00.000	test
3	153716	6	9	2000-01-21 00:00:00.000	test
4	120951	4	5	2004-03-28 00:00:00.000	test
5	44412	17	8	2006-11-08 00:00:00.000	test
6	471371	10	8	2001-03-07 00:00:00.000	test
7	285297	15	7	2007-04-19 00:00:00.000	test
8	224925	31	6	2001-02-18 00:00:00.000	test
9	355452	14	5	2006-07-29 00:00:00.000	test
10	77211	11	7	2001-09-27 00:00:00.000	test

- 1 M rows
- Indexes: **student_id** and **exam_date**

Demo- Requirements

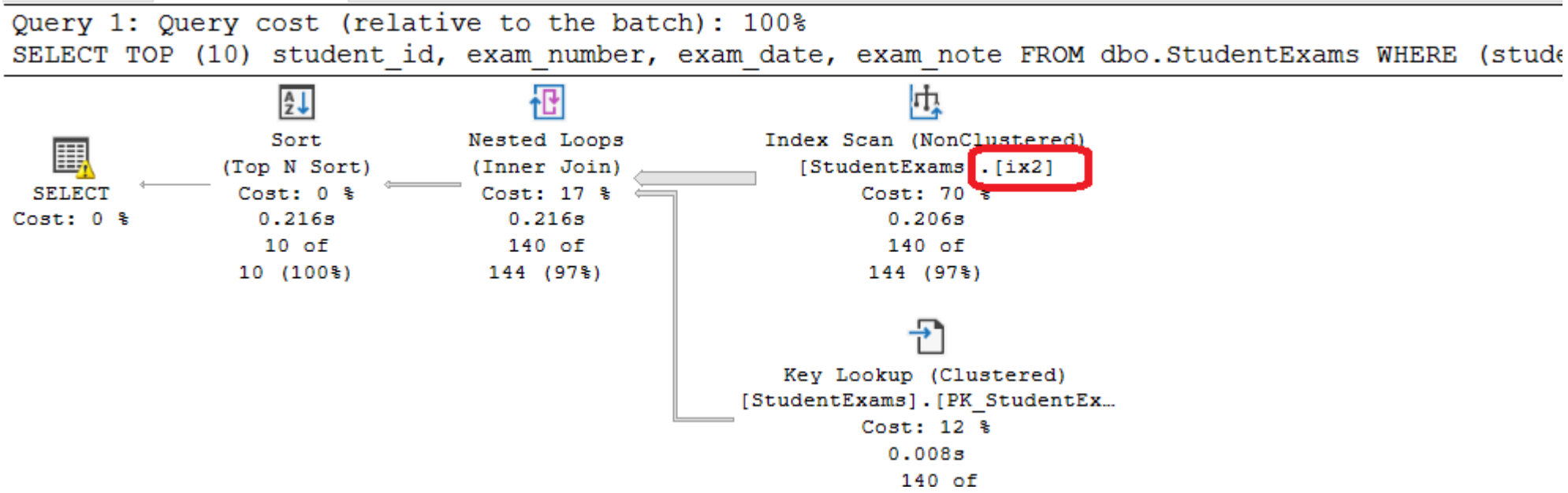
- Two input parameters: Student Id and Order Date
- Both parameters are optional
- The result set should contain up to 10 rows sorted by Exam Note descending

Common Solution

```
CREATE OR ALTER PROCEDURE dbo.GetExams
@student_id INT = NULL,
@exam_date DATETIME = NULL
AS
BEGIN
      SELECT TOP (10) student_id, exam_number, exam_date, exam_note
      FROM  dbo.StudentExams
      WHERE (student_id = @student_id OR @student_id IS NULL)
            AND (exam_date = @exam_date OR @exam_date IS NULL)
      ORDER BY exam_note DESC
END
```

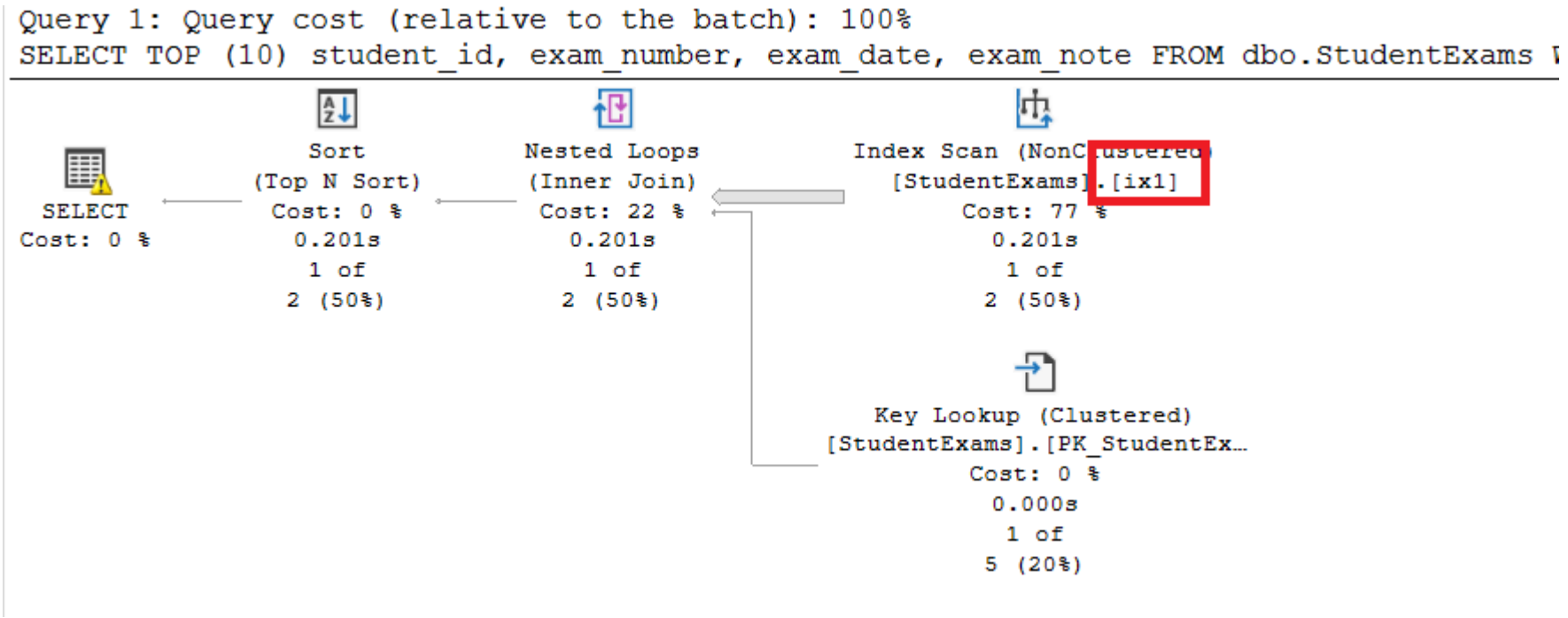
Execution Plans – Plan 1

```
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE;
GO
EXEC dbo.getExams NULL, '20010731';
GO
```



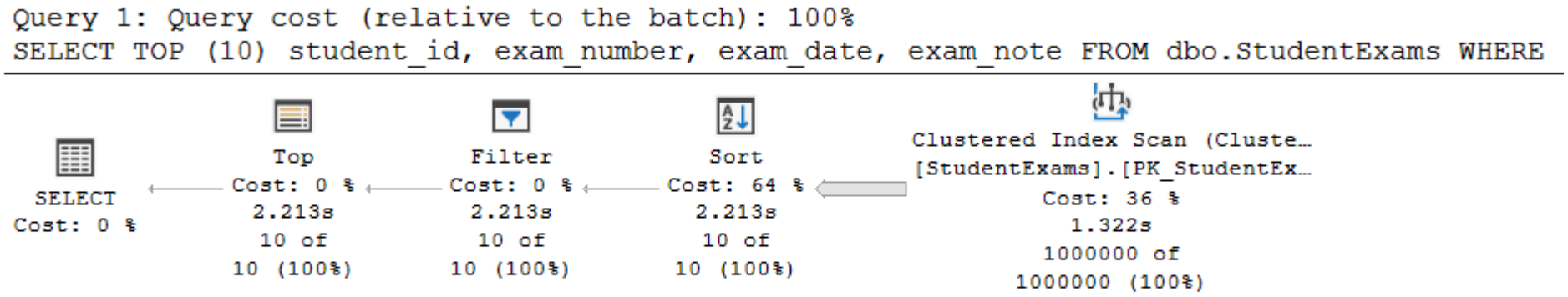
Execution Plans – Plan 2

```
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE;
GO
EXEC dbo.getExams 31302, NULL;
```

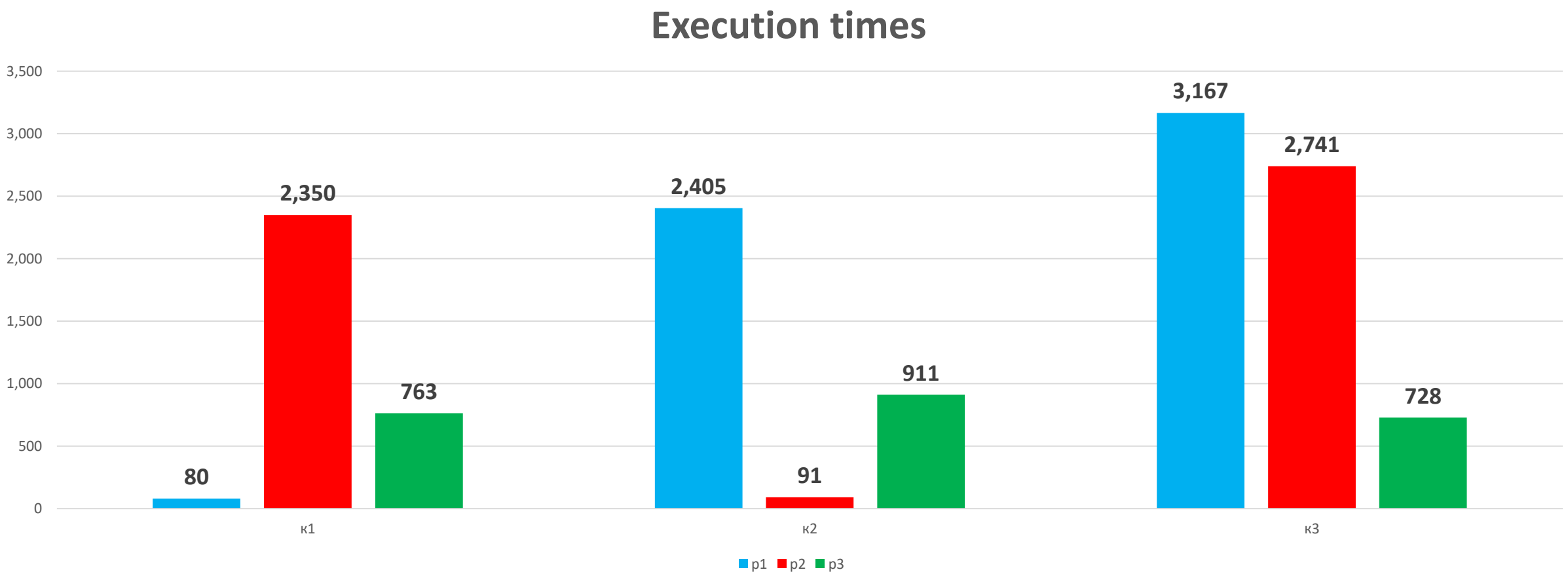


Execution Plans – Plan 3

```
ALTER DATABASE SCOPED CONFIGURATION CLEAR PROCEDURE_CACHE;
GO
EXEC dbo.getExams NULL, NULL;
```



Results


Execution times
3,500
3,000
2,500
2,000
1,500
1,000
500
0
80
2,350
763
2,405
91
911
3,167
2,741
728
к1
к2
к3
p1
p2
p3

What is Parameter Sniffing?

- During stored procedure compilation, the values from parameters are evaluated (sniffed) and used to create an execution plan
- Future executions will re-use this plan
- This is a behaviour, not a bug
  - **It is good for invocations with similar parameters**
  - **It can significantly, sometimes dramatically degrade the performance for non-common parameters**

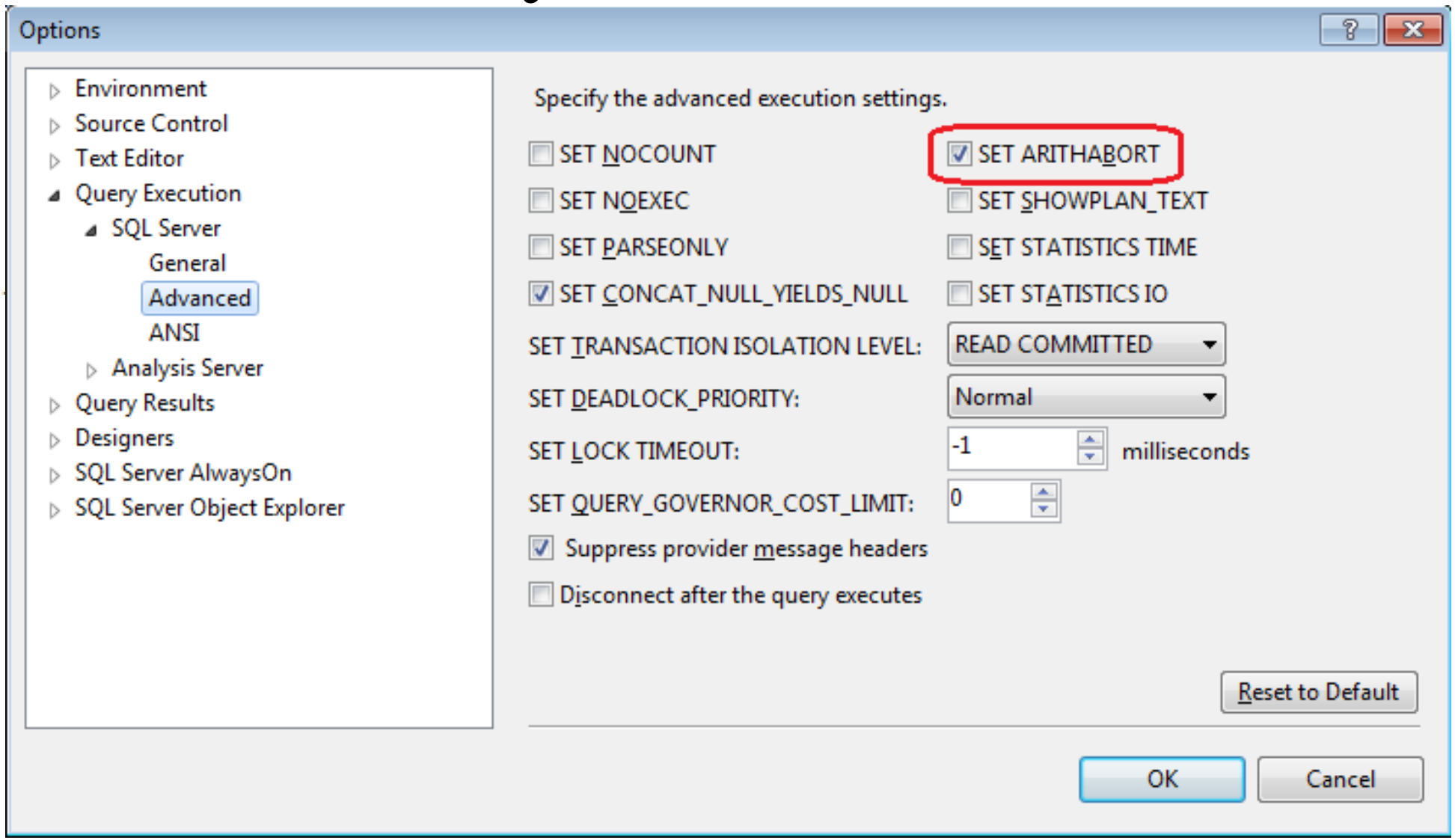
When PS is a Problem?

- When several execution plans are possible
- Stored procedures prone to parameter sniffing
  - with parameters participating in range operators
  - with optional parameters
- Not only limited to stored procedures
  - Static parameterized queries
  - Dynamic queries executed with **sp_executesql**

SSMS Mistery

- It works instantly in the SSMS, but it takes 5 seconds in the application
  - A new execution plan is created for the stored procedure invocation within SSMS!
- Factors that affect plan-reuse
- ANSI_NULLS
- ANSI_PADDING
- ANSI_WARNINGS
- ARITHABORT
- QUOTED_IDENTIFIER
- CONCAT_NULL_YIELDS_NULL
- DATEFORMAT

SSMS Mistery



Solution 1 – Disable Parameter Sniffing Effect

- Disable Parameter Sniffing Effect
  - SQL Server uses average distribution statistics to choose an execution plan
  - Neutralize parameter values – an average solution
  - It does not work always!
- How to implement?
  - Using the OPTIMIZE FOR UKNOWN query hint
  - Wrap parameters in local variables
  - ALTER DATABASE SCOPED CONFIGURATION SET PARAMETER_SNIFFING = OFF;
  - ~~Using TF 4136~~

Solution 1a – Disable Parameter Sniffing Effect

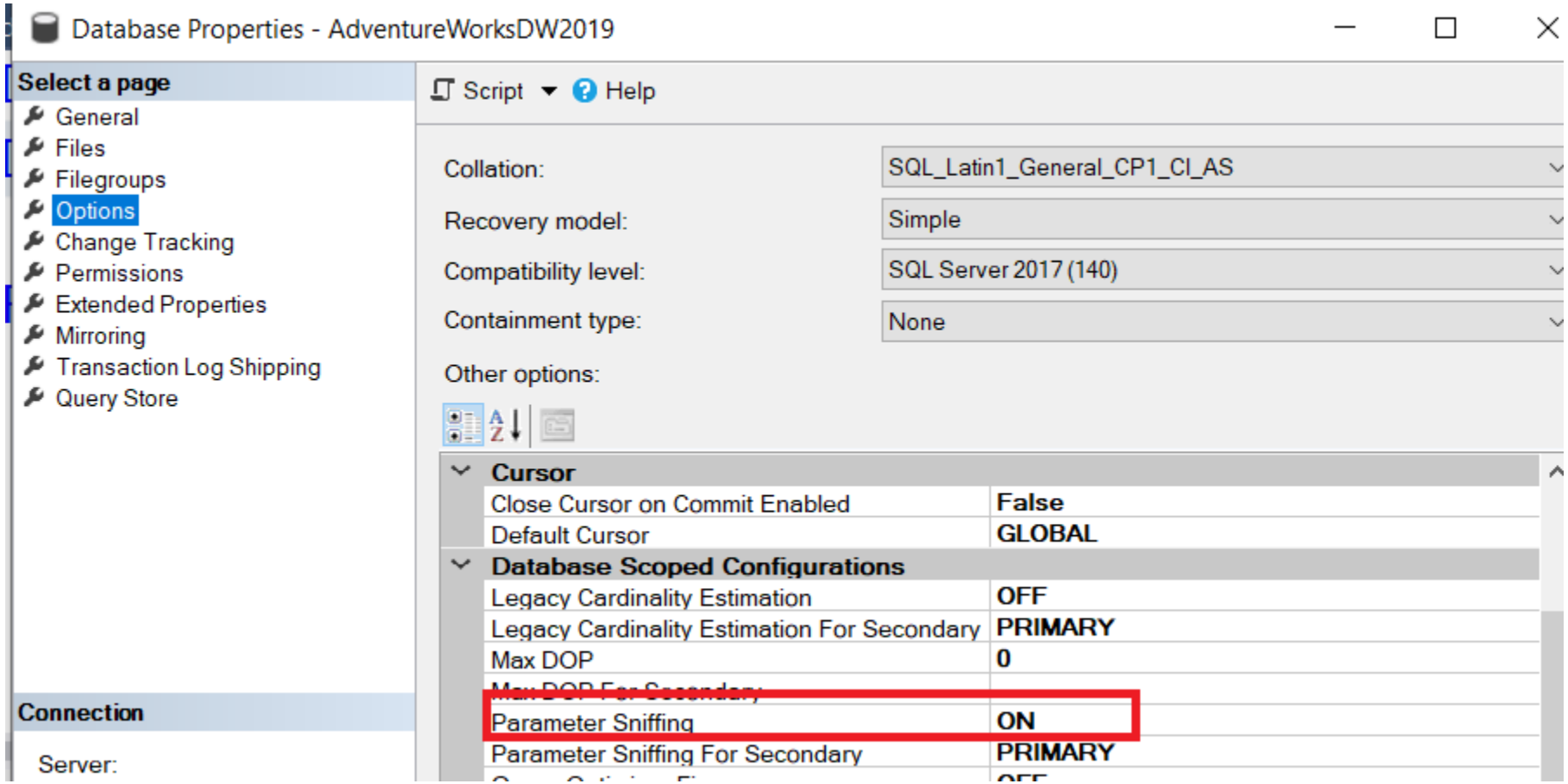
```
CREATE OR ALTER PROCEDURE dbo.GetExams
@student_id INT = NULL,
@exam_date DATETIME = NULL
AS
BEGIN
      SELECT TOP (10) student_id, exam_number, exam_date, exam_note
      FROM  dbo.StudentExams
      WHERE (student_id = @student_id OR @student_id IS NULL)
            AND (exam_date = @exam_date OR @exam_date IS NULL)
      ORDER BY exam_note DESC
   OPTION (OPTIMIZE FOR UNKNOWN)
END
```

Solution 1b – Disable Parameter Sniffing Effect

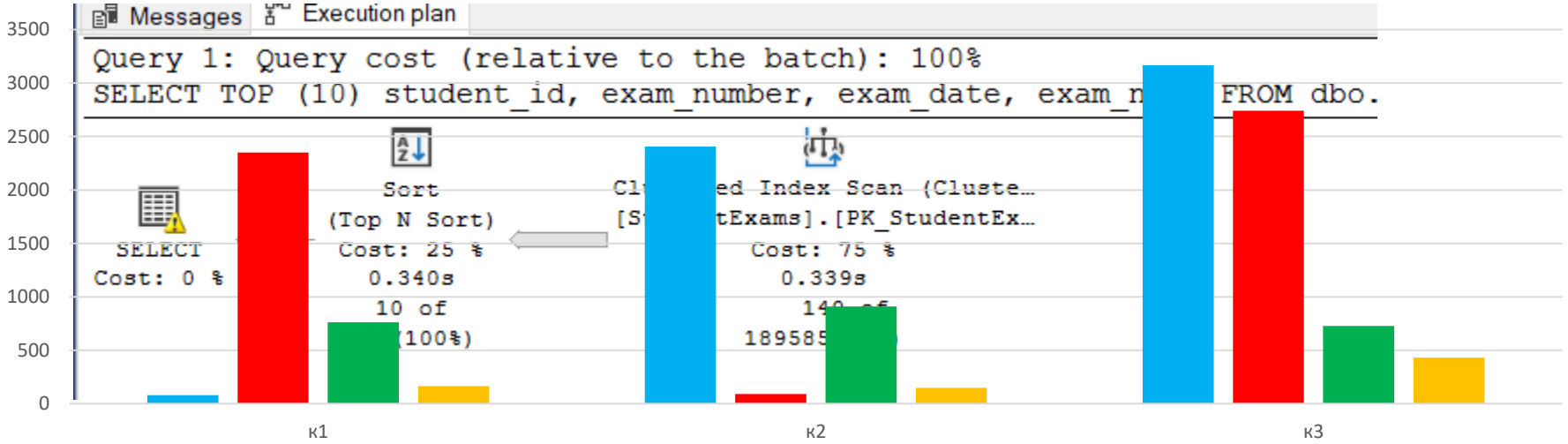
```
CREATE OR ALTER PROCEDURE dbo.GetExams
@student_id INT = NULL,
@exam_date DATETIME = NULL
AS
BEGIN
      DECLARE @student_id_local INT = @student_id,
        @exam_date_local DATETIME = @exam_date
      SELECT TOP (10) student_id, exam_number, exam_date, exam_note
      FROM  dbo.StudentExams
      WHERE (student_id = @student_id_local OR @student_id_local IS NULL)
            AND (exam_date = @exam_date_local OR @exam_date_local IS NULL)
      ORDER BY exam_note DESC

END
```

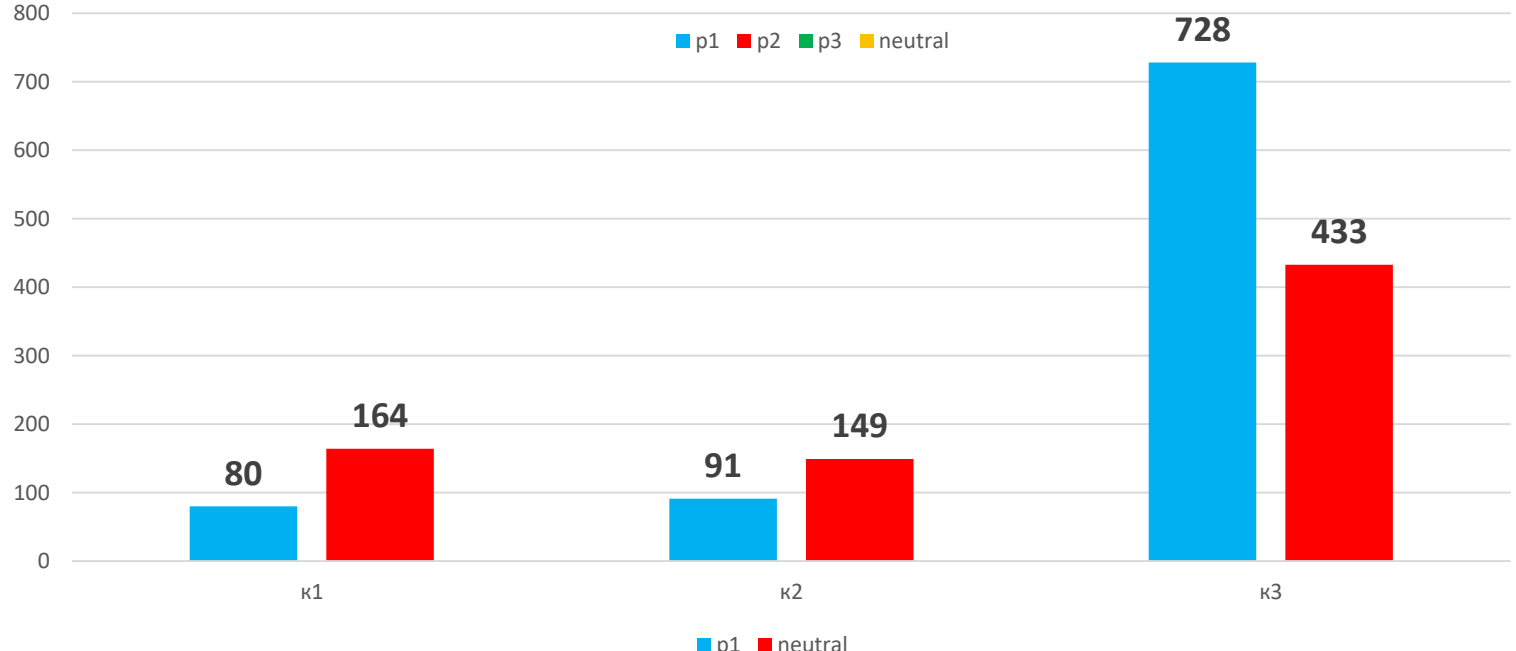
Solution 1c – Disable Parameter Sniffing Effect

```
ALTER DATABASE SCOPED CONFIGURATION SET PARAMETER_SNIFFING = OFF;
```



Results – Disable Parameter Sniffing Effect





Solution 2 – Choose Favorite Combinations

- Goal: To work perfect for the most common or important combination(s)
  - SQL Server generates an optimal execution plan for it and reuses it
  - Need to contact business people
- How to implement?
  - Using the OPTIMIZE FOR query hint
  - Query Decomposition (IF)

Solutions – Choose Favorite Combinations

```
ALTER PROCEDURE dbo.getExams
@student_id INT = NULL,
@exam_date DATETIME = NULL
AS
BEGIN
    SELECT
        TOP (10) student_id, exam_number, exam_date, exam_note
    FROM
    dbo.StudentExams
  WHERE
    (student_id = @student_id OR @student_id IS NULL)
  AND
    (exam_date = @exam_date OR @exam_date IS NULL)
  ORDER BY
    exam_note DESC
  OPTION (OPTIMIZE FOR (@student_id = 1))
END
```

Solution 3 – Disable Execution Plan Re-use

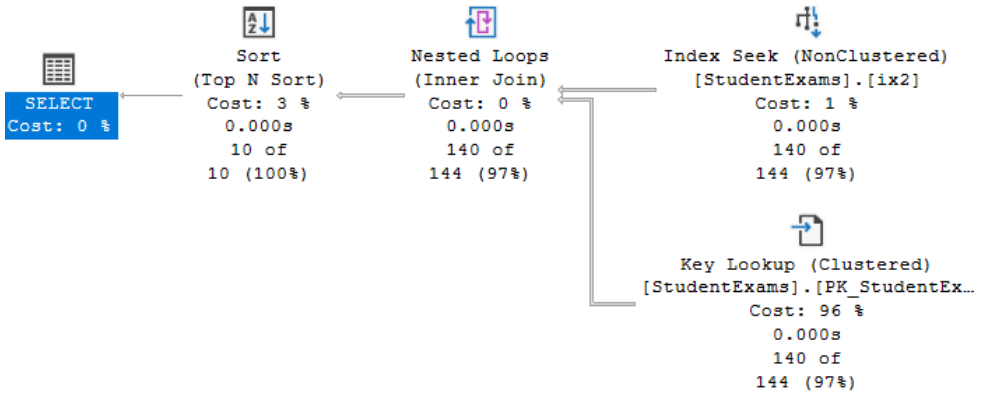
- Goal: Always get the optimal execution plan
  - Pros:
    - Optimal plan for each execution
    - **The plan can be better than the best plan in the initial solution!!!**
  - Cons:
    - Compiled by each execution
- How to implement?
  - ~~Define SP with the option WITH RECOMPILE (**not recommended!**)~~
  - OPTION (RECOMPILE) at the statement level

Solution 3 – OPTION (RECOMPILE)

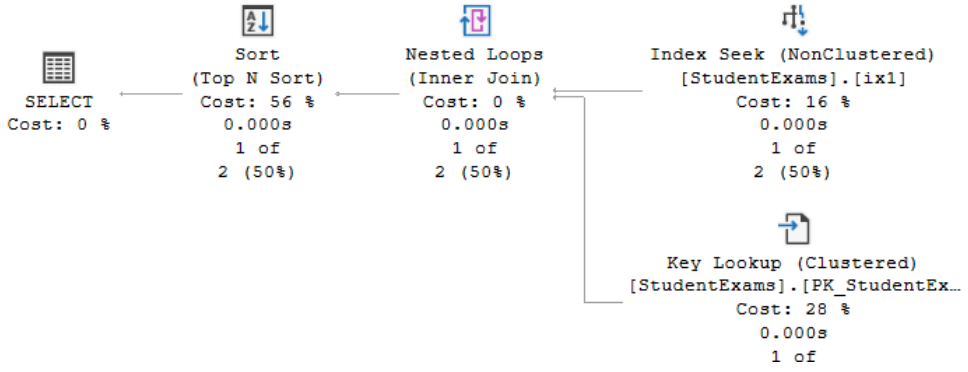
```
ALTER PROCEDURE dbo.getExams
@student_id INT = NULL,
@exam_date DATETIME = NULL
AS
BEGIN
    SELECT TOP (10) student_id, exam_number, exam_date, exam_note
    FROM dbo.StudentExams
    WHERE (student_id = @student_id OR @student_id IS NULL)
    AND (exam_date = @exam_date OR @exam_date IS NULL)
    ORDER BY exam_note DESC
    OPTION (RECOMPILE, MAXDOP 1)
END
```

Results – OPTION (RECOMPILE)

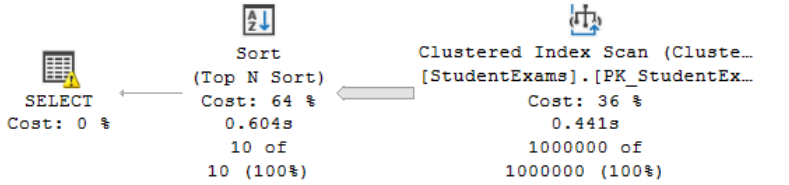
Query 1: Query cost (relative to the batch): 0%
SELECT TOP (10) student_id, exam_number, exam_date, exam_note FROM dbo.



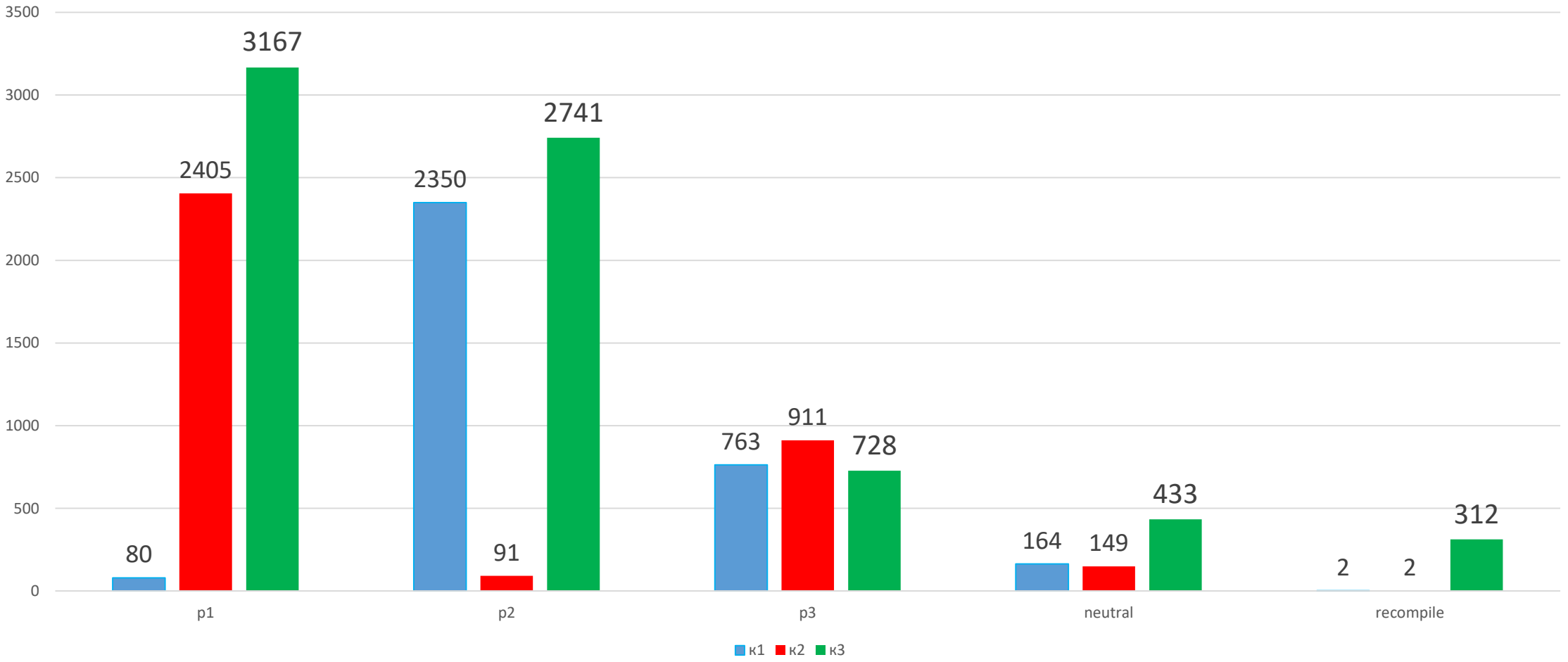
Query 2: Query cost (relative to the batch): 0%
SELECT TOP (10) student_id, exam_number, exam_date, exam_note FROM dbo.



Query 3: Query cost (relative to the batch): 100%
SELECT TOP (10) student_id, exam_number, exam_date, ex



Results – OPTION (RECOMPILE)



Solutions – Static or Dynamic Query Decomposition

- Goal: Always get the optimal execution plan and avoid recompilation
  - Pros:
    - Optimal plan for each execution
    - Reuse
    - **The plan can be better than the best plan in the initial solution!!!**
  - Cons:
    - Maintenance problems, SQL Injection…
- How to implement?
  - Static SQL (Decission Tree Implementation)
  - Dynamic SQL

Solution 4 – Decision Tree

```
ALTER PROCEDURE dbo.GetExams
@student_id INT = NULL,
@exam_date DATETIME = NULL
AS
BEGIN
    IF @student_id IS NOT NULL EXEC dbo.getExams1
    ELSE
    IF @exam_date IS NOT NULL EXEC dbo.getExams2
    ELSE EXEC dbo.getExams3
END
```

Solution 5 – Decision Tree Dynamic SQL

```
ALTER PROCEDURE dbo.getExams
@student_id INT = NULL,
@exam_date DATETIME = NULL
AS
BEGIN
    DECLARE @sql nvarchar(600) = N'SELECT TOP (10) student_id, exam_number, exam_date, exam_note
FROM dbo.StudentExams WHERE 1 = 1  '
    IF @student_id IS NOT NULL
        SET @sql+=' AND student_id = @sid '
    IF @exam_date IS NOT NULL
        SET @sql+=' AND exam_date = @ed '
    SET @sql+=' ORDER BY exam_note DESC '

EXEC sp_executesql @sql,  N'@sid INT, @ed DATETIME',  @sid = @student_id, @ed = @exam_date;
END
```

Solution 6 – A Combined Solution

- Goal: Always get the optimal execution plan and reuse it for most common parameters
  - Pros:
    - Optimal plan for most important executions
    - Reuse
    - **The plan can be better than the best plan in the initial solution!!!**
- How to implement?
  - Static SQL Decission Tree Implementation combined with OPTION (RECOMPILE)

Conclusion

- If you are OK with an average execution plan, you can disable parameter sniffing
- To get the best possible plan use the OPTION (RECOMPILE)
- If the compilation is to expensive, use query decomposition
- If you cannot use neither RECOMPILE, nor static decomposition, you can use dynamic SQL, but you can expect security issues
- You can also make a compromise and optimize just a small set of parameter values

THANK YOU AND #STAYHOME