Parameter Sniffing is a common performance issue in SQL Server, especially in environments with highly variable query parameters.

Let's break it down clearly:

1. What is Parameter Sniffing?

When a **stored procedure** or **parameterized query** is executed, SQL Server's query optimizer generates an execution plan based on the **first set of parameter values it receives**. This plan is then cached for reuse.

- If the initial parameter value is typical, the plan may work well for similar values.
- If later executions use very different parameter values, the cached plan may be **suboptimal**, causing slow queries.

Example:

```
CREATE PROCEDURE GetOrders
  @CustomerID INT
AS
BEGIN
  SELECT *
  FROM Orders
  WHERE CustomerID = @CustomerID;
END
```

- First execution: @CustomerID = 1 → small dataset → optimizer picks **nested loops join**.
- Later execution: @CustomerID = 10000 → large dataset → cached plan (nested loops) is inefficient → performance drops.

This behavior is **Parameter Sniffing**: SQL Server "sniffs" the first parameter to build its plan, sometimes causing poor performance for other parameter values.

2. How to Fix Parameter Sniffing Permanently

There are multiple strategies. Some are query-level, others involve SQL Server features:

a) Optimize for Typical or Generic Value

- Use OPTION (OPTIMIZE FOR (@param = value)) to force a specific plan.

```
SELECT *
FROM Orders
WHERE CustomerID = @CustomerID
OPTION (OPTIMIZE FOR (@CustomerID = 1)); -- typical value
```

b) Use OPTIMIZE FOR UNKNOWN

- Forces SQL Server to create a **generic plan** that is not based on the first parameter.

```
SELECT *
FROM Orders
WHERE CustomerID = @CustomerID
OPTION (OPTIMIZE FOR UNKNOWN);
```

- Works well when parameter values vary widely.

c) Recompile Stored Procedure per Execution

- Forces SQL Server to generate a fresh plan each time.

```
CREATE PROCEDURE GetOrders
  @CustomerID INT
WITH RECOMPILE
AS
BEGIN
```

```
    SELECT *
    FROM Orders
    WHERE CustomerID = @CustomerID;
END
```

- Downside: increases CPU usage because plan is compiled every execution.

d) Local Variables Inside Procedure

- Assign parameter to a local variable to prevent sniffing.

```
CREATE PROCEDURE GetOrders
    @CustomerID INT
AS
BEGIN
    DECLARE @LocalCustomerID INT = @CustomerID;
    SELECT *
    FROM Orders
    WHERE CustomerID = @LocalCustomerID;
END
```

- Optimizer uses **average statistics** instead of parameter value.

e) Plan Guides / Query Hints

- Advanced: Use plan guides to control plan choice for specific queries without changing code.

f) Index & Statistics Optimization

- Update statistics regularly: UPDATE STATISTICS Orders WITH FULLSCAN;
- Ensure indexes match typical queries, reducing sensitivity to parameter variations.

3. Best Practices

- Use OPTION (OPTIMIZE FOR UNKNOWN) for high-variability queries.
- Use WITH RECOMPILE only for queries with highly skewed parameter distributions.
- Keep statistics up-to-date.
- Avoid over-indexing, as it can exacerbate suboptimal plans.
- Monitor execution plans via sys.dm_exec_query_stats and Query Store (SQL Server 2016+).

Parameter Sniffing in SQL Server & Permanent Solutions

Understanding the Issue and How to Fix It

The Problem

Parameter Sniffing: SQL Server caches an execution plan based on the first parameter value it "sniffs", which might not be optimal for subsequent uses.

1st Execution

```
GetOrders @CustomerID INT AS
SELECT * FROM Orders
    WHERE CustomerID = @
```

Cached Execution Plan (Optimized for Small Dataset)

2nd Execution

```
GetOrders @CustomerID = 10000
SELECT * FROM Orders
    WHERE CustomerID = @CustomerID;
```

Plan May Not Work Well for Other Parameters

Permanent Solutions

Optimize for Typical Value

OPTIMIZE FOR (@param = value)

```
SELECT ...
FROM Orders
WHERE CustomerrID =
@CustomerID
OPTION (OPTIMIZE
FOR (@CustomerID = 1);
```

Hint SQL Server to use a specific common value for optimization

Optimize for Unknown

OPTIMIZE FOR UNKNOWN

```
SELECT ...
FROM Orders
WHERE CustomerID =
@CustomerID
OPTION (OPTIMIZE
FOR UNKNOWN);
```

Generate a plan that is not based on the first parameter value

Always Recompile

WITH RECOMPILE

```
CREATE PROCEDURE
GetOrders @CustomerID
    INT RECOMPILE
AS BEGIN
  SELECT Orders
  WHERE CustomerrID
  END
```

Force SQL Server to recompile the query every time it's executed

Use Local Variables


```
CREATE PROCEDURE GetOrders
  @CustomerID INT WITRECOMPILE
AS BEGIN
SELECT *
   FROM Orders
WHERE CustomeerID
   := localCustomerIDI)
END
```

Assign parameter to a local variable to prevent sniffing

Other Best Practices:

- ✔ Update Statistics Regularly
- ✔ Optimize Indexes & Keep Current
- ✔ Monitor Query Performance
- ✔ Monitor Query Performance