SQL Server Stored Procedure coding and .NET interoperability best practices

A best practice guide in coding stored procedures in SQL Server 2005 and SQL Server 2008

# Humble foreword

These are best practices of SQL Server 2005 and above best practices, but in accordance to interoperability with Entity Framework 4 in .NET 4.0 and also provide broadest interest in interoperability with other database such as Oracle 8i (8.1.6+) and IBM DB2 8.2+.

These rules are provided in Design Pattern style but adapted with no ”where to apply” condition and post condition.

# Overview of this best practices guide

## Goals of this best practices guide

The goals of this best practices guide are:

1. Predictable behavior of stored proc and its mapping to EF counterpart
2. Moving towards side effect free or minimal side effect in stored proc
3. Testable result from side effect free or minimal side effect of stored proc

## Reasoning background of why

# Software compatibilities and requirements

This guide will require:

* SQL Server 2005 SP1 or SQL Server 2008 and above
* Microsoft .NET Framework 4

Optional recommendation:

* Visual Studio 2010 Professional, Premium or Ultimate

# Guidelines of achieving less or no side effects

## Overview of functional programming

Functional programming means programming with functions as functions in math. Functions in this context (math context), ***must*** return a value.

For example:

f(x) = x \* x

When x = 5, then f(5) will always returns 25

This means:

1. A function has a number of defined parameters with a single strong typed value
2. A function has desired constant same result when called many times with the same parameter value.

The f(x) function can then be considered as side effect free function. This is different when you call other function or method that returns changing/variable values.

For example:

f(x) = DateTime.Now

The result of f(x) can’t be guaranteed the same, although it’s correct behavior (***as you would expected to return current date***) but this call is not side effect free.

# Common rules

## RULE #1

AVOID have stored procedure that returns SELECT \*

### CRITICAL LEVEL WHEN NOT APPLIED:

Very Critical

### CODE SAMPLE/ILLUSTRATION

### REASON

PERFORMANCE

* Only returns columns you need, especially when dealing with tables with more than 50 columns

DESIGN PATTERN IN .NET

* Avoid construct new type that has hundreds of properties mapped from hundreds of table columns
* Type safety can’t be guaranteed, hence there will be no possible way to reason your code

EF INTEROP:

* EF mapping will be easier to understand and easier to be reasoned when the return value is constrained with specific properties mapped from specific columns returned

SQL SERVER CONSIDERATIONS:

* SELECT \* will inflict larger cost on query plan, instead of using specific columns

### FUTURE COMPATIBILITY PROBLEMS:

None

### POSSIBLE SIDE EFFECTS:

None

### APPLIES TO:

All database vendors

## RULE #2

AVOID have stored procedure that contains IF.. ELSE construct

### CRITICAL LEVEL WHEN NOT APPLIED:

Very Critical

### REASON

PERFORMANCE:

* Can be a problem if there’s a complex nested IF inside the stored procedures

DESIGN PATTERN IN .NET:

* No concern/relation, but IF should be only included in business logic, not coded in database. Therefore, the logic is hard to reason outside the physical data layer

EF INTEROP:

* EF mapping will have no guarantee which value is to be returned from any IF and ELSE branches

SQL SERVER CONSIDERATIONS:

* Hard to reason the returned value (if any)
* Can incur costly query plan

### FUTURE COMPATIBILITY PROBLEMS

* Can imply incompatibility between code path problems
* There’s no guarantee the resultant returned type is the same in the IF ELSE

### POSSIBLE SIDE EFFECTS:

* No guarantee that the stored procedure will return the same single type as expected

### APPLIES TO:

All database vendors

## RULE #3

AVOID have stored procedure that parameters that has specific SQL Server only type such as currency

### CRITICAL LEVEL WHEN NOT APPLIED:

Very Critical

### REASON

PERFORMANCE:

* No observable consequences

DESIGN PATTERN IN .NET:

* There’s no direct map from currency to any .NET types

EF INTEROP:

* EF mapping will have no guarantee which value is to be returned from any IF and ELSE branches

SQL SERVER CONSIDERATIONS:

* None, only specific in SQL Server 2005 and above
* Can infer error when using as input to other currency columns

### FUTURE COMPATIBILITY PROBLEMS

* Can imply incompatibility for other database beside SQL Server for the same stored procedure, because of SQL Server specifics
* Table used itself should not include currency typed column, otherwise it will not have cross database compatibility

### POSSIBLE SIDE EFFECTS:

* None

### APPLIES TO:

All database vendors

## RULE #4

AVOID have stored procedure that calls other stored procedure

### CRITICAL LEVEL WHEN NOT APPLIED:

Very Critical

### REASON

PERFORMANCE:

* No observable consequences, but can be slow

DESIGN PATTERN IN .NET:

* Multiple indirections make your code less ceremonial, but it’s harder to reason your code

EF INTEROP:

* No observable consequences

SQL SERVER CONSIDERATIONS:

* Hard to reason the code in the stored procedure, because we can’t debug it
* Can incur costly query plan

### FUTURE COMPATIBILITY PROBLEMS

* Can imply incompatibility for other database beside SQL Server for the same stored procedure, because of SQL Server specifics
* Table used itself should not include currency typed column, otherwise it will not have cross database compatibility

### POSSIBLE SIDE EFFECTS:

* If the called stored procedure is inflicting error, then the stored procedure can’t be guaranteed safe at the first level

### APPLIES TO:

All database vendors

## RULE #5

AVOID have stored procedure that has 1 or 2 letters variables

### CRITICAL LEVEL WHEN NOT APPLIED:

Very Critical

### REASON

PERFORMANCE:

* No observable consequences

DESIGN PATTERN IN .NET:

* Multiple indirections make your code less ceremonial, but it’s harder to reason your code

EF INTEROP:

* No observable consequences

SQL SERVER CONSIDERATIONS:

* Hard to read the code, and it will always be hard to reason the code
* No direct observable effect on query plan

### FUTURE COMPATIBILITY PROBLEMS

* Can imply incompatibility for other database beside SQL Server for the same stored procedure, because of SQL Server specifics
* Table used itself should not include currency typed column, otherwise it will not have cross database compatibility

### POSSIBLE SIDE EFFECTS:

* If the called stored procedure is inflicting error, then the stored procedure can’t be guaranteed safe at the first level

### APPLIES TO:

All database vendors

## RULE #6

AVOID have stored procedure that use deprecated table hint on Transact SQL queries

### CRITICAL LEVEL WHEN NOT APPLIED:

Critical

### REASON

PERFORMANCE:

* No observable consequences

DESIGN PATTERN IN .NET:

* Multiple indirections make your code less ceremonial, but it’s harder to reason your code

EF INTEROP:

* No observable consequences

SQL SERVER CONSIDERATIONS:

* Hard to read the code, and it will always be hard to reason the code
* Some keywords can’t have query plan/explained

### FUTURE COMPATIBILITY PROBLEMS

* Can’t use the code in future release of SQL Server

### POSSIBLE SIDE EFFECTS:

* Can be observable, depends on the hints used

### APPLIES TO:

All database vendors

### SAMPLES AND NOTES:

The use of NOLOCK in UPDATE and DELETE will not supported in future releases; see “Table Hints (Transact SQL)” also in online: <http://msdn.microsoft.com/en-us/library/ms187373.aspx>

The use of FASTFIRSTROW will be deprecated: <http://msdn.microsoft.com/en-us/library/ms187373.aspx>

## RULE #7

USE MERGE instead of INSERT.. SELECT or UPDATE.. SELECT

### CRITICAL LEVEL WHEN NOT APPLIED:

Critical

### REASON

PERFORMANCE:

* No observable consequences

DESIGN PATTERN IN .NET:

* Multiple indirections make your code less ceremonial, but it’s harder to reason your code

EF INTEROP:

* No observable consequences

SQL SERVER CONSIDERATIONS:

* Embracing ANSI SQL support (in ANSI SQL:2003)
* Only available in SQL Server 2008 and above (including SQL Server 2008 R2)

### FUTURE COMPATIBILITY PROBLEMS

* Can’t use the code in older release of SQL Server before SQL Server 2008, but this can be mitigated using INSERT..SELECT or UPDATE.. SELECT
* No future compatibility problems

### POSSIBLE SIDE EFFECTS:

* Always observable, this statement *will always return numbers of affected rows result* with side effects if there’s any constraint violations
* MERGE has the same side effect as above note

### APPLIES TO:

All database vendors

### SAMPLES AND NOTES:

None