

---



# Quality Assurance *For Dynamic SQL*

Sponsored by:



<http://www.cdbsoftware.com>



**Craig S. Mullins**  
Mullins Consulting, Inc.

<http://www.craigsmullins.com>



# Author

This presentation was prepared by:

**Craig S. Mullins**  
**President & Principal Consultant**

Mullins Consulting, Inc.  
15 Coventry Ct  
Sugar Land, TX 77479  
Tel: 281-494-6153  
Skype: cs.mullins  
E-mail: [craig@craigsmullins.com](mailto:craig@craigsmullins.com)  
Web: <http://www.CraigSMullins.com>



This document is protected under the copyright laws of the United States and other countries as an unpublished work. Any use or disclosure in whole or in part of this information without the express written permission of Mullins Consulting, Inc. is prohibited.

© 2011 Craig S. Mullins, Mullins Consulting, Inc. All rights reserved.

## About CDB Software

- Founded in 1985
- Focused on DB2 for z/OS
- Corporate Philosophy
  - CDB enables businesses to meet growing DB2 demands by improving data management processes with intelligent solutions
- High Speed Automated Utilities that keep your database in top condition while you focus on your applications



## CDB/Auto-Online Reorg

- High Speed Fully Online Reorg
  - Applications never leave R/W
  - Patented No-Sort Process
  - Handles any size objects
  - Full SQL Subselect capability to Purge Data
  - Utilizes Real Time Statistics without the need to ever generate JCL
    - Only objects that have to be reorged are processed
      - Never waste CPU reorganizing objects that don't need it
        - » Indexes or Tablespaces
  - Database is fully maintained using the least amount of resources (both CPU and Human) possible

# QA for Dynamic SQL -- Agenda

## Dynamic SQL – The Basics

- › Static vs. Dynamic SQL
- › When to Use Dynamic vs. Static SQL

## Drivers of Dynamic SQL Growth

- › Packaged Applications (e.g. ERP)
- › Modern Application Development
  - GUI, web, etc.

## Performance Issues

- › Understanding the Dynamic Statement Cache
- › BIND options

## Solving Your Dynamic SQL Woes with QA+



# Types of SQL

## Functionality

---

DCL	Control of data and security
DDL	Data definition
DML	Data manipulation

## Execution Type

---

Production	Planned
Ad hoc	Unplanned

## Existence

---

Embedded	Requires a program
Stand-alone	No program used

## Dynamism

---

Dynamic SQL	Changeable at run time
Static SQL	Unchangeable at run time



# Dynamic SQL – Basics

## Static SQL

- Is unique to DB2
- Access paths established before execution
- Creates a consistent access path
- Requires a BIND to create SQL access paths

## Dynamic SQL:

- Usually more flexible but access path can be inconsistent
- Not as common
  - At least not in traditional COBOL applications
  - Modern development uses dynamic SQL much more frequently, though
- A “mini-bind” is done automatically by DB2 before execution

# Dynamic SQL versus Static SQL

Capability	Dynamic SQL	Static SQL
Access paths created at:	Run time	BIND time <sup>1</sup>
Access paths in PLAN_TABLE		✓
BIND <u>not</u> required before execution	✓	
Dynamic Statement Cache	✓	
Uses latest RUNSTATS <sup>2</sup>	✓	
SQL errors detected at BIND		✓
Authorization to tables <u>not</u> required <sup>3</sup>		✓
Flexible: SQL can change	✓	
Supported in interpreted languages (e.g. REXX) <sup>4</sup>	✓	

# So to Summarize...

## With Static SQL:

- > **Beneficial** because Access paths are formulated at BIND time and EXECUTE authority on plans and packages eliminates need for privileges for all objects in the SQL
- > **But** binding before running program can become burdensome and many of current development tools provide better support for dynamic APIs



## With Dynamic SQL:

- > **Beneficial** because IDEs support Java better, the Dynamic Statement Cache can alleviate performance penalty of re-formulating access paths and dynamic SQL will always use the latest RUNSTATS (which may produce a better access paths)
- > **But** compiling statements each time they are executed increases total statement execution time and SQL errors may not be detected until the program is executed.

# Static vs. Dynamic SQL: When To Use Each

## > Performance sensitivity of the SQL statement

- Dynamic SQL will incur a higher initial cost per SQL statement due to the need to prepare the SQL before use. But once prepared, the difference in execution time for dynamic SQL compared to static SQL diminishes.

## > Use of range predicates

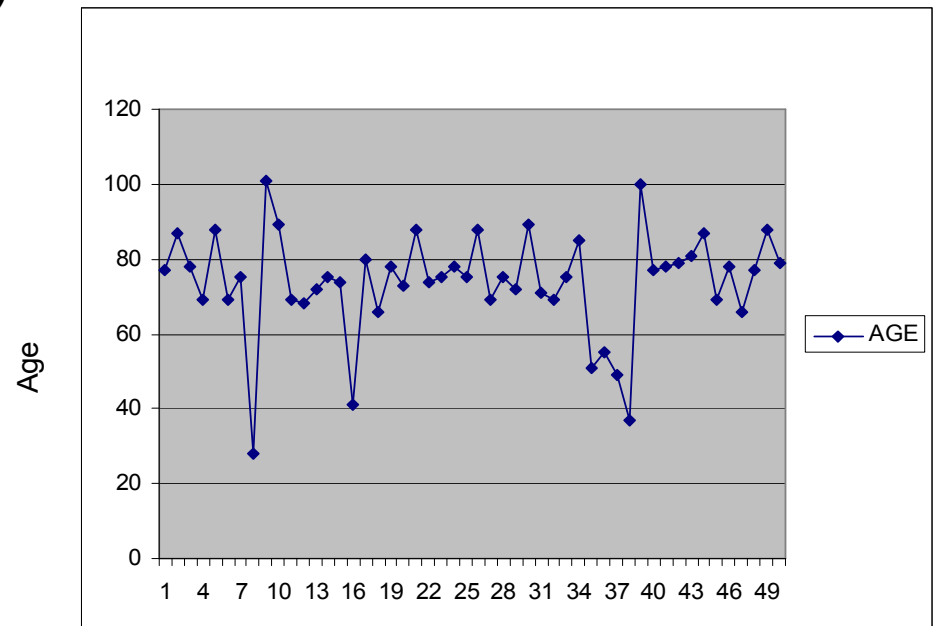
- The more frequently you need to use range predicates (<, >, <=, >=, BETWEEN, LIKE) the more you should favor dynamic SQL.
- The optimizer can take advantage of distribution statistics & histogram statistics to formulate better access paths because the actual range will be known.



# When to Use Static vs. Dynamic SQL (continued)

## > Data uniformity

- Dynamic SQL can result in more efficient access paths than static SQL is whenever data is:
  - Non-uniformly distributed.  
(e.g. cigar smokers skews male, mortality age skews older)
  - Correlated  
(e.g. CITY, STATE, and ZIP\_CODE data will be correlated)



# When to Use Static vs. Dynamic SQL (continued)

## > Repetitious Execution

- As the frequency of execution increases, then you should favor static SQL (or perhaps dynamic SQL with local dynamic statement caching (KEEPDYNAMIC YES)).
- The cost of the PREPARE becomes a smaller and smaller percentage of the overall run time of the statement the more frequently it runs (if the cached prepare is reused).



## > Nature of Query

- When you need all or part of the SQL statement to be generated during application execution favor dynamic over static SQL.

## When to Use Static vs. Dynamic SQL (continued)

### > **Run Time Environment**

- Dynamic SQL can be the answer when you need to build an application where the database objects may not exist at precompile time. Dynamic might be a better option than static specifying `VALIDATE(RUN)`.



### > **Frequency of RUNSTATS**

- When your application needs to access data that changes frequently and dramatically, it makes sense to consider dynamic SQL.

# Drivers of Dynamic SQL Growth

## Packaged applications (COTS) use dynamic SQL

- SAP R/3, Peoplesoft, Siebel, CSC/Hogan, etc.
- Easier to support multiple DBMSes that way



## Many newer applications use dynamic SQL

- Developed on distributed platforms and for the web
  - New developers are more familiar with GUI-based programming environments
  - Many of the current development tools provide better support for dynamic APIs (like JDBC), than they do for static SQL
  - Many developers never even sign on to the mainframe
    - Java and .NET developers
  - Some developers never write SQL, but have it generated for them by a framework (e.g. Hibernate)



# Let's Talk About Dynamic SQL Performance

## Many aspects of tuning dynamic SQL are similar to tuning static SQL

- Find the problematic SQL
- Determine who executed it and from which program
  - if possible
- EXPLAIN the statement
- Tune the statement
- Repeat as needed

*We'll speak about each of these on the ensuing slides.*

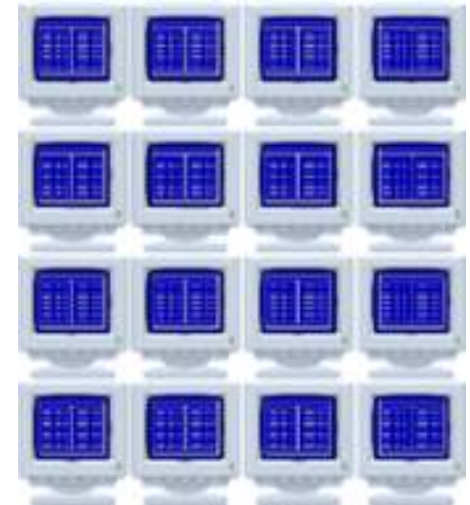
## Building best practices for tuning dynamic SQL

# Finding the Problematic Dynamic SQL

**There are no DBRMs, packages, or access paths in PLAN\_TABLE(s) to utilize.**

**So, how do you find problems?**

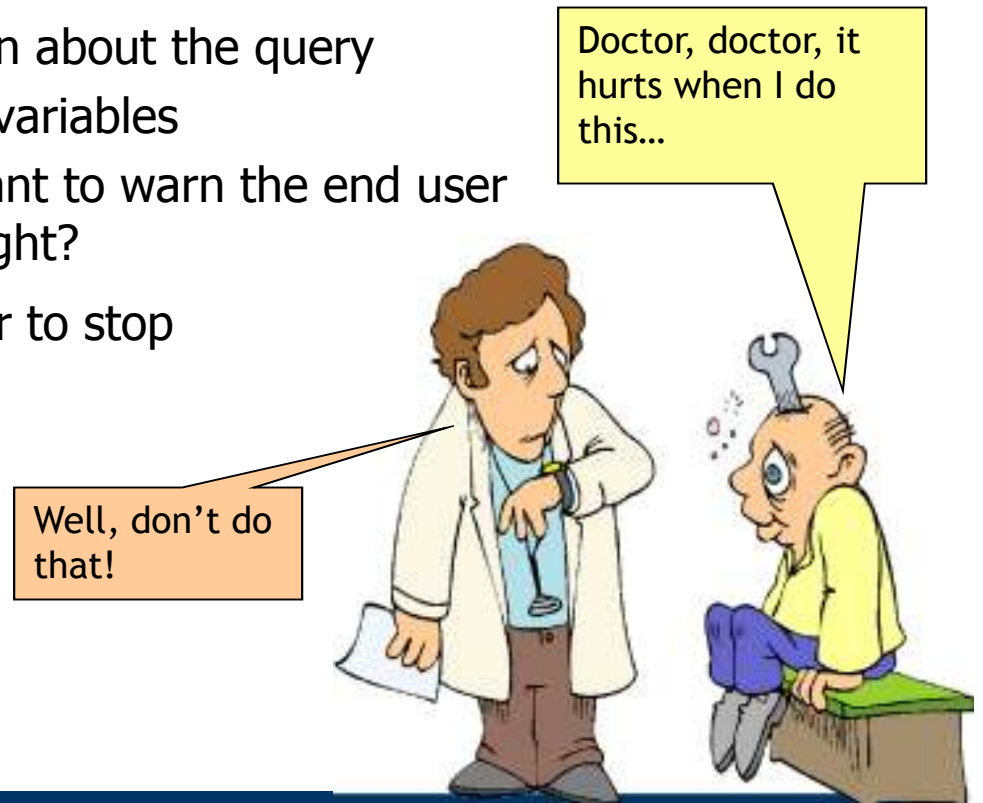
- > A problem usually starts with a phone call
- > Could also arise from performance reports and/or performance monitor(s)
  - Typically after a problem occurs, though
  - May require costly traces



# Who Executed the Dynamic SQL?

## Can you determine *who* executed the *problem* dynamic SQL statement(s)?

- Why would you want/need to know this?
  - Can help to gather information about the query parameter markers and host variables
  - If it is still running you will want to warn the end user before you kill the thread... right?
  - Remediate by getting the user to stop running that particular query ...or form of the query.



## Who Executed the Dynamic SQL? (continued)

### But it can be difficult to determine *who* executed dynamic SQL

- > Why?
  - Much dynamic SQL comes from an application server
  - Single connection ID problem
- > There are four client identifiers that *can* be passed to DB2 for z/OS when dynamic SQL is executed:
  - Client workstation name
  - Client application name
  - Client user
  - Client accounting information.



These can be set using DRDA or JDBC...

# Tuning/Fixing Dynamic SQL Statements

## Take advantage of information in the Dynamic Stmt Cache

- We will cover the DSC in more detail momentarily

## Create a statement cache table

- Run EXPLAIN STMTCACHE STMTID
  - Needs SYSADM authority

## Apply traditional SQL tuning best practices

- If you are able to modify the SQL

## What if you cannot modify the SQL?

- Add or modify indexes
- Refresh statistics by executing RUNSTATS
- Reorganize table spaces and/or indexes
- Lobby your application vendors to change the SQL



# An Additional Dynamic SQL Tuning Thought

## Consider stored procedures for frequently executed dynamic SQL statements

- Stored procedures can reduce network traffic
- Native SQL stored procedures are relatively easy to code and are more efficient than earlier types of stored procedures

### Caveats:

- Requires modification of the application
  - To call the stored procedure instead of issuing the dynamic SQL
- Few tools to manage DB2 stored procedures
- Requires mainframe skills
  - May not be helpful if all programmers use an IDE

# Dynamic SQL Best Practices

## Evaluate access paths before moving to production

- › Dynamic EXPLAIN of most common statements/varieties

## Threshold and parameter-controlled alerting

- › Tag “problem” SQL and watch over time
- › Compare past/average versus current

## Retain historical details

- › Easy identification of poor performing SQL
- › Average CPU and elapsed times for each SQL
- › Group top x SQL statements for tuning
  - Concentrate on biggest consumers for biggest return on tuning investment
- › Identify trends



# Introduction to Dynamic Statement Caching (DSC)

## DB2 uses the DSC to minimize (perhaps even eliminate) dynamic prepares

- There are three (*OK, four*) types of caching supported:
  - No caching
  - Local Dynamic Statement Caching
  - Global Dynamic Statement Caching
  - Full Caching
- The prepared SQL and statement text for dynamic SQL statements are cached in the DBM1 address space
  - Local Statement Cache
  - Global Dynamic Statement Cache

# Implementing Dynamic Statement Caching

## Controlled by several different parameters

- > BIND options
- > DSNZPARMs
- > Application constructs

**Will discuss the parameters on the upcoming slides in the appropriate place**



# No Caching

## This is the default

- › And the way DB2 worked before the DSC was introduced

## Prepared statements do not persist across COMMIT

- › The prepared mini-plan is discarded after a commit
  - Except for CURSOR with HOLD statements
- › The next time the statement is executed it will have to be prepared again



# Local Dynamic Statement Caching

## Eliminates need for the application to issue multiple PREPAREs for the same statement

- › Implicit PREPAREs are done by DB2

## To enable Local Statement Caching

- › Set KEEP DYNAMIC(YES) Bind Parameter
- › The MAXKEEPD DSNZPARM controls maximum prepared statements
  - Does not affect statement text which is always kept

## Not really a big performance help

- › Useful because programmers do not need to keep track of when COMMITs are issued in order to perform another PREPARE
  - DB2 does the implicit PREPARE
- › Some reduction in message traffic in a distributed environment is possible

# Global Dynamic Statement Caching

## Will reuse prepared statements across units of work

- › Within and across program executions
- › Must be the EXACT same SQL statement, though
- › Prepared statement cached in global dynamic statement cache
  - Skeleton Dynamic Statement (SKDS)
  - Short Prepare

## To enable Global Statement Caching

- › Set DSNZPARM CACHEDYN=YES

## Global Dynamic Statement Caching can offer significant performance improvement for applications with frequent reuse of dynamic SQL

- › No coding changes required

# Prerequisites for Global Dynamic Caching

## Statement text must be **EXACTLY** the same

- › Can use parameter markers
- › Literals will not work (unless it is always the SAME literal)<sup>1</sup>

## Other things that must be the same or compatible

- › Bind rules
- › Special registers
- › Authorizations

**Packaged application vendors (e.g. ERP) have designed their applications to make use of DSC**

<sup>1</sup> DB2 V10 changes this where literals can be treated like variables

# SQL Has to be "The Same"

## Are These Statements The Same?

```
SELECT COL1, COL2
FROM TEST_TABLE
WHERE COL3 = ?
AND COL4 = ?
```

```
SELECT COL1, COL2
FROM TEST_TABLE
WHERE COL3 = ?
AND COL4 = ?
```

## How About These?

```
SELECT COL1, COL2
FROM TEST_TABLE
WHERE COL3 = ? AND COL4 = ?
```

```
SELECT COL1,
       COL2
FROM TEST_TABLE
WHERE COL3 = ?
AND COL4 = ?
```

## Literals can be treated as “the same” in DB2 10

- As we just learned, in order to take advantage of the DSC, the dynamic SQL statement must be exactly the same
- If a literal changes, it is not the same. For example:

### NOT THE SAME

```
SELECT NAME, ADDRESS  
FROM CUST  
WHERE CUSTNO = 1234;
```

```
SELECT NAME, ADDRESS  
FROM CUST  
WHERE CUSTNO = 5678
```

- As of DB2 V10, dynamic SQL with literals can be reused in the DSC
  - It is still generally better to use parameter markers for dynamic SQL than to use literals and rely on this update though

## Full Caching *(Both Local and Global)*

### Combines the benefits of Local and Global Dynamic Statement Caching

- Can completely avoid PREPARE operations
- Prepared statement kept in local thread storage and not invalidated across commits
  - Prepare Avoidance

### Enabling global statement caching

- CACHEDYN=YES
- MAXKEEPD>0
- KEEP DYNAMIC(YES)



# Flushing the DSC

## Invalidate the dynamic statement cache by:

### 1. Executing RUNSTATS

### 2. RUNSTATS UPDATE NONE REPORT NO

- Causes any statement in the DSC which is dependent on the affected table space or index space to be removed from the cache.
- This is done to allow users who manually update DB2 Catalog statistics to invalidate the related dynamic SQL in the cache
  - The next prepare will re-evaluate the access paths.

## The granularity is at the table space and index level

- Not the table level

# Reoptimization (REOPT)

**Reoptimization settings can make dynamic SQL more static ...and sometimes vice versa.**

**You can gain additional optimization for (mostly dynamic) SQL using the REOPT parameter of the BIND command.**

**REOPT specifies whether to have DB2 determine an access path at run time by using the values of host variables, parameter markers, and special registers.**

**As of DB2 9, there are four options from which to choose when specifying REOPT.**



# REOPT Parameter Choices

NOREOPT(VARS) can be specified as a synonym of REOPT(NONE).

DB2 V8

REOPT(VARS) can be specified as a synonym of REOPT(ALWAYS).

DB2 9

**REOPT (NONE)** –PREPARE determines the access path and no reoptimization is performed. The bound statement can be moved to the dynamic statement cache (DSC), if the cache is being used.

**REOPT (ONCE)** – PREPARE determines an initial access path before the host variable values are available. When the statement is first executed and the host variable values are known, the statement is reoptimized one time. The hope is that the one-time reoptimization will provide a better access path than the initial PREPARE. The statement can be placed in the dynamic statement cache and reused multiple times.

**REOPT (ALWAYS)** – The SQL statement is re-optimized each time it is executed, always using the latest host variable values.

**REOPT (AUTO)** – Leave it up to DB2 (autonomic?). If changes in the filter factors for the statement predicates warrant, DB2 can re-prepare the statement. The newly prepared statement would be executed and would replace the prepared statement currently in the Global Dynamic Statement cache.

# REOPT Applicability: Dynamic vs. Static SQL

REOPT Parameter	Dynamic SQL	Static SQL
NONE	YES	YES
ALWAYS	YES	YES
ONCE	YES	NO
AUTO	YES	NO

Consider binding static programs with REOPT(ALWAYS) when the values for your program's host variables or special registers are volatile and make a significant difference for access paths.

ONCE and AUTO are not valid for static SQL because they work with the dynamic statement cache, which does not apply to static SQL.

# Dynamic SQL Best Practices

## Understand the difference between static & dynamic

- › Use appropriate parameters for each

## Implement Dynamic Statement Cache

- › Train programmers how to write code that benefits from DSC

## Build threshold and parameter-controlled alerts

- › Tag “problem” SQL and watch over time
- › Compare past/average versus current

## Retain historical details

- › Easy identification of poor performing SQL
- › Average CPU and elapsed times for each SQL
- › Group “Top  $x$ ” SQL statements for tuning
  - Concentrate on biggest consumers for biggest return on tuning investment
- › Identify trends

# CDB Helps to Resolve Your Dynamic SQL Woes



## DB/IQ - Family

### DB/IQ Family in Version 4.92

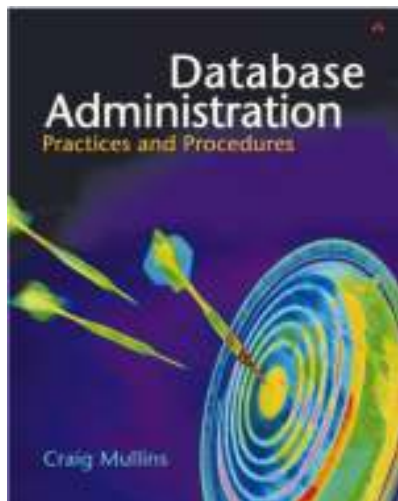
- ✉ QA      Quality Assurance (Base Product)
- ✉ IA+     Index Administrator + (Add-On)
- ✉ PM      PackMan - Package Management
- ✉ QA+     QA Plus      (Add-On)
- ✉ WL+     WorkLoad Detector + (Add-On)

# Contact Information



**Phone: 281-920-3305**  
**[cdbsales@cdbsoftware.com](mailto:cdbsales@cdbsoftware.com)**

**<http://www.cdbsoftware.com>**



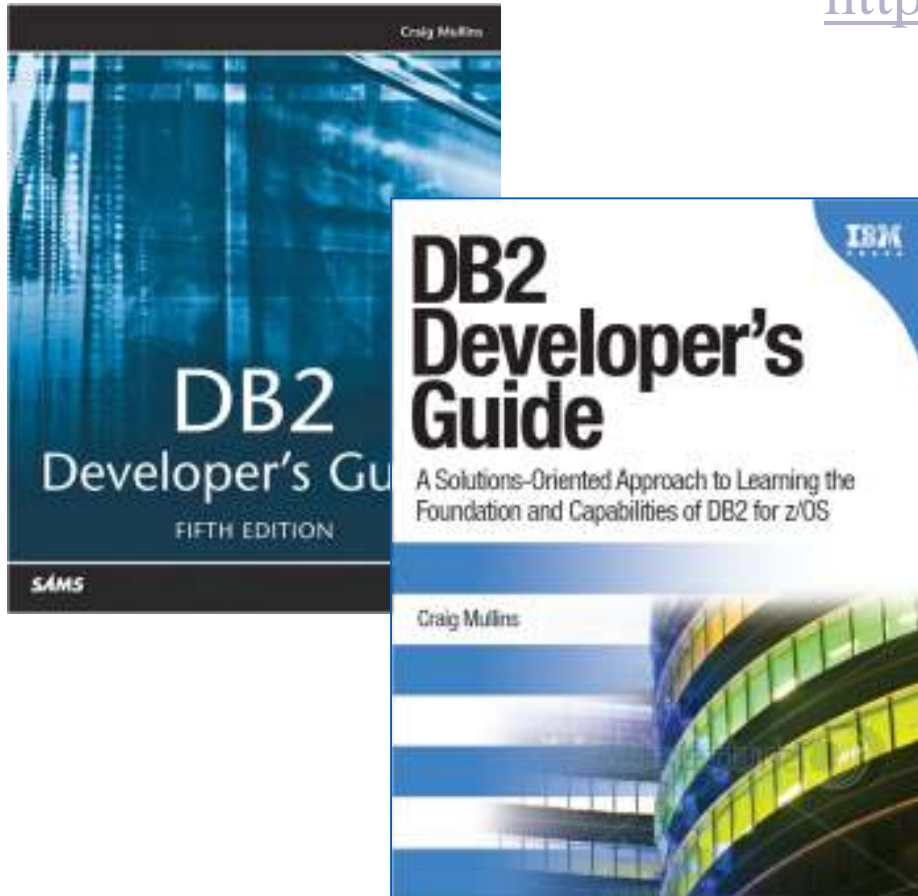
**Questions?**  
**[info@cdbsoftware.com](mailto:info@cdbsoftware.com)**

**Craig S. Mullins**  
***Mullins Consulting, Inc.***  
**15 Coventry Court**  
**Sugar Land, TX 77479**  
**[craig@craigsmullins.com](mailto:craig@craigsmullins.com)**  
**<http://www.craigsmullins.com>**

**<http://www.craigsmullins.com/cm-book.htm>**

# DB2 Developer's Guide, 6<sup>th</sup> edition

<http://www.craigsmullins.com>



← 6<sup>th</sup> edition coming soon...  
covering up thru DB2 V10!