Let Me Make This Clear (Things That Plenty of DB2 for z/OS People Get Wrong)

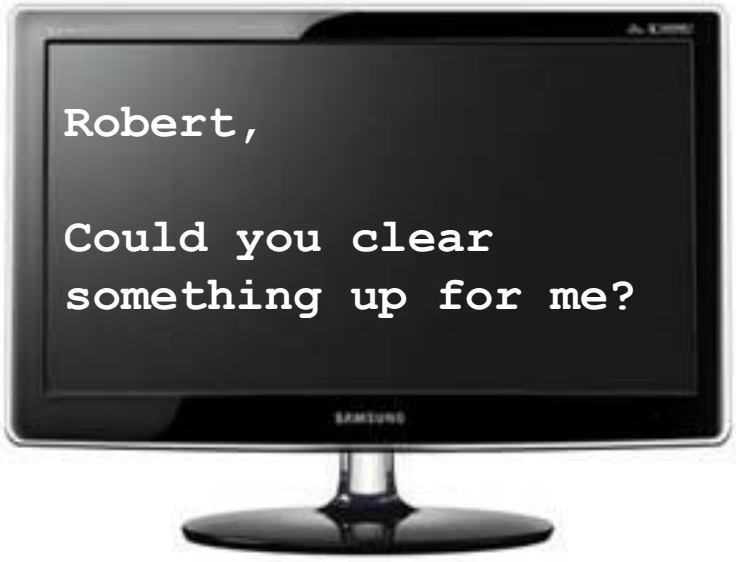
Robert Catterall
IBM

Session Code: A15
May 26, 2016 9:15 AM | Platform: DB2 for z/OS

Photo by Steve from Austin, TX, USA

Introduction

- In the course of my work, I get a lot of questions from a lot of DB2 for z/OS people



- Some of these questions suggest widespread misunderstanding of certain DB2 features and functions
- In this presentation, I'll highlight some of these misunderstandings and provide (I hope) some clarity
  - I'll highlight misunderstandings in *dark red italics*

Agenda

- zIIP offload and native SQL procedures
- zIIP offload and dynamic SQL
- Selective query parallelism
- Java stored procedures
- High-performance DBATs
- RELEASE(DEALLOCATE) “break-in”
- Buffer pool monitoring
- Group buffer pool monitoring
- Page-fixed buffer pools and 1 MB page frames
- Inactive DBATs versus inactive connections
- Partition-by-growth and smaller tables
- Dynamic versus ad-hoc SQL
- DB2 address space prioritization
- DB2-managed archiving versus system-time temporal
- The IBM Data Server Driver versus DB2 Connect
- DB2 Connect versus z/OS Connect

zIIP offload and native SQL procedures

- *A lot of people are under the impression that native SQL procedure execution is always zIIP-eligible*
- In fact, native SQL procedure execution is only zIIP-eligible when the caller is a DRDA requester – in other words, when the CALL comes through DDF
  - Why? Because zIIP eligibility depends on a process running under a preemptible SRB versus a TCB or a non-preemptible SRB
  - A native SQL procedure runs under the task of the process that called it, and when the caller is a DRDA requester, that task is a preemptible SRB in the DB2 DDF address space – that makes the native SQL procedure zIIP-eligible

More on native SQL procedure zIIP eligibility

- When a native SQL procedure is called by a process that runs under a TCB (e.g., a CICS transaction or a batch job), it will run under that TCB and so will not be zIIP-eligible
- Question: if a DRDA requester calls an external DB2 stored procedure, and that stored procedure calls a native SQL procedure, will the native SQL procedure's execution be zIIP-eligible?
  - Answer: NO, because an external stored procedure always runs under its own TCB in a stored procedure address space, and the nested native SQL procedure will run under that TCB and so will not be zIIP-eligible

Note: when an external stored procedure is called by a DRDA requester, you will see a little zIIP offload, because associated send/receive processing is done under preemptible SRB in DDF

Still on the topic of native SQL procedures and zIIPs

- *Some people think that native SQL procedures, when they are zIIP-eligible, are 100% zIIP-offload-able*
- Nope – when a native SQL procedure is zIIP-eligible (i.e., when it is called by a DRDA requester), it will be up to 60% zIIP-offload-able
  - Why? Because SQL statements that execute under preemptible SRBs in the DB2 DDF address space are up to 60% zIIP-offload-able, and a native SQL procedure is just SQL
  - An implication: going from SQL DML statements issued directly from DRDA clients to packaging SQL DML in native SQL procedures is not a way to boost zIIP offload, since the SQL is up to 60% zIIP-offload-able either way
    - You can boost zIIP offload when you change external stored procedures called by DRDA requesters to native SQL procedures

zIIP offload-o-meter

zIIP offload and dynamic SQL

- *Some people have this idea that there's something about dynamic SQL that affects zIIP offload-ability*
- The zIIP eligibility of a SQL statement – whether dynamic or static, depends on the type of task under which the statement executes
  - Preemptible SRB: zIIP-eligible
  - TCB: not zIIP-eligible
- Note: SQL statements issued by DRDA requesters (or by native SQL procedures called by DRDA requesters) aren't the only ones that run under preemptible SRBs
  - When a query – static or dynamic – is parallelized by DB2, the "pieces" of the spilt query run under preemptible SRBs and are up to 80% zIIP-offload-able

"Look the same to me"

Dynamic vs. Static

Selective query parallelism

- Some people are interested in DB2 query parallelism as a means of getting zIIP offload for processes, such as batch jobs, that have "local" (i.e., not through DDF) connections to DB2, but…
  - *…they think that there is no good option for granular control of query parallelism*
    - *Think that, for dynamic SQL, either ALL queries are made candidates for parallelization via specification of CDSSRDEF=ANY in ZPARM, or queries are selectively made candidates for parallelism via SET CURRENT DEGREE = 'ANY'*
    - *Think that, for static SQL, only option is bind of package with DEGREE(ANY)*
    - *Think that, for all queries, maximum degree of parallelization is whatever is specified for PARAMDEG in ZPARM*
- What these people don't know about is the SYSQUERYOPTS table in the DB2 catalog

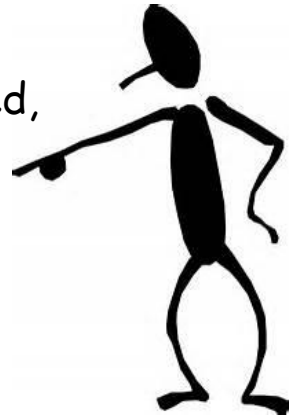
More on SYSIBM.SYSQUERYOPTS

- Introduced with DB2 10 for z/OS in new-function mode
- Used in conjunction with the BIND QUERY command and the SYSIBM.SYSQUERY table
- Lets you specify that a specific query (static or dynamic) is to be a candidate for parallelization, along with a maximum degree of parallelization for that specific query – with NO code changes needed
- Here's a blog entry with more details:

  http://robertsdb2blog.blogspot.com/2016/02/statement-level-control-of-db2-for-zos.html

"OK, static query ABC can be parallelized, with a maximum degree of 4.

Dynamic query XYZ can be parallelized, with a maximum degree of 8."



Java stored procedures

- I've seen 2 misunderstandings pertaining to Java stored procedures:
  - *They're a bad idea: they perform poorly and are CPU and memory hogs*
  - *They are 100% zIIP-offload-able*
- The "poor-performing, resource hog" view likely has roots in the situation of not-too-many years ago, which has changed
  - z/OS is now a great Java environment: 1200% performance improvement from Java 5 on a z9 mainframe to Java 7 on an EC12
    - Even better performance with Java 8 on a z13, thanks to features such as SIMD (Single Instruction Multiple Data) and SMT (Simultaneous Multi-Threading)
  - And, Java doesn't "hog" memory – it exploits large memory resources (as does DB2 for z/OS), and z Systems memory gets cheaper all the time
  - And, DB2 11 delivered important enhancements for Java stored procedures
    - One 64-bit multi-threaded JVM per Java stored procedure address space, versus a single-threaded 31-bit JVM per Java stored procedure in an address space

Java stored procedures and zIIP eligibility

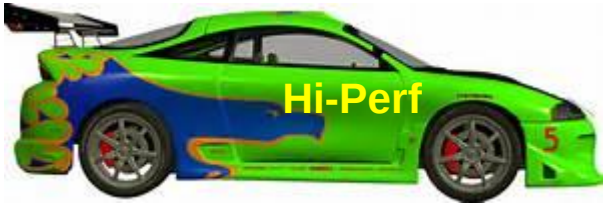
- No, they are not 100% zIIP-eligible
  - Yes, Java code execution in a z/OS system is zIIP-eligible, but SQL is not Java, so SQL statements issued by a Java stored procedure are not zIIP-eligible
    - Recall that SQL statements are zIIP-eligible when they execute under a preemptible SRB – SQL statements issued by a Java stored procedure execute under the TCB of the Java stored procedure
    - In truth, you would likely get a small amount of zIIP offload for SQL statements issued by a Java stored procedure, because the zIIP engine used to execute the procedure's Java code is "held on to" for a little while when SQL starts executing

zIIP offload-o-meter

High-performance DBATs

- Some people think, *"We can't use high-performance DBATs, because we wouldn't be able to get ALTERs and BIND/REBIND stuff done"*
- It is true that any combination of persistent threads (i.e., threads that persist through COMMITs) and RELEASE(DEALLOCATE) packages can interfere with ALTER, BIND/REBIND, and more, but…
  - …you have to keep in mind that these operations might be specifically blocked by high-performance DBATs, as opposed to being generally blocked
    - For example, if package PKG1 is bound with RELEASE(DEALLOCATE) and is allocated to a high-performance DBAT, and the package is not dependent on table T1, an ALTER of T1 will not be blocked because of package PKG1
  - If you determine that an ALTER (or BIND/REBIND, or online REORG that would materialize a pending DDL change) would be blocked by a high-performance DBAT, use command to temporarily "turn off" high-performance DBATs:

```
-MODIFY DDF PKGREL(COMMIT)
```



More on RELEASE(DEALLOCATE) and concurrency

- *Some think they can't use RELEASE(DEALLOCATE) packages at all – not with high-performance DBATs, not with anything – because they will cause concurrency problems*
- First, get the concurrency facts straight
  - Does RELEASE(DEALLOCATE) cause page or row locks to be retained until thread deallocation?
    - NO – X locks on pages and rows are always released at COMMIT; S locks are typically released when DB2 moves to the next page or row
  - Table space locks are held longer with RELEASE(DEALLOCATE) – is that a problem?
    - Generally speaking, NO, because table space locks are almost always of the intent variety (e.g., IX, IS), and intent locks do not interfere with each other
    - DB2 utilities have long been able to "break in" on RELEASE(DEALLOCATE) packages, by way of drain locking (claims are always released at COMMIT)

DB2 11 and RELEASE(DEALLOCATE) "break in"

- The real concurrency concern has been related to packages
  - A package cannot be replaced or invalidated when it is in use
  - A RELEASE(DEALLOCATE) package is considered to be continuously in-use as long as the thread to which it is allocated exists
  - That being the case, it used to be that any operation that would replace or invalidate package XYZ would fail if package XYZ were bound with RELEASE(DEALLOCATE) and allocated to a persistent thread
    - Failing operation could be a BIND/REBIND, an ALTER, or an online REORG that would materialize a pending DDL change
- This changed with DB2 11 (in new-function mode)
  - If a BIND/REBIND, ALTER or pending DDL-materializing online REORG would be blocked by a RELEASE(DEALLOCATE) package allocated to a persistent thread, DB2 can "break in" to let blocked operation proceed
    - Package's behavior will be temporarily changed to RELEASE(COMMIT)

A little more on RELEASE(DEALLOCATE) "break-in"

- *Some people think that this is only important for RELEASE(DEALLOCATE) packages*

  - Not so – in addition to switching package behavior to RELEASE(COMMIT), the new DB2 11 functionality will "drain" package to get its use count to 0
  - That provides relief from blockages caused by RELEASE(COMMIT) packages that would otherwise be "always in use" due to frequency of execution
- *Some folks think that this applies to all kinds of persistent threads*
  - They're actually half right
    - RELEASE(DEALLOCATE) "break-in," as it pertains to "in-DB2" threads, does apply to all types of persistent threads, high-performance DBATs included – the package's behavior will be changed to RELEASE(COMMIT) at next commit
    - That said, "break-in," as it pertains to threads that are not processing in DB2 but have RELEASE(DEALLOCATE) packages allocated to them, is something that does NOT apply to high-performance DBATs
    - So, even with DB2 11, it's best to issue -MODIFY DDF PKGREL(COMMIT) to clear out high-performance DBATs as needed for DBA tasks

Buffer pool monitoring

- *Lots of people think that the "hit ratio" is the key metric when it comes to buffer pool monitoring and tuning*
- As far as I'm concerned, the hit ratio is of very little value

- I'd much rather focus on a buffer pool's total read I/O rate
  - That's total synchronous reads plus total prefetch reads (sequential, list, dynamic) for a buffer pool, per second
    - Can get these numbers from DB2 monitor statistics long report or online display of buffer pool activity, or from DB2 command -DISPLAY BUFFERPOOL DETAIL
  - What's your objective for a buffer pool?
    - Total read I/O rate < 1000 per second is good, < 100 per second is great
    - Of course, for a buffer pool used to "pin" one or more objects (i.e., cache them in memory in their entirety), your objective is a total read I/O rate of zero

Group buffer pool monitoring

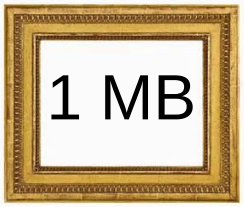
- *Some think that the only metrics that matter are the "double zeroes" (0 directory entry reclaims, 0 write failures due to lack of storage)*
- These are indeed important, but another valuable metric is often overlooked: the "XI read hit ratio" ← This is a hit ratio that does matter to me
  - That's the percentage of the time that synchronous read requests directed to a GBP because of local buffer cross invalidation (XI) resulted in "page found"
    - (sync reads due to XI, data returned) / (total sync reads due to XI)
    - Numbers can be found in DB2 monitor statistics long report or online display of GBP activity, or via DB2 command -DISPLAY GROUPBUFFERPOOL MDETAIL
  - Buffer invalidations happen when directory entry reclaims occur, and when a page cached locally by DB2 member X is changed on member Y
    - If there are no directory entry reclaims, buffer invalidations must be due to pages being changed on other members of the data sharing group
    - If a page was changed on another DB2 member, it had to have been written to the GBP – when you go to the GBP looking for that page, you're hoping it's still there

More on the GBP XI read hit ratio

- The more data entries there are in a GBP, the longer pages written to the GBP will stay there, and the higher the XI read hit ratio will go
  - I've often seen XI read hit ratios above 80%, even above 90%
  - GBP XI read hits are good, because a GBP read will generally be two orders of magnitude faster than a disk read
- If ALLOWAUTOALT(YES) is specified for a GBP in the CFRM policy, check the GBP's ratio of directory entries to data entries
  - Default ratio is 5:1
  - I've seen ratios in excess of 250:1 with ALLOWAUTOALT(YES) in effect
  - If you see a super-high ratio of directory entries to data entries for a GBP, one effect may be a low XI read hit ratio
  - If that's the case, consider enlarging the GBP (given sufficient CF memory) and changing the ratio of directory to data entries to something closer to 5:1
  - Low XI read hit ratio no big deal if few GBP reads due to XI (e.g., < 1000/hour)

Page-fixed buffer pools and 1 MB page frames

- *Some people think that page-fixing buffers is only good for buffer pools that have a high read I/O rate*
- It is good for such pools (because they make I/Os cheaper), but it is also good for high-activity pools, IF the buffers can reside in 1 MB real storage page frames (true even if pool has low read I/O rate)
  - I'd say that a pool with more than 1000 GETPAGEs/second is "high activity"
  - Availability of 1 MB page frames depends on the value of the LFAREA parameter in the IEASYSxx member of SYS1.PARMLIB
  - When a buffer pool is defined with PGFIX(YES), DB2 will automatically seek to have the pool backed by 1 MB page frames
  - Why 1 MB page frames are good for high-activity pools: they make translation of virtual storage to real storage addresses more CPU-efficient



Inactive DBATs versus inactive connections

- *LOTS of people think that DBATs go "inactive" when they are not being used – the DB2 documentation even refers to inactive DBATs*
- In fact, DBATs do not go inactive
  - It's connections that go inactive when they are not in use
  - When a transaction using a "regular" DBAT (as opposed to a high-performance DBAT) completes, the DBAT goes into a disconnected – not an inactive – state
  - What's important: a disconnected DBAT (a DBAT in the DBAT pool) takes up a thread "slot" – it counts towards the MAXDBAT limit

"I'm not inactive – I'm disconnected"

Partition-by-growth and smaller tables

- *Some people think that partition-by-growth table spaces are only appropriate for large tables*
- Not so
  - People under this impression may be influenced by the word "partition," which traditionally (before universal table spaces) was associated with large tables
  - A PBG table space's DSSIZE (smallest value is 1 GB) is the space-used value that triggers allocation of an additional partition for the table space
  - A small table won't grow to the DSSIZE value, so the PBG will be a one-partition table space
  - The DSSIZE value doesn't determine disk space utilization – that's determined by amount of data in table, PRIQTY, and SECQTY

OK  PBG

PBG  Also OK

Dynamic versus ad-hoc SQL

- *Some DB2 people use the terms interchangeably, and end up opposing applications that will issue dynamic SQL because that's equated with ad-hoc SQL*
  - Result: developers can get the impression that their applications are not wanted on the z Systems platform (not good)
- Yes, ad-hoc SQL is dynamic, but the reverse is not necessarily true
  - Consider a Java application that would access DB2 data via JDBC calls
    - That'll be dynamic SQL on the DB2 side, but more than likely the queries are hard-coded in the Java programs – not ad-hoc
  - Keep in mind that "static" SQL is a DB2 concept – many current developers who don't have a mainframe heritage are not familiar with this concept
  - Bottom line: don't paint all dynamic SQL with the same brush
    - Most important: developers should know that their applications are welcome on the DB2 for z/OS platform

DB2 address space prioritization

- *At plenty of sites, one or more DB2 address spaces are given a too-low priority in the z/OS system's WLM policy*
  - Result is degraded throughput for DB2-accessing applications
- First of all, IRLM should be assigned to the super-high-priority SYSSTC service class
  - When IRLM has work to do, it needs a processor right away; otherwise, lock acquisition and release is delayed
  - IRLM uses very little CPU, so no worries about it getting in the way of other address spaces if it has a higher priority than those other address spaces
- Should any other DB2 address spaces be assigned to SYSSTC?
  - I say, "No" – remember what "Syndrome" said in "The Incredibles?"

"When everyone is super, no one will be"

Prioritizing DB2 address spaces other than IRLM

- DIST and any stored procedure address spaces should have the same priority as MSTR and DBM1, and that should be below SYSSTC but a little higher than CICS AORs (or IMS message regions)
  - *Some people give these DB2 address spaces a priority below CICS AORs, fearing that the DB2 address spaces will block CICS access to processors*
    - In fact, if DB2 tasks wait behind CICS tasks for processor time, CICS-DB2 transaction performance will suffer (CICS monitor will show higher "wait for DB2" times)
  - *Some people give the DIST address space (DDF) a lower priority than other DB2 address spaces, fearing that a higher priority will be too high for SQL coming through DDF*
    - In fact, the priority of the DIST address space applies only to the DDF system tasks, and these use very little CPU
    - The priority of SQL statements coming through DDF depends on the service class (or classes) to which DDF-using applications are mapped in the WLM policy – if they are not mapped to a service class, they get discretionary priority by default

Very low

Prioritizing DB2 address spaces other than IRLM (cont'd)

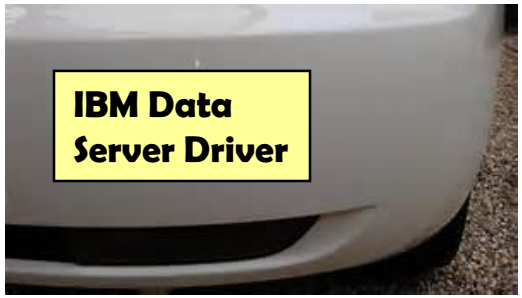
- *At some sites, DB2 stored procedure address spaces are given a lower priority than other DB2 address spaces, because people don't want DB2 stored procedures executing at a too-high priority*
  - In fact, the priority at which a DB2 stored procedure executes is inherited from the process that calls the stored procedure
  - If a stored procedure address space has a too-low priority, that can result in delays in scheduling called stored procedure for execution (that, in turn, negatively impacts the performance of the callers of stored procedures)
  - Note: native SQL procedures, like external stored procedures, run at the priority of the calling process, but they execute in DBM1 under the caller's task

DB2-managed archiving vs. system-time temporal

- *Some people get DB2-managed archiving (introduced with DB2 11) and system-time temporal support (DB2 10) mixed up*
- That's understandable – both are forms of archiving, if what you mean by "archiving" is long-term retention of historical data
  - With system-time temporal, a base table has an associated history table, and the history table holds the "before" image of rows that were made non-current via UPDATE and DELETE operations
    - Implemented to enable viewing of data in a "current as of (some date)" fashion
  - With DB2-managed archiving, a base table has an associated archive table, and the archive table holds rows that are current (i.e., still in effect) but relatively old and relatively infrequently accessed
    - Implemented to improve performance of retrieval of "newer" rows
  - For both features, DB2 can make the base and "adjunct" tables appear to programs to be one logical table

The IBM Data Server Driver vs. DB2 Connect

- *Some people think that DB2 Connect gateway servers are the way to go*
- Those people may think that way because they don't know about a better alternative: the IBM Data Server Driver
  - Simplified IT infrastructure, better performance
    - Eliminates a "hop" between application server and DB2 for z/OS, as IBM Data Server Driver is of the "type 4" variety – straight to DB2 from the app server
  - Lighter weight client, easier to configure and upgrade versus DB2 Connect
- How can you get the IBM Data Server Driver?
  - Easy: if you're licensed for DB2 Connect, you are entitled to use the Data Server Driver
    - Exception: DB2 Connect concurrent user license requires use of DB2 Connect gateway server



A little more on the IBM Data Server Driver

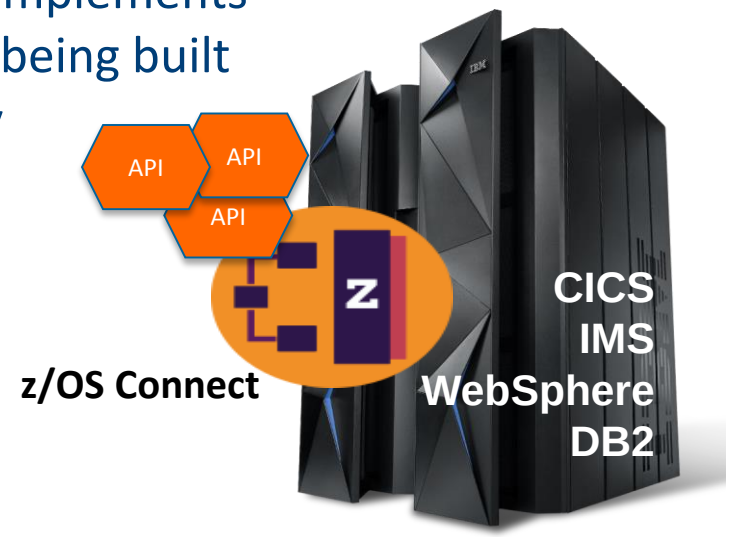
- Functionality-wise, just about everything in DB2 Connect is also provided by the IBM Data Server Driver – for example:
  - Connection pooling
  - Transaction pooling
  - Sysplex workload balancing
  - Drivers for lots of languages (not just Java and C# .NET, but others, too, including Perl, Python, PHP, Ruby…)
- About the only exception of which I'm aware: if an application requires 2-phase commit capability and the application server uses a dual-transport processing model, DB2 Connect is needed
  - WebSphere Application Server, among others, uses a single-transport processing model

DB2 Connect versus z/OS Connect

- *Some people are unclear as to the difference between DB2 Connect (and the IBM Data Server Driver) and z/OS Connect*
- DB2 Connect (and the Data Server Driver) continue to do what they have long done:
  - They enable applications to access DB2 for z/OS data over a network connection, using industry-standard relational database interfaces such as JDBC and ODBC
    - In other words, they enable these applications to be DRDA requesters
- z/OS Connect, newer on the scene, allows client applications to invoke z/OS-based services (e.g., CICS and IMS transactions, and DB2 stored procedures) in the form of APIs
  - These APIs are invoked by clients using REST calls (i.e., they are RESTful services), and data payloads are in JSON format

A little more on z/OS Connect and DB2 for z/OS

- DB2 for z/OS supports z/OS Connect V1
  - A client application could use a REST call to invoke an API that would in turn drive execution of a SQL statement (which could be a stored procedure CALL)
- DB2 for z/OS support for z/OS Connect V2 is not yet there, but is in the works
  - This support will be delivered in a way that complements and leverages the native REST support that is being built into the DB2 for z/OS Distributed Data Facility



Robert Catterall
IBM
rfcatter@us.ibm.com

Please fill out your session evaluation before leaving!

Let Me Make This Clear (Things That Plenty of DB2 for z/OS People Get Wrong)

Photo by Steve from Austin, TX, USA