



Db2 Night Show

Db2 Java Performance Best Practices

Dave Beulke,
Dave Beulke @ Associates,
Division of Pragmatic Solutions, Inc.

Dave @ DaveBeulke.com - 703 798-3283

- Member of the inaugural IBM Db2 Information Champions
- One of 45 IBM Db2 Gold Consultant Worldwide
- Past President of International Db2 Users Group - IDUG
- Best speaker at CMG conference & former TDWI instructor

- Former co-author of certification tests
- Db2 Certification test
- IBM Business Intelligence certification test
- Former columnist for Db2 Magazine
- Former editor of the IDUG Solutions Journal

Performance BLOG:
www.DaveBeulke.com

- **Consulting**
 - CPU Demand Reduction – Guaranteed!
 - Db2 Performance Reviews
 - Database Design Review
 - Security Audit & Assessments
 - Migration Assistance
- **Education Seminars**
 - Db2 Version 10 Transition
 - Db2 Performance for Java Developers
 - Data Warehousing Designs for Performance
 - How to Do a Performance Review
 - Data Studio and Others
- Extensive experience in architecture & performance of large systems, databases and DW systems
 - Working with Db2 on z/OS since V1.2
 - Working with Db2 on LUW since OS/2 Extended Edition
 - First data warehouse in 1988 for E.F. Hutton
- Programming in Java for **Syspedia** since 1996 Find, understand and integrate your data faster!

I am honored to have been a presenter at 30+ years of Db2 conferences

- 2021 – Java Db2 Performance Best Practices
Security Best Practices volume III
- 2020 - SQL Performance for Big Data
- 2019 - Best Design and Performance Practices for Analytics
- 2018 – Philadelphia - Security Best Practices Volume II
-Best Design and Performance Practices for Analytics
- 2017 – Anaheim -Understand IDAA Performance and Justify an IDAA Appliance
- 2016 – Austin Performance Enterprise Architectures for Analytic Design Patterns
How to do your own Db2 Security Audit
- 2015 - Valley Forge Db2 Security Practices
Big Data Performance Analytics Insights
- 2014 – Phoenix Big Data SQL Considerations
- 2013 – Orlando Big Data Disaster Recovery Performance
- 2012 – Denver Agile Big Data Analytics
- 2011 – Anaheim Db2 Temporal Tables Performance Designs
- 2010 - Tampa - Java DB2 Developer Performance Best Practices
- 2009 – Denver -Java Db2 Perf with pureQuery and Data Studio
Improve Performance with Db2 Version 9 for z/OS
- 2008 – Dallas - Java pureQuery and Data Studio Performance
- 2007 - San Jose - Developing High Performance SOA Java Db2 Apps
Why I want Db2 Version 9
- 2006 - Tampa - Class - How to do a Db2 Performance Review
Db2 Data Sharing
Data Warehouse Designs for Performance
- 2005 – Denver - High Performance Data Warehousing
- 2004 – Orlando – Db2 V8 Performance
President of IDUG
- 2003 - Las Vegas - Db2 UDB Server for z/OS V8 Breaking all the Limits
Co-author IBM Business Intelligence Certification Exam
- 2002 - San Diego - Db2 UDB for LUW 8 - What is new in Db2 Version 8
Data Warehouse Performance
- 2001 – Orlando -Data Sharing Recovery Cookbook
Designing a Data Warehouse for High Performance
Co-authored the first IBM Db2 z/OS Certification Exam
- 2000 – Dallas - Db2 Data Warehouse Performance Part II
- 1999 – Orlando - Store Procedures & Multi-Tier Performance
Developing your Business Intelligence Strategy
Evaluating OLAP Tools
- 1998 - San Francisco - Db2 Version 6 Universal Solutions
Db2 Data Warehouse Performance
Db2 & the Internet Part II
- 1997 – Chicago - Db2 & the Internet
- 1996 – Dallas- Sysplex & Db2 Data Sharing
Best Speaker Award at CMG Conference Mullen Award
- 1995 – Orlando - Practical Performance Tips
Improving Application Development Efficiency
- 1994 - San Diego - Database Design for Time Sensitive Data &
Guidelines for Db2 Column Function Usage
- 1993 – Dallas - High Availability Systems: A Case Study &
Db2 V3: A First-Cut Analysis
- 1992 - New York -Db2 –CICS Interface Tuning
- 1991 - San Francisco - Pragmatic Db2 Capacity Planning for DBAs
- 1990 – Chicago - Performance Implication of Db2 Design Decisions
- 1989 – Chicago - Db2 Performance Considerations

Db2 Java Performance Best Practices

- Understand the Db2 coding best practices to enhance performance, avoid problems and enhance debugging.
- Realize all the application coding options and java class frameworks that can help and hinder performance.
- Realize the debugging methods available, the java tracing tools and the easy and fast best practices to find performance issues.
- Understand the standard performance characteristics and special java statistics to monitor to determine quickly whether a performance problem or improvement opportunity exists.
- How to bring some of these java performance tuning best practices to your shop and enhance your standard development procedures.



Strategic and Tactical Discoveries & Recommendations

Story behind every performance problem

- Performance issues always have a story
 - Need to understand the background, context and issues
- Performance issues are thrust upon us
 - Seems that every time it is an emergency
 - Always time to fix it later
- The stories you are about to hear are true. The names and circumstances have been changed to protect the innocence.
My name is Dave Beulke and I fix these situations.
- After learning from this session you will be able to fix them too.



Started Monday morning....

Number of Vice Presidents MS Team
me first thing in the morning

- Processing has been executing for 2+ hours

Always ask these five questions!

- **Who** are the developers?
- **What** was the performance history in development and QA?
- **When** was the process supposed to finish? Runtime expectations
- **Where** was it tested? Where did it come from-home grown/vendor?
- **Why** is it a performance issues? SNAFU

Start the investigation of performance

Gather all statistics from every monitor available

- Best if the performance problem is still running

Plan on the worst case scenario

- Production access
- Authorization for the monitor access
- Monitor always on/started
- Correct traces turned on via zOS, zLinux, Server, Db2 etc....
- Capture the Web Server Logs & application logs
- Traceable situation – dump of the applicationabend?

1 out of 7 isn't bad

- The Framework to partition the work along the data partition boundaries
- Monitor shows
 - One job doing the work
 - Seven others doing nothing

+ Elapsed		PlanName	DB2	Status	GetPg	Update	Commit	CORRID/JOBNM
+ -----		-----	-----	-----	-----	-----	-----	-----
+ 02:03:52.1	P	J312NT	DSN3	WAIT-SYNC-IO	21311K	0	0	J312NT09
+ 02:03:51.9	*	J312NT	DSN3	WAIT-LOCKPQS	0	0	0	DSN3DBM1
+ 02:03:51.9	*	J312NT	DSN3	WAIT-LOCKPQS	0	0	0	DSN3DBM1
+ 02:03:51.9	*	J312NT	DSN3	WAIT-LOCKPQS	0	0	0	DSN3DBM1
+ 02:03:51.9	*	J312NT	DSN3	WAIT-LOCKPQS	0	0	0	DSN3DBM1
+ 02:03:51.9	*	J312NT	DSN3	WAIT-LOCKPQS	0	0	0	DSN3DBM1
+ 02:03:51.9	*	J312NT	DSN3	WAIT-LOCKPQS	0	0	0	DSN3DBM1
+ 02:03:51.9	*	J312NT	DSN3	WAIT-LOCKPQS	0	0	0	DSN3DBM1



Architecture Frameworks

Discoveries and Recommendations

Frameworks

- All frameworks can perform efficiently
 - Pattern for the processing always works
- Beware of the “*lightweight*” frameworks
 - Beware of “Demo” ware
 - Only need their java libraries copied into your development
 - Beware of their security vulnerabilities especially if open source
 - Beware of their update cycle, versions and dependencies
- Beware of “*Object generators*” or “*Web Engines*”
 - Also Event or Validation engines

Avoid Spring Batch! Db2 can do it faster? but if you must consider it

- Architecture – Read, Process, Write concept
 - Know your database partitioning ranges
- Calculate result set size (#Rows * Column width)
 - Align JVM/Chunk with partitioning
 - Realize updates lock – *isolate all locking* appropriately
- Adjust minimize JVM settings per chunk
- Runtime Memory allocations appropriate
 - Optimize the Chunk Size
- Must have a CPU processor per JVM
- Chunk size determined by
 - One chunk “read” at time processes
 - Logical Read Process Write



Avoid Hibernate!

- UOW & Transaction Scope
 - One hibernate definition many uses
- Persistence cache control
 - How much cache is enough
- Hibernate and persistence layer issues
 - Lazy, Evict, etc.....
 - Regular SQL versus Hibernate SQL
 - Optimistic-lock
- Logging Considerations
 - How many logs are your transactions writing to?



Avoid Hibernate – Avoid/eliminate if possible

- UOW & Transaction Scope
 - Many object oriented application persist the data for easy object programming languages. Generic persistence of SOA objects leads to deadlocks, data integrity and usually poor performance.
- Hibernate and persistence layer issues
 - The Hibernate interface, persistence layer and its performance problems are a full presentation by itself. Many companies are having difficulties with the Hibernate settings, its handling of persistence, its SQL issues. Hibernate can cause many performance problems if setup badly or used inappropriately. Check your configuration settings, customize them for your application and minimize the amount of data Hibernate persists are the best practices for performance.
- Logging Considerations
 - How much logging is happening in your distributed environments? Check the UNIX and Windows connections because their logging can be 75% of the transaction time.

Framework Configuration & Deployment Issues

Configuration data is difficult to deploy

- Has repeatedly delayed testing efforts
- Beware of tooling or loading configuration data to development, testing, and production database
- Configuration/reference data will not accelerate application development effort

Incremental deployment can't easily be done

- Can't be shared - usually multiple versions on server add complexity to the issues
- Incremental improvements or changes can't be done online
- Framework doesn't provide availability, security and reliability
 - Fragile frameworks can ruin your application availability, reliability and scalability

Collect more info on the running process

- Spring configuration & framework spawned the other processes
- Monitor shows
 - One job UOW is out of control, syncing with other 7 processes

+ Elapsed		PlanName	DB2	Status	GetPg	Update	Commit	CORRID/JOBNM
+ -----		-----	-----	-----	-----	-----	-----	-----
+ 02:03:52.1	P	J312NT	DSN3	WAIT-SYNC-IO	21311K	0	0	J312NT09
+ 02:03:51.9	*	J312NT	DSN3	WAIT-LOCKPQS	0	0	0	DSN3DBM1
+ 02:03:51.9	*	J312NT	DSN3	WAIT-LOCKPQS	0	0	0	DSN3DBM1
+ 02:03:51.9	*	J312NT	DSN3	WAIT-LOCKPQS	0	0	0	DSN3DBM1
+ 02:03:51.9	*	J312NT	DSN3	WAIT-LOCKPQS	0	0	0	DSN3DBM1
+ 02:03:51.9	*	J312NT	DSN3	WAIT-LOCKPQS	0	0	0	DSN3DBM1
+ 02:03:51.9	*	J312NT	DSN3	WAIT-LOCKPQS	0	0	0	DSN3DBM1

Module Considerations

- Java class overall module size
- Discover the Synchronizing within the application
 - Static method forces each calling thread to block until no other thread is executing that static method
 - Is your framework and application thread safe?
- Where within the application is the module Serialization?
 - Application waits for processing of another framework UOW portion
- Duplicate java classes within projects
 - Or duplicate packages or paths within the applications



Java Debugging

Discoveries and Recommendations

Gather runtime information is the best

- Thread details are critical performance & debugging information

THREAD HISTORY SQL COUNTS

HPLN

```
+ Thread:  Plan=?RRSAF      Connid=RRSAF      Corrid=J312NT09      Authid=J312MSP
+ Attach:  RRSAF           DB2=DSN3          MVS=LPAR1
+ Time   :  Start=01/19/2021 05:29:59.162472      End=01/19/2021 07:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit      =      44      Abort      =      2      Select      =      0
+ Open Cursor = 1344387      Close Cursor = 183722      Fetch      = 1163383
+ Insert      = 168902      Delete      = 38088      Update      = 1940714
+ Describe    = 3909142      Lock Table  = 7      Prepare     = 3492098
+ Grant       = 0      Revoke      = 0      Set Rules   = 0
```

Java Debugging display

- Debug display showing JDBC Db2 prepared statement
 - No parameter markers
 - Using invalid java types to receive Db2 column values
- SQLCODE ignored, copied and set
 - Variety of SQLCODE(s) - +100, -100, -811 etc.....
 - Framework runtime documentation there are no memory errors logged
- Need to remember to CLOSE all SQL objects
 - stmt(s), resultSet(s) and Connection
- Verify that the Java processing is THREADSAFE
 - Subroutine integrity

Java debugging resources

- Developer testing environment
 - Not enough data/performance testing resources
 - No End-to-End performance documentation for fixes
 - 1 in a Million data situations -100s of billions of rows in the cloud
- Performance needs to be given enough time and be a priority
 - Changes documented, need to be given priority for performance analysis
- Object point analysis/Process point
 - Need to monitor memory objects within the application
 - Arrays, Hash Maps, Vectors, caching within any java object or framework component
- Management support structures
 - CM procedures drive performance analysis – checklist of components tested
 - Documentation gathered while going through the business requirements and functionality

Testing Tools

- Many tools can help discover problems
 - Copied code
 - Dead code
 - Open Source Patch Maintenance issues
- Static code analysis
 - PMD, SONAR , IntelliJ and etc.....reveal problems
 - <https://java-source.net/open-source/code-analyzers>
 - Most are free
 - Cloud can host analysis
- Need automated tests with common data set that helps developers test services
 - Automated JUnit testing?
 - Thread safe testing?

Debugging is a matter of Dev/QA

Automated tools should be a requirement

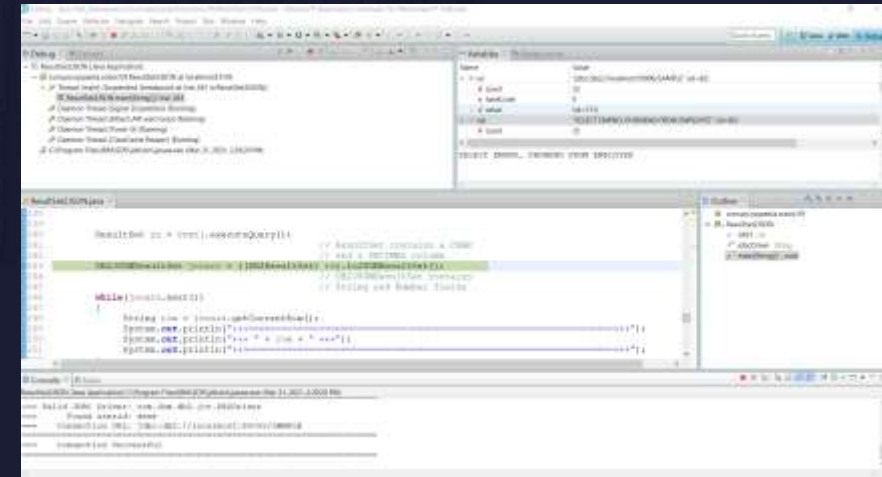
- Assign resources from development team
- Integrate automated tests during development
 - Develop bulk and unique test data for validation
 - SQL EXPLAINS captured within the test environments

Generated test data for services

- Understanding of business service usage
- Data missing keys, codes and proper optional types parameters
- Invalid type data flowing into services parameters
- Platform to platform conversions
 - (EBCDIC to UNICODE) or (UNICODE to UNICODE) or (ASCII to ASCII to ASCII) or XX to YY

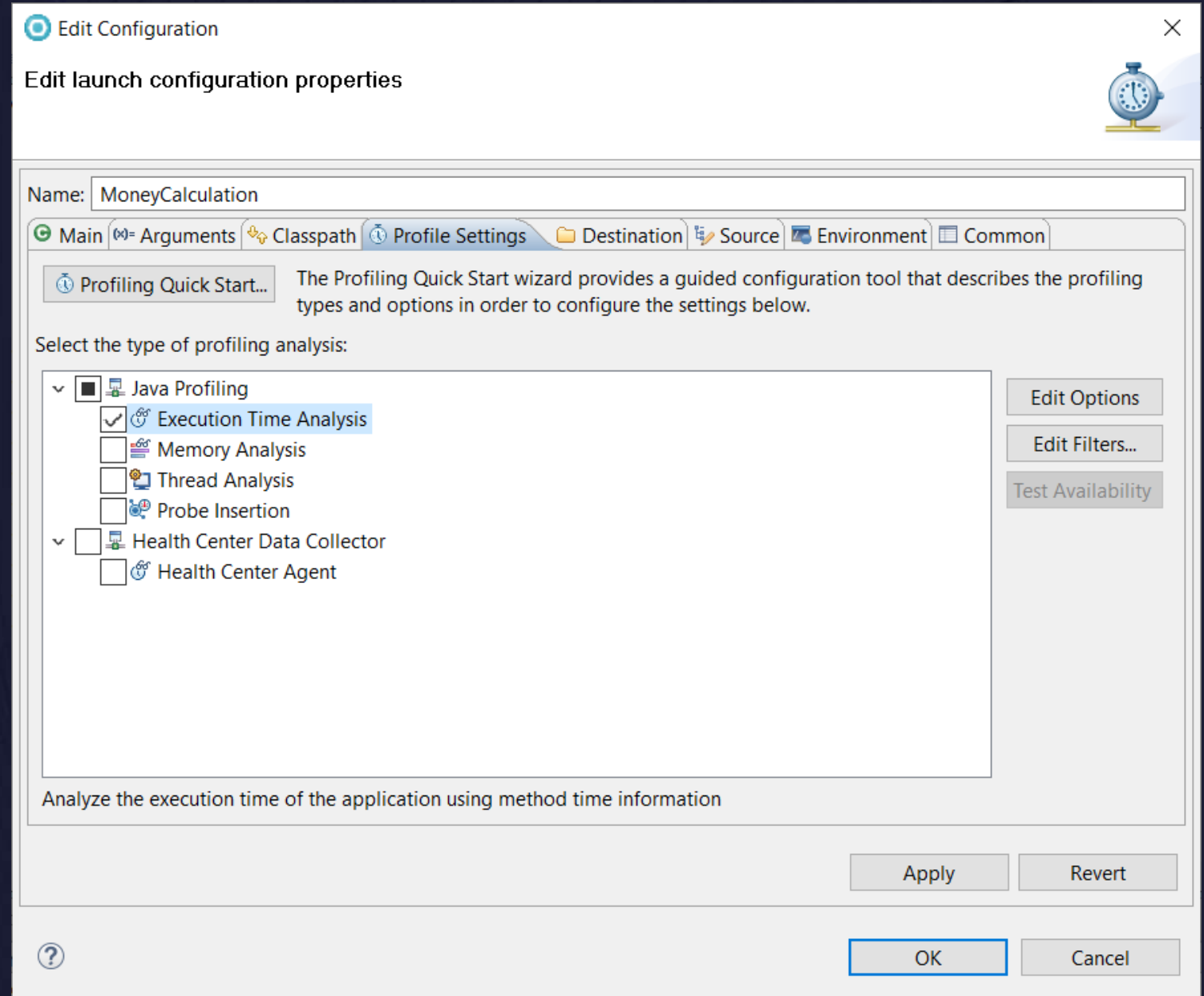
Benefits of Profiling Tools

- Integrated Profiling within RAD and <insert your IDE here>
 - Provide a trace of the normal flow of the application
 - Discover the exceptional java classes
 - Discovered 60,000+ SQL calls for process
 - Uncovered java classes used from old releases
 - Realize where cache is being used for transactions
 - Discovered data validation being done multiple times
- Web services/applications logic flow being used
 - Uncover the performance truth about an application
 - See the time spent within each java package, *class* and *method*
 - Understand the modules referenced
 - Discover unnecessary looping, arrays and tables being built



Profiling Statistics

- Thread Visualizer





Java Profiling

- Is there a test history of Profiling the application
 - Thread Statistics
 - Monitor Statistics
 - Threads Visualizer

[illegible]

Thread Statistics

- More Statistics

Execution Statistics   Object Allocations

Execution Statistics

Call Tree

Thread name	<Percent Per T...	Cumulative Time (s...	Min Time	Avg Time	Max Tim...	Calls
main[14060]	100.00%	0.112389				
MoneyCalculation()	16.69%	0.018755	0.018755	0.018755	0.018755	1
main(java.lang.String[]) void	0.69%	0.000775	0.000775	0.000775	0.000775	1
doCalculations() void	0.62%	0.000695	0.000695	0.000695	0.000695	1
log(java.lang.String) void	0.26%	0.000298	0.000025	0.000043	0.000105	7
getPercentage() java.math.BigDecimal	0.04%	0.000043	0.000043	0.000043	0.000043	1
rounded(java.math.BigDecimal) java.math.BigDecimal	0.01%	0.000017	0.000017	0.000017	0.000017	1
getAverage() java.math.BigDecimal	0.03%	0.000039	0.000039	0.000039	0.000039	1
getSum() java.math.BigDecimal	0.01%	0.000008	0.000008	0.000008	0.000008	1
getPercentageChange() java.math.BigDecimal	0.03%	0.000038	0.000038	0.000038	0.000038	1
getDifference() java.math.BigDecimal	0.01%	0.000008	0.000008	0.000008	0.000008	1
rounded(java.math.BigDecimal) java.math.BigDecimal	0.01%	0.000007	0.000007	0.000007	0.000007	1
getDifference() java.math.BigDecimal	0.02%	0.000021	0.000021	0.000021	0.000021	1
getSum() java.math.BigDecimal	0.01%	0.000012	0.000012	0.000012	0.000012	1
MoneyCalculation(java.math.BigDecimal, java.math.BigDecimal)	0.04%	0.000041	0.000041	0.000041	0.000041	1
rounded(java.math.BigDecimal) java.math.BigDecimal	0.02%	0.000017	0.000004	0.000008	0.000013	2

< Previous

Next >

Session summary

Execution Statistics

Call Tree

Method Invocation Details

Method Invocation

SUSPENDS, DEADLOCKS & ROLLBACKS

- Usually undocumented and/or tracked when Java is involved!
- No documentation on any errors
 - Email all error(s) to the developers!
- Trying to mimic a database within the application
 - Cache involved in problem situations/transactions
 - How often is a memory/cache error reported/involved in these errors?
- How many insert/update/deletes within application
 - When is the processing done within the UOW – Commit points



Class, Method Dependencies

Discoveries and Recommendations

Looking at your Java threads in detail

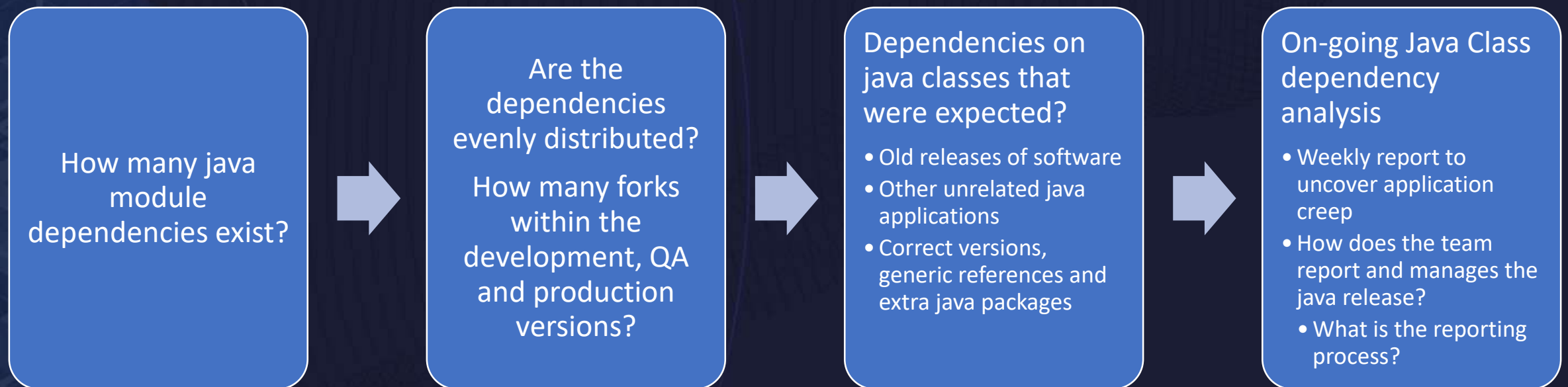
- Thread details are critical performance & debugging information

THREAD HISTORY SQL COUNTS

HPLN

```
+ Thread:  Plan=?RRSAF      Connid=RRSAF      Corrid=J312NT09      Authid=J312MSP
+ Attach:  RRSAP          DB2=DSN3          MVS=LPAR1
+ Time   :  Start=01/19/2021 05:29:59.162472      End=01/19/2021 07:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit      =      44      Abort      =      2      Select      =      0
+ Open Cursor = 1344387      Close Cursor = 183722      Fetch      = 1163383
+ Insert      = 168902      Delete      = 38088      Update      = 1940714
+ Describe    = 3909142      Lock Table  = 7      Prepare     = 3492098
+ Grant       = 0      Revoke      = 0      Set Rules   = 0
```

Java Class Dependencies

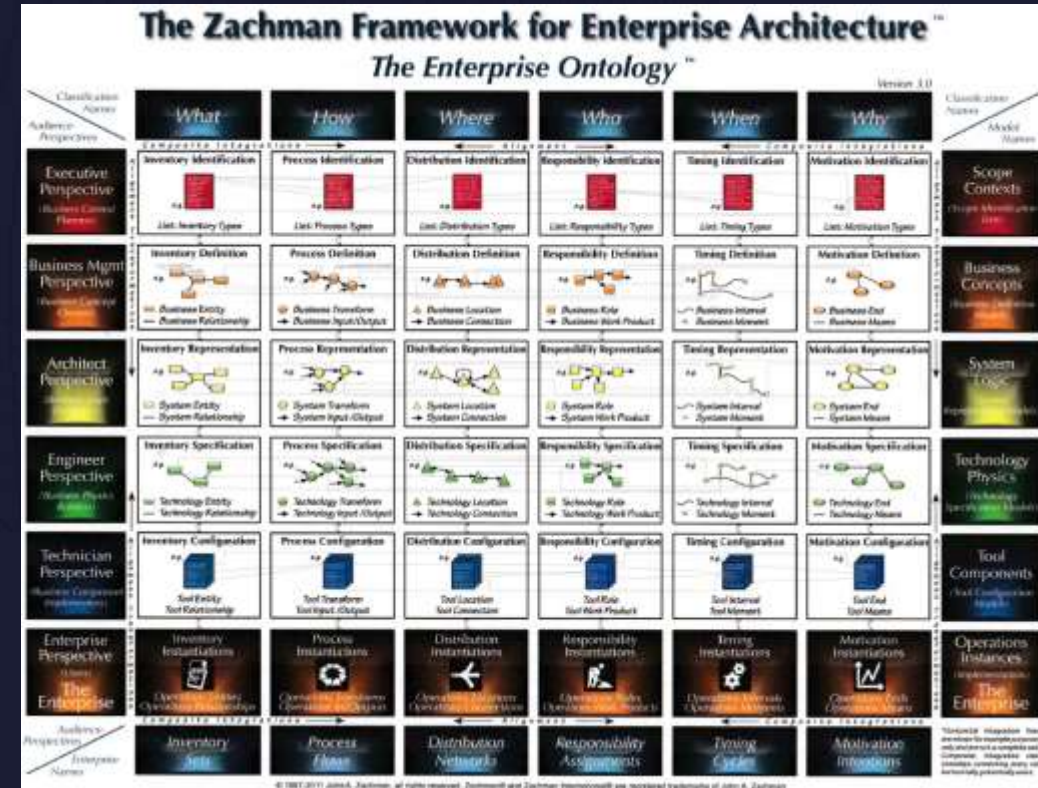


Framework Java Class Dependencies

- How many frameworks is your project dependent on?
 - Have all the frameworks been copied in every module?
- How many framework versions is project dependent on?
 - Sometime multiple versions introduced within application
- Have the frameworks been integrated into modules where they aren't used or needed?

Database Performance Dependencies

- Which Java classes reference the database
 - Dependencies on the SQL or Hibernate or ???
 - Cross Reference SQL statements to the Java Classes
 - Beware of using the wrong SQL statement for their data
 - Might supply the data but doesn't do it efficiently -
- Database designs not Best Practices
 - Insufficient structures
 - # of Partitions, Index structure, Index Columns
- Application processing and flow
 - Amount of data processed
 - Order of processing flow impacts amount of deadlocks



Dependency on Hibernate Issues

- Convert Hibernate to Db2 Native Stored Procedures
 - Remove dynamic Hibernate SQL statements
 - Convert into stored procedures will dramatically reduce CPU & overall TCO
 - Exploits Db2 zIIP CPU resources that are free
- Beware of stale cache data problems
 - Dependent on the cache for data
 - Reference data
 - Slowly changing data within the reference data
 - Transaction UOW data
- Local cache and failover dependencies
 - Beware of multiple servers/clouds for performance

Cache Data Persistence



How much memory is required for the peak number of transactions?



Multiply the numbers out to understand the memory needed

- Transactions * cache memory * concurrent = peak server memory requirements



Remember to leave enough headroom

- Server/Cloud cache requirements
- Server balancing, Fail-over and capacity planning



Spawn another server/Kubernetes instance

- When, where, and how is it spawned?



Client Situations

Discoveries and Recommendations

Every picture tells a story don't it!

- First impressions

THREAD HISTORY SQL COUNTS

HPLN

```
+ Thread:  Plan=?RRSAF      Connid=RRSAF      Corrid=J312NT09      Authid=J312MSP
+ Attach:  RRSAF           DB2=DSN3          MVS=LPAR1
+ Time   :  Start=01/19/2021 05:29:59.162472      End=01/19/2021 07:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit      =      44      Abort      =      2      Select      =      0
+ Open Cursor = 1344387      Close Cursor = 183722      Fetch      = 1163383
+ Insert      = 168902      Delete      = 38088      Update      = 1940714
+ Describe    = 3909142      Lock Table  = 7      Prepare     = 3492098
+ Grant       = 0      Revoke      = 0      Set Rules   = 0
```

Data Integrity

- ABORTS=2

THREAD HISTORY SQL COUNTS

HPLN

+ Thread: Plan=?RRSAF Connid=RRSAF Corrid=J312NT09 Authid=J312MSP
+ Attach: RRSAF DB2=DSN3 MVS=LPAR1
+ Time : Start=01/19/2021 05:29:59.162472 End=01/19/2021 07:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2

+ Commit	=	44	Abort	=	2	Select	=	0
+ Open Cursor	=	1344387	Close Cursor	=	183722	Fetch	=	1163383
+ Insert	=	168902	Delete	=	38088	Update	=	1940714
+ Describe	=	3909142	Lock Table	=	7	Prepare	=	3492098
+ Grant	=	0	Revoke	=	0	Set Rules	=	0

SQL Loops

- Open Cursor \neq Closed Cursor
 - Difference spells trouble – 1.1m

THREAD HISTORY SQL COUNTS

HPLN

```
+ Thread:  Plan=?RRSAF      Connid=RRSAF      Corrid=J312NT09      Authid=J312MSP
+ Attach:  RRSAF            DB2=DSN3          MVS=LPAR1
+ Time   :  Start=01/19/2021 05:29:59.162472      End=01/19/2021 07:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit           =          44      Abort           =           2      Select           =           0
+ Open Cursor      = 1344387      Close Cursor     = 183722      Fetch             = 1163383
+ Insert           = 168902      Delete           = 38088      Update            = 1940714
+ Describe         = 3909142      Lock Table       =           7      Prepare           = 3492098
+ Grant            =           0      Revoke           =           0      Set Rules         =           0
```

Lock Usage

- Why or how are Lock tables?
- Which table(s)?

THREAD HISTORY SQL COUNTS

HPLN

```
+ Thread:  Plan=?RRSAF      Connid=RRSAF      Corrid=J312NT09      Authid=J312MSP
+ Attach:  RRSAF            DB2=DSN3          MVS=LPAR1
+ Time   :  Start=01/19/2021 08:29:59.162472      End=01/19/2021 10:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit      =      44      Abort      =      2      Select      =      0
+ Open Cursor = 1344387      Close Cursor = 183722      Fetch      = 1163383
+ Insert      = 168902      Delete      = 38088      Update      = 1940714
+ Describe    = 3909142      Lock Table  = 7          Prepare     = 3492098
+ Grant       =      0      Revoke      =      0      Set Rules   =      0
```

SQL Statement reuse

- Number of Describes = 3.9m
- Why?

THREAD HISTORY SQL COUNTS

HPLN

```
+ Thread:  Plan=?RRSAF      Connid=RRSAF      Corrid=J312NT09      Authid=J312MSP
+ Attach:  RRSAF            DB2=DSN3          MVS=LPAR1
+ Time   :  Start=01/19/2021 05:29:59.162472      End=01/19/2021 07:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit      =      44      Abort      =      2      Select      =      0
+ Open Cursor  = 1344387      Close Cursor = 183722      Fetch      = 1163383
+ Insert       = 168902      Delete      = 38088      Update      = 1940714
+ Describe     = 3909142      Lock Table  = 7          Prepare     = 3492098
+ Grant        = 0          Revoke      = 0          Set Rules   = 0
```

Be Prepared

- How about some thread or SQL reuse?

THREAD HISTORY SQL COUNTS

HPLN

```
+ Thread:  Plan=?RRSAF      Connid=RRSAF      Corrid=J312NT09      Authid=J312MSP
+ Attach:  RRSAF           DB2=DSN3          MVS=LPAR1
+ Time   :  Start=01/19/2021 05:29:59.162472      End=01/19/2021 07:34:00.761584
+ Luwid=PRODNET.J312BS0.D8D8602315E2
+
+ Commit      =      44      Abort      =      2      Select      =      0
+ Open Cursor  = 1344387      Close Cursor = 183722      Fetch      = 1163383
+ Insert       = 168902      Delete      = 38088      Update      = 1940714
+ Describe     = 3909142      Lock Table  =      7      Prepare     = 3492098
+ Grant        =      0      Revoke      =      0      Set Rules   =      0
```

Java logic issues - Cursor loop control

1. The number of ABORTS=2. How are these impacting UOW and integrity of the data INSERTED/DELETED? Aborts should ALWAYS be = 0.
2. OPEN CURSOR<>CLOSE CURSOR which is causing the -479 error.
3. LOCK TABLE - Why do there need to LOCK TABLE statements? Which tables and where in the processing logic?
4. The number of DESCRIBES is extraordinarily high and needs to be investigated. Usually there are only a few.
5. The number of SQL PREPAREs is high. Is the logic PREPAREing for every SQL statement for execution, usually these PREPAREs are based on logic unit-of-work (UOW) processing. The number of PREPAREs doesn't seem to reflect UOW logic, CURSOR usage or number of DESCRIBEs.

Reduced SQL calls by 99%

- Performance problem symptoms
 - Slow response time
 - Deadlocking application
 - Huge I/O rate against the database
- Profile the application uncovered the issue
 - Discovered 60,000+ SQL calls for process
 - Wrong looping within the application
 - Web pages showed the correct data
- Understand what java classes are calling database/table(s)/SQL



Can transaction cache be transferred?

- Where do the services go?
- Services Architecture
 - Connection validation & reuse
 - Transaction/Data integrity
 - Security authorization
 - Thread caching and reuse
 - Plan/Package authorization caching



Number of Clouds, Servers, UOW & Connections

- How many does the application really need?
 - Database(s)
 - Db2 LUW, Db2 z/OS & Oracle in one transaction
 - Queues
 - Inbound and outbound
 - Web and App Server connection threads
- Parallelism and connection state
 - Mind the state of all of these connections
 - How long each is active
 - How long the transaction UOW is!
 - What is the UOW within each OS, database, within which table(s).

Developer Bad Habits

- Chaos - Developers want their own environment
 - Helps team communicate
 - Test and debug without worrying about locking rows used by others
- Test data problems
 - Isolated testing for only a small variety of data
 - Don't have full range or set of data or complexity of data needed for all tests
 - How many different tests are runs? How many are enough?
- How many network round trips to get the transaction completed?
 - Within the application, UOW and java class(es)

Summary - Java

- Avoid any and all Java Frameworks
- Understand usage of Java Packages, Classes and versions – hopefully only 1
- Minimize Java open source packages – updates & security liabilities?
- Understand usage of JVM, memory cache, java types ie; Hash Maps etc.
 - Use IDE Profiling to understand memory usage
 - In terms of database information cached
 - Use IDE Debug mode to fully document logic looping
 - Automate Junit testing
 - Understanding how much Garbage your program produces during UOW
 - Fully document GC at full capacity performance load
- Test your java processing, transaction and UOW for being Thread Safe
- Understand failover and scalability liabilities limitations – Cloud/server/app

Summary - Java Db2

- Fully understand connection usage and UOW scope
 - Logic loop reflection of UOW
 - Understand Locking scope of UOW
 - Locks usage, locks acquired, retained and released during UOW and full runtime
 - Understand COMMIT scope and failover
- SQL etiquette
 - SQL column to Java data type usage
 - SQL parameter markers
 - SQL EXPLAIN captured
 - SQLCODE checking/handling immediately after SQL executed
 - CLOSE clean up all database resources – SQL Stmts, ResultSets, Connection

Thank you!!



✓ Send any questions or comments to Dave @ Dave Beulke . com

Performance BLOG:
www.DaveBeulke.com

Twitter (@DBeulke)
LinkedIn (www.linkedin.com/in/davebeulke)