Dave Beulke & Associates
A division of Pragmatic Solutions, Inc
Dave@DaveBeulke.com
3213 Duke Street
Suite 805
Alexandria, VA 22314
703 798-3283

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

Co-Author of certification tests
• DB2 Certification test
• IBM Business Intelligence certification test

Columnist for DB2 Magazine
• Former editor of the IDUG Solutions Journal

BLOG:
www.DaveBeulke.com

Consulting
• CPU Demand Reduction Guaranteed!
• DB2 Performance Review
• Database Design Review
• Security Audit & Assessments
• Migration Assistance

Teaching Educational Seminars
• DB2 Version 10 Transition
• DB2 Performance for Java Developers
• Data Warehousing Designs for Performance
• How to Do a Performance Review
• Data Studio and pureQuery

Extensive experience in 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

Programming in Java for Syspedia since 2001 Find, understand and integrate your data faster!
Java DB2 Developer Performance Best Practices
By Dave Beulke

Outline

- Understand Java DB2 performance best practices
- Learn Java performance components
- Understand servlet and JSP considerations
- Learn blocking, caching and Java techniques
- Understand my many client experiences solving their Java performance problems

Notes:

- Can your Java, J2EE DB2 application sustain a large number of client requests simultaneously? Or do they deadlock, become sluggish, or have painfully slow response times? There are many reasons for java performance bottlenecks and many ways to prevent them. However, sometimes it's just a matter of following some simple best practices that can make all the difference.

- This presentation will discuss the java developer best practices, coding for optimum DB2 access and some simple changes you can make, some in the design and some in the coding phases, that can help your developers build faster, more robust applications.
Patterns for Performance

Reuse and object oriented programming

Web Services considerations

Trends, Fades and Reality

Notes:

These topics will be detailed during this presentation.

Patterns for Performance

- The pattern of your transactions and using a MVC architecture is very important for object oriented programming languages.

Reuse and object oriented programming

- Object oriented programming offers many opportunities for application reuse and function flexibility. Sometimes these issues lead to performance problems.

Web Services considerations

- Object oriented programming offers many opportunities for application reuse and function flexibility. Sometimes these issues lead to performance problems.

Trends, Fades and Reality

- How the CPU, storage, bandwidth and object oriented programming improvements will affect your systems and their performance.
Patterns for Performance

- Minimizing coupling between components
  - Understand the critical path of your main processing
  - Analyze your components dependencies
    - Cookies, session variables, persistent data
    - Domain, range and data
    - Other

- Understand the method mapping to other methods
  - Understand the method or JAR dependencies
  - Analyze the method overlap or method extensions
  - Research the Classpath hierarchy and method duplication

Notes:

- Minimizing coupling between components
  - The internet is billions of loosely coupled items. As your application and data architecture continues to expand, this pattern will persist. How are you dealing with bringing data and applications together?

- Understand the method mapping to other methods
  - How is sharing data and application assets across your enterprise done? Today, integration is a major expense and given lip service in new application efforts. Understanding the best performing objects and assets is paramount to research.
Patterns for Performance

Distributed processing considerations
- Verify outside methods are good performance partners
- Have your performance critical path as local as possible

Component should communicate with a limited number of other components
- Components are local or remote or distance
  - Same server, another server in data center or remote data center

Notes:

- Distributed processing considerations
  - Since the new SOA: Service Oriented Architecture uses all the internet billions of loosely coupled items, how many objects do you access with outside partners? What is the performance profile of those partner applications?

- Component should communicate with a limited number of other components
  - Limit the number of objects referenced to improve performance. Where are the objects located in your application? How many servers does your application touch to provide that sub-second response time?
Server Connections - Basics

- Use a server connection pool
  - One per ??
    - Web page
    - Application
    - Web server

- Always time out threads after a certain period
  - Only define the number needed
    - Know your minimum and maximum expected

- Recycle and reuse all the threads connected

Notes:

- Use a server connection pool
  - No direct JDBC connections should made to your database systems. It is a huge security risk and auditing and compliance will find you. Always separate connection pools for different applications. Generic pools can only provide generic or bad performance.

- Always time out threads after a certain period
  - A thread left unmanaged will hold locks and other resources. Make sure to time the threads out. The recommended timeout period is 3 or 5 minutes for all thread connections.

- Recycle and reuse all the threads connected
  - Make sure to recycle the thread pools within your servers. This can be done through different servers to provide application availability.
Server Connections

How many does the application really need?
- Database
  - 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!

Notes:
- How many does the application really need?
  - Some SOA applications get a tremendous amount of connections to a wide variety of systems, databases, files and other resources. Many of these connections go to different platforms, vendors, applications and interfaces.
- Parallelism and connection state
  - The different connections have a wide variety of settings and usage patterns. Connection parallelism is vital for overall performance. Make sure your system settings provide enough connections for the peak workload, that they timed out appropriately and provide the security to thwart attacks.
**Server Application Scope**

- UOW & Transaction Scope
- Persistence cache control
- Hibernate and persistence layer issues
  - Lazy, Evict, etc.....
  - Regular SQL versus Hibernate SQL
  - Optimistic-lock
- Logging Considerations
  - How many logs are your TX writing to?

**Notes:**

- **UOW & Transaction Scope**
  - Many object oriented application persist the data for easy object programming languages. Generic persistence of SOA objects leads to locking, data integrity and usually poor performance.

- **Hibernate and persistence layer issues**
  - The Hibernate interface, persistence layer and its performance problems is a full presentation by itself. Many companies are having difficulties with the Hibernate settings, its handling of persistence, its SQL issues. Hibernate is a good technology but can cause many performance problems if setup badly or used poorly. Check your 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.
Server Considerations

Is the server sliced virtually?
- Memory and CPU for each slice
- Peak, average, CPU, memory, I/O utilization
- Monitoring

Amount of server memory
- Number of concurrent transactions
- Amount of parallel application threads
- Amount of persistence kept within each transaction

Amount of CPU available
- How many cores can the application leverage
- What common resources do transaction share?

Notes:

Is the server sliced virtually?
- Server virtualization is the normal standard operating procedure these days. The machine your application is running on is also running 12 other applications. This causes your important production transaction to wait for resources such as CPU, I/O and connection bandwidth. Find out how big is your virtual slice of the server.

Amount of server memory
- The amount of memory in a server is vital for all the components performance. Find out how many concurrent transaction are being serviced, how big their threads are and how much data each uses. This will show your performance is suffering from too much workload being used by too many transactions. It only gets worst if your server is supporting virtual environments.

Amount of CPU available
- How many concurrent transaction can your CPU process? Do you know if you don’t why not? Ask and you will be surprised no one really knows because capacity planning is not done for distributed systems management just buys another server.
Where did the transaction come from?

Services Architecture
- Connection validation & reuse
- Transaction/Data integrity
- Security authorization
- Thread caching and reuse
- Plan/Package authorization caching

Where did the transaction come from?

Servers virtualization spreads servers and applications all over the data center and among business partners. For critical performance of application transactions try to minimize the number of connections used. This will minimize connections, security checks, thread caching, and execution authorizations and help insure data integrity.
Logging Common Errors

Exception Error handling may not be appropriate

- All errors are not exceptions
  - `-811` and `+100 SQLCode`

GET DIAGNOSTICS statement

- Information about the last SQL statement
- How many rows effected/errrors

Example

- Retrieve information that is similar to what is returned by the SQLCA plus Row Count

```
EXEC SQL GET DIAGNOSTICS CONDITION 1
  :dasqlcode = DB2_RETURNED_SQLCODE,
  :datokencnt = DB2_TOKEN_COUNT,
  :datoken1 = DB2_ORDINAL_TOKEN_1,
  :datoken2 = DB2_ORDINAL_TOKEN_2,
  :datoken3 = DB2_ORDINAL_TOKEN_3,
  :datoken4 = DB2_ORDINAL_TOKEN_4,
  :datoken5 = DB2_ORDINAL_TOKEN_5,
  :dasqlerrdb = DB2_MESSAGE_ID,
  :damsgtext = MESSAGE_TEXT,
  :dasqlerrp = DB2_MODULE_DETECTING_ERROR,
  :rcount = ROW_COUNT,
  :dasqlstate = RETURNED_SQLSTATE;
```

Notes:

Exception Error handling may not be appropriate

- Error handling within object oriented applications is very critical. It is critical that SOA modules communicate when to back out transactions within their different processes to retain data integrity.

- Using the GET DIAGNOSTICS module helps the application understand all the rows that were processed along with their associated error.

It is vital to check ALL POSSIBLE error codes and messages. It is very surprising to see so many application modules not check for any error codes. Does your application check for errors? If so what percentage of modules?
Logging Common Errors

Commit Scope is a problem within java applications
- The connection auto-commit mode to false:
  - (Data) data.setAutoCommit(false);

Number of connections within the code executed
- Every module is creating a connection instance
  - Understand which modules get a connection to the database
  - Minimize the times a connection is created

Very important for transaction integrity, rollback and service scope analysis
- db.rollback();

Notes:
- Logging Common Errors
  - Error reporting and handling within object oriented module is important but logging the errors is vital to understand data integrity issues and what step your applications are failing. Shut off different components in your application and log the error codes. You would be surprised to find out sometimes no errors are reported.

- Commit Scope is a problem within java applications
  - Some modules auto-commit after a SQL statement because of their configuration, its settings or application coding. Understand what pieces need to be committed to preserve data integrity.
Cache Management

Data persistence has many names
- JPA, JDO, Hibernate, J2EE, POJO
- Content as Java Objects - ORM
  - Flattens out hierarchy of the content into objects
  - No matter what type of source

Storing data so you don’t have to wait
Storing data because don’t know SQL

Reason DBMSs are still around
- Standard access
- Process independent

Notes:

Data persistence has many names
- JPA, JDO, Hibernate, J2EE, POJO are all persistence models and all work well. The problem usually is that the persistence performs well for the applications it was designed for. When it is used for another transaction or application the data does not really fit, the persistence is too small or too big.

Reason DBMSs are still around
- Persistence should not be confused with database processes. Do not let your persistence carry any TO-BE-INSERTED or other flags that mimic database processing. Database handle locking and data integrity. Persistence speeds data retrieval and there is a big difference.
Cache Management

Foundation is your MVC pattern

Example:

```
1: getProductInfo(productID)
1.1: getProductInfo(productID)
1.3: removeProductInfo(productID)
1.4: addProductInfo(productInfo)
```

Good application pattern promotes performance

- Cache the right data
- Correct amount of data

Books:
- Good Design Patterns: Elements of Reusable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides
- Patterns in Java by Mark Grand

Examples

---

Notes:

Foundation is your MVC pattern

- Model View Controller is the standard object oriented Java application flow. Make sure your application utilizes this industry standard method. Socialize the MVC methodology and understand where the different MVC phases hand off different components for improved performance.

Good application pattern promotes performance

- Understanding the MVC pattern hand off phases helps the designer and the developer understand the amount and right data for minimizing the persistence within the system.
Cache Data Persistence

- Only cache reference data
  - Non transaction data/oriented
  - Should have only a single Unit-of-Work
  - Should be no locking concerns – don’t duplicate a DBMS

How much data to cache per transaction?
- What are the typical and extreme processing needs?
- How many peak or concurrent transaction?
- Where is the cache being done?
  - Client Cookies - App Server – Web Server – Host

Notes:

- Only cache reference data
  - Cache or persist only the reference data. Persisting transaction tempts developers to recreate DBMS functionality. By persisting reference data it can sometimes be read-only and unlocked throughout the transactions.

- How much data to cache per transaction?
  - Some developers want all their transaction data cached and then at transaction completion put into the database. This is a very bad idea because it can sometimes be too much data and cause deadlocking. This can also cause huge persistence requirements that cause memory problems during peak processing. Only cache what is needed.
Cache Data Persistence

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

Multiple the numbers out to understand the memory needed

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

Remember to leave enough headroom

- Server Cache requirements
- Server balancing, Fail-over and capacity planning

Notes:

- How much memory is required for the peak number of transactions?
  - The peak number of transactions should be at least guessed during the project design phase. Take that peak number of transaction guess and multiple it out by the size of the persistence that the transaction will each have. Does a server exist that can have that much memory for an application?

- Remember to leave enough headroom
  - While the servers continue to be bigger and DB2 10 for z/OS fully exploits 64-bit processing and memory allocations leave room memory after your application persistence requirements are calculated.
Cache Anomalies

- Transient-transactional instances – BEWARE!
- Read consistency and write consistency
  - Frequency of cache refresh should be within pattern
  - Integrity of the system, application and data
- Know what is process, cache or database
  - Each will have read and write consistency issues
  - What keys are critical – process, cache or database
- Know where the data is coming from!
  - Know what is transactional

Notes:

- Transient-transactional instances
  - Beware of these transient-transactional instances because they can cause data integrity issues and lose important data within your application. The problem usually occurs when a single transaction tries to share a persisted object with another transaction and an error occurs. Once an error occurs it is very hard to rollback something that is in memory only.
- Know what is process, cache or database
  - Understand the different types of objects and where their origin is within the application transactions. This knowledge will help the developer understand their status before and after transactions. This is important so that everyone knows what the data should look like when it is rolled back.
Many servers manage the Beans

Distributed processing considerations
- Make sure your outside methods are good performance partners
- Have your performance critical path as local as possible

Component should communicate with a limited number of other components
- Components are local or remote or distance
  - Same server, another server in data center or remote data center

Biggest memory users are always cleaned up - java.lang.OutOfMemoryError
- ResultSet
- Vectors & Arrays
- HashTables
  - All within static classes
- Event Listener

Notes:

Distributed processing considerations
- Error reporting and handling within object oriented module is important but logging the errors is vital to understand data integrity issues and what step your applications are failing. Shut off different components in your application and log the error codes. You would be surprised to find out sometimes no errors are reported.

Biggest memory users are always cleaned up
- There is always the transactions that work that clean up their memory usage. Verify that the transactions that fail are cleaned up. Applications that use large arrays, vectors and Result Sets should get special attention.
pureQuery programming advantages

- Single pureQuery API can work with any java
  - Improved developer productivity by generating data access modules through Data Studio
    - XML, in-memory, simple or complex SQL
- Uses standard JDBC - so portable across databases
  - Produces true data ‘objects’ for the OO designs
    - Goes beyond ‘get’ and ‘set’ routines
- Can use all other types of java modules
  - Generic, XML, JSON or custom coded

Notes:

- PureQuery is the best way for applications to perform static SQL in Java applications.
- In this inline style example, it shows the connection information, the SQL and the resultSet iterator to retrieve all the database table SQL data. This nice routine quickly retrieves the data and presents it.
- Note the generated AutoCommit(false) within the code. Special analysis and handling of the commit scope of a module needs to be done to make sure the module or service is handling the work properly.
- Also note the rollback within the module that could be referenced if there are no rows retrieved through the SQL statement. These generated statements are fine the way the code is generated and works now but may need to changed if the module or transaction logic changes.
### SQL Capture Feature

- Existing dynamic JDBC application to Static SQL
  - Improve security with Static SQL
  - Improved performance
  - Improve debugging & maintenance

- Run existing application and capture SQL
  - No changes necessary to the application
    - Add pureQuery libraries to the external jars
    - Execute application
    - Bind capture file
  - Run static and new statements can be
    - Run dynamically or
    - Rejected and not executed

### Notes:

- Existing dynamic JDBC application to Static SQL
  - Converting dynamic SQL to static SQL is easy with pureQuery. By tracing the applications the SQL is captured and then bound to make it static. This improves performance by eliminating the security, object, verification and developing an access path.
Cross Reference SQL with source module

- Impact analysis for java SQL modules
  - Improves debugging analysis
  - SQL traceability back to the source module

- Improve problem SQL faster
  - Trace back to source module and improve SQL

- Understand module and data dependencies
  - Through DS code outline display
  - Through SQL DB2 Catalog dependencies
  - Debugging and performance testing is much easier

Notes:

- Impact analysis for java SQL modules
  - Having static SQL helps everyone understand the dependencies, improve SQL performance and debug their application faster. PureQuery provides extraordinary performance with no changes or impact to the application code.

- Within Data Studio with pureQuery you can cross reference the SQL statements to the source module. Within Outline mode of the module you can quickly see the SQL reference table and the various columns used by the module.
**Java DB2 Developer Performance Best Practices**  
By Dave Beulke

**pureQuery programming advantages**

- Static Bind removes java dynamic overhead
  - Reduces the Dynamic Statement Cache
  - Reduces EDM Pool overhead
  - Allows better memory tuning capabilities

- Faster overall execution by .001 per transaction
  - 20 million java dynamic transactions per day
  - 333 minutes of CPU saved per day!
    - Chargeback $10-$38 per minute = $12,654.00/day
    - 250 business days > $3 million per year
    - Reduced CPU demand > 5.5 hours of CPU per day
      - Your mileage may vary

**Notes:**

- Static Bind removes java dynamic overhead
- Static SQL removes the dynamic Statement Cache, EDM Pool and memory overhead requirements. This is paramount for memory constrained systems.
- Static SQL also performs better saving clients huge amounts of CPU.
- The most important advantage of Data Studio and pureQuery are its capabilities to do a static bind for java SQL applications. In addition to all the static bind advantages highlighted on the previous slide, having a static java application environment helps the system reduces memory allocations. Since the workload is static it no longer requires a large system Dynamic Statement Cache, a large EDM Pool or a large number of server connections.
- Static SQL and static processes within DB2 system reduce the system resources required to execute the SQL processing. For example it can reduce the overall CPU demand and result in significant charge back savings within an enterprise.
- Retry logic is a common mistake that I see more and more in client installations. Try to remove any retry logic in any module. The application is only hiding a performance or programming issue that needs to be addressed.
Questions for your system administrator

- How many SOA transactions are we configured for?
  - We are expecting 5M transactions doing 1M updates, 2M inserts with 15M-20M page views per day

- What are the hardware components of the server/LPAR?
  - Number of CPUs – Cores and the speed of the CPUs
  - Memory allocation for the LPAR
  - I/O connection speed for the Network

- What monitoring facilities are being used?
  - What CPU load statistics are available?
  - Current network traffic utilization is ???

- Web server platform software
  - Operating system, server level and patch level(s)

Questions for application development

- What framework are you using? For persistence?
  - How much persistence per user, session, transaction or idle thread?

- What automated testing tools are going to be used?
  - What are the performance expectations of the web services?

- Is the web content new or where does it exist now?
  - The services are 75% dynamic returning 500 rows of data each

- What are a list of the error conditions that are produced?
  - How can I help with the testing of the database conditions?

- What retry logic is within the web services?
  - What methods perform the retry logic? How many times?
  - How are the methods insuring transaction rollback/commit integrity?