Database 3: Improving Database Processing

Charlie Arehart
Founder/CTO Systemanage
carehart@systemanage.com
SysteManage: our practice makes you perfectSM

www.systemanage.com

Agenda

- Eight Measures of Architectural Quality
- DB Performance and Scalability:
  - Query Caching
  - BlockFactor
  - Indexes
- DB Reliability:
  - Constraints
  - Triggers
  - Transaction Management
  - Bind Parameters
- DB Extensibility and Maintainability:
  - Stored Procedures
- The Other Measures of Quality
- Where to Learn More and Q&A

www.systemanage.com
Part 3 of 3

- This seminar is part 3 of 3 presented today
  - Previous two were in conference “beginner” track
- Part 3 is in “Advanced” track
  - Won’t lose those who’ve made it this far
  - May discuss things that advanced developers have already heard (more than once)
    - May hear it in a different way today
    - Or leave thinking about it differently than before
    - May simply trigger your putting them into effect
- More than just “how to”
  - Focus as much on why, architectural perspective
  - 50% is CF-specific, rest meaningful to other developers

Databases & Overall Architecture

- Database processing is just part of your overall system and information architecture including:
  - Web server, CF server, DB server
  - As well as DB design, SQL code, CF code
- Should evaluate entire system in terms of quality
Eight Measures of Architectural Quality

- Sun Microsystems defines eight measures of architectural quality
  - Offered in regard to Java Enterprise (J2EE) platform
  - Apply just as well to considering CF/DB architecture

<table>
<thead>
<tr>
<th>Performance</th>
<th>Maintainability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalability</td>
<td>Availability</td>
</tr>
<tr>
<td>Reliability</td>
<td>Security</td>
</tr>
<tr>
<td>Extensibility</td>
<td>Manageability</td>
</tr>
</tbody>
</table>

- Create a backdrop considering various techniques to improving database processing

Performance & Scalability

- **Performance:**
  - A measure of the effectiveness of your application (and database design and server platform), in terms of response time, transaction throughput, and/or resource usage
  - Always involves tradeoffs of cost/benefit

- **Scalability:**
  - Ability to support the required quality of service as load (number of users, volume of data) increases
  - Today’s small application (or your tests) may not reflect future
Reliability, Extensibility & Maintainability

➢ Reliability:
  – Assurance of the integrity and consistency of the application and all its transactions
  – May suffer with increased load
    • But ensuring reliability may negatively affect scalability

➢ Extensibility
  – Ability to add/modify additional functionality without impacting existing functionality
  – Given the high effort involved in maintenance, this is more important than many recognize

➢ Maintainability
  – Ability to correct flaws in the existing functionality without impacting other components/systems
  – Includes modularity, documentation

Other Measures of Architecture

➢ Not really the focus of topics in this seminar
  – Some tips at conclusion

➢ Availability
  – Assurance that a component/resource is always available
  – Can be enabled with redundancy and failover

➢ Security
  – Ability to ensure that the system has not been compromised
  – By far the most difficult to address
  – Involves protecting confidentiality, integrity, availability, more

➢ Manageability
  – Ability to manage the system in order to ensure continued health with respect to previous measures
  – Involves both monitoring and ability to improve systemic qualities dynamically without changing system
Addressing the Challenges

- One approach to scalability/performance concerns:
  - Add more memory/processors
    - Tends to have good impact on all parts of system with little negative
Another solution:
- Distribute processing across multiple servers
  - May be simply segregating CF Server and DB server
    - Again, generally a very good idea
  - May involve creating cluster for web server
    - Tends to add complexity to design and implementation
Improving Design & Implementation

- May be able to improve performance/scalability without new hardware
  - Features in DB design, SQL, and CF can help
  - Many are useful even in relatively small applications
    - Should design for performance, keeping in mind cost/benefit tradeoffs

- Design/implementation choices impact other facets
  - Reliability, extensibility, maintainability, security

- Some features revolve around design of database
  - Most simply involve more effective use of db

DB Processing: Key for CF App

- DB processing is single biggest bottleneck in most CF apps
  - Sadly, many will blame CF itself
  - Usually, the problems are preventable

- Typical things that can degrade quality of DB processing:
  - Poor database and table design
  - Use of non-relational tables
  - Use of incorrect data types
  - Poorly written SQL
  - Lack of indexes
  - Not using stored procedures, triggers
  - Repeatedly requesting the same data
  - And much more

- Previous talks have addressed some of these
  - Today we’ll cover some of the rest, and more
DB Performance and Scalability Solutions

- Some DB performance and scalability solutions:
  - Query Caching
  - BlockFactor
  - Indexes

Repeatedly Requesting the Same Data

- Many web apps suffer from unnecessarily requesting the same data over and over
  - Doesn’t really matter if DB is well-designed

- Examples include:
  - Providing drop-down list of states on a reg. form
    - When did we last add a new state?
  - A company phone directory
    - How often are employees added/removed?
  - Reporting management information
    - Does it need to be accurate to the second?
  - Showing search results n-records at a time
    - Search criteria doesn’t change for “next 10” records
Query Caching

- **CF provides two means of caching query results for re-use**
  - **Variable-based query caching**
    - Leverages ability to store any variable in server, application, or session scope
    - Since a query resultset is a variable, it can be scoped as such
    - May surprise those who never thought of it
  - **Time-triggered query caching (a.k.a. “query result caching”)**
    - New attributes for CFQUERY to indicate that any code executing that query should create/use cached copy for given timeframe
    - Will show how to use each of these

- **Also look into CFCACHE and CFSAVECONTENT tags**
  - These cache the entire CF page or page portions
  - Not covered in this seminar but important to performance

Variable-based Query Caching

- **ColdFusion offers 3 scopes for storing persistent variables:**
  - **Session scope**
    - Persists for the life of a single user’s session until server is restarted or session times out
  - **Application scope**
    - Persists for all users of a given application until server is restarted
  - **Server scope**
    - Persists for all users of entire CF server until server is restarted

- I’ll have to presume for this class that you understand setup and use of these
Variable-based Query Caching

- Just as we can assign variables to these scopes
  - we can declare that a CFQUERY NAME value use a persistent scope, as in:

```cfml
<CFQUERY DATASOURCE="ProdPrsnl" NAME="application_GetStates">
  SELECT State, StateAbbrev
  FROM States
</CFQUERY>
```

- Now, this query result set is stored with all other application variables
  - Can be referred to by any code anywhere in this application
    - meaning, under control of same CFAPPLICATION

```cfml
<SELECT NAME="state">
  <CFOUTPUT QUERY="application_GetStates">
    <OPTION VALUE="#StateAbbrev">#State#
  </CFOUTPUT>
</SELECT>
```

Avoid Recreating Cached Resultset

- Once cached, query shouldn’t be executed again
  - At least not until the data it reflects changes

- How to avoid executing query if already “cached”?
  - Test if query already exists, with IsDefined()

```cfml
<CFIF NOT IsDefined("application_GetStates")>
  <CFQUERY DATASOURCE="ProdPrsnl" NAME="application_GetStates">
    SELECT State, StateAbbrev
    FROM States
  </CFQUERY>
</CFIF>
```

- Now this query will be executed only once but be available for the life of its indicated scope
Where to Create/Update Variable-based Cached Query?

- Where might it be sensible to put query creation code to be cached for all app users?
  - Application.cfm

- When should the query be re-executed?
  - Whenever its underlying database table changes
    - In whatever template performs changes to data
    - Only dilemma: if code outside your control updates the database

- Consider use of session scope to hold a user’s search results over many “next n” pages?
  - Create/cache it on the search action page

Another Challenge: Locking Issues

- Shared scope variables should be locked when written to
  - Should probably instead code query as:

    ```
    <CFQUERY DATASOURCE="ProdPrsnl" NAME="GetStates">
      SELECT State, StateAbbrev
      FROM States
    </CFQUERY>
    <CFLOCK SCOPE="APPLICATION" TYPE="EXCLUSIVE" TIMEOUT="5">
      <CFSET application.GetStates= GetStates>
    </CFLOCK>
    ```

  - Note use of “exclusive” type of lock
    - Not wrapping query in lock because you should avoid holding locks any longer than needed
      - Why make lock wait for query to run?
      - It should just be locked for however long it takes to assign the result set to the persistent variable
Locking Issues (cont.)

- Should also lock when reading
  - Could code CFOUTPUT loop as:
    ```html
    <CFLOCK SCOPE="APPLICATION" TYPE="READONLY" TIMEOUT="5">
    <SELECT NAME="state">
      <CFOUTPUT QUERY="application.GetStates">
        <OPTION VALUE="#StateAbbrev#">#State#</OPTION>
      </CFOUTPUT>
    </SELECT>
    </CFLOCK>
    ```
  - Note use of “readonly” type of lock
  - Note too that TIMEOUT attribute in each case has nothing to do with how long this lock will take
    - It’s how long this lock will wait for lock being held by others
  - Could instead assign cached result to local variable within lock (locking just that assignment) and loop over that
    - Will likely release lock faster (for benefit of others updating same-scoped variables)
    - Comes at cost of creating local copy of resultset each time

More Challenges

- More challenges of variable-based cached queries
  - You’re responsible for managing cache (creating, updating)
    - To delete cache, delete variable
      ```html
      <CFSET x = StructDelete(application,"GetEmployees")>
      ```
  - Be careful about creating too many
    - They’re just stored in memory
      - Large queries could take a lot of memory
    - No way for admin to limit memory used
More Challenges

- **More challenges of variable-based cached queries**
  - You’re relying on previous code to have created the cache, such as application.cfm in one example
    - Can look confusing to developers unfamiliar with this form of caching
    - And what if it didn’t exist? Hadn’t been run?
  - Consider how CFPARAM creates a variable only if it doesn’t exist
    - Wouldn’t it be nice if you could just do the query where you need it?
    - and if it hadn’t been cached, it would be?
    - And, further, it would automatically re-cache itself at defined intervals (after x minutes, or after certain date)

- Next alternative to query caching solves these problems

Time-triggered Query Caching: CACHEDAFTER

- Referred to in “Certified CF Developer Study Guide” as “Query Result Caching”
- Does not involve creating variables
  - Instead, specify a caching attribute on CFQUERY
    - CACHEDAFTER or CACHEDWITHIN
  - Example:
    ```
    <CFQUERY DATASOURCE="ProdPrsnl" NAME="GetSales"
      CACHEDAFTER="09-01-01 10:00 pm">
    SELECT * FROM SalesStats
    </CFQUERY>
    ```
  - This would cache the result the first time the query is run after specified date/time (and use the cache from then on)
    - Meant to be used with fixed date/time, in the future
    - Might be useful when you know data is updated at 10pm
**Time-triggered Query Caching: CACHEDWITHIN**

- **CACHEDWITHIN works differently**

  ```cfml
  <CFQUERY DATASOURCE="ProdPrsnl" NAME="GetEmployees"
  CACHEDWITHIN="#CreateTimeSpan(0,0,5,0)#">
  SELECT * FROM Employees
  </CFQUERY>
  ``

  - This would cache the result the first time the query is run and reuse the cache each time query is executed
    - until specified timespan has passed since it was first cached
    - will re-cache it the next time it's run after specified timespan
    - Meant to be used with relative time span
      - Can be specified in either days, hours, mins, secs
    - Useful to cache for a specific amount of time from the first time it's cached
    - CFML reference mistakenly indicates this should "define a period of time from the present backwards"

**Time-triggered Query Caching: Issues**

- **Can observe if query was taken from cache**
  - If debugging is turned on, query time shows "cached query"
    - Note that CFQUERY.ExecutionTime variable does NOT show this value
      - Shows "0", doesn't always mean it was a cached query

- **Important difference from variable-based caching**
  - Query remains where it normally would appear
  - No need to test existence, no shared variables used, no need to worry about <CFLOCK>
Time-triggered Query Caching: Dynamic Queries

- A single CFQUERY may generate multiple cached results
  - If SQL is built dynamically, each unique SQL statement is cached separately
    - Consider search action page driven by form fields
      - Same CFQUERY with different resulting SQL will create separate cached result
  - Pro
    - Means more potential to benefit from cache
  - Con
    - Means lots of cached results could be created

Time-triggered Query Caching: Admin Settings

- Time-triggered caching is governable by admin settings
  - Can restrict total number of cached queries allowed
  - Limit the maximum number of cached queries on the server to xxx queries
    - When the limit is exceeded, oldest query is dropped and replaced
    - Defaults to 100 on installation of CF
  - Can disable this sort of caching by setting to 0
Time-triggered Query Caching: Sharing Cached Results

- Mentioned previously that unique SQL in same query will result in different cached results
  - Conversely, and perhaps unexpectedly to many, cached result for given SQL can be reused by another CFQUERY
    - To reuse another query’s cached result, query must have identical SQL and DATASOURCE
      - And, if specified, identical DBTYPE and Login info
    - Doesn’t need to have same query NAME
    - Of course, doesn’t need to be in same template
      - Nor even in same application

More About CachedAfter

- CF docs are very sparse about CACHEDAFTER
  - Both the docs and the Certification Study Guide say it supports only a date
    - Will support a date and time
      - Can specify date as any valid CF date, then add time
        » such as “09/01/01 10:00pm” or “09-01-2001 22:00”
    - To cache each day as of 10pm, use
      - CACHEDAFTER="#{dateformat(now())}# 22:00"
    - Can’t, however, just specify a time
Another Performance Factor: BlockFactor

- BLOCKFACTOR gets a lot of press by some as important performance factor
  - May not bring value for most
  - Also easily misunderstood
- When CF and database communicate to create result set, may transfer only one record at a time
  - Applies to some DB drivers
    - ODBC, Oracle according to docs
  - BLOCKFACTOR is an attribute on CFQUERY
    - Allows specifying a number of records to transfer at a time
    - Does NOT control HOW MANY records are retrieved
  - If not supported by DB driver, won’t cause error
    - but could degrade performance
  - If supported but set too large, could degrade performance
  - Many feel it’s best to not set at all

About DB Column Indexes

- When column in table is searched, does the DBMS look at each record in entire table, one at a time?
  - Yes, if the column is not indexed
  - Think of index as similar to a book’s index
    - Just as we can find info quickly, so can DBMS
    - Can have dramatic impact on performance of queries
  - In small tables, lack of index may not be noticeable
    - Then again, with more users doing more queries, could become a problem
  - Whether a column is indexed is optional
    - Except that primary key is always indexed
    - Should consider adding index to columns frequently searched
      - May also improve sorting by a given column
  - Beware: indexing a column isn’t always a good idea
Indexing Cautions

Before rushing off to create indexes on too many columns, consider a few cautions:
- Each index requires time to be maintained during record insert/update operations
- Not all data is suitable for indexing
  - Depending on indexing technique used by DBMS, data without many unique values may not benefit
    - State may not be good index while lastname is
- Indexed data does add to storage requirement for DB

Creating/Adding Indexes

To add an index to a table for a given column

```
CREATE INDEX indexname
ON tablename (columnname)
```

- `Indexname` must be unique within given table
- Can create index before or after populating table with data

CF and even SQL coding isn’t typically changed by adding indexes
- Just see improved query performance (at tradeoff of aforementioned cautions)
DB Reliability Solutions

- Some DB reliability solutions:
  - Constraints
  - Triggers
  - Transaction Management
  - Bind Parameters

About DB Column Constraints

In Database 2 seminar, we learned about inter-related tables and how to create JOINs between them

- Learned that, in this example, values of Employees.DeptID reflect those in Departments.DeptID
  - Can be used to lookup Dept name by way of joining them
- What ensures that the only values stored in Employees.DeptID come from Departments.DeptID?
  - Many developers don’t take steps to ensure this
Problems Managing Related Table Values

- Others take responsibility to manage it themselves
  - Trying to maintain this form of integrity is challenging
  - Need to do it everywhere data may be updated
  - Also need to do it for updates/deletes
  - Take effort to code, then execute, such checks
- Far better to let DBMS manage this itself

Creating/Adding Constraints

- Can create *constraints* for and between such related table columns

  ```sql
  ALTER TABLE Employees
  ADD CONSTRAINT FK_DeptID
  FOREIGN KEY (DeptID)
  REFERENCES Departments (DeptID)
  ```

  - With this in place, an attempt to insert invalid value for DeptID in Employees (a value not in Departments.DeptID column), DB will throw error
  - Can catch this error in CF with CFTRY
    - Surround CFQUERY doing insert/udpate
About Unique Constraints

- Similar dilemma arises when you want unique values for a given column
  - May want to prevent multiple records with same email address
    - Learned in previous seminar that primary key values are guaranteed to be unique
    - But what if column (like email) is not the primary key?
  - Again, could try to manage this yourself
    - Doing test before doing insert/update to ensure email address value doesn’t already exist
  - Or could have DBMS manage it, with unique constraint
    - May be created with CREATE UNIQUE INDEX or with another kind of CONSTRAINT

About Check Constraints

- Still another reliability option is that some databases allow creation of Check Constraints
  - These are defined for a given column to ensure values meet some defined criteria
  - Examples include:
    - minimum/maximum values
    - range of values
    - List of possible values
Visually Defining Indexes, Constraints

- SQL statements will work for nearly all DBMS’s
  - Many DBMS’s offer visual interface for managing these
    - MS Access “Design Table” and “Tools>Relationships” features
    - SQL Server Enterprise Manager
    - And more
  - Again, be aware that in many instances, the defaults are to not define indexes, constraints
    - If you’d like to use them, you may need to add them

Ensuring Further Data Reliability

- We know that constraints can ensure that data meets certain criteria during insert/update
- May need to ensure further integrity
  - May want to convert data to uppercase during insert/update
  - May need to write data to another table on insert/update
    - keeping accountbalance column in account table updated for each deposit/withdrawal tracked in transaction table
  - May need to check data in another table before allowing insert/update
Triggers

- Some DBMS's allow creation of triggers to perform these sort of integrity checks and cross-table update
  - Specified in form of SQL statements
  - Stored in database, associated with given table
  - Typically can define separate triggers to act upon insert, update, and/or delete against that table
  - Syntax will differ between DBMS's. An example:

```
CREATE TRIGGER triggername
ON tablename
FOR INSERT|UPDATE|DELETE
AS
UPDATE tablename SET columnname=UPPER(columnname)
WHERE tablename.columnname = INSERTED.columnname
```

- When performing similar actions, constraints typically execute more quickly than triggers (use them instead)

Transaction Management

- Multiple users can (and generally do) update data in databases at the same time
  - Transaction processing prevents them updating the exact same data at the same time
  - Also allows a group of related updates to be packaged such that if they don’t all succeed, none will succeed

- Generally controlled by the DBMS for us
  - We can influence it from within CF by way of the CFTRANSACTION tag

- See Chapter 19 of Certification Study Guide for more details and code samples
Grouping Updates

- When multiple updates must take place, otherwise none should take place, use CFTRANSACTION

  ```html
  <CFTRANSACTION>
  <CFQUERY ...>
  UPDATE Checking SET Balance=Balance -100
  WHERE AccountID = 1234
  </CFQUERY>
  <CFQUERY ...>
  UPDATE Savings SET Balance=Balance+100
  WHERE AccountID = 1234
  </CFQUERY>
  </CFTRANSACTION>
  ```

- This simplest and oldest form simply ensures that if the first update fails, the second will as well
  - Called backing out or “rolling back” the first update
  - Up to the database to handle the rollback
    - More advanced DBMS will handle rollback even after recovering from crash of DB server that may have caused transaction to fail in the first place

Isolation Levels

- When performing a group of transactions, need to be careful about other users reading the data we update, and vice-versa
  - Databases generally define up to 4 isolation levels that can influence these sort of cross-user locks, from
    - Serializable (default)
      - Can indicate that no reads/updates by others take place during our update
    - Through Repeatable_Read and Read_Committed
      - Not supported by all DBMS’s
    - Read_Uncommitted
      - Or can indicate that we don’t care if others are reading/updating

- We can specify a desired isolation level with CFTRANSACTION ISOLATION attribute
Programmable Commit/Rollback

- Mentioned that CFTRANSACTION would rollback all updates if any failed
  - Didn’t mention, but COMMIT takes place at end of transaction
    - Commit tells DBMS to consider update finished
      - CFQUERY updates outside CFTRANSACTION also do COMMIT at end of CFQUERY
  - Release 4.5 added ability to perform BACKOUT (and COMMIT) programatically within transaction
    - `<CFTRANSACTION ACTION="Backout|Commit"/>`
      - This tag is designed to be used within other CFTRANSACTION tag
        - Doesn’t allow embedded tags of its own, but needs to be closed to avoid confusion with surrounding CFTRANSACTION
        - Could use closing `<CFTRANSACTION>` tag or just closing slash at end of tag, as above

Using Bind Parameters

- ColdFusion is a loosely typed language
  - Numbers considered string until used for math
- Databases are strongly typed
  - Column expecting numbers will want numbers
  - But CF will be passing a string that looks like number
    - Database can do conversion to fix that
    - But we can help the database to know the datatype
    - Can help performance by specifying bind parameters

```cfml
<CFQUERY ...>
  SELECT * FROM EMPLOYEES
  WHERE EmpID = "<CFQUERYPARAM CFSQLTYPE="CF_SQL_INTEGER" VALUE="#url.empid#"/>
</CFQUERY>
```
Bind Parameter for Reliability

- When passing form or URL variables on some SQL statements (with some DB drivers)
  - User can pass strings to add unexpected SQL
  - Bind parameters can stop that
    - If expecting to bind numeric data and user passes text (including SQL statements), bind will strip them

DB Extensibility Solutions

- One DB extensibility and maintainability solution:
  - Stored Procedures
About Stored Procedures

- We typically specify SQL statements within CFQUERY tags within our CF templates
  - What if multiple templates would execute same SQL?
  - While we could use CFINCLUDE to re-use this code, there are options in most DBMS’s to store that code in the DBMS
  - Then would call upon it much like we call a custom tag
    - But instead of executing CF code, it just executes SQL
  - Each DBMS has its own language for the SQL to be used for such stored procedures, for instance:
    - Oracle: PL/SQL
    - SQL Server: T/SQL

Creating Stored Procedures

- Other benefits:
  - Stored procedure typically compiled and stored in DBMS
  - Parameters can be passed to procedure to be used in SQL execution
  - Can create and use variables, pass data among statements, and perform conditional processing within the SQL
  - Can execute multiple statements in one procedure
  - Stored procedure may be able to return multiple record sets
  - Example might be:

```
CREATE PROCEDURE procedurename in/outparms
ON tablename
AS
    SQL statements

```

- Can create Stored Procedures using CFQUERY
  - More typically created in DBMS, managed by DB Admin
Executing Stored Procedures

- Once stored in a DBMS, we can execute the stored procedure by calling upon it, in either:
  - CFQUERY
  - CFSTOREDPROC
- Procedure executes in the DBMS (just as if we’d passed the SQL)
- Returns one or more result sets to process (just as with normal CFQUERY)
- Working with SPs in Oracle has complications
  - See Macromedia Knowledge Base articles
- Though MS Access doesn’t have stored procedures, there are ways to fake it
  - use Access “parameter queries” feature
  - See my CFDJ article from Oct 99: “Stored Procedures in Access? Yes Indeed”

Other Measures of Architecture

- **Availability**
  - Assurance that a component/resource is always available
  - Can be enabled with redundancy and failover
    - Some may know that CF Servers can be clustered
  - From DB standpoint, no built-in CF features
    - On simple level, could use CFTRY to catch failures and attempt query/update of alternate DB
    - On larger level, enable backup/restore
      - Often ignored by CF developers
    - Replication may play a part
      - Some DBMS implementations better than others
Other Measures of Architecture

- **Security**
  - Ability to ensure that the system has not been compromised
  - By far the most difficult to address
  - Involves protecting confidentiality, integrity, availability, more
  - Will be highly influenced by DBMS, configuration, perhaps programming

- **Manageability**
  - Ability to manage the system in order to ensure continued health with respect to performance, scalability, reliability, availability and security
  - Involves both monitoring and ability to improve systemic qualities dynamically without changing system
    - ColdFusion 5 offers monitoring features to observe system, servers, and even successful execution of probing templates
    - Most DBMS’s and operating systems also offer monitoring tools
Some Other Tidbits for You to Investigate

- Query of Queries
- VIEWs
- DB Security management

Where to Learn More

- **Version 5 CF manuals:**
  - Installing and Configuring ColdFusion Server
  - Developing ColdFusion Applications
  - CFML Reference
- **Books by Ben Forta:**
  - Teach Yourself SQL in 10 Minutes
  - Certified ColdFusion Developer Study Guide
  - ColdFusion Web Application Construction Kit
  - Advanced ColdFusion Development
- **Many other CF and SQL books available, including**
  - Practical SQL Handbook (new edition available)
  - SQL For Smarties (any Joe Celko book)
Contact Information

Contact for follow-up issues
- **Email:** carehart@systemanage.com
- **Phone:** (301) 604-8399
- **Web:** www.systemanage.com

Also available for
- Training (custom or pre-written)
  - CF, DB, Jrun/J2EE, Javascript, wireless, and more
- Consulting (very short-term engagements)
  - best practices, architecture, setup, troubleshooting, etc.
- Developer Group Mentoring, and more

Q&A

?