Understanding, Improving and Resolving Issues with Database Prepared Statements

What really happens when we do/don't use cfqueryparam?

Charlie Arehart, Independent Consultant charlie@carehart.org

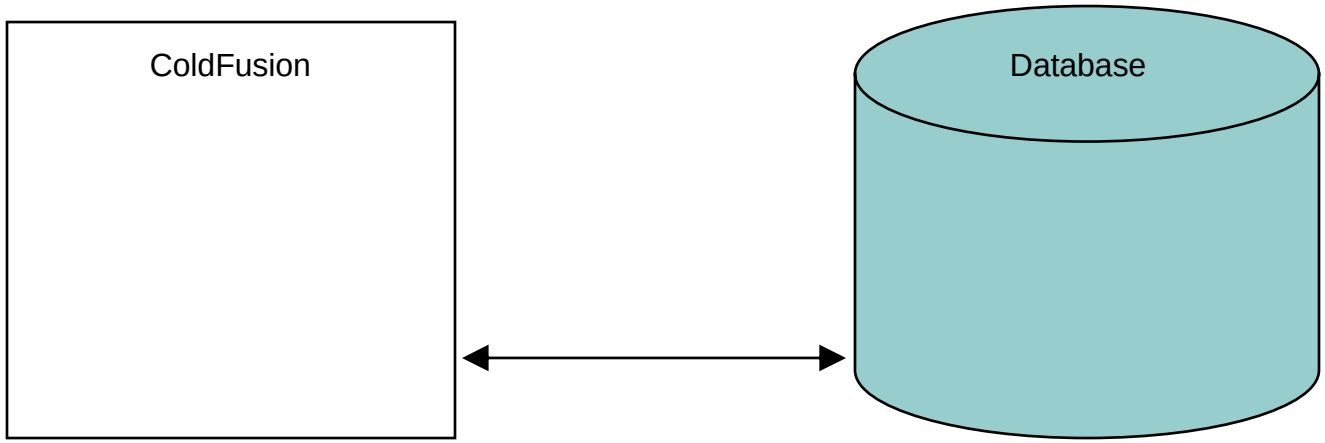
Topics

- Thinking through DB request processing
  - Understanding DB query plan creation
  - Understanding DB prepared statements
  - Influence of CFQUERYPARAM
  - Options for query plan/statement cache
- Monitoring & Performance implications
- Comparison of SQL Server & MySQL
  - Most applies to all DBMSs and versions
  - Will explain differences, where I know them

About Charlie Arehart

- Independent consultant since April 2006
- 10 yrs CF experience (25 in Enterprise IT)
  - Member, Adobe Community Experts
  - Certified Adv CF Developer (4 - 7), Cert. Adobe Instructor
  - Frequent speaker to user groups, conferences worldwide
  - Contributor to Ben Forta's CF8 books, past *ColdFusion MX Bible*
  - Run the Online ColdFusion Meetup (coldfusionmeetup.com)
  - Living in Alpharetta, Georgia (north of Atlanta)
- Web home at www.carehart.org
  - Hosts 175+ blog entries, 50+ articles, 70+ presentations, more
  - UGTV: recordings of presentations by over 100 CFUG speakers
  - AskCharlie: per-minute telephone & web-based CF support

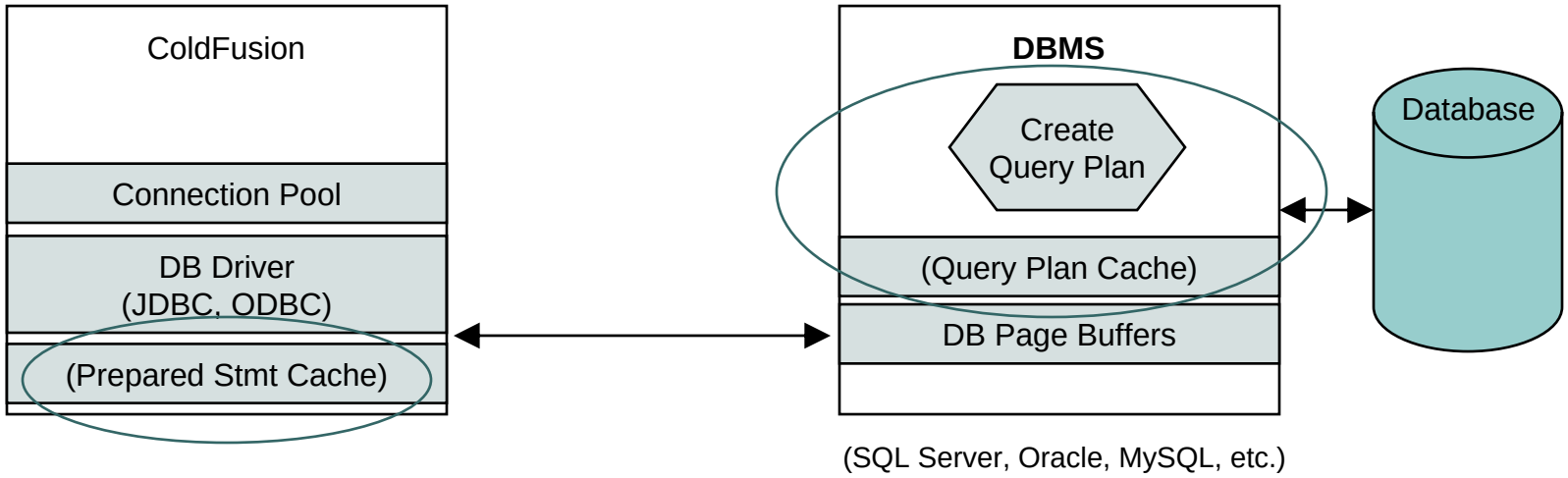
Thinking through DB request processing



A Simple Perspective

Of course, it’s more than “just a database”

Thinking through DB request processing



A More Evolved Understanding

CFQUERYPARAM can affect the last three

Understanding DB query plan creation

- When request reaches DBMS, it needs to calculate best way to execute the SQL
  - Influenced by indexing, row count, value distribution within a column, and more
- This takes time, separate from getting the data
  - Is good that DBMS does this for us
  - But we can influence *how much* it does this
- Let's see simple query in action
  - Using SQL Server Profiler
    - Stored Procedures event (yes, even for dynamic SQL)
      - RPC:Starting/Completed; SQL:StmtStarting/Completed
    - Performance event
      - Showplan All/XML for Query Compile,
  - Same concept applies in Oracle, MySQL, etc.

Understanding DB prepared statements

- DB creates new query plan for each unique SQL statement
  - Using prepared statements, we can cause DBMS to reuse previous query plan
    - If a given statement has variable values
    - Also reused by multiple users, if same SQL statement
  - Could save significant time (for query compilation)
  - Can be requested in CFML code via CFQUERYPARAM
  - Can also be *performed automatically by DBMS* (more later)
- Let's see it in action
- Notes
  - Concern with SELECT * and CFQUERYPARAM
    - DB driver stores metadata reflecting columns in play at plan creation
    - If you add columns after that, CFQUERY could fail
  - Can be done for more than just SELECT
    - Just as useful for INSERT, UPDATE, DELETE
      - If given SQL statement executed often
      - But varies by some input parameter
  - Offer security benefits as well
  - Can provide performance benefit (not always, as discussed later)

Performance implication of prepared statements

- Could save significant time (for query compilation)
- Could also affect memory
  - Since you're reusing query plans for varying arguments
- Prepared statements also known as
  - Using "bind variables", parameterized queries
  - Opposite of "direct" or "immediate" execution

b,c

DB query plan caching

- Some DBMS' saves query plans for reuse
  - Called variously "plan cache", "procedure cache", "statement cache"
  - All statements, whether using prepared statements or not, are cached for reuse
  - Like any cache, want high ratio of hits/misses
- Let's see it in action (see next slide)

DB query plan caching in SQL Server

- Can view effort for each query in SQL Profiler
  - Stored Procedures event (again, even for dynamic SQL)
    - SP:CacheHit/CacheInsert/CacheMiss/CacheRemove
- Can view aggregate data in new DMVs in 2k5
  - sys.dm_exec_cached_plans
  - sys.dm_exec_sql_text
- Also available data in PerfMon
  - SQLServer:PlanCache event
    - “Cache Hit Ratio” for “SQL Plans” instance
    - “Cache Object Counts”/”Cache Objects in use”
- Can clear cache with DBCC FREEPROCCACHE

SQL Server Profiler w/out cfqueryparam

- First call when cache cleared
  - SP:**CacheMiss** (@1 varchar(8000))
    SELECT * FROM [person].[address] WHERE [city]=@1
  - SP:**CacheInsert**
    select * from person.address
    where city='Dallas'
  - SQL:BatchCompleted
    select * from person.address where city='Dallas'
- Second call
  - SP:**CacheHit**
    select * from person.address where city='Dallas'
  - SQL:BatchCompleted
    select * from person.address where city='Dallas'

SQL Server Profiler w/ cfqueryparam

- First call when cache cleared
  - SP:**CacheInsert** (@P1 varchar(8000))
    select * from person.address where city=@P1
  - SP:StmtCompleted
    select * from person.address where city=@P1
  - RPC:Completedexec sp_execute 1,'Dallas'
  - Exec Prepared SQL
- Second call
  - SP:**CacheHit** (@P1 varchar(8000))
    select * from person.address where city=@P1
  - SP:StmtCompleted
    select * from person.address where city=@P1
  - RPC:Completedexec sp_execute 1,'Dallas'
  - Exec Prepared SQL

Performance implication of query plan caching

- The query plan cache requires memory
  - So using it most effectively can be important
- Failure to use prepared statements (CFQUERYPARAM) means unique entries for statements varying by some arg
  - Prepared statements would reuse one cached statement
- Regarding multiple users (different logins)
  - if no schema used on tablename, SQL Server will not reuse same plan
    - since different users could have different default schemas
    - “Select xxx from employees” versus “select * from dbo.employees”
  - Could cause much larger volume of query plan creation
- DMV analysis can help address evaluating performance
  - For determining which so parameterize, find SQL statements that look alike
    - But vary by some one parameter
  - For multiple user/no schema problem, will find entries with identical sql

SQL Server 2k5 DMV SQL

- SQL to view top 100 used compiled plans

```
SELECT TOP 100 objtype, usecounts, size_in_bytes, cacheobjtype,
REPLACE (REPLACE ([text], CHAR(13), ' '), CHAR(10), ' ') AS
   sql_text
FROM sys.dm_exec_cached_plans AS p
CROSS APPLY sys.dm_exec_sql_text (p.plan_handle)
WHERE p.objtype in ('Proc','Prepared','Adhoc') AND cacheobjtype =
   'Compiled Plan'
ORDER BY objtype, usecounts DESC
```

- I’m hiding all by those I’m running with additional WHERE clauses

```
and REPLACE (REPLACE ([text], CHAR(13), ' '), CHAR(10), ' ') like
   '%select *%'
and REPLACE (REPLACE ([text], CHAR(13), ' '), CHAR(10), ' ') not
   like '%SELECT TOP 100%'
```

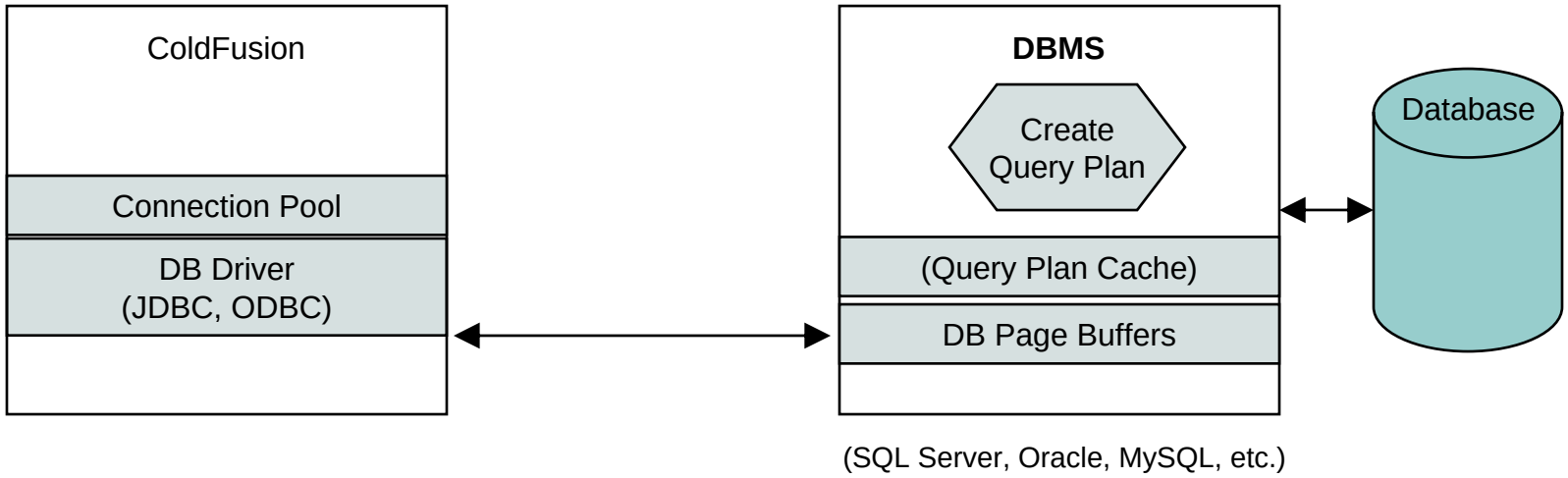
SQL Server 2k5 DMV SQL

- SQL to get CPU times and memory for those

```
SELECT TOP 10 usecounts, size_in_bytes, cacheobjtype,
SUM (total_worker_time / 1000) AS total_cpu_time_in_ms,
SUM (total_physical_reads) AS total_physical_reads,
SUM (total_logical_reads) AS total_logical_reads,
SUM (total_logical_writes) AS total_logical_writes,
REPLACE (REPLACE ([text], CHAR(13), ' '), CHAR(10), ' ') AS
   sql_text
FROM sys.dm_exec_cached_plans AS p
INNER JOIN sys.dm_exec_query_stats stat ON p.plan_handle =
   stat.plan_handle
CROSS APPLY sys.dm_exec_sql_text (p.plan_handle)
WHERE p.objtype in ('Proc','Prepared','Adhoc')  AND cacheobjtype =
   'Compiled Plan'
GROUP BY usecounts, size_in_bytes, cacheobjtype, [text]
ORDER BY usecounts DESC
```

- Again, I'm hiding all by those I'm running with the same additional WHERE clauses

Thinking through DB request processing



Adding query plan caching

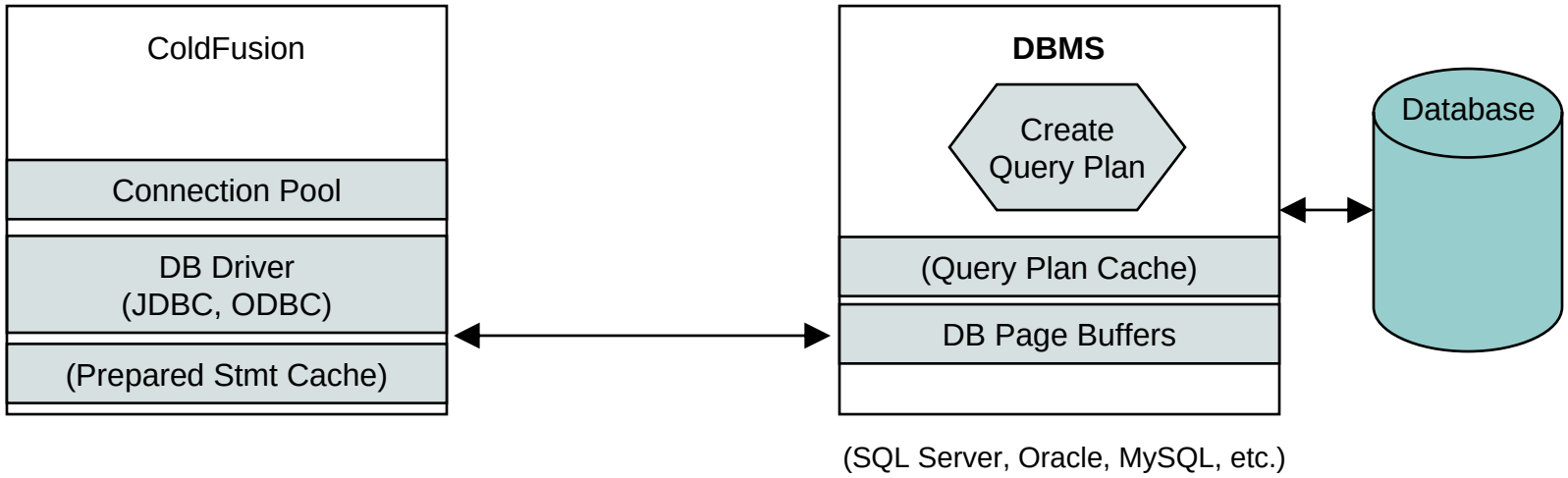
DB driver prepared statement caching

- Some DB drivers in CFMX offer “prepared statement caching”
- Not exposed in Admin DSN page
  - But settable on connection string
- Have not explored this fully

MySQL Driver Prepared Statement Cache

- Prepared statement cache in connector/j
  - Not enabled by default
  - See http://dev.mysql.com/doc/refman/5.0/en/connector-j-reference-configuration-properties.html
  - See also many other references to “prepared” there

Thinking through DB request processing



Adding prepared statement caching

Observations on MySQL

- Support for prepared statements new as of 4.1
  - Prior to that, could/would be emulated by DB driver
- One discussion explains benefits
  - (http://dev.mysql.com/doc/refman/5.0/en/connector-net-using-prepared.html)
  - Prepared execution is faster than direct execution for statements executed more than once, primarily because the query is parsed only once. In the case of direct execution, the query is parsed every time it is executed. Prepared execution also can provide a reduction of network traffic because for each execution of the prepared statement, it is necessary only to send the data for the parameters.
  - Another advantage of prepared statements is that it uses a binary protocol that makes data transfer between client and server more efficient
- Monitoring
  - See Com_stmt_execute to Com_stmt_prepare ratio
- Can view prepared statements in the “general query log”
  - Enable as discussed in docs, or using log in my.ini
  - View in “mysql administrator”, under “server logs”

MySQL Gotchas

- Some gotchas (http://dev.mysql.com/doc/refman/5.0/en/c-api-prepared-statement-problems.html)
  - Prepared statements not supported if using MySQL's query **results** cache feature
    - Fixed in 5.1.17: http://bugs.mysql.com/bug.php?id=735
  - Prepared statements do not support multi-statements (that is, multiple statements within a single string separated by ';' characters)
    - This also means that prepared statements cannot invoke stored procedures that return result sets, because prepared statements do not support multiple result sets.
- Others from elsewhere
  - Use of prepared statements involves extra round-trip between driver and DBMS on first execution of statement
    - Prepare returns an identified to client, which then executes it (optionally filling in bind parms)
    - http://dev.mysql.com/tech-resources/articles/4.1/prepared-statements.html
  - There is no prepared statement cache
    - "You can allocate multiple copies of same prepared statement and they will each use separate structures on the server. It does not matter if you do it from same connection or multiple connections."
      - http://www.mysqlperformanceblog.com/2006/08/02/mysql-prepared-statements/

Observations on using SQL 2005 features

- If SQL offered for 2k5 fails with
  - “incorrect syntax near '.'”
  - Check the compat level of the DB
    - sp_dbcmptlevel *dbname*
  - To change it to 2k5 level
    - sp_dbcmptlevel *dbname*, 90
- Permissions needed to run DMV SQL from within CF
  - In sql server, add permissions for the user used in the CF DSN
    - Sorry, can’t recall what I changed

When Prepared Statements Can Hurt

- Query plan may be created for one value that does not work well for other values
  - Obvious example is NULL
  - A query plan created for SELECT ... WHERE col=null would not use index
    - So could force tablescan
  - Subsequent query using “real” value could end up using that query plan...BAD!
  - Could happen for other than null (value with low distribution across records vs value with high)
- SQL Server offers some solutions to help
  - Can offer WITH RECOMPILE hint
  - Can also create plan guides and template plan guides
  - Beyond the scope of this talk to discuss in details

Sql server driver SPY log

- When ARE using CFQUERYPARAM
  - Connection[12].isClosed()
  - OK (false)
  - Connection[12].getAutoCommit()
  - OK (true)
  - Connection[12].getMetaData()
  - OK (DatabaseMetaData[24])
  - DatabaseMetaData[24].supportsMultipleResultSets()
  - OK (true)
  - DatabaseMetaData[24].supportsGetGeneratedKeys()
  - OK (true)
  - **Connection[12].prepareStatement(String sql, int autoGeneratedKeys)**
  - **sql = select * from person.address**
  - **where city=?**
  - **autoGeneratedKeys = 1**
  - **OK (PreparedStatement[6])**
  - **PreparedStatement[6].setMaxFieldSize(int max)**
  - **max = 64000**
  - **OK**
  - **PreparedStatement[6].setObject(int parameterIndex, Object x, int targetSqlType)**
  - **parameterIndex = 1**
  - **x = Dallas**
  - **targetSqlType = 12**
  - **OK**
  - **PreparedStatement[6].execute()**
  - **OK (true)**
  - PreparedStatement[6].getUpdateCount()
  - OK (-1)
  - PreparedStatement[6].getResultSet()
  - OK (ResultSet[6])

Sql Server driver SPY log

- When NOT using CFQUERYPARAM
  - Connection[16].isClosed()
  - OK (false)
  - Connection[16].getAutoCommit()
  - OK (true)
  - Connection[16].getMetaData()
  - OK (DatabaseMetaData[32])
  - DatabaseMetaData[32].supportsMultipleResultSets()
  - OK (true)
  - DatabaseMetaData[32].supportsGetGeneratedKeys()
  - OK (true)
  - **Connection[16].createStatement()**
  - **OK (Statement[2])**
  - **Statement[2].setMaxFieldSize(int max)**
  - **max = 64000**
  - **OK**
  - **Statement[2].execute(String sql, int autoGeneratedKeys)**
  - **sql = select * from person.address**
  - **where city='Dallas'**
  - **autoGeneratedKeys = 1**
  - **OK (true)**
  - Statement[2].getUpdateCount()
  - OK (-1)
  - Statement[2].getResultSet()
  - OK (ResultSet[8])

Summary

- Thought through DB request processing
- Reviewed
  - DB query plan creation
  - DB prepared statements
  - Influence of CFQUERYPARAM
  - Options for query plan/statement cache
- Discussed performance implications
- Compared of SQL Server & MySQL

Questions on presentation

- Charlie Arehart
  - charlie@carehart.org
- I'd really appreciate your feedback
  - http://carehart.org/feedback/
- Also available for setup and implementation consulting
  - Also other developer productivity coaching, system admin and tuning support, and more
  - Remote or on-site
- New Per-minute Phone/Web support
  - http://carehart.org/askcharlie/