# Dynamic Statement Cache In A Nutshell

**Bill Arledge**

**Principal Consultant, Technical Sales**

**CA Technologies**

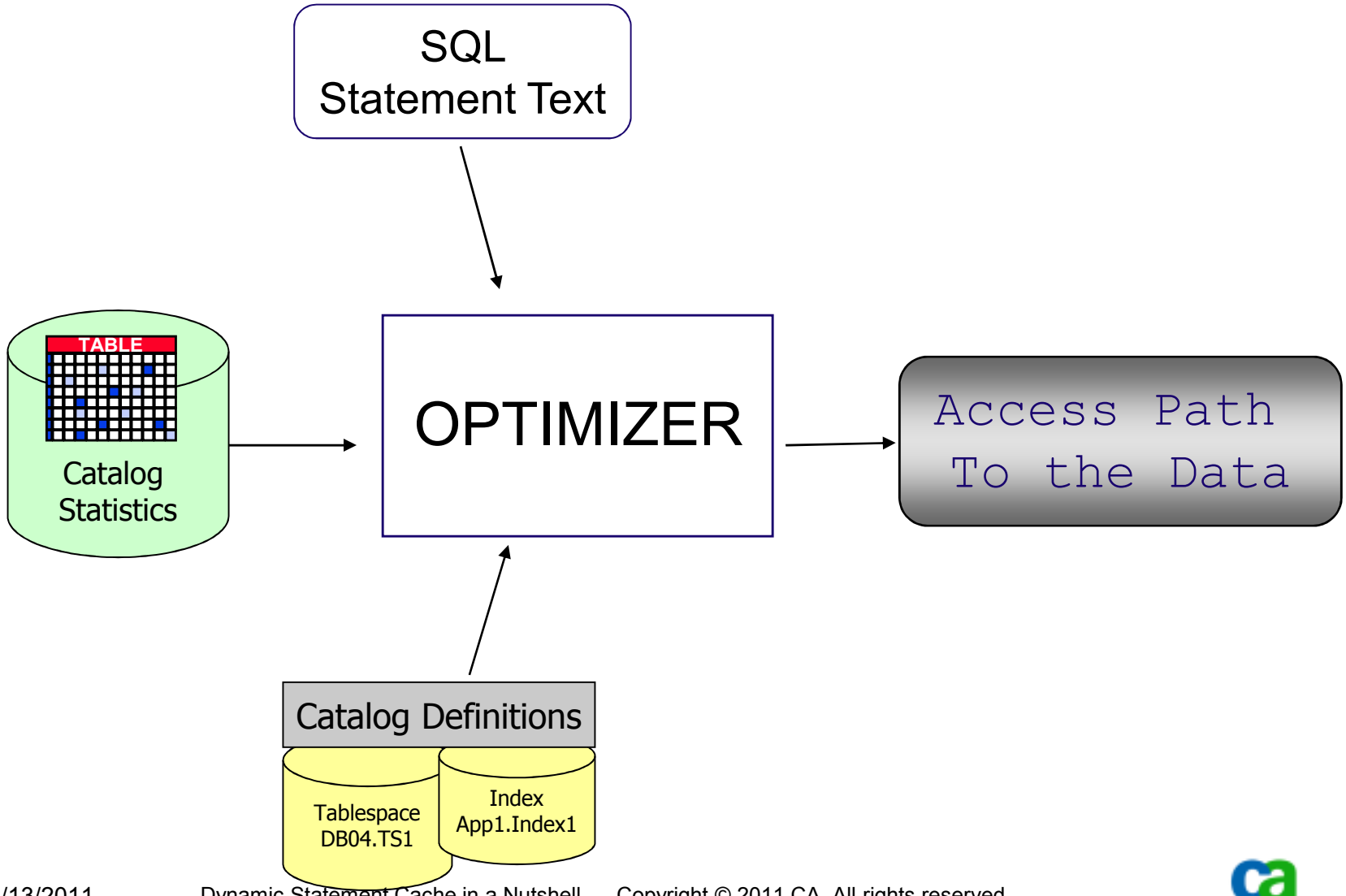
**BW DB2 User Group**
**September 14, 2011**

# What Will We Talk About?

- Some SQL Tuning Fundamentals
- Dynamic SQL in More Detail
- Introduction to DB2 Statement Caching
- DB2 10 Caching Enhancements
- Mining for Gold in the Global Statement Cache

# SQL Tuning Fundamentals
# DB2 Optimizer Determines SQL Performance
# (Initial



# SQL Tuning Fundamentals
# Access Path Selection

**Static SQL**

- Access path determined at bind time – better performance at run time
- Authorization for execution at the plan/package level
- Qualifiers passed via host variables
- SQLJ provides for bound static SQL in Java applications

**Dynamic SQL**

- Access Path Selection determined at execution
  - PREPARE operation
  - Exceptions to the Rule
    - KEEPDYNAMIC bind option
    - Global Dynamic Statement Cache
- Build and execute SQL on the fly
- User requires authorization to all accessed objects
- Parameter markers for passing variables

ca technologies

# Trends in the Marketplace
# Static vs. Dynamic SQL

—Dynamic SQL is > 50% of the workload at many shops

—What drives dynamic SQL usage?

- Dynamic SQL offers flexibility that can simplify developing complex applications
- New applications being developed on distributed platforms using connections that only support dynamic SQL
  - DB2 CONNECT, etc.
- ERP applications implemented with dynamic SQL
  - SAP, PeopleSoft, Siebel
- New applications being developed on distributed platforms
  - New developers are much more familiar with GUI and/or web-based programming environments and don't even sign on to the mainframe
    - –More Java and C++

# SQL Fundamentals - Static SQL CURSOR Processing

- Data access requirements well defined and predictable
  - Define the Cursor

```
EXEC SQL
    DECLARE INDCSR
      SELECT E.EMPNO , D.DEPTNO , D.DEPTNAME
   FROM DSN8910.EMP E
      INNER JOIN DSN8910.DEPT D ON E.WORKDEPT = D.DEPTNO
  WHERE D.DEPTNAME = :DEPTNAME-VAR
END-EXEC.
```

Host variable defined In working storage

  - Open the cursor

```
EXEC SQL
  OPEN INDCSR
```

  - Fetch the rows from the result set

```
EXEC SQL
  FETCH INDCSR INTO :DCLCR-EMP-DEPT-JOIN
```

  - Close the cursor

# SQL Fundamentals - Dynamic SQL

- Data access requirements are ad hoc in nature and identified on the fly
  - SELECT Operations

```
01 DYNAMCSR-A.
      49  DYNAMSQL-L              PIC S9(4) USAGE COMP-4.
      49  DYNAMSQL-D1             PIC X(300).
01 DYNAMCSR-LIT.
   05 DYNSQL-TEXT.
      10 DYNSQL-TEXT-SECT-1       PIC X(57) VALUE
      'SELECT E.EMPNO, D.DEPTNO, D.DEPTNAME  FROM DSN8910.EMP E '.
      10 DYNSQL-TEXT-SECT-2       PIC X(50) VALUE
      'INNER JOIN DSN8910.DEPT ON E.WORKDEPT = D.DEPTNO '.
      10 DYNSQL-TEXT-SECT-3       PIC X(20) VALUE
      'WHERE D.DEPTNAME = ?'
01 DEPTNAME-VAR                  PIC X(10) VALUE 'ACCOUNTING'.
```

Parameter marker provides placeholder for later substitution

```
PREPARE STMT FROM :DYNAMCSR-A
```

```
EXEC SQL DECLARE EDCSR FOR STMT
```

```
OPEN EDCSR USING :DEPTNAME-VAR
```

At Cursor OPEN the variable value will be substituted

# Introduction to Dynamic Statement Caching

- **Goal is to reduce or eliminate SQL Prepare operations required for dynamic SQL statements**
- **Implementation**
  - Four kinds of caching
    - No caching
    - Local Dynamic Statement Caching
    - Global Dynamic Statement Caching
    - Full Caching
  - Cache prepared SQL statement and statement text for dynamic SQL statements in DBM1address space
    - Local Statement Cache
    - Global Dynamic Statement Cache
  - Controlled by various parameters
    - Bind options
    - DSNZPARMs
    - Application constructs

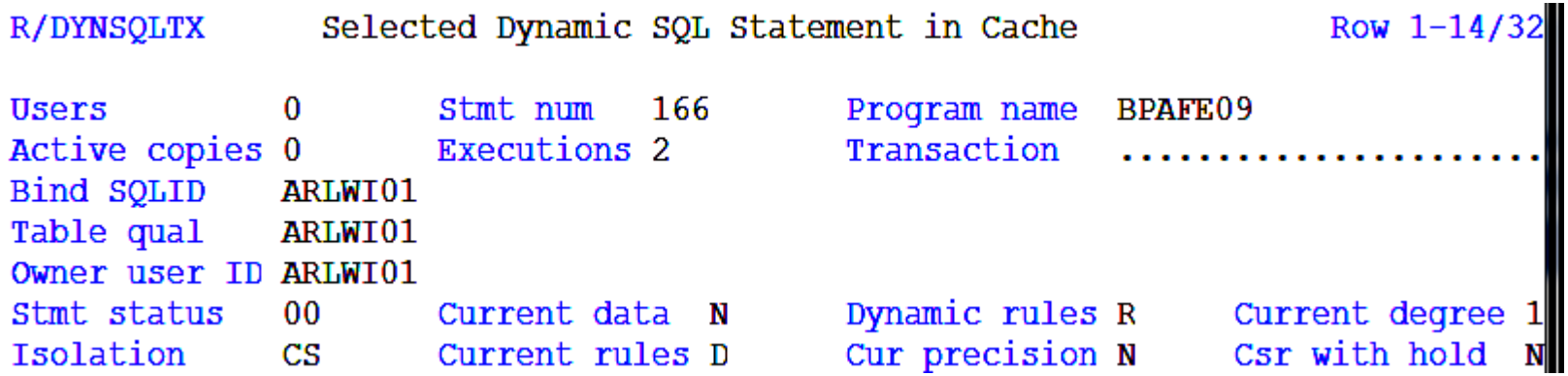
 

# Dynamic Statement Caching
# Global Statement Caching

- Allows reuse of prepared statements across UOWs
  - Within and across program executions
  - Prepared statement (SKDS) cached in global dynamic statement cache
- Enabling global statement caching
  - CACHEDYN=YES DSNZPARM value
  - Storage allocation discussed later
- PREPARE Flavors
  - Full prepare – Statement is not found in the cache and must be prepared from scratch
  - Short prepare – Statement is found in the cache and can be reused; no need to regenerate access path
  - Prepare avoidance – Access path information is still in the thread's local storage and requires no flavor of prepare
    - MAXKEEPD parameter specifies how many of these statement can exist

# Dynamic Statement Caching Where Cached Statements can be Reused



```
R/DYNSQLTX      Selected Dynamic SQL Statement in Cache          Row 1-14/32

Users          0       Stmt num    166      Program name  BPAFE09
Active copies  0       Executions  2        Transaction   ....................
Bind SQLID     ARLWI01
Table qual     ARLWI01
Owner user ID  ARLWI01
Stmt status    00      Current data  N      Dynamic rules R     Current degree 1
Isolation      CS      Current rules D      Cur precision N     Csr with hold  N
```

- Statement text must be 100% the same
  - Use parameter markers
  - Literals won't work (usually)
- Additional items must be 100% the same or compatible
  - Bind rules
  - Special registers
  - Authorizations
  - Others
- You may not get any benefit out of the dynamic statement cache at all
  - Most likely to benefit if you using an ERP or some other application that uses dynamic SQL extensively

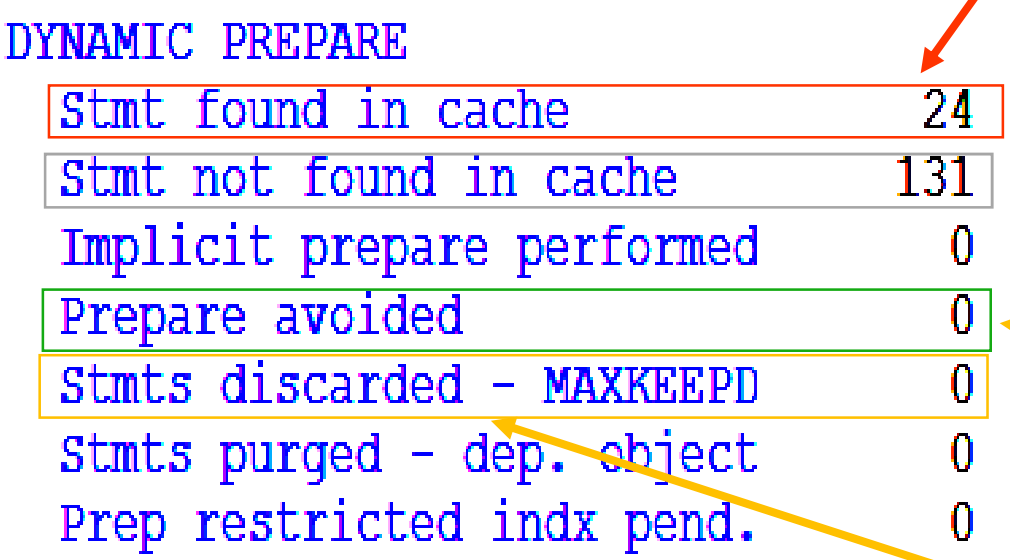
# Dynamic Statement Caching
# Prepare Costs

- Full Prepare
  - Statement not in cache
  - Global statement caching not active
- Short Prepare
  - Dynamic statement (SKDS) in the global cache
  - Global caching active
- Avoided Prepare
  - Local and global caching active

| SQL_CALL | STMT# | SECT# | INDB2_CPU |
|---|---|---|---|
| PREPARE | 00324 | 00001 | 00:00.003628 |
| FETCH | 00391 | 00001 | 00:00.000680 |
| OPEN | 00398 | 00001 | 00:00.000045 |
| CLOSE | 00405 | 00001 | 00:00.000023 |

| SQL_CALL | STMT# | SECT# | INDB2_CPU |
|---|---|---|---|
| FETCH | 00391 | 00001 | 00:00.000454 |
| PREPARE | 00324 | 00001 | 00:00.000231 |
| OPEN | 00398 | 00001 | 00:00.000020 |
| CLOSE | 00405 | 00001 | 00:00.000007 |

# Dynamic SQL Statement Caching
# DB2 Cache Statistics (from DB2 statistics)




**Requested statement was found in cache (nearly best case)**
- Short Prepare
- Higher the better

**Statement Not In Cache (worst case)**
- Full prepare
- Should be low depending on dynamic workload

**Prepare Avoided**
Specific for Applications bound with KEEPDYNAMIC(YES)
- High is good

**Statement Discarded**
Shoot for 0
Increase MAXKEEPD if > 0

# The Global Dynamic Statement Cache More Details

- Dynamic Statements
  - Inserted in the cache if:
    - CACHEDYN=YES – Global Cache is active
- Bind Options Impact on the cache
  - REOPT ALWAYS – prepared statements aren't placed in the cache
  - REOPT ONCE - Prepares the statement the first time the statement is executed with host variables available at that time
  - REOPT AUTO – DB2 9 feature where DB2 will examine the host variable values and will generate new access paths only when host variable values change and DB2 has not already generated an access path for those values
  - REOPT NONE – DB2 will not re-optimize the SQL at run time
- Cached statements reside in the cache until:
  - DROP or ALTER of any object
  - Authorization Revoked
  - LRU - Least Recently Used
  - RUNSTATS

# DB2 10 Caching Enhancements

- New STMT_ID (existed prior for dynamic cached statements)
  - For dynamic and static SQL
  - Included in IFCIDs for dynamic and static statements
  - Generated for dynamic statements that enter the cache
  - For static the STMT_ID column in SYSPACKSTMT
- New Monitor Class 29
  - Turns on IFCID 318 (more later)
  - IFCID 316 new behavior – when statement is ejected from pool the 316 is written
  - IFCID 400 – similar to IFCID 318 except for static statements
  - IFCID 401 – similar to IFCID 316 when statement is ejected from EDM Pool the 401 IFCID is written

# DB2 10
# Dynamic Statement Cache Enhancement

- Literals vs. parameter marker issue mitigated using new CONCENTRATE STATEMENTS WITH LITERALS clause
  - Can be added to the PREPARE Statement
  - Can be set in JCC driver on the client side
    - enableliteralReplacement='YES'
  - ODBC initialization file in z/OS can set LITERALREPLACEMENT
  - New column in statement cache 'LITERAL_REPL' indicates literal replacement
- Sequence of events
  - Incoming SQL with literals is looked up in the cache (DB2 9 behavior)
  - If not found, literals are replaced and the new SQL is searched for
  - If not found, new SQL is prepared and stored in the cache.
- Actual literals are stored in the cache as well

# Retrieving Data From the Global Cache

- As shown previously
  - Statement caching performance data in DB2 statistics records
  - Metrics show details about cache hit ratios and other useful data points that help you evaluate overall performance of your statement caches
- For more detail on Global Statement Cache usage the following instrumentation is provided
  - IFCID 316 – Provides details on statements in the cache
    - First 60 bytes of SQL text
    - Includes execution statistics (0 if not being collected)
  - IFCID 317 can then be used to retrieve the entire SQL statement from the cache once you have identified the statement of interest
- EXPLAIN STMTCACHE
  - V8 feature that exports Dynamic Statement Cache information to the DSN_STATEMENT_CACHE_TABLE
  - Nearly identical to the detail in IFCID 316 & 317
  - Multiple options including ALL, stmt-id, and stmt-token

# Reviewing Global Statement Cache Information IFCID 316 Results

- First 60 Bytes of SQL Text
  - IFCID 317 gives full text
- Bind Options
- Statement Statistics (more later)

```
                          SQL Text                                    Current  Active
                   User String (on 2nd line if present)                 Users  Copies
 ------------------------------------------------------------------- ------- -------
_SELECT E.EMPNO , D.DEPTNO , D.DEPTNAME                                     0       0
_UPDATE ANDRO16.BIGEMP1 SET PHONE = 5712321635 WHERE EMPKEY =               0       0
 UPDATE ANDRO16.BIGEMP1 SET PHONE = 4363427866 WHERE EMPKEY =               0       0
```

```
Users          0       Stmt num     324      Program name  DSNESM68
Active copies  0       Executions   3        Transaction   ......................
Bind SQLID     ARLWI01
Table qual     ARLWI01
Owner user ID  ARLWI01
Stmt status    00      Current data Y        Dynamic rules R     Current degree 1
Isolation      CS      Current rules D       Cur precision N     Csr with hold  N
```

```
Elapsed time      0.004      0.001  N/A   Getpages          48      16.0
CPU time          0.001      0.000 36.7   Rows examined    150      50.0
Sync I/O time     0.000      0.000  0.0   Rows processed     6       2.0
```

# Reviewing Global Statement Cache Information IFCID 317 Results

- IFCID 317 returns the full text of the SQL statement
- Notice that the two statements are functionally equivalent but have different formatting
  - 2 statements in the cache

```
R/DYNSQLTX      Selected Dynamic SQL Statement in Cache          Row 20-33/33

                                                Sync writes          0        0.0



Enter E to EXPLAIN the following SQL ====> _

       SELECT E.EMPNO , D.DEPTNO , D.DEPTNAME
          FROM DSN8910.EMP E INNER JOIN DSN8910.DEPT D ON E.WORKDEPT = D.DEPTNO
         WHERE LASTNAME IN ('PULASKI','HAAS')
```

```
       SELECT E.EMPNO , D.DEPTNO , D.DEPTNAME
          FROM DSN8910.EMP E INNER JOIN DSN8910.DEPT D ON E.WORKDEPT = D.DEPTNO
         WHERE LASTNAME
          IN ('PULASKI','HAAS')
```

# Reviewing Global Statement Cache Information Literals vs. Parameter Markers

| | | |
|---|---|---|
| SELECT A.NAME, A.CREATOR, A.TYPE, A.DBNAME, A.TSNAME, A.COLC | 0 | 0 |
| UPDATE ANDRO16.BIGEMP2 SET PHONE = 2683278951 WHERE EMPKEY = | 0 | 0 |
| UPDATE ANDRO16.BIGEMP2 SET PHONE = 6566111862 WHERE EMPKEY = | 0 | 0 |
| UPDATE ANDRO16.BIGEMP2 SET PHONE = 5782209117 WHERE EMPKEY = | 0 | 0 |
| SELECT STATE, CITY, COUNT(*) AS POPULATION FROM ANDRO16.BIG | 0 | 0 |

```
UPDATE ANDRO16.BIGEMP2 SET PHONE = 6566111862 WHERE EMPKEY =  603117

   UPDATE ANDRO16.BIGEMP2 SET PHONE = 2683278951 WHERE EMPKEY =  592725

      UPDATE ANDRO16.BIGEMP2 SET PHONE = 5782209117 WHERE EMPKEY =  21639
```

- In this example we have multiple statements in the cache that are exactly the same except for the literals used in the statement
- What's the downside
  - No re-use of the prepared statement so more CPU
  - Other statements in the cache may be tossed out if many of these statements run in a given period of time

# Mining the Dynamic Statement Cache EXPLAINing the SQL

- EXPLAIN STMTCACHE ALL
  - Inserts one row for each entry in the global DSC
    - Populates DSN_STATEMNT_CACHE_TABLE only
    - STMT_ID column matches the Unique ID in the global statement cache
    - Nearly exact match to the DSC with a few additional columns
    - COLLID set to DSNDYNAMICSQLCACHE
- EXPLAIN STMTCACHE STMT_ID
  - Extracts a single statement from the global DSC
    - Populates PLAN_DSN_DYNAMIC_STATEMNT, DSN_STATEMENT, and DSN_FUNCTION tables if they exist
    - Access path is current access path for statement in the cache
    - Numeric literal or host variable from program
- EXPLAIN STMTCACHE STMTTOKEN
  - Extracts a group of statements from the global DSC
    - Populates PLAN_DSN_DYNAMIC_STATEMNT, DSN_STATEMENT, and DSN_FUNCTION tables if they exist
    - Access path is current access path for statement in the cache
    - Based on STMT_TOKEN value in the cache
    - Alphanumeric literal or host variable in program

# Reviewing Global Statement Cache Information IFCID 318

- **Execution statistics for dynamic SQL statements**
- **Turn on collection with Monitor trace IFCID 318**
  - Begins collecting statistics and accumulates them for the length of time the monitor trace is on
  - Stop Monitor trace resets all statistics
  - 2-4% overhead per dynamic SQL statement stored in the cache
- **Recommended approach**
  - Run the trace only when actively monitoring the cache
- **Use EXPLAIN STMTCACHE to externalize data for evaluation**

```
-START TRACE (MON) IFCID(318)
```

```
DSNW130I  !S91A MON TRACE STARTED, ASSIGNED TRACE NUMBER 06
DSN9022I  !S91A DSNWVCM1 '-START TRACE' NORMAL COMPLETION
```

| | Total | Average | %Elp |
|---|---|---|---|
| Elapsed time | 0.000 | 0.000 | N/A |
| CPU time | 0.000 | 0.000 | 76.7 |
| Sync I/O time | 0.000 | 0.000 | 0.0 |
| Other read | 0.000 | 0.000 | 0.0 |
| Other write | 0.000 | 0.000 | 0.0 |
| Lock suspend | 0.000 | 0.000 | 0.0 |
| Global lock | 0.000 | 0.000 | 0.0 |
| Exec unit sw | 0.000 | 0.000 | 0.0 |

| | Total | Average |
|---|---|---|
| Getpages | 16 | 16.0 |
| Rows examined | 50 | 50.0 |
| Rows processed | 2 | 2.0 |
| Sorts | 0 | 0.0 |
| Index scans | 3 | 3.0 |
| Tblspace scans | 0 | 0.0 |
| Parallel grps | 0 | 0.0 |
| Sync reads | 0 | 0.0 |
| Sync writes | 0 | 0.0 |

# Summary

- Dynamic SQL is a major part of many workloads
  - ERP Vendors
  - Distributed applications
- DB2 offers multiple options for reducing the overhead traditionally associated with dynamic SQL
- These options include multiple types of statement caching
  - Local statement caching
  - Global statement caching
  - Full statement caching
- DB2 9 and 10 both introduced new functionality to enhance dynamic statement cache usage