IDUG

2024 EMEA Db2 Tech Conference

Tuning SQL Stored Procedures (LUW Edition)

Philip Nelson

Lloyds Banking Group / ScotDB Limited

Session F04

Theme: AppDev

Platform: Db2 (LUW)

Agenda

- What lies behind a SQL stored procedure deployment
- How to gather stored procedure performance information
- Tuning individual SQL stored procedure components

Stored Procedure DDL Template

```
CREATE OR REPLACE <procedure-name>
(<parameters>)
<config-parameters>
BEGIN
  <declarations>
  <body code>
END
```

Common SP Styles

- Scalar
- Single update
- Result Set
- Complex stored procedures

Example: "Scalar" Stored Procedure

```
CREATE PROCEDURE SP_SCALAR_CustSearch
(IN paramCustNumber INTEGER
,OUT paramCustName  VARCHAR(50))
BEGIN
  -- Validate input
  -- Retrieve data to return
  SELECT CUSTNAME INTO paramCustName
  FROM CUSTOMER
 WHERE CUSTNUM = paramCustNumber;
END#
```

Example : Result Set Stored Procedure

```
CREATE PROCEDURE SP_RS_CUSTTX
(IN paramCustNumber INTEGER)
BEGIN
  -- Declare cursor to return data
  DECLARE C01 CURSOR WITH RETURN FOR
  SELECT TXID, TXVALUE FROM CUSTOMER_TRANSACTION
  WHERE CUSTNUM = paramCustNumber
  ORDER BY CUST_TX_TIMESTAMP;
  -- Validate inputs
  -- Open cursor
   OPEN C01;
END#
```

Complex Stored Procedures

- Encapsulates complex processing within a stored procedure
- Many SQL statements
- Mixture of reading and updating
- Complex non-SQL logic
- Often replaces code in "traditional languages"
- "Batch as a stored procedure"
  - Triggered from job scheduling software
- "Services as a stored procedure"
  - Moving complex calculations down into the data layer

Some Other Notes on SP Patterns

- Some SPs are often better written as UDFs
  - Some UDF specific functions : particular PIPE
    - https://www.dbisoftware.com/db2nightshow/20230414DB2Night254.pdf
- Even for complex SPs most of the resource use is in SQL execution

Stored Procedure Catalog Entries (1) SYSCAT.ROUTINES

- Selected important columns
  - ROUTINESCHEMA,ROUTINENAME : name used to call SP
  - SPECIFICNAME : name used for admin if multiple SPs with same name
  - LIB_ID : basis of link between SP and associated package
    - ‘P’ CONCAT CHAR(LIB_ID) yields PKGNAME
      - But better to derive package name from SYSCAT.ROUTINEDEP

Stored Procedure Catalog Entries (2) SYSCAT.ROUTINEDEP

- Lists routine dependencies
- Selected important columns
  - ROUTINESCHEMA,SPECIFICNAME : SP details (link from ROUTINES)
  - ROUTINENAME : can be used as link if only one SP of this name
  - BTYPE : type of dependent object
    - ‘K’ = package, ‘T’ = table
  - BSCHEMA, BNAME : dependent object details
- Use this to find packages rather than old 'P' + LIB_ID trick

Stored Procedure Catalog Entries (3) SYSCAT.PACKAGES

- Used to look at various bind parameters
  - A few bind parameters can be set using SET_ROUTINE_OPTS (q.v.)
- Link to individual sections (statements) from SYSCAT.STATEMENTS

Stored Procedure Catalog Entries (4) SYSCAT.STATEMENTS

- One record for every SQL statement in the package / SP
- Important columns –
  - PKGSCHEMA, PKGNAME : link back to package
  - STMTNO : pointer to statement number in source (unique statement ID)
  - SECTNO : package section number
  - TEXT : SQL text
  - VERSION : version of the package
- Note that SPs with no SQL will have entries in PACKAGES but not in STATEMENTS

Monitoring Stored Procedure Performance

- Setting Up The Monitoring Environment
- Functions to Monitor Stored Procedures
- Statistics to Look At

Default Monitoring DB CFG Parameters

```
Request metrics                          (MON_REQ_METRICS) = BASE
Activity metrics                         (MON_ACT_METRICS) = BASE
Object metrics                           (MON_OBJ_METRICS) = EXTENDED
Routine data                                (MON_RTN_DATA) = NONE
  Routine executable list                (MON_RTN_EXECLIST) = OFF
Unit of work events                         (MON_UOW_DATA) = NONE
  UOW events with package list           (MON_UOW_PKGLIST) = OFF
  UOW events with executable list        (MON_UOW_EXECLIST) = OFF
Lock timeout events                      (MON_LOCKTIMEOUT) = NONE
Deadlock events                             (MON_DEADLOCK) = WITHOUT_HIST
Lock wait events                            (MON_LOCKWAIT) = NONE
Lock wait event threshold                  (MON_LW_THRESH) = 5000000
Number of package list entries            (MON_PKGLIST_SZ) = 32
Lock event notification level             (MON_LCK_MSG_LVL) = 1
```

MON_RTN_DATA DB CFG Parameter

- Controls capture of routine invocations
- Default is NONE (collects nothing)
- Should be set to BASE
- Information visible via MON_GET_ROUTINE(_DETAILS) functions
- Required as a prereq for MON_RTN_EXECLIST

MON_RTN_EXECLIST DB CFG Parameter

- Enables monitoring of “routine executable lists”
  - Compiled SQL statements executed by routines
- Provides a set of metrics for each “section” within a routine
- Information visible via MON_GET_RTN_EXEC_LIST function
- Additional info via MON_GET_PKG_CACHE_STMT
  - Linked via an “executable ID” (EXECUTABLE_ID column)
  - Note DYN_COMPOUND_EXEC_ID also available
    - For dynamically prepared SQL or anonymous blocks (PL/SQL)

Enabling Monitors : Warning

- IBM indicates that these monitors are low overhead
- BUT … tread carefully especially if you have high transaction rates
  - Internal data buffers must be updated (so there is SOME impact)
  - Beware of using monitors which externalize to disk

MON_GET_ROUTINE and MON_GET_ROUTINE_DETAILS

- Both are table functions that return aggregated routine details
  - MON_GET_ROUTINE is a relational result set
  - MON_GET_ROUTINE_DETAILS is an XML structure
- Example execution –

```
SELECT * FROM TABLE(
   MON_GET_ROUTINE(
      'P','MYSCHEMA',null,null,-2
   )
) AS M;
```

- Returns all statistics for stored procedures (type 'P') in schema 'MYSCHEMA'

MON_GET_ROUTINE() Outputs

- One row per executed stored procedure
- All executions of a stored procedure accumulated
  - Divide details by TOTAL_TIME_ROUTINE_INVOKED to get averages
- Total of 334 data points (V11.5.9)

MON_GET_RTN_EXEC_LIST Function

- Table function with identical input as MON_GET_ROUTINE()
- Returns aggregated statistics for every SQL statement in SP
- Link with MON_GET_PKG_CACHE_STMT() for more info
  - Link column is EXECUTABLE_ID
  - Extra information includes SQL statement text
  - Some duplicated information between two table functions

MON_GET_RTN_EXEC_LIST() Sample Execution

```
select
m.routine_name          , m.section_type   , m.section_number,
m.stmtno                , m.num_routines   , m.num_executions,
m.num_exec_with_metrics, m.total_act_time, m.total_cpu_time,
m.rows_read,
p.num_executions, p.section_number,
p.rows_Read     , p.stmt_text
from
table(mon_get_routine_exec_list('P','DBAIR001',null,null,-2)) as m
inner join
table(mon_get_pkg_cache_stmt(null,null,null,-1)) as P
on m.executable_id = p.executable_id
;
```

Some Hints On Using Monitor Statistics

- Compare routine executions (MON_GET_ROUTINE) to SQL statement executions (MON_GET_RTN_EXEC_LIST)
  - Particularly useful to highlight SPs with internal loops
- Consider both total statistics and average statistics per execution to differentiate between –
  - SP with large resource usage but infrequently executed
  - SP with smaller individual resource usage but running often
  - Often the case that the latter of these is more problematic

Alternatives to Table Functions

- Graphical tools should show the same as MON_GET_ROUTINE
  - dmctop : statistics don’t match MON_GET_ROUTINE
  - dsmtop (no longer in V11.5.9) : returned nothing
  - db2top (deprecated) : returns multiple records in some cases
- Concerns about discrepancies
  - Suggests taking data from different sources?

Obtaining Explain Plans for SQL in SPs

- Setting Up the Explain Tables
- Producing an Explain for a Stored Procedure
- Viewing the Explain Information Produced

Setting Up Explain Tables

- Explain tables change frequently
  - Additional tables or columns added
  - Most recent changes in V11.5.9 (add support for EXPLAIN_FORMAT)
- Best maintained using SYSINSTALLOBJECTS

```
CALL SYSPROC.SYSINSTALLOBJECTS
('EXPLAIN', 'C'
,CAST (NULL AS VARCHAR(128))
,CAST (NULL AS VARCHAR(128)));
```

- This creates tables under default schema (SYSTOOLS)
- To clean up existing tables swap ‘C’ for ‘D’ in second parameter
  - This will remove tables (and all existing data will be lost)

Producing Explains for Stored Procedures

- At stored procedure deployment time
- After deployment from catalog

Explaining During SP Deployment

- SP deployment options set using SET_ROUTINE_OPTS() SP
  - Multiple options can be specified in one invocation

```
CALL SYSPROC.SET_ROUTINE_OPTS
  ('EXPLAIN YES DATETIME ISO');
```

Explaining During SP Deployment (Controlling At Deployment Time)

- DevOps Pipeline – based on shell scripts and CLPPLUS
  - Collects whether to explain as an input parameter
  - Exports value of parameter (allows reading in CLPPLUS)
  - Gets value in CLPPLUS as parameter
  - Uses this parameter inside SET_ROUTINE_OPTS call

EXPLAIN Control
Shell Script (Selective Extract)

```
if [ $# = 7]; then
  …
  db2expl=“$7”;
else
  …
  print –n “Explain SPs (YES/NO)”; read db2expl;
Fi
# (before exporting validate contains YES/NO)
export DB2EXPL=$db2expl;

clpplus –nw $userid$location @deploy.clpplus
```

EXPLAIN Control
CLPPLUS Script (Selective Extract)

```
-- Set stored procedure parameters
CALL SYSPROC.SET_ROUTINE_OPTS('EXPLAIN $DB2EXPL');

-- Deploy stored procedure
STA @SP_BASIC.spsql

-- Exit CLPPLUS Script
QUIT
```

Obtaining Explain Information

- Information has been written to explain tables
- Extract information using db2exfmt

```
db2exfmt –d <dbname> -s <schema> -n <spname>
  -ot P –o <output-file> -w -1 –no_prompt
```

Explaining from Catalog after Deployment

- Process prior to V11.5.9 –
  - Obtain SP package for stored procedure
  - Check individual statements in catalog
  - Explain individual statements
  - Uses EXPLAIN_FROM_CATALOG() stored procedure
  - Extract information using db2exfmt
- Last step greatly simplified in V11.5.9
  - Use EXPLAIN_FORMAT() stored procedure
  - Writes explain (db2exfmt format) to new column in explain tables

EXPLAIN_FROM_CATALOG

```
CALL EXPLAIN_FROM_CATALOG
-- Input parameters
(<pkgschema>,<pkgname>,<pkgversion>,<sectno>
-- InOut parameter (set to null on input)
,<explain_schema>
-- Output parameters : explain details
,<requester>,<time>
-- Output parameters : source details
,<name>,<schema>,<version>
);
```

EXPLAIN_FROM_CATALOG Example

```
CALL EXPLAIN_FROM_CATALOG
('DBAIR001','P1234567890','',1,NULL
,?,?,?,?,?);
```

- Package P1234567890, Version 1
- Returns –
  - EXPLAIN_SCHEMA : SYSTOOLS
  - EXPLAIN_REQUESTER : DB2INST1
  - EXPLAIN_TIME : 2024-09-01-13.22.26.153233
  - SOURCE_NAME : P1234567890
  - SOURCE_SCHEMA : DBAIR001
  - SOURCE_VERSION :

Generating EXPLAIN_FROM_CATALOG

```
SELECT 'CALL EXPLAIN_FROM_CATALOG(''' CONCAT
RTRIM(d.bschema) CONCAT ''','''  CONCAT
RTRIM(d.bname)   CONCAT ''','''  CONCAT
RTRIM(s.version) CONCAT ''','''  CONCAT
s.sectno         CONCAT ','      CONCAT
'NULL,?,?,?,?,?);'
FROM SYSCAT.ROUTINES R
LEFT JOIN SYSCAT.ROUTINEDEP
ON R.SPECIFICNAME = D.ROUTINENAME AND
   R.ROUTINESCHEMA = D.ROUTINESCHEMA
LEFT JOIN SYSCAT.STATEMENTS S
ON S.PKGSCHEMA = D.BSCHEMA AND
   S.PKGNAME = D.BNAME
WHERE
D.BTYPE = 'K' AND R.ROUTINESCHEMA = 'DBAIR001' AND R.ROUTINENAME = 'SP_CURSOR';
```

- Takes in SP details
- Finds package for SP
- Finds all statements (sections)
- Generates one call per section

After EXPLAIN_FROM_CATALOG

- EXPLAIN_FROM_CATALOG populates explain tables
- Must then format output
  - Use same db2exfmt command as before
- Could generate db2exfmt commands
  - If multiple SPs have been deployed / explained

EXPLAIN_FORMAT() : New in V11.5.9

- Updates explain output into new column in explain tables
  - EXPLAIN_STATEMENT.EXPLAIN_FORMAT_TEXT
- Core data must have already been formatted
  - Still need EXPLAIN_FROM_CATALOG or EXPLAIN YES at deployment
- Can take in SP name (or specific name) as well as package
  - Avoids having to convert SP names to packages
- Explain data can then be exported to files
  - Useful for bulk explanations

EXPLAIN_FORMAT() : New in V11.5.9

```
EXPLAIN_FORMAT(
 -- Inputs : explain (table) details
 <schema>,<requester>,<time>
-- Inputs : source to explain
,<name>,<schema>,<version>,<section>
,<type>,<module>
-- Inputs : formatting flags
,<format-flags>, <graph-flags>
-- Output : SQL to extract explain from table
,<extract_sql>);
```

EXPLAIN_FORMAT()
Stored Procedure Specific Notes

- Explain table search parameters
  - Setting to null will use defaults (based on auth ID in use)
- Object types : (P)=Procedure, (SP)=Specific Procedure
- To explain all sections use 0 for section number
- Typical invocation –

```
CALL EXPLAIN_FORMAT
('SYSTOOLS',null,null,'SP_CURSOR','DBAIR001'
,null,0,'P',null,null,null,?);
```

Typical EXTRACT_SQL Output Parameter

```
SELECT EXPLAIN_FORMAT_TEXT
FROM SYSTOOLS.EXPLAIN_STATEMENT EXSTMT
WHERE
EXSTMT.EXPLAIN_REQUESTER = ‘DB2INST1’ AND
EXSTMT.EXPLAIN_TIME = ‘2024-08-29-20.26.33.611648’ AND
EXSTMT.SOURCE_NAME = ‘P1234567890’ AND
EXSTMT.SOURCE_SCHEMA = ‘DBAIR001’ AND
EXSTMT.SOURCE_VERSION = ‘’ AND
EXSTMT_SECTNO = 1 AND
EXSTMT_EXPLAIN_LEVEL = ‘O’
FOR READ ONLY;
```

Exporting Explain Data (Bulk Explains)

```
EXPORT TO explains.del OF DEL
LOBS TO . MODIFIED BY LOBSINSEPFILES
SELECT * FROM SYSTOOLS.EXPLAIN_STATEMENT
WHERE <appropriate filters>;
```

- Each (non-null) LOB is written to a separate file
  - For each statement this will be STATEMENT_TEXT and EXPLAIN_FORMAT_TEXT
    - Potentially also SNAPSHOT if not null
- LOB file formats are explains.del.<nnn>.lob
  - Mappings to individual files in explains.del

What About Non-SQL Components?

- Most SPs are “SQL intensive”
- What about SPs which are “logic intensive”
  - Particularly with loops that invoke SQL multiple times
- Monitors differentiate between “section” and “non-section” (or “user”) resource consumption
- Non-SQL components are “non-section” / “user” values
  - Virtually always this is a very small value (even if containing loops) when compared to the resource usage for SQL

IDUG

2024 EMEA Db2 Tech Conference

Tuning SQL Stored Procedures (LUW Edition)

Philip Nelson

teamdba@scotdb.com

Session F04

Please fill out your session evaluation!

@IDUGDb2
#IDUG_EMEA24