Abstract

DB2 z/OS V8 and 9 have introduced us to many new EXPLAIN tables. This presentation will introduce users some of the new EXPLAIN tables used by these tools for advanced query analysis, and demonstrate how SQL queries can be used to gather and analyze the advanced information.
**EXPLAIN and EXPLAIN Tools and Queries**

- EXPLAIN has been enhanced greatly for DB2 V8 and DB2 9
- Several EXPLAIN tools are available
  - Visual Explain
  - Optimization Service Center
    - Replaces Visual Explain
    - Works with V8 NFM
  - Data Studio
    - Visual Explain Plug-Ins
- However, DB2 EXPLAIN facility is built-in to DB2
  - No additional tools are needed to perform an EXPLAIN
  - EXPLAIN tables contain access path information
  - Anyone can read the access path information from the EXPLAIN tables

---

**Visual Explain Tool (DB2 V8)**

- A GUI interface to the information in the EXPLAIN tables
- Provides information about statements in the system
- Can dynamically EXPLAIN statements
- Can get Statistics for accessed objects
- Ability to browse subsystem parameters
- Compatible with V7 subsystems
- With V8 subsystems
  - More information than the PLAN_TABLE provides
  - Cardinality and Filter Factor information
  - Additional of predicate information
  - Some rewritten predicates exposed
- Method of transmission of information to IBM
  - XML File
  - Service SQL Feature

---
Optimization Service Center (DB2 9)

- A workstation-based tool
  - Easy interaction with DB2 EXPLAIN and the explain tables
  - Allows you to analyze
    - SQL statements
    - Objects
    - Statistics
    - The statement cache
    - Workloads
- Replaces Visual Explain
- Has Many of the features of Visual Explain
- Compatible with V8 Subsystems in New Function Mode
- Has additional features that can be purchased

Why Not Use These Products for EXPLAIN?

- PC workstation maybe does not have enough power
  - These tools use a lot of the machine
- DB2 Connect Enterprise Edition is required
- Remote connections to mainframe are required
  - This raises security concerns
    - Especially for production
- SPUFI or QMF is often preferred
  - This is a fast and easy method
Additional Explain Tables

- Most available in V8
- Now very accessible in V9 via Optimization Service Center or SQL
- Documented in IBM DB2 9 Performance Guide

```
DSN_VIEWREF_TABLE
DSN_QUERY_TABLE
DSN_PGRANGE_TABLE
DSN_SORTKEY_TABLE
DSN_SORT_TABLE
DSN_DETCOST_TABLE
DSN_FILTER_TABLE
DSN_PTASK_TABLE
DSN_PGROUP_TABLE
DSN_STRUCT_TABLE
DSN_PREDICAT_TABLE
```

EXPLAIN Tables Explained

The DB2 EXPLAIN facility populates the EXPLAIN tables
- If any of the EXPLAIN Tables exist for the Userid, then they will be populated automatically when you EXPLAIN
- The EXPLAIN Tables can be queried just like any other table

Visual Explain and Optimization Service Center define the EXPLAIN tables
- However, once created you can use them independent of these products
- You can also migrate or create them yourself

It is possible to have a “Green Screen” advanced EXPLAIN
- Helps those especially without distributed access to the mainframe
EXPLAIN Tables and Columns of Primary Interest

When EXPLAINing there are just a few primary tables and columns that should be initially focused on

— There is much more, but this is just for initial analysis
  
  Your first look at the Access Path
  
  More details are available, but the first look simplifies the view and can answer many questions

WARNING!

— Most EXPLAIN table analysis should be left in the hands of IBM of professional EXPLAIN products!

— Any “Tweaking” of EXPLAIN is done without IBM support

---

PLAN_TABLE

This is the base access path information

Columns for initial analysis

- METHOD
- MERGE_JOIN_COLS
- CREATOR
- TNAME
- CORRELATION_NAME
- ACCESS_TYPE
- ACCESSNAME
- INDEXONLY
- MIXOPSEQ
- MATCHCOLS
- SORT#### Cols
- PREFETCH
- PAGE_RANGE
- JOIN_TYPE
- PARENT_OBLOCKNO
- TABLE_TYPE
- BIND_TIME

First table accessed, join method, or sort

Number of merge scan joined columns

Name of table accessed

Index access or table space scan

‘Y’ if index-only access

Quantity of index matching columns used

‘Y’ if table space scan and partition elimination

Prefetch type if used

Join type if used

Table, work file, MQT, subquery, etc.

Table correlation name in query

Index name if used

Multi-index step number (if used)

Sort reasons

Parent query block number (subqueries, etc.)
Warnings About Using the Other EXPLAIN Tables

Can quite easily query these tables via SQL
Should limit what is viewed in the tables
These tables exist in support of the IBM query analysis products
— IBM can change/eliminate them at any time
IBM does NOT support your use of these tables beyond the authorized tools
— Will not respond to issues about these tables
— You are using them on your own and at your own risk

DSN_FILTER_TABLE

Contains information about how predicates are used during query processing
— One row per simple and compound predicate

Columns for initial analysis

<table>
<thead>
<tr>
<th>COLLID</th>
<th>PROGNAME</th>
<th>QUERYNO</th>
<th>EXPLAIN_TIME</th>
<th>PREDNO</th>
<th>STAGE</th>
</tr>
</thead>
</table>

These columns used for join to PLAN_TABLE
The relative predicate number used to join to DSN_PREDICAT_TABLE
The predicate stage (where processed within DB2). Matching, Screening, Stage 1, or Stage 2
**DSN_PREDICAT_TABLE**

Contains information about the predicates in a query
— One row per simple and compound predicate

Columns for initial analysis

- COLLID
- PROGNAME
- QUERYNO
- EXPLAIN_TIME
- PREDNO
- FILTER_FACTOR
- BOOLEAN_TERM
- TEXT

- These columns used for join to PLAN_TABLE
- The relative predicate number used to join to DSN_FILTER_TABLE
- The predicate filter factor (percent of rows to be returned based on this predicate alone)
- Whether or not the predicate is Boolean Term (when it evaluates to false does it make the entire WHERE clause false)
- The actual predicate text including rewritten or generated predicates

**DSN_DETCOST_TABLE**

Provides detailed cost information about the mini-plans in a query
— There can be multiple rows per query block and plan step
  Take the one with the smallest value

Columns for initial analysis

- COLLID
- PROGNAME
- QUERYNO
- EXPLAIN_TIME
- QBLOCKNO
- PLANNO
- ONECOMPROWS

- These columns used for join to PLAN_TABLE
- The number of rows the optimizer thinks will qualify after only local predicates are applied (take the lowest value)
Basic PLAN_TABLE Query

```sql
SELECT SUBSTR(DIGITS(QUERYNO), 5) CONCAT ' ' CONCAT
SUBSTR(DIGITS(QBLOCKNO), 4) CONCAT ' ' CONCAT SUBSTR(DIGITS(PLANNO), 4) AS
Qubit, PROGNAME AS PNAME, SUBSTR(CHAR(METHOD), 1, 1) AS MT
, SUBSTR(TNAME, 1, 18) AS TNAME , CHAR(TABNO) AS T_NO , ACCESSTYPE AS AT
, SUBSTR(TNAME, 1, 18) AS TNAME , CHAR(TABNO) AS T_NO , ACCESSTYPE AS AT
QQB_PL , PROGNAME AS PNAME , SUBSTR(CHAR(METHOD), 1, 1) AS MT
, CHAR(MATCHCOLS) AS MC , SUBSTR(ACCESSNAME, 1, 8) AS ACC_NM , INDEXONLY AS IX
, CHAR(MATCHCOLS) AS MC , SUBSTR(ACCESSNAME, 1, 8) AS ACC_NM , INDEXONLY AS IX
SUBSTR(DIGITS(QBLOCKNO), 4) CONCAT ' ' CONCAT SUBSTR(DIGITS(PLANNO), 4) AS
CONCAT SUBSTR(DIGITS(PPLANNO), 4) AS
----WHERE PROGNAME = 'YLAPROG1' <-- TARGET A SPECIFIC PROGRAM HERE
FROM PLAN_TABLE A
WHERE BIND TIME = (SELECT MAX(BIND TIME) FROM PLAN_TABLE B WHERE
A.PROGNAME = B.PROGNAME AND A.COLLID = B.COLLID)
ORDER BY PROGNAME, BIND_TIME, Qubit, MIX;
```

Advanced EXPLAIN Query

```sql
WITH MAXTIME (COLLID, PROGNAME, QUERYNO, BIND_TIME) AS (SELECT COLLID, PROGNAME, QUERYNO,
MAX(BIND_TIME) FROM PLAN_TABLE WHERE COLLID = 'DSNESPCS' AND PROGNAME = 'DSNESM68' GROUP BY
COLLID, PROGNAME, QUERYNO)
SELECT SUBSTR(PROGNAME, 1, 8) AS PROGNAME, SUBSTR(DIGITS(P.QUERYNO), 6) CONCAT ' ' CONCAT
SUBSTR(DIGITS(P.QBLOCKNO), 4) CONCAT ' ' CONCAT SUBSTR(DIGITS(P.PLANNO), 4) AS
QB, SUBSTR(CHAR(METHOD), 1, 1) AS MH , SUBSTR(DISASSEMBLY, 1, 8) AS AS
P.PROGNAME, SUBSTR(HRESULT, 1, 8) AS HRES , SUBSTR(CHAR(METHOD), 1, 1) AS MH
, SUBSTR(ACCESSNAME, 1, 8) AS ACC_NM , INDEXONLY AS IX , CHAR(MATCHCOLS) AS MC
, SUBSTR(ACCESSNAME, 1, 8) AS ACC_NM , INDEXONLY AS IX
P.QUERYNO = X.QUERYNO AND P.QBLOCKNO = X.QBLOCKNO AND P.PLANNO = X.PLANNO AND P.BIND_TIME = X.BIND_TIME AS D
ON 1 = 1 ORDER BY PROGNAME, BIND_TIME, QB, MIX, F.PREDNO;
```

© YL&A 1997-2009
Sample EXPLAIN Statement

Running this statement will populate the EXPLAIN tables

—Then we can run the two EXPLAIN reporting queries

```sql
EXPLAIN PLAN SET QUERYNO = 3 FOR
SELECT A.FIRSTNAME, A.LASTNAME, B.DEPTNAME
FROM   DSN8710.EMP A
INNER JOIN
DSN8710.DEPT B
ON A.WORKDEPT = B.DEPTNO
WHERE A.EMPNO = '000010';
```

EXPLAIN Results

**PLAN_TABLE Query**

<table>
<thead>
<tr>
<th>PROGRAM</th>
<th>QQP</th>
<th>MTH</th>
<th>P_NO</th>
<th>MJC</th>
<th>TBCREATOR</th>
<th>TBNAME</th>
<th>CORR_NM</th>
<th>ROWS_POST_FILTER</th>
<th>ATYP</th>
<th>A_NM</th>
<th>IXO</th>
<th>MIX</th>
<th>MCOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNESM68</td>
<td>00003-01-01</td>
<td>0</td>
<td>2</td>
<td></td>
<td>DSN8710</td>
<td>EMP</td>
<td>A</td>
<td></td>
<td>I</td>
<td></td>
<td>1.0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>DSNESM68</td>
<td>00003-01-02</td>
<td>1</td>
<td>3</td>
<td></td>
<td>DSN8710</td>
<td>DEPT</td>
<td>B</td>
<td></td>
<td>I</td>
<td></td>
<td>14.0</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

The number of rows the optimizer thinks will qualify after only local predicates are applied (take the lowest value)

The query's predicate number

Advanced EXPLAIN Query

```sql
EXPLAIN PLAN SET QUERYNO = 3 FOR
SELECT A.FIRSTNAME, A.LASTNAME, B.DEPTNAME
FROM   DSN8710.EMP A
INNER JOIN
DSN8710.DEPT B
ON A.WORKDEPT = B.DEPTNO
WHERE A.EMPNO = '000010';
```

Whether or not the predicate is Boolean Term

The actual predicate text including rewritten or generated predicates

The predicate stage (where processed within DB2).
Matching, Screening, Stage 1, or Stage 2

The predicate filter factor (percent of rows to be returned based on this predicate alone)
Simple Query Improvement EXPLAINed!

The following query has no Boolean Term predicates

```
EXPLAIN PLAN SET QUERYNO=1 FOR
SELECT *
FROM DSN8710.EMP
WHERE EMPNO = ?
OR (EMPNO = ? AND LASTNAME >= ?)
OR EMPNO > ?
ORDER BY EMPNO, LASTNAME;
```

This improved query has a redundant predicate added

```
EXPLAIN PLAN SET QUERYNO=2 FOR
SELECT *
FROM DSN8710.EMP
WHERE (EMPNO = ?
OR (EMPNO = ? AND LASTNAME >= ?)
OR EMPNO > ?)
AND EMPNO >= ?
ORDER BY EMPNO, LASTNAME;
```

This redundant predicate was added to the query to get index access on the EMPNO indexed column.

Advanced Explain Query Results

Advanced EXPLAIN for Query 1

```
<table>
<thead>
<tr>
<th>PROGRAM</th>
<th>QQP</th>
<th>MTH</th>
<th>P_NO</th>
<th>MJC</th>
<th>TBCREATOR</th>
<th>TBNAME</th>
<th>CORR_NM</th>
<th>ROWS_POST_FILTER</th>
<th>ATYP</th>
<th>A_NM</th>
<th>IXO</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNESM68 00001-01-01 0 1</td>
<td>DSN8710.EMP</td>
<td>--------</td>
<td>15.3</td>
<td>I</td>
<td>XEMP1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSNESM68 00001-01-01 0 4</td>
<td>DSN8710.EMP</td>
<td>--------</td>
<td>15.3</td>
<td>I</td>
<td>XEMP1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Advanced EXPLAIN for Query 2

```
<table>
<thead>
<tr>
<th>PROGRAM</th>
<th>QQP</th>
<th>MTH</th>
<th>P_NO</th>
<th>MJC</th>
<th>TBCREATOR</th>
<th>TBNAME</th>
<th>CORR_NM</th>
<th>ROWS_POST_FILTER</th>
<th>ATYP</th>
<th>A_NM</th>
<th>IXO</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNESM68 00002-01-01 0 1</td>
<td>DSN8710.EMP</td>
<td>--------</td>
<td>5.1</td>
<td>I</td>
<td>XEMP1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSNESM68 00002-01-01 0 4</td>
<td>DSN8710.EMP</td>
<td>--------</td>
<td>5.1</td>
<td>I</td>
<td>XEMP1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The improvement is detailed in the advanced EXPLAIN output via the access method, matchcols, boolean term, and text columns.
Complicated Output

The examples shown here are not perfect
Output can be influenced by
— Compound predicates
— Multi-Index access
More rows of data than are needed can appear
— Be careful and adjust queries as needed
Joining to additional EXPLAIN tables can further complicate the output

Reading the Tables Separately

If the output from the Advanced Query is too complicated
— Can reading each table separately
— Always read the PLAN_TABLE first to get the ‘Big Picture’ on the access path
Reading each table separately
— Easier to do
— Have to put the data together yourself
— Can provide more information than the advanced explain join query, or even the Explain Products
**Reading Tables Separately Example**

Here are queries to read the Filter and Predicate Tables

---

**Do this when EXPLAINing a single statement and be sure to Delete all rows from the Explain Tables first**

```sql
SELECT SUBSTR(F.PROGNAME, 1, 8) AS PROGNAME,
       SUBSTR(DIGITS(F.QUERYNO), 6) CONCAT '-'
       SUBSTR(DIGITS(F.QBLOCKNO), 4) CONCAT '-'
       SUBSTR(DIGITS(F.PLANNO), 4) AS QQ,
       PREDNO, STAGE, EXPLAIN_TIME AS B_TM
FROM   DSN_FILTER_TABLE F
ORDER  BY PROGNAME, B_TM, QQ, PREDNO;
```

- **The predicate number and stage is pulled from the DSN_FILTER_TABLE**

```sql
SELECT SUBSTR(P.PROGNAME, 1, 8) AS PROGNAME,
       SUBSTR(DIGITS(P.QUERYNO), 6) CONCAT '-'
       SUBSTR(DIGITS(P.QBLOCKNO), 4) AS QQ,
       PREDNO, FILTER_FACTOR, BOOLEAN_TERM AS BT,
       JOIN, AFTER_JOIN AS AJ, ADDED_PRED AS AP,
       REDUNDANT_PRED AS RP,
       KEYFIELD, substr(TEXT,1,40), EXPLAIN_TIME AS B_TM
FROM   DSN_PREDICATE_TABLE P
ORDER  BY PROGNAME, B_TM, QQ, PREDNO;
```

- **The predicate number, filter factor and boolean term flag is pulled from the DSN_PREDICATE_TABLE**
- **Addition predicate information is pulled: join predicate, after join predicate, added (generated) predicate, and redundant predicate flags**

---

**Example Query 1**

```sql
SELECT A.FIRSTNME, A.LASTNAME, B.DEPTNAME
FROM   DSN8810.EMP a
        INNER JOIN
        DSN8810.DEPT B
ON A.WORKDEPT = B.DEPTNO
WHERE  A.EMPNO = '000010';
```

- **Our join predicate is identified as using a key field**

**DSN_FILTER_TABLE Query Output**

<table>
<thead>
<tr>
<th>PROGNAME</th>
<th>QQ</th>
<th>PREDNO</th>
<th>STAGE</th>
<th>B_TM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNESM68</td>
<td>00001-01-01</td>
<td>2 MATCHING</td>
<td>2008-08-06-13.25.46.040000</td>
<td></td>
</tr>
<tr>
<td>DSNESM68</td>
<td>00001-01-02</td>
<td>3 MATCHING</td>
<td>2008-08-06-13.25.46.040000</td>
<td></td>
</tr>
</tbody>
</table>

**DSN_PREDICATE_TABLE Query Output**

<table>
<thead>
<tr>
<th>PROGNAME</th>
<th>QQ</th>
<th>PREDNO</th>
<th>FILTER_FACTOR</th>
<th>BT</th>
<th>AJ</th>
<th>AP</th>
<th>RP</th>
<th>KEYFIELD</th>
<th>TEXT</th>
<th>B_TM</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSNESM68</td>
<td>00001-01</td>
<td>1 0.1000000000000000E+01</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
<td>2008-08-06-13.25.46.040000</td>
</tr>
<tr>
<td>DSNESM68</td>
<td>00001-01</td>
<td>2 0.2380952239036560E-01</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td></td>
<td></td>
<td>2008-08-06-13.25.46.040000</td>
</tr>
<tr>
<td>DSNESM68</td>
<td>00001-01</td>
<td>3 0.7142859697341919E-01</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td></td>
<td></td>
<td>2008-08-06-13.25.46.040000</td>
</tr>
</tbody>
</table>

- **The compound predicate is displayed**
**Reading Tables Separately Example**

EXPLAIN PLAN SET QUERYNO=1 FOR
SELECT A.FIRSTNME, A.LASTNAME, B.DEPTNAME
FROM DSN8810.EMP a
INNER JOIN
  DSN8810.DEPT B
ON A.WORKDEPT = B.DEPTNO
WHERE A.WORKDEPT = 'B01';

Our join predicate is identified as stage 2

This fourth predicate was generated by DB2 via predicate transitive closure

**Addition Information Not Covered**

This is only an introduction to the use of these tables
May require more information
— Compliments the PLAN_TABLE information
Filter, Detailed Cost, and Predicate tables provide enhanced information

Views and MQTs used in a query
Query text before and after query transformation
Qualified partitions for a page range scan
The keys for all sorts in a query
Sort operations required
Information about query parallelism
Information about each query block in a query
Global Prepare Cache – V8

Explain has been enhanced

- Allowing for EXPLAINs to be done for statements in the global prepare cache
  - The access path used by the statement is written to the PLAN_TABLE
    - COLLID - DSNDYNAMICSQLCACHE
      - The statement does NOT go through access path selection (as opposed to normal explain processing of dynamic SQL)
  - Statement Id (STMTID) available from trace records with IFCID 316, 124
  - Statement Token (STMTTOKEN) assigned by application that prepares statement
     - RRSAF SET_ID function
     - SQLESETI function for remote applications

```
EXPLAIN STMTCACHE
STMTID = int

EXPLAIN STMTCACHE
STMTOKEN = string
```

DSN_STATEMENT_CACHE_TABLE – V8

To populate use keyword ALL is on EXPLAIN STMTCACHE

DSN_STATEMENT_CACHE_TABLE is created to hold the output of IFCID 316 and IFCID 318

Two different sets of information that can be collected from the SQL statements in the dynamic statement cache

STMTCACHE with the STMTID or STMTTOKEN

- Traditional access path information to be written to the PLAN_TABLE for the associated SQL statement
- Single row written to DSN_STATEMENT_CACHE_TABLE if it exists

STMTCACHE with the ALL keyword

- Information is written only DSN_STATEMENT_CACHE_TABLE
- Consists of one row per SQL statement in the dynamic statement cache for which the current authorization ID is authorized to execute.

```
EXPLAIN STMTCACHE ALL
```
### DSN_STATEMENT_CACHE_TABLE – V8

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STMT_ID</td>
<td>Statement ID, EDM unique token</td>
</tr>
<tr>
<td>STMT_TOKEN</td>
<td>Statement token. User-provided identification string</td>
</tr>
<tr>
<td>COLLID</td>
<td>Collection id value is DSN_DYNAMICSQLCACHE</td>
</tr>
<tr>
<td>PROGRAM_NAME</td>
<td>Program name, Name of package or DBRM that performed the initial PREPARE</td>
</tr>
<tr>
<td>INV_DROPALT</td>
<td>Invalidated by DROP/ALTER</td>
</tr>
<tr>
<td>INV_REVOKE</td>
<td>Invalidated by REVOKE</td>
</tr>
<tr>
<td>INV_LRU</td>
<td>Removed from cache by LRU</td>
</tr>
<tr>
<td>INV_RUNSTATS</td>
<td>Invalidated by RUNSTATS</td>
</tr>
<tr>
<td>CACHED_TS</td>
<td>TS Timestamp when statement was cached</td>
</tr>
<tr>
<td>USERS</td>
<td>Number of current users of statement. These are the users that have prepared or executed the statement during their current unit of work.</td>
</tr>
<tr>
<td>COPIES</td>
<td>Number of copies of the statement owned by all threads in the system</td>
</tr>
<tr>
<td>LINES</td>
<td>Precompiler line number from the initial PREPARE</td>
</tr>
<tr>
<td>PRIMAUTH</td>
<td>User ID - Primary authorization ID of the user that did the initial PREPARE</td>
</tr>
<tr>
<td>CURSPOOLID</td>
<td>CURRENT SQLID of the user that did the initial prepare</td>
</tr>
<tr>
<td>BIND_QUALIFIER</td>
<td>Bind Qualifier, object qualifier for unqualified table names</td>
</tr>
<tr>
<td>BIND_C</td>
<td>DATA CURRENTDATA BIND option: 'Y' - CURRENTDATA(YES), 'N' - CURRENTDATA(NO)</td>
</tr>
<tr>
<td>BIND_DYNRL</td>
<td>DYNAMICRULES BIND option: 'B' - DYNAMICRULES(BIND), 'R' - DYNAMICRULES(RUN)</td>
</tr>
<tr>
<td>BIND_DEGREE</td>
<td>CURRENT DEGREE value: 'A' - CURRENT DEGREE = ANY, 'I' - CURRENT DEGREE = 1</td>
</tr>
<tr>
<td>BIND_SQLRL</td>
<td>CURRENT RULES value: 'D' - CURRENT RULES = DB2, 'S' - CURRENT RULES = SQL</td>
</tr>
<tr>
<td>BIND_HOLD</td>
<td>Cursor WITH HOLD bind option 'Y' - initial PREPARE was done for a cursor WITH HOLD, 'N' - initial PREPARE was not done for a cursor WITH HOLD</td>
</tr>
<tr>
<td>STAT_TS</td>
<td>Timestamp of stats when IFCID 318 is started</td>
</tr>
<tr>
<td>STAT_EXEC</td>
<td>Number of executions of statement. For a cursor statement, this is the number of OPENs</td>
</tr>
<tr>
<td>STAT_GPAG</td>
<td>Number of getpage operations performed for statement</td>
</tr>
<tr>
<td>STAT_SYNCR</td>
<td>Number of synchronous buffer reads performed for statement</td>
</tr>
<tr>
<td>STAT_WRT</td>
<td>Number of buffer write operations performed for statement</td>
</tr>
<tr>
<td>STAT_EROW</td>
<td>Number of rows examined for statement</td>
</tr>
<tr>
<td>STAT_PROW</td>
<td>Number of rows processed for statement</td>
</tr>
<tr>
<td>STAT_SORT</td>
<td>Number of sorts performed for statement</td>
</tr>
<tr>
<td>STAT_INDEX</td>
<td>Number of index scans performed for statement</td>
</tr>
<tr>
<td>STAT_RSCN</td>
<td>Number of table space scans performed for statement</td>
</tr>
</tbody>
</table>
**DSN_STATEMENT_CACHE_TABLE (cont.) – V8**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STAT_PGRP</td>
<td>Number of parallel groups created for statement</td>
</tr>
<tr>
<td>STAT_ELAP</td>
<td>Accumulated elapsed time used for statement</td>
</tr>
<tr>
<td>STAT_CPU</td>
<td>Accumulated CPU time used for statement</td>
</tr>
<tr>
<td>STAT_SUS_SYNIO</td>
<td>Accumulated wait time for synchronous I/O</td>
</tr>
<tr>
<td>STAT_SUS_LOCK</td>
<td>Accumulated wait time for lock and latch request</td>
</tr>
<tr>
<td>STAT_SUS_SWIT</td>
<td>Accumulated wait time for synchronous execution unit switch</td>
</tr>
<tr>
<td>STAT_SUS_GLCK</td>
<td>Accumulated wait time for global locks</td>
</tr>
<tr>
<td>STAT_SUS.Other</td>
<td>Accumulated wait time for read activity done by another thread</td>
</tr>
<tr>
<td>STAT_SUS_OTHW</td>
<td>Accumulated wait time for write activity done by another thread</td>
</tr>
<tr>
<td>STAT_RIDLMT</td>
<td>Number of times a RID list was not used because the number of RIDs would have exceeded one or more DB2 limits</td>
</tr>
<tr>
<td>STAT_RIDSTOR</td>
<td>Number of times a RID list was not used because not enough storage was available to hold the list of RIDs</td>
</tr>
<tr>
<td>EXPLAIN_TS</td>
<td>When statement cache table is populated</td>
</tr>
<tr>
<td>SCHEMA</td>
<td>CURRENT SCHEMA value</td>
</tr>
<tr>
<td>STMT_TEXT</td>
<td>Statement text</td>
</tr>
<tr>
<td>STMT_ROWID</td>
<td>Statement ROWID</td>
</tr>
</tbody>
</table>

**DSN_STATEMENT_CACHE_TABLE – V9**

**New Columns in V9**

**BIND_RA_TOT**
- Total number of rebinds that have been issued for the dynamic statement due to REOPT(AUTO)

**BIND_TO_TYPE**
- N – REOPT(NONE) or its equivalent
- 1 – REOPT(ONCE) or its equivalent
- A – REOPT(AUTO)
- O – Current plan is deemed as optimal and no need for further REOPT(AUTO)
**DSN_STATEMENT_CACHE_TABLE Usage**

```sql
SELECT cached_ts, STAT_EXEC,
       dec(stat_elap,12,2) as stat_elap,
       dec(STAT_CPU,12,2) as stat_cpu,
       left(STMT_TEXT,100) as short_text
FROM UID1.DSN_STATEMENT_CACHE_TABLE
WHERE primauth = 'ABC'
AND explain_ts > '2008-08-27-00.00.00.000000'
ORDER BY stat_cpu DESC
```

An example statement extracting elapsed and CPU time for queries in the dynamic SQL cache, including the first 100 characters of the SQL text.

Could also get STMTID and then obtain the access path information from the cache.

---

**Example Output**

<table>
<thead>
<tr>
<th>CACHED_TS</th>
<th>STAT_EXEC</th>
<th>STAT_ELAP</th>
<th>STAT_CPU</th>
<th>SHORT_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008-08-26 01:55:29</td>
<td>341295</td>
<td>7050.42</td>
<td>1924.97</td>
<td>SELECT</td>
</tr>
<tr>
<td>2008-08-26 01:55:24</td>
<td>252134</td>
<td>5508.16</td>
<td>1741.26</td>
<td>SELECT</td>
</tr>
<tr>
<td>2008-08-26 01:55:30</td>
<td>193977</td>
<td>2943.68</td>
<td>930.89</td>
<td>SELECT</td>
</tr>
</tbody>
</table>

An example statement extracting elapsed and CPU time for queries in the dynamic SQL cache, including the SQL text (just the first few characters in this example).
Conclusions

IBM’s advanced SQL analysis tools
— Nice if you have them available

The EXPLAIN tables explained
— Many new tables with additional information

Defining the EXPLAIN tables you need
— Can be defined and populated outside the tools

Running EXPLAINs under SPUFI
— Or anywhere an EXPLAIN can be executed

Queries for the EXPLAIN tables
— Basic and advanced

Selecting only at the fields needed for query tuning
— Many very useful fields for additional information

EXPLAIN facilities for dynamic SQL tuning
— Ability to look at statement cache
DB2 Information on the Web

IBM
  ibm.com
IBM Software
  ibm.com/software
DB2 Family
  ibm.com/software/db2
DB2 Solutions Directory Applications
  ibm.com/developerworks/db2
"Red Books"
  ibm.com/redbooks
DB2 for z/OS
  ibm.com/software/db2zos
DB2 Support
  ibm.com/software/db2zos/support.html
DB2 for z/OS Papers
DB2 Magazine
  http://www.db2mag.com
DB2 Certification
  http://www.ibm.com/certify
DB2 Experts
  www.db2expert.com

Courses by YL&A - Taught by skilled instructors world-wide!

DB2 V9 for z/OS Transition
DB2 V8 for z/OS Transition
  ― Application or DBA or both
SQL (z/OS and LUW)
  ― Basic SQL
  ― Advanced and Complex SQL
  ― SQL Performance Tuning and Optimization
Application Development
  ― Stored Procedure Development and Implementation
  ― UDFs and Triggers Development
Database Design
  ― Physical Design, Logical Design
Data Sharing
  ― Implementation, Performance, Recovery
High Performance Design and Tuning
  ― Application, Database, Systems
DB2 V9 for z/OS Certification Crammer Course

We customize all classes based upon customer requirements

ONLY YL&A offers you the DB2 V9 DBA Certification Crammer Course to help you become certified!!
# CPU Reduction Through Performance Audits

**DB2 Performance Audits**
- Existing or new database designs and applications
- Certification of design and implementation acceptance
- Evaluation of all the performance ‘points’ in a DB2 environment
  - Physical Design
  - Subsystem
  - Application Code and SQL
- Help with bringing legacy application to an e-business environment – the rules have changed!
  - What was acceptable performance in the past is NOT acceptable in an e-business environment
- Experienced in ‘fighting fires’ – many performance problems do not become reality until production
- **Results:** problems identified, solutions proposed (many implemented immediately), continual knowledge transfer during the process

---

## Cost Avoidance Through Performance Tuning!!!!