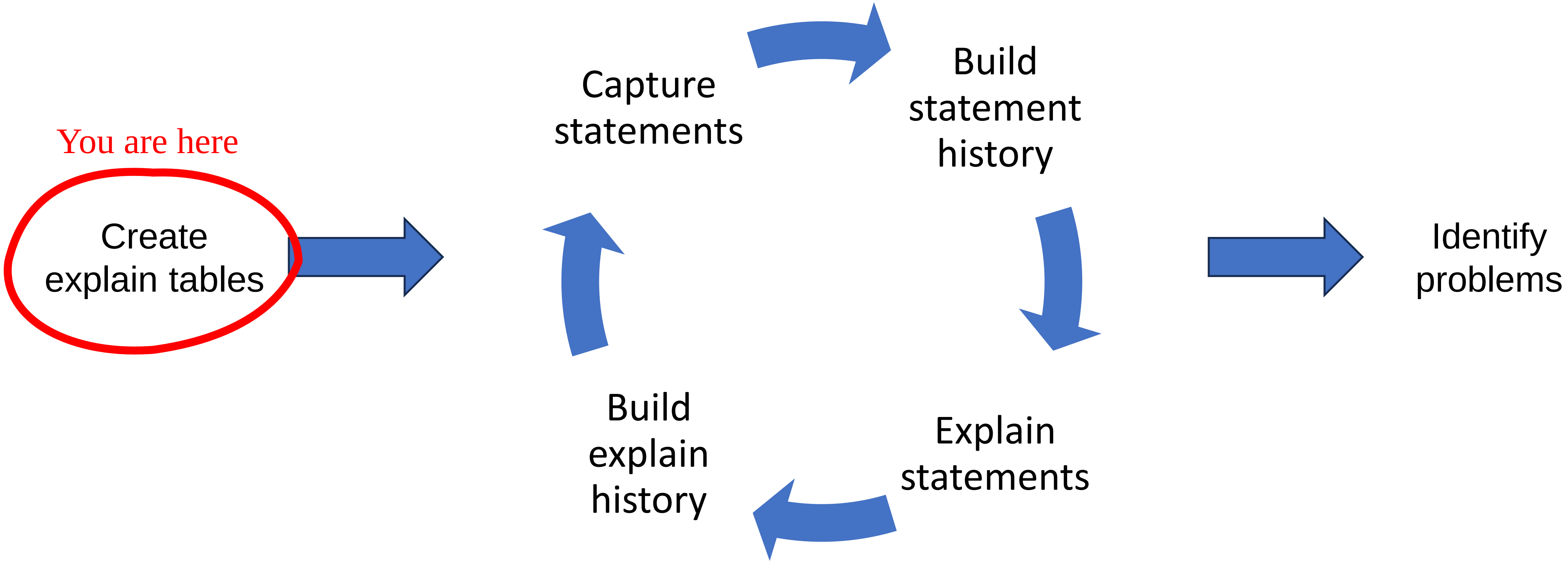
Build a lightweight monitor to identify SQL workload tuning potential

Kai Stroh
Product Manager
UBS Hainer
kai.stroh@ubs-hainer.com

- Objectives:
  - Capture SQL statistics from the DSC
  - Explain the statements found in the DSC
  - Save results to build a history
  - Combine statistics and access path information
  - Learn how to spot potential performance problems



You are here

Create explain tables

Capture statements

Build statement history

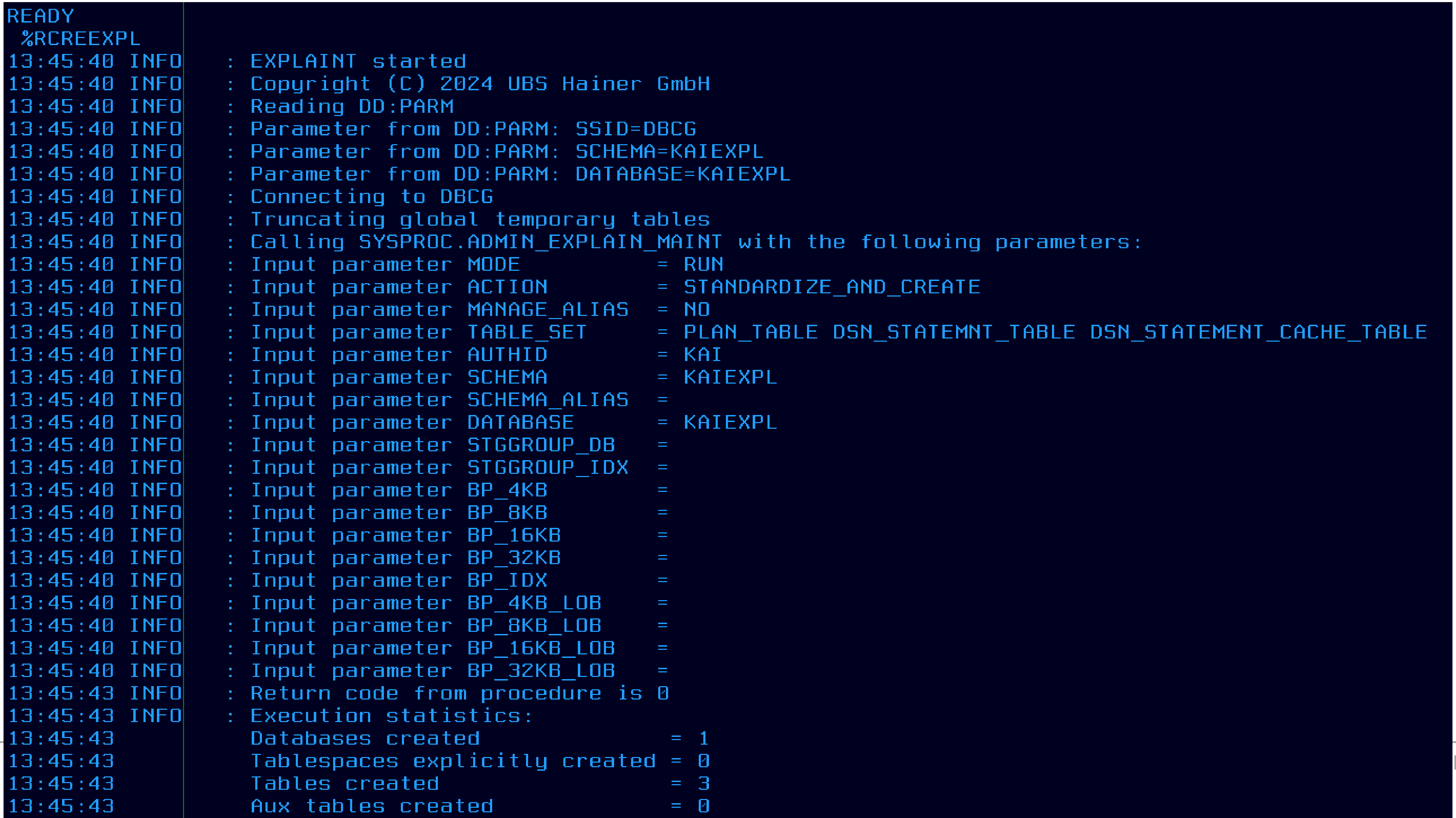
Explain statements

Build explain history

Identify problems

Create the explain tables

- Use stored procedure SYSPROC.ADMIN_EXPLAIN_MAINT with the following parameters:
  - *action* = STANDARDIZE_AND_CREATE
  - *table-set* = PLAN_TABLE
DSN_STATEMNT_TABLE
DSN_STATEMENT_CACHE_TABLE
- Creates 3 explain tables and 7 indexes as of Db2 V13 FL505
- Existing explain tables are updated to the newest structure
- **REXX available for download**

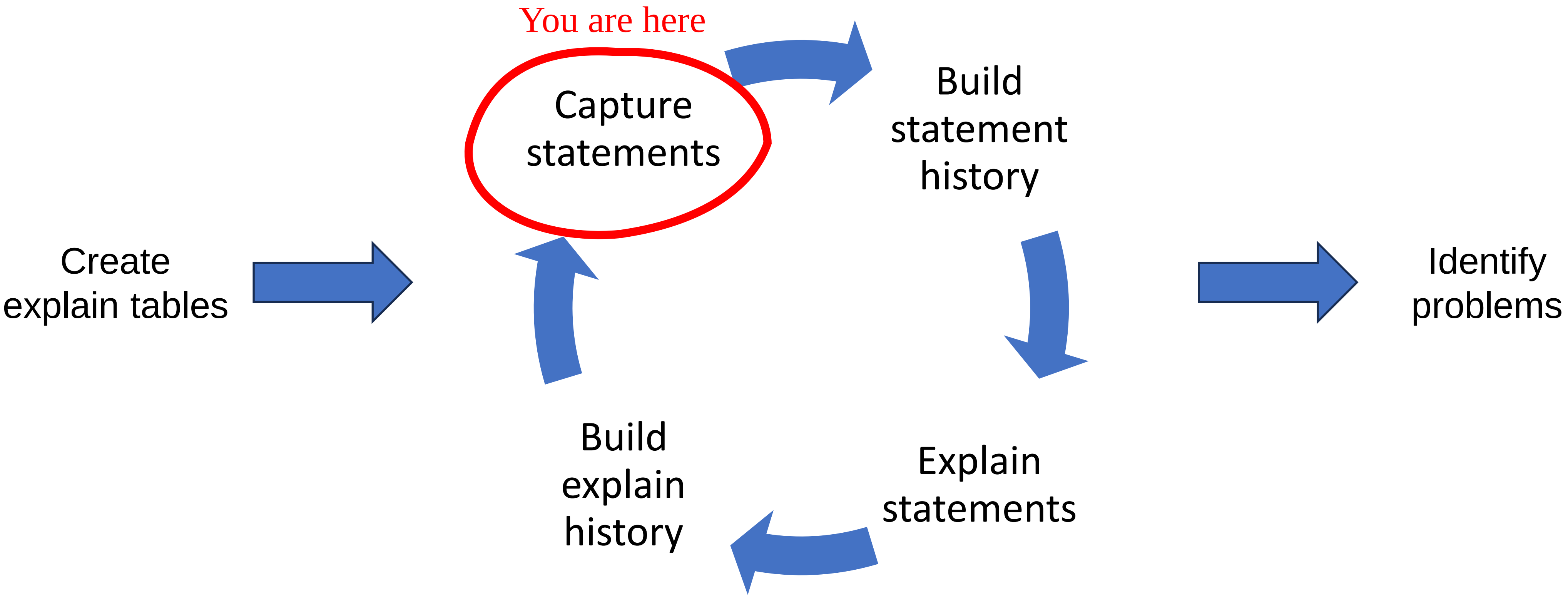


```
READY
 %RCREEXPL
13:45:40 INFO   : EXPLAINT started
13:45:40 INFO   : Copyright (C) 2024 UBS Hainer GmbH
13:45:40 INFO   : Reading DD:PARM
13:45:40 INFO   : Parameter from DD:PARM: SSID=DBCG
13:45:40 INFO   : Parameter from DD:PARM: SCHEMA=KAIEXPL
13:45:40 INFO   : Parameter from DD:PARM: DATABASE=KAIEXPL
13:45:40 INFO   : Connecting to DBCG
13:45:40 INFO   : Truncating global temporary tables
13:45:40 INFO   : Calling SYSPROC.ADMIN_EXPLAIN_MAINT with the following parameters:
13:45:40 INFO   : Input parameter MODE          = RUN
13:45:40 INFO   : Input parameter ACTION        = STANDARDIZE_AND_CREATE
13:45:40 INFO   : Input parameter MANAGE_ALIAS  = NO
13:45:40 INFO   : Input parameter TABLE_SET     = PLAN_TABLE DSN_STATEMNT_TABLE DSN_STATEMENT_CACHE_TABLE
13:45:40 INFO   : Input parameter AUTHID        = KAI
13:45:40 INFO   : Input parameter SCHEMA        = KAIEXPL
13:45:40 INFO   : Input parameter SCHEMA_ALIAS  =
13:45:40 INFO   : Input parameter DATABASE      = KAIEXPL
13:45:40 INFO   : Input parameter STGGROUP_DB   =
13:45:40 INFO   : Input parameter STGGROUP_IDX  =
13:45:40 INFO   : Input parameter BP_4KB        =
13:45:40 INFO   : Input parameter BP_8KB        =
13:45:40 INFO   : Input parameter BP_16KB       =
13:45:40 INFO   : Input parameter BP_32KB       =
13:45:40 INFO   : Input parameter BP_IDX        =
13:45:40 INFO   : Input parameter BP_4KB_LOB    =
13:45:40 INFO   : Input parameter BP_8KB_LOB    =
13:45:40 INFO   : Input parameter BP_16KB_LOB   =
13:45:40 INFO   : Input parameter BP_32KB_LOB   =
13:45:43 INFO   : Return code from procedure is 0
13:45:43 INFO   : Execution statistics:
13:45:43          Databases created              = 1
13:45:43          Tablespaces explicitly created = 0
13:45:43          Tables created                 = 3
13:45:43          Aux tables created             = 0
```

Add tables to store historic information

- Creating history tables makes it easier to spot trends
- Add column for collect time

```
SET SCHEMA = "your-schema";

CREATE TABLE PLAN_TABLE_H
        LIKE PLAN_TABLE
        IN DATABASE "your-dbname";

CREATE TABLE DSN_STATEMENT_CACHE_TABLE_H
        LIKE DSN_STATEMENT_CACHE_TABLE
        IN DATABASE "your-dbname";

COMMIT;

ALTER TABLE DSN_STATEMENT_CACHE_TABLE_H
  ADD COLLECT_TS TIMESTAMP NOT NULL;

COMMIT;
```


You are here
Capture statements
Build statement history
Create explain tables
Identify problems
Build explain history
Explain statements

Capture SQL statistics from the DSC

- Make sure IFCID 318 is active
- To get all statements, SYSADM authorization is required

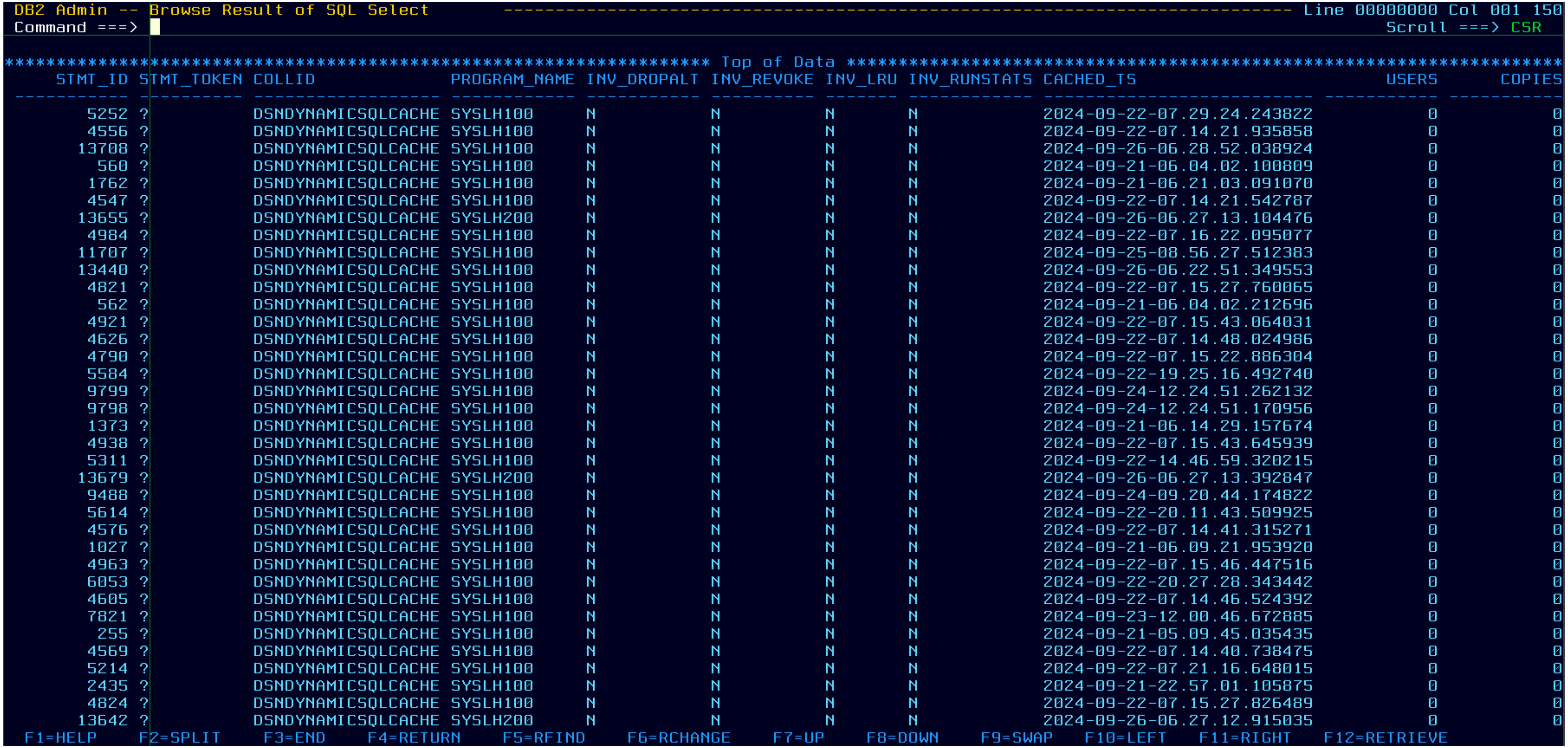
Run this Db2 command:

```
-STA TRACE(MON) CLASS(30) IFCID(318)
```

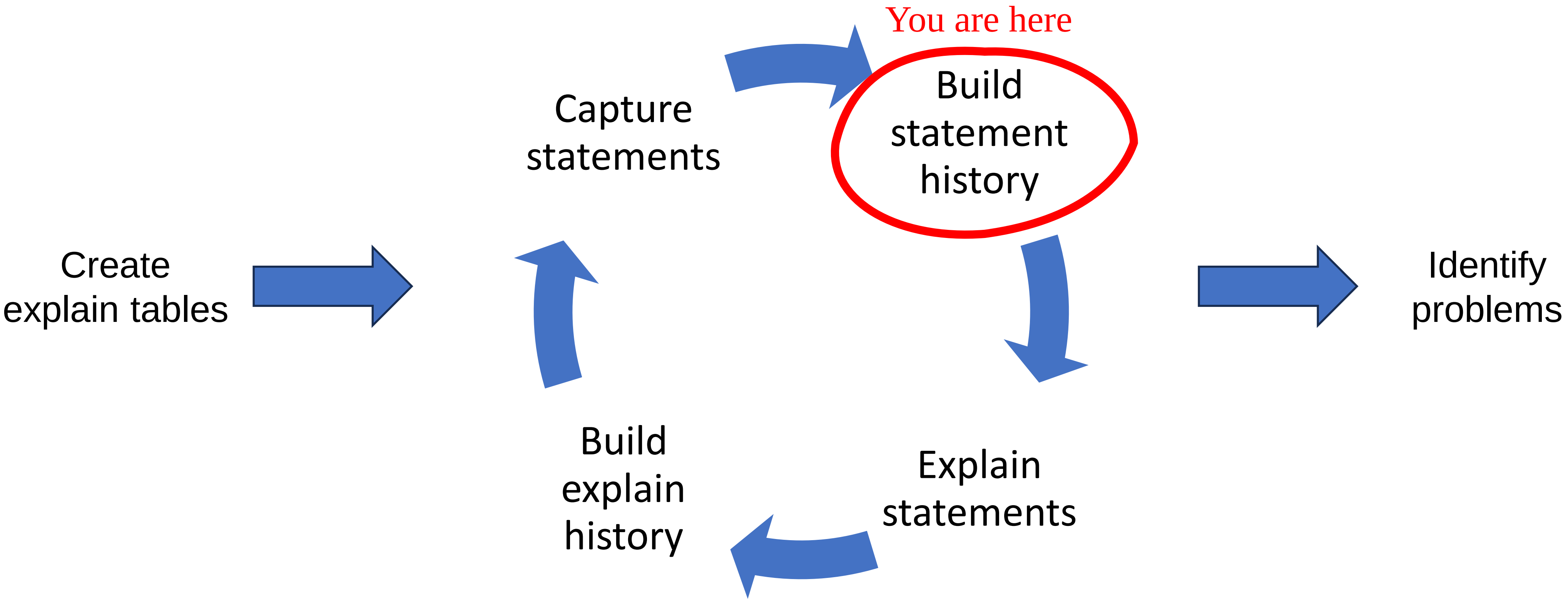
After some time, run these SQL statements:

```
SET SCHEMA = "your-schema";
TRUNCATE TABLE "DSN_STATEMENT_CACHE_TABLE";
SET CURRENT SQLID = 'your-schema';
EXPLAIN STMTCACHE ALL;
```

Build a lightweight monitor to identify SQL workload tuning potential



```
 DB2 Admin -- Browse Result of SQL Select       ---------------------------------------------------------------------------- Line 00000000 Col 001 150
 Command ===>                                                                                                                      Scroll ===> CSR

******************************************************************** Top of Data *********************************************************************
     STMT_ID STMT_TOKEN COLLID             PROGRAM_NAME INV_DROPALT INV_REVOKE INV_LRU INV_RUNSTATS CACHED_TS                        USERS      COPIES
 ----------- ---------- ------------------ ------------ ----------- ---------- ------- ------------ -------------------------- ----------- -----------
        5252 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.29.24.243822           0           0
        4556 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.14.21.935858           0           0
       13708 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-26-06.28.52.038924           0           0
         560 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-21-06.04.02.100809           0           0
        1762 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-21-06.21.03.091070           0           0
        4547 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.14.21.542787           0           0
       13655 ?          DSNDYNAMICSQLCACHE SYSLH200     N           N          N       N            2024-09-26-06.27.13.104476           0           0
        4984 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.16.22.095077           0           0
       11707 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-25-08.56.27.512383           0           0
       13440 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-26-06.22.51.349553           0           0
        4821 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.15.27.760065           0           0
         562 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-21-06.04.02.212696           0           0
        4921 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.15.43.064031           0           0
        4626 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.14.48.024986           0           0
        4790 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.15.22.886304           0           0
        5584 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-19.25.16.492740           0           0
        9799 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-24-12.24.51.262132           0           0
        9798 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-24-12.24.51.170956           0           0
        1373 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-21-06.14.29.157674           0           0
        4938 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.15.43.645939           0           0
        5311 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-14.46.59.320215           0           0
       13679 ?          DSNDYNAMICSQLCACHE SYSLH200     N           N          N       N            2024-09-26-06.27.13.392847           0           0
        9488 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-24-09.20.44.174822           0           0
        5614 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-20.11.43.509925           0           0
        4576 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.14.41.315271           0           0
        1027 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-21-06.09.21.953920           0           0
        4963 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.15.46.447516           0           0
        6053 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-20.27.28.343442           0           0
        4605 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.14.46.524392           0           0
        7821 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-23-12.00.46.672885           0           0
         255 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-21-05.09.45.035435           0           0
        4569 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.14.40.738475           0           0
        5214 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.21.16.648015           0           0
        2435 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-21-22.57.01.105875           0           0
        4824 ?          DSNDYNAMICSQLCACHE SYSLH100     N           N          N       N            2024-09-22-07.15.27.826489           0           0
       13642 ?          DSNDYNAMICSQLCACHE SYSLH200     N           N          N       N            2024-09-26-06.27.12.915035           0           0
  F1=HELP    F2=SPLIT    F3=END     F4=RETURN    F5=RFIND    F6=RCHANGE     F7=UP     F8=DOWN    F9=SWAP    F10=LEFT    F11=RIGHT   F12=RETRIEVE
```


You are here
Build statement history
Capture statements
Create explain tables
Identify problems
Build explain history
Explain statements

Build a history

- DSC dump is now in DSN_STATEMENT_CACHE_TABLE
- Be aware that counters are accumulative!
- Counters are set back to 0 when trace is stopped and restarted
- Stopping and restarting the trace does not remove the statements from the DSC!

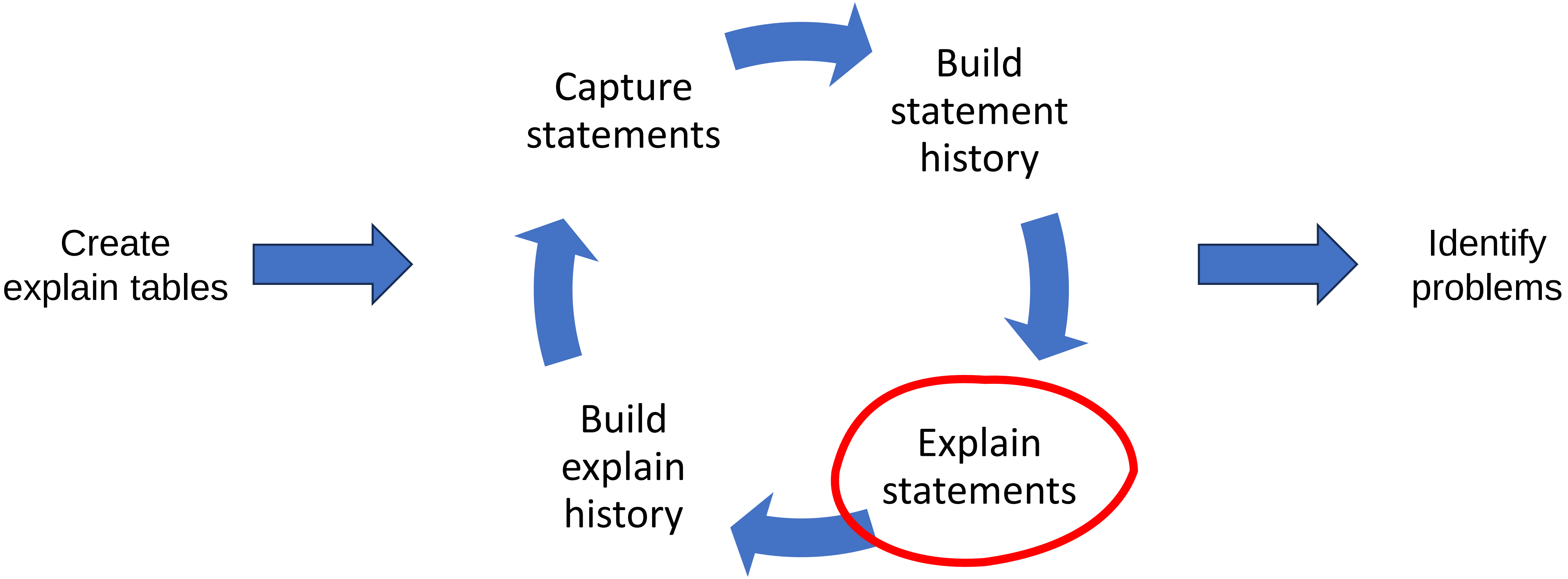
Cycle the trace in order to reset the counters:

```
-STO TRACE(MON) CLASS(30)
-STA TRACE(MON) CLASS(30) IFCID(318)
```

Build a history

- Copy rows from DSN_STATEMENT_CACHE_TABLE into DSN_STATEMENT_CACHE_TABLE_H

```
INSERT INTO DSN_STATEMENT_CACHE_TABLE_H (
  <list of 103 columns>
, COLLECT_TS
)
SELECT
 <list of 103 columns>
, CURRENT TIMESTAMP AS COLLECT_TS
FROM DSN_STATEMENT_CACHE_TABLE
WHERE STAT_EXECB > 0;
```



Create explain tables

Capture statements

Build statement history

Explain statements

Build explain history

Identify problems

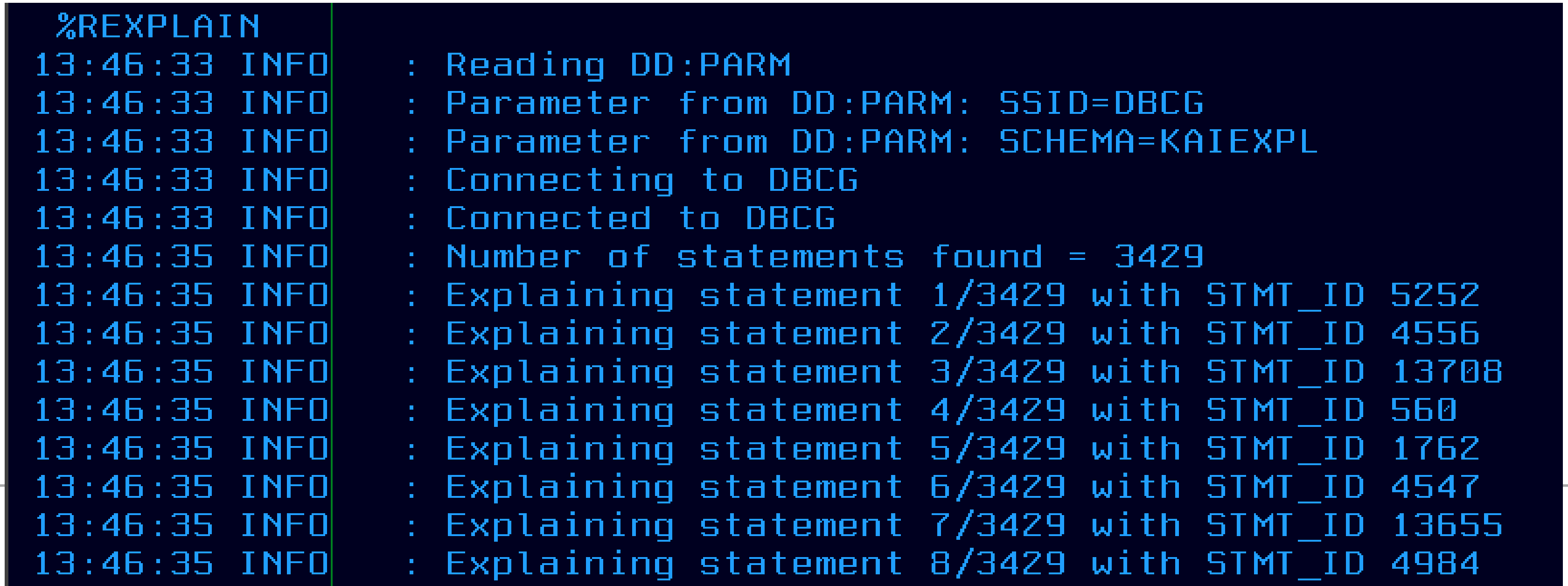
You are here

Explain the data

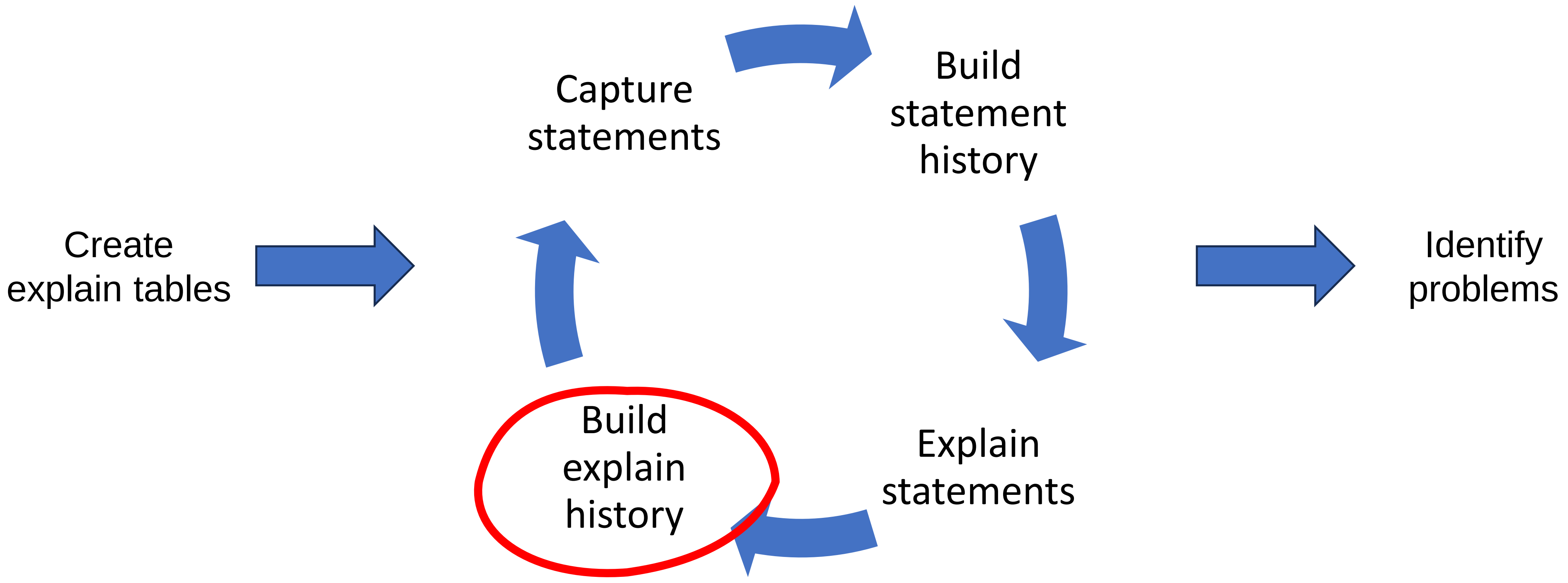
- EXPLAIN the statements from the DSC dump
  - Read the DSN_STATEMENT_CACHE_TABLE and explain each statement
  - Use EXPLAIN STMTCACHE STMTID xxx (xxx is the STMT_ID)
- This gives you the access path that Db2 **actually used** when the statement was prepared and entered the cache, **not** the access path Db2 would use now

Explain the data

- Some explains will fail because the statement has just been removed from the cache – should not be too many
- **REXX available for download**



```
 %REXPLAIN
13:46:33 INFO   : Reading DD:PARM
13:46:33 INFO   : Parameter from DD:PARM: SSID=DBCG
13:46:33 INFO   : Parameter from DD:PARM: SCHEMA=KAIEXPL
13:46:33 INFO   : Connecting to DBCG
13:46:33 INFO   : Connected to DBCG
13:46:35 INFO   : Number of statements found = 3429
13:46:35 INFO   : Explaining statement 1/3429 with STMT_ID 5252
13:46:35 INFO   : Explaining statement 2/3429 with STMT_ID 4556
13:46:35 INFO   : Explaining statement 3/3429 with STMT_ID 13708
13:46:35 INFO   : Explaining statement 4/3429 with STMT_ID 560
13:46:35 INFO   : Explaining statement 5/3429 with STMT_ID 1762
13:46:35 INFO   : Explaining statement 6/3429 with STMT_ID 4547
13:46:35 INFO   : Explaining statement 7/3429 with STMT_ID 13655
13:46:35 INFO   : Explaining statement 8/3429 with STMT_ID 4984
```



Create explain tables

Capture statements

Build statement history

Explain statements

Build explain history

Identify problems

You are here

Build a history

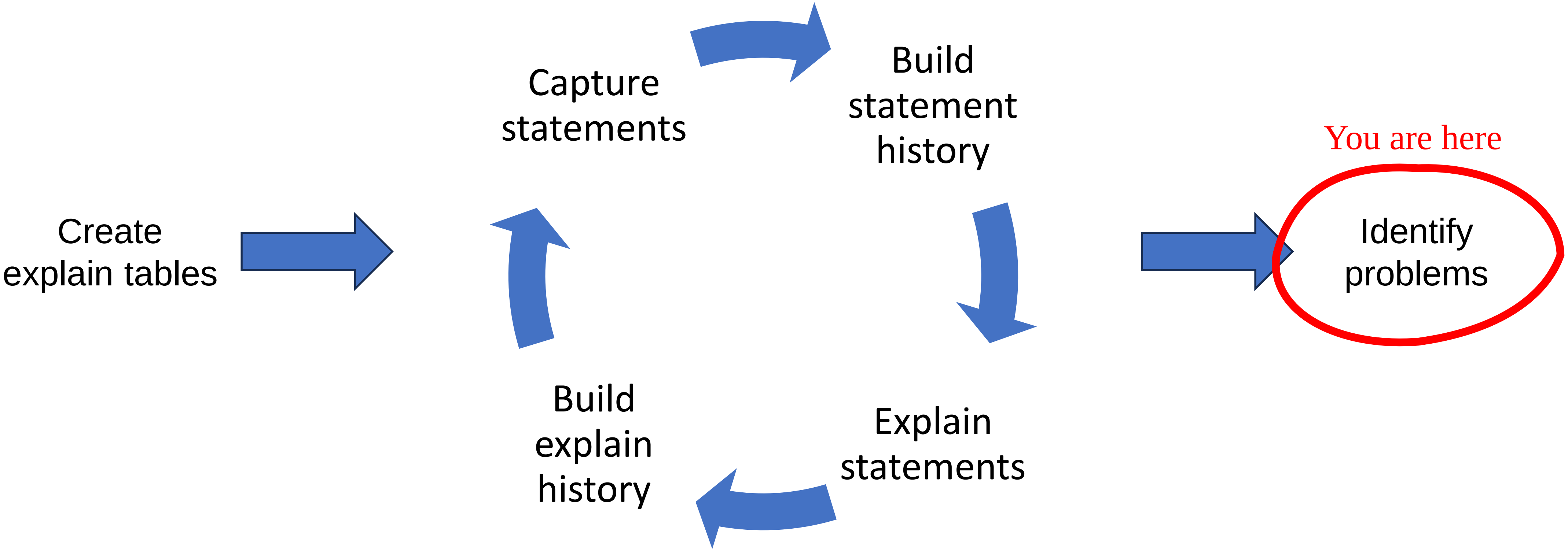
- Copy the result of the PLAN_TABLE to PLAN_TABLE_H to build the access path history
- If you would like to have a history of the optimizer estimates, you can do the same with DSN_STATEMNT_TABLE

Result

- A history of dynamic statements, which makes it easy to build trending statistics for each SQL
- Statements that just entered the cache:
  - Statistical columns contain values from since the statement was cached
- Statements that were already in the cache last time:
  - Statistical columns contain values from since the trace was cycled
- → Can calculate sums over statistical columns

Result

- By running these DSC dumps and explain 3-4 times a day, you create a history of SQL statements with information about CPU, I/O, LOCK, LOG…
- You have a history of the access path
    - Find out when an access path change occurred
    - Which indexes were in use before and after


Create
explain tables
Capture
statements
Build
statement
history
Explain
statements
Build
explain
history
You are here
Identify
problems

Combining the data

- Performance data:
  - You only need DSN_STATEMENT_CACHE_TABLE_H
  - A statement may change its STMT_ID over time
  - But it is uniquely identified by STMT_HASHID2
- You can now join DSN_STATEMENT_CACHE_TABLE_H (SCT) and PLAN_TABLE_H (PT)
  - SCT.STMT_ID = PT.QUERYNO
  - SCT.GROUP_MEMBER = PT.GROUP_MEMBER

Identify problems

- Statements without parameter markers
- Highest CPU / elapsed time
- Access path changes
- RID list problems
- Other indicators

Statements without parameter markers

- If your installation is running many SQLs without parameter markers, you get different hash-keys for what is essentially the same SQL statement
- Example: The next SQL with another CUST_NO occupies another slot in the DSC

```
SELECT * FROM CUSTOMER WHERE CUST_NO = 492954;
SELECT * FROM CUSTOMER WHERE CUST_NO = 82397;
SELECT * FROM CUSTOMER WHERE CUST_NO = 104329;
```

Statements without parameter markers

```
DO I = 1 TO CANDIDATES.0
  SQLSTMT = "UPDATE EMP SET SALARY = SALARY * 1.1 WHERE EMPNO = " || CANDIDATES.I
  ADDRESS DSNREXX "EXECSQL EXECUTE IMMEDIATE :SQLSTMT"
END
```

The “proper” way to handle this would be to convert the above code to:

```
SQLSTMT = "UPDATE EMP SET SALARY = SALARY * 1.1 WHERE EMPNO = ?"
ADDRESS DSNREXX "EXECSQL PREPARE S1 FROM :SQLSTMT"
DO I = 1 TO CANDIDATES.0
  EMPNO = CANDIDATES.I
  ADDRESS DSNREXX "EXECSQL EXECUTE S1 USING :EMPNO"
END
```

Statements without parameter markers

- Using literals generates **a lot** of overhead – preparing such a simple SQL is most likely more expensive than its execution
- Quick and dirty programming → **significant** avoidable overhead
- REXX programs tend to use literals instead of parameter markers
- Recommendation: Consider using the parameter CONCENTRATE for your packages

Solution

- A good starting point is to rebind the REXX packages
- Ways to enable CONCENTRATE:
  - As discussed, as Bind parameter
  - SQL PREPARE as additional attribute
  - JDBC on connection level, setDBStatementConcentrator(2)
  - In ODBC init file: LITERALREPLACEMENT=1
- If your installation is using many simple SQL statements with literals, you should already see a significant CPU reduction

Effect of CONCENTRATE for DSC dumps

- After applying CONCENTRATE, you will find SQL statements with an ampersand (&) in DSN_STATEMENT_CACHE_TABLE
- Db2 is now replacing the literals with ampersands while the statements with real parameter markers still have question marks
- Not a problem for EXPLAIN STMTCACHE STMTID xxx
- To explain the statement manually, replace “&” with “?” before EXPLAIN

Enable CONCENTRATE for REXX and Java programs

- REXX: Run the REBIND commands below
- Java programs:
  - Bind copy the packages into a new collection – see SDSNSAMP(DSNTIJLC)
  - Either: Set the jdbcCollection connection property
  - Or: Use profile tables to set a collection for a given Java application

```
REBIND PACKAGE(DSNREXCS.DSNREXX.(*)) CONCENTRATESTMT(YES)
REBIND PACKAGE(DSNREXRR.DSNREXX.(*)) CONCENTRATESTMT(YES)
REBIND PACKAGE(DSNREXRS.DSNREXX.(*)) CONCENTRATESTMT(YES)
REBIND PACKAGE(DSNREXUR.DSNREXX.(*)) CONCENTRATESTMT(YES)
REBIND PACKAGE(DSNREXX.DSNREXX.(*)) CONCENTRATESTMT(YES)
```

Minor caveat

- In **really rare** cases, you need to have control over the re-optimization (by using CONCENTRATE, you lose a detailed check on the host variable/literal)
- Might happen if column values are skewed
- This can be solved by adding a REOPT
  - REOPT(ONCE): Access path is calculated when the statement is first executed and stays as it is until the statement leaves the cache
  - REOPT(AUTO): Access path is re-optimized when parameter values change significantly
  - REOPT(ALWAYS): Access path is calculated every time the statement is executed (no caching in the DSC)

Identify problems

- Statements without parameter markers
- Highest CPU / elapsed time
- Access path changes
- RID list problems
- Other indicators

- Most relevant: Top CPU consumers, top elapsed time
- STMT_HASHID2 allows tracking a statement even if it leaves / reenters the cache and gets a new ID

```
SELECT
  HEX(STMT_HASHID2)                 AS STMT_HASHID2
, SUM(STAT_EXEC)                    AS EXECUTIONS
, SUM(STAT_CPU)                     AS CPU_TIME
, SUM(STAT_CPU)  / SUM(STAT_EXEC)   AS CPU_PER_EXEC
, SUM(STAT_ELAP)                    AS ELAPSED_TIME
, SUM(STAT_ELAP) / SUM(STAT_EXEC)   AS ELAP_PER_EXEC
, SUM(STAT_GPAGB)                   AS GETPAGES
, SUM(STAT_EROWB)                   AS ROWS_EXAMINED
, SUM(STAT_PROWB)                   AS ROWS_PROCESSED
, DOUBLE(SUM(STAT_EROWB)) / SUM(STAT_PROWB) * 100 AS RATIO
, SUM(STAT_RIDLIMTB)                AS STAT_RIDLIMTB
, SUM(STAT_RIDSTORB)                AS STAT_RIDSTORB
, VARCHAR(STMT_TEXT, 200)           AS SQL
FROM DSN_STATEMENT_CACHE_TABLE_H
WHERE COLLECT_TS > CURRENT TIMESTAMP - 3 DAYS
GROUP BY STMT_HASHID2, VARCHAR(STMT_TEXT, 200)
HAVING SUM(STAT_EXEC) > 0
ORDER BY 3 DESC
```

Break down elapsed time

- Elapsed time = time required for processing + wait times
- High SYNCIO / ELAPSED (> 50-60%)
  - Rows are processed in an order different from clustering
  - Tablespace may just need a REORG
  - Bufferpool size or parameters incorrect (separate analysis)
- High LOCK_WAITS / ELAPSED
  - Other Queries are locking tables or pages
  - Order of processing? Commit frequency?
  - Quick and dirty solution: Use row level locking (but watch for lock escalation)
  - Java: Choose isolation level (SYSLH100 / SYSLH200 / SYSLH300 SYSLH400)

Identify problems

- Statements without parameter markers
- Highest CPU / elapsed time
- Access path changes
- RID list problems
- Other indicators

Access path changes

- Reasons:
  - Indexes created or dropped
  - RUNSTATS updated
  - Table sizes changed significantly
  - …
- Static SQL: Access path calculated at BIND time
- Dynamic SQL: Access path calculated as statement enters the DSC
- Created an index, but your dynamic SQL is not using it? Invalidate DSC (or wait for the statement to leave and re-enter the DSC)

Finding access path changes

- First, identify the STMT_HASHID2 of your statement

```
SELECT
  HEX(STMT_HASHID2)        AS STMT_HASHID2,
  STAT_CPU / STAT_EXEC     AS CPU_PER_EXEC,
  COLLECT_TS               AS COLLECT_TS,
  VARCHAR(STMT_TEXT, 200) AS STMT_TEXT
FROM DSN_STATEMENT_CACHE_TABLE_H
WHERE VARCHAR(STMT_TEXT, 1000) LIKE
'SELECT * FROM TMF.S1306#11_1_TB1%'
ORDER BY COLLECT_TS
```

I have a feeling that this statement is slower than it used to be…

Aha! Over 20 times the CPU per execution

```
STMT_HASHID2      STAT_CPU  CPU_PER_EXEC             COLLECT_TS          STMT_TEXT
----------------  --------  ------------  --------------------------  ---------------------------------------------------------------
95286BCA05F8FFB7   0.00189       0.00189  2024-09-20 15:10:08.112918  SELECT * FROM TMF.S1306#11_1_TB1 WHERE NAME = 'LE' ORDER BY DOB
95286BCA05F8FFB7   0.03804       0.03804  2024-09-20 15:15:08.34794   SELECT * FROM TMF.S1306#11_1_TB1 WHERE NAME = 'LE' ORDER BY DOB
```

Finding access path changes

- Then look at the historic access paths

```
SELECT ...
FROM DSN_STATEMENT_CACHE_TABLE_H H
INNER JOIN PLAN_TABLE_H P
ON H.STMT_ID = P.QUERYNO
WHERE HEX(H.STMT_HASHID2) = '95286BCA05F8FFB7'
ORDER BY P.QUERYNO, P.PLANNO
```

STMT_HASHID2 identifies your SQL statement

We went from index access to tablespace scan

```
QUERYNO  QBLOCKNO  PLANNO  METHOD  CREATOR  TNAME            TABNO  ACCESSTYPE  MATCHCOLS  ACCESSCREATOR  ACCESSNAME
-------  --------  ------  ------  -------  ---------------  -----  ----------  ---------  -------------  -----------------
    211         1       1       0  TMF      S1306#11_1_TB1       1  I                   1  TMF            S1306#11_1_TB1_X1
    211         1       2       3                                0                      0
    215         1       1       0  TMF      S1306#11_1_TB1       1  R                   0
    215         1       2       3                                0                      0
```

Identify problems

- Statements without parameter markers
- Highest CPU / elapsed time
- Access path changes
- RID list problems
- Other indicators

RID list problems

- RID list: List containing row positions (RIDs) of candidate rows that Db2 builds when evaluating an expression using an index
- Size of a RID list is limited (~ 16.7 million RIDs)
- Size of the RID pools is limited (ZPARM MAXRBLK), overflow to DSNDB07 possible
- If Db2 wants to use a RID list but can’t, an alternative access path is calculated on the fly

RID list problems

- Difficult to analyze
- Explain looks different from the real access plan
- Worst case: Tablespace scan even though explain looks good
- STAT_RIDLIMTB: Number of times RID list exceeded maximum allowed for a query
- STAT_RIDSTORB: Number of times RID list ran out of space
- Increasing the RID pool size is a possible solution (or fix the index setup – better filtering can help)

Identify problems

- Statements without parameter markers
- Highest CPU / elapsed time
- Access path changes
- RID list problems
- Other indicators

Other indicators

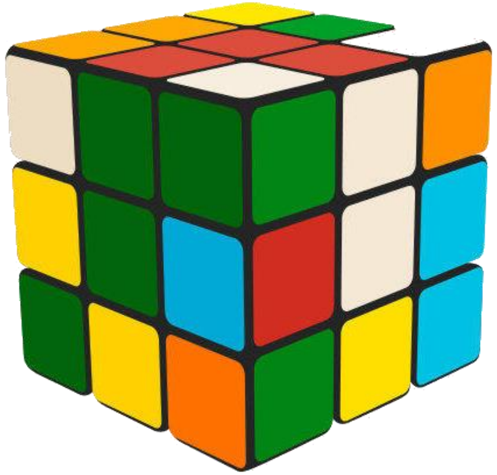
- DSC contains real values (not estimates), for example:
  - STAT_SYNRB → Synchronous Buffer Reads (sync. IO)
  - STAT_SORTB → Number of Sorts for each SQL statement
  - STAT_RSCANB → Number of tablespace scans
  - STAT_INDXB → Number of index scans
  - STAT_GPAGB → Number of getpages
- Monitor these over time
- Look at the top consumers, also look at values *per execution*

Other indicators

- Examined rows (STAT_EROWB) vs. processed rows (STAT_PROWB): Ratio is helpful to find bad access patterns
- High ratio is an indicator for access patterns with index issues
- Example: Tablespace scan, 100,000 rows, only five row matches the WHERE condition
- STAT_EROWB = 100,000, STAT_PROWB = 5, Ratio = 20,000
- Good starting point for the analysis, but also look at absolute values

Final words

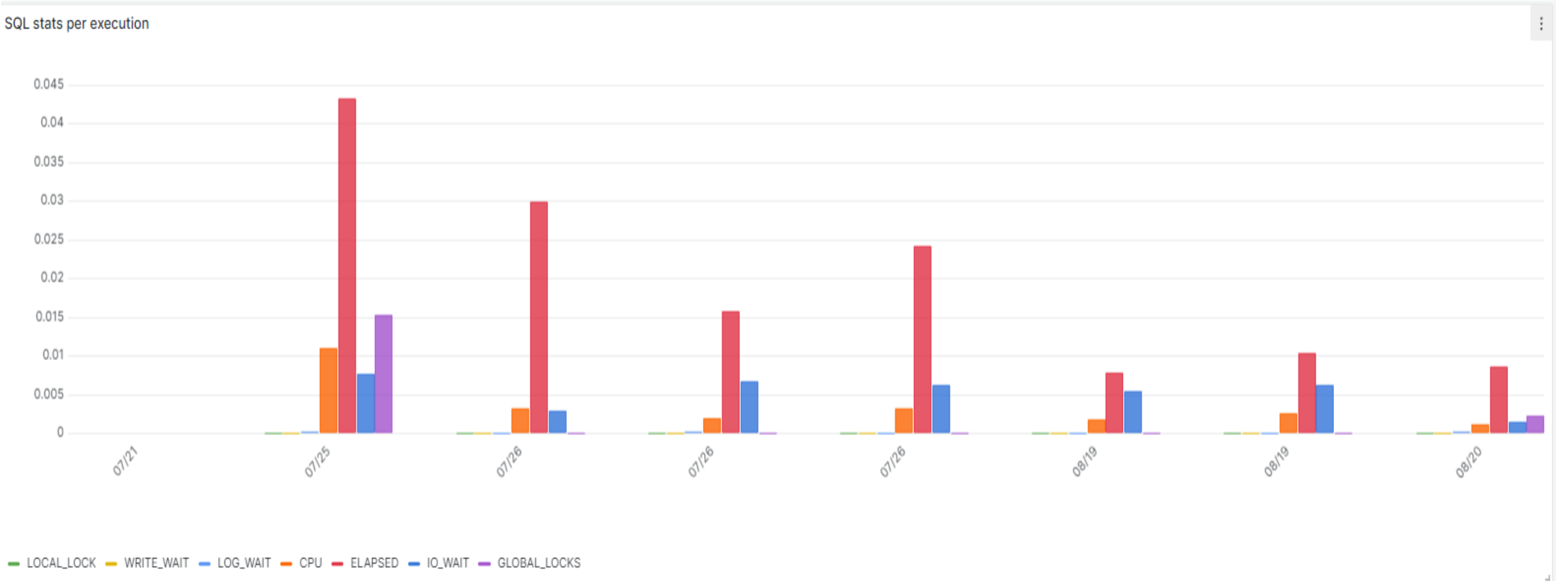
- This just scratched the surface. There are many more statistics to look at, and also things like static SQL, bufferpool configuration, etc.
- It’s great to look at the current state of affairs, but *much* more value can come from trends (=changes over time)
- Implement changes **carefully** as they never affect a single statement only
- Measure, measure, measure



Picture: deviantart.com, Creative Commons Attribution 3.0 License

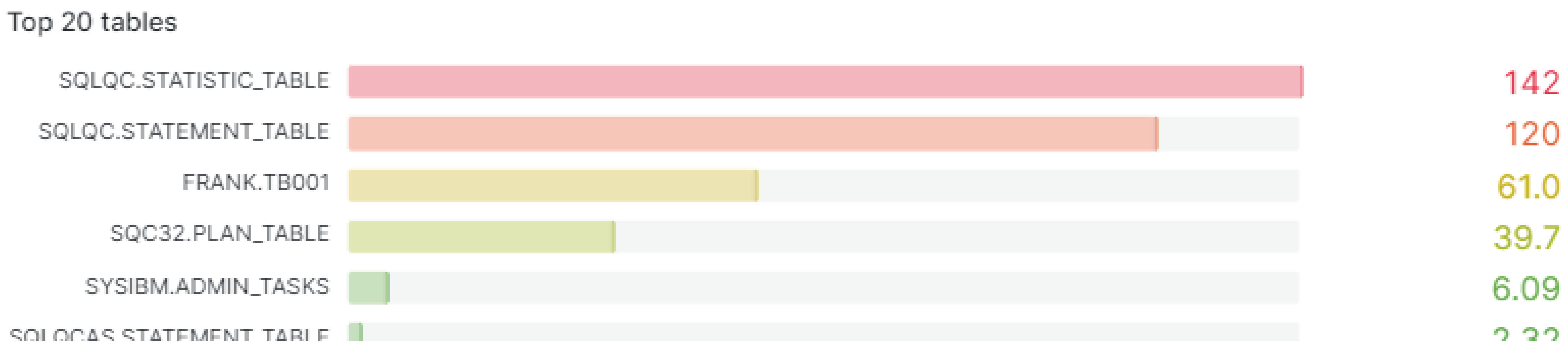
Don’t feel like doing it yourself?

- Software that does the heavy lifting for you is available
- Ours is called **SQLQC**.



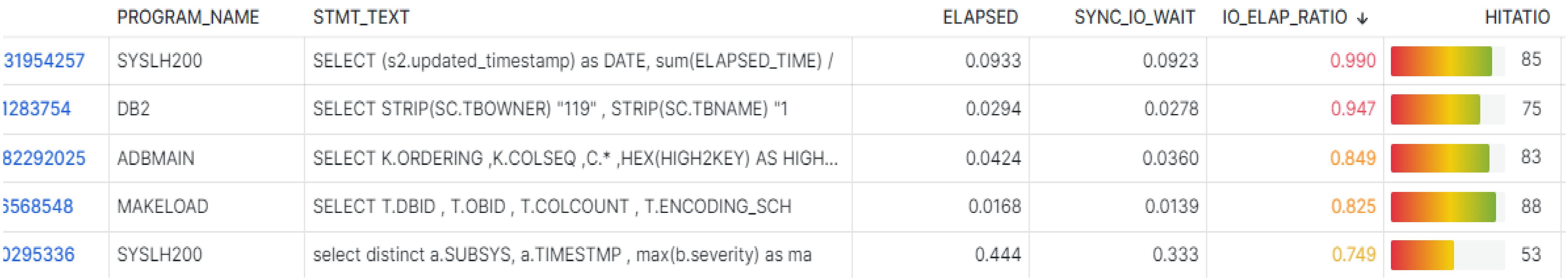
Don’t feel like doing it yourself?

- Software that does the heavy lifting for you is available
- Ours is called **SQLQC**.



Don’t feel like doing it yourself?

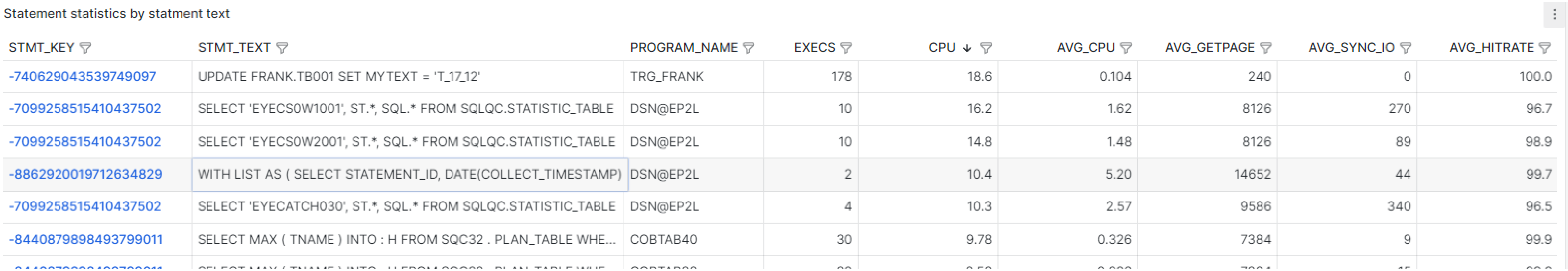
- Software that does the heavy lifting for you is available
- Ours is called **SQLQC**.



	PROGRAM_NAME	STMT_TEXT	ELAPSED	SYNC_IO_WAIT	IO_ELAP_RATIO ↓		HITATIO
31954257	SYSLH200	SELECT (s2.updated_timestamp) as DATE, sum(ELAPSED_TIME) /	0.0933	0.0923	0.990		85
1283754	DB2	SELECT STRIP(SC.TBOWNER) "119" , STRIP(SC.TBNAME) "1	0.0294	0.0278	0.947		75
82292025	ADBMAIN	SELECT K.ORDERING ,K.COLSEQ ,C.* ,HEX(HIGH2KEY) AS HIGH...	0.0424	0.0360	0.849		83
5568548	MAKELOAD	SELECT T.DBID , T.OBID , T.COLCOUNT , T.ENCODING_SCH	0.0168	0.0139	0.825		88
0295336	SYSLH200	select distinct a.SUBSYS, a.TIMESTMP , max(b.severity) as ma	0.444	0.333	0.749		53

Don’t feel like doing it yourself?

- Software that does the heavy lifting for you is available
- Ours is called **SQLQC**.



Statement statistics by statment text

STMT_KEY	STMT_TEXT	PROGRAM_NAME	EXECS	CPU ↓	AVG_CPU	AVG_GETPAGE	AVG_SYNC_IO	AVG_HITRATE
-740629043539749097	UPDATE FRANK.TB001 SET MYTEXT = 'T_17_12'	TRG_FRANK	178	18.6	0.104	240	0	100.0
-7099258515410437502	SELECT 'EYECS0W1001', ST.*, SQL.* FROM SQLQC.STATISTIC_TABLE	DSN@EP2L	10	16.2	1.62	8126	270	96.7
-7099258515410437502	SELECT 'EYECS0W2001', ST.*, SQL.* FROM SQLQC.STATISTIC_TABLE	DSN@EP2L	10	14.8	1.48	8126	89	98.9
-8862920019712634829	WITH LIST AS (SELECT STATEMENT_ID, DATE(COLLECT_TIMESTAMP)	DSN@EP2L	2	10.4	5.20	14652	44	99.7
-7099258515410437502	SELECT 'EYECATCH030', ST.*, SQL.* FROM SQLQC.STATISTIC_TABLE	DSN@EP2L	4	10.3	2.57	9586	340	96.5
-8440879898493799011	SELECT MAX (TNAME) INTO : H FROM SQC32 . PLAN_TABLE WHE...	COBTAB40	30	9.78	0.326	7384	9	99.9

Feel like doing it yourself?

- Download sample JCL and REXX: https://www.ubs-hainer.com/downloads/NEDB2UG.zip
- Contains four jobs and two REXX programs
- Look at readme.txt for installation and usage instructions

Thank you for your attention

Questions?

kai.stroh@ubs-hainer.com