DB2 LUW Powerful Monitoring Tools and Procedures

DB2 LUW

- Db2 night show #264
- C Raghavendra
- Senior staff software engineer
- IBM Software labs Data and AI

Achievements in IBM

1. Hall of Fame Award in the year -2019
2. Outstanding Technical Achievement Award – 2018
3. IBM Rock Star Award – 2018
4.Michelin Client Award for Automation – 2018
5.Successfull Completion of ONC Data Center Move project – 2017
6.GCH Approved ideas for db2 luw automation ideas – 2017
7.Eminence and Excellence Cash Prize Award - 2017
8.Eminence and Excellence Cash Prize Award – 2016
9. Top Innovator and Mentor Award – 2016
10. Top Innovator and Mentor Award – 2015
11.Michelin Client Certification of Appreciation – 2014
12.Appreciation from Database Service Engineering Team – 2020
13.Db2 LUW python Tool kit contribution of a performance shell script appreciation – 2020
14.Top Innovator and Mentor Award – 2020
15.IBM Gold Champion Learner -2020
16. IBM Silver Champion Learner -2023
17. IDUG Bronze Member Award – 2021
18. IDUG Silver Member Award – 2022
19. Best Automation Award – 2013
20. Best Automation Award - 2014

DB2 LUW

Built-in routines and views

Built-in administrative routines and views provide a simplified programmatic interface to administer and use databases and database objects through structured query language (SQL).

Built-in routines encompass procedures, scalar functions, and table functions.
You can run these built-in routines and views from an SQL-based application, a command line, or a command script.

To help ensure your successful use of the built-in routines and views, certain coding practices are recommended. These practices are especially important because routines might change from release to release and also within releases, such as through fix packs, as enhancements are made.

Authorizations for using built-in routines and views

For all built-in routines in the SYSPROC schema, you need EXECUTE privilege on the routine.

For all built-in views in the SYSIBMADM schema, you need SELECT privilege on the view.

Built-in routines

For all built-in routines in the SYSPROC schema, you need EXECUTE privilege on the routine. You can use the following query to check whether your authorization ID, or a group or a role to which you belong, has EXECUTE privilege

```
SELECT A.SPECIFICNAME, GRANTEE, GRANTEETYPE FROM SYSCAT.ROUTINEAUTH A, SYSCAT.ROUTINES R WHERE
A.SCHEMA = R.ROUTINESCHEMA AND A.SPECIFICNAME = R.SPECIFICNAME AND A.SCHEMA = 'SYSPROC' AND
R.ROUTINENAME = 'routine_name' AND A.EXECUTEAUTH <> 'N’
```

Built-in views

For all built-in views in the SYSIBMADM schema, you need SELECT privilege on the view. You can use the following query to check whether your authorization ID, or a group or a role to which you belong, has SELECT privilege:

```
SELECT GRANTEE, GRANTEETYPE FROM SYSCAT.TABAUTH WHERE TABSCHEMA = 'SYSIBMADM' AND TABNAME =
'view_name' AND SELECTAUTH <> 'N'
```

IBM

DB2 LUW

Snapshot Routines and Views

This grouping of routines and views can be used to retrieve information about the database and any connected applications at a specific time.

Some of the important Routines and Views are :

QUERY_PREP_COST administrative view - Retrieve statement prepare time information

SNAPDYN_SQL administrative view and SNAP_GET_DYN_SQL table function - Retrieve dynsql logical group snapshot information

SNAPUTIL_PROGRESS administrative view and SNAP_GET_UTIL_PROGRESS table function - Retrieve progress logical data group snapshot information

SNAPTAB_REORG administrative view and SNAP_GET_TAB_REORG table function - Retrieve table reorganization snapshot information

TOP_DYNAMIC_SQL administrative view - Retrieve information about the top dynamic SQL statements

DB2 LUW

QUERY_PREP_COST administrative view

The QUERY_PREP_COST administrative view returns a list of statements with information about the time required to prepare the statement.

The schema is SYSIBMADM.

Authorization
One of the following authorizations is required:

- SELECT privilege on the QUERY_PREP_COST administrative view
- CONTROL privilege on the QUERY_PREP_COST administrative view
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- ACCESSCTRL authority
- SECADM authority

Default PUBLIC privilege
In a non-restrictive database, SELECT privilege is granted to PUBLIC when the view is automatically created.

Example

Retrieve a report on the queries with the highest percentage of time spent on preparing.

```
SELECT NUM_EXECUTIONS, AVERAGE_EXECUTION_TIME_S, PREP_TIME_PERCENT,
   SUBSTR(STMT_TEXT, 1, 30) AS STMT_TEXT, DBPARTITIONNUM
   FROM SYSIBMADM.QUERY_PREP_COST ORDER BY PREP_TIME_PERCENT
```

The following is an example of output for this query.

```
NUM_EXECUTIONS     AVERAGE_EXECUTION_TIME_S ...
--------------...- ------------------------ ...
                 1                       25 ...

  1 record(s) selected.
```

Output for this query (continued).

```
... PREP_TIME_PERCENT STMT_TEXT                      DBPARTITIONNUM
... ----------------- ------------------------------ --------------
...               0.0 select * from dbuser.employee               0
```

DB2 LUW

SNAPDYN_SQL administrative view and SNAP_GET_DYN_SQL table function

SNAPDYN_SQL administrative view

This administrative view allows you to retrieve dynsql logical group snapshot information for the currently connected database.

This view returns information equivalent to the **GET SNAPSHOT FOR DYNAMIC SQL ON database- alias** CLP command.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required to use the view:

- SELECT privilege on the SNAPDYN_SQL administrative view
- CONTROL privilege on the SNAPDYN_SQL administrative view
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- ACCESSCTRL authority
- SECADM authority

Example

Retrieve a list of dynamic SQL run on the currently connected database, ordered by the number of rows read.

```
SELECT PREP_TIME_WORST, NUM_COMPILATIONS, SUBSTR(STMT_TEXT, 1, 60)
   AS STMT_TEXT, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPDYN_SQL ORDER BY ROWS_READ
```

The following is an example of output from this query.

```
PREP_TIME_WORST      NUM_COMPILATIONS     ...
-------------------- -------------------- ...
                  98                    1 ...
                   9                    1 ...
                   0                    0 ...
                   0                    1 ...
                   0                    1 ...
                   0                    1 ...
                   0                    1 ...
                   0                    1 ...
                  40                    1 ...

  9 record(s) selected.
```

Output from this query (continued).

```
... STMT_TEXT                                                    ...
... ------------------------------------------------------------ ...
... select prep_time_worst, num_compilations, substr(stmt_text,  ...
... select * from dbuser.employee                                ...
... SET CURRENT LOCALE LC_CTYPE = 'en_US'                        ...
... select prep_time_worst, num_compilations, substr(stmt_text,  ...
```

SNAP_GET_DYN_SQL table function

The SNAP_GET_DYN_SQL table function returns the same information as the SNAPDYN_SQL administrative view, but allows you to retrieve the information for a specific database on a specific database member, aggregate of all database members or all database members.

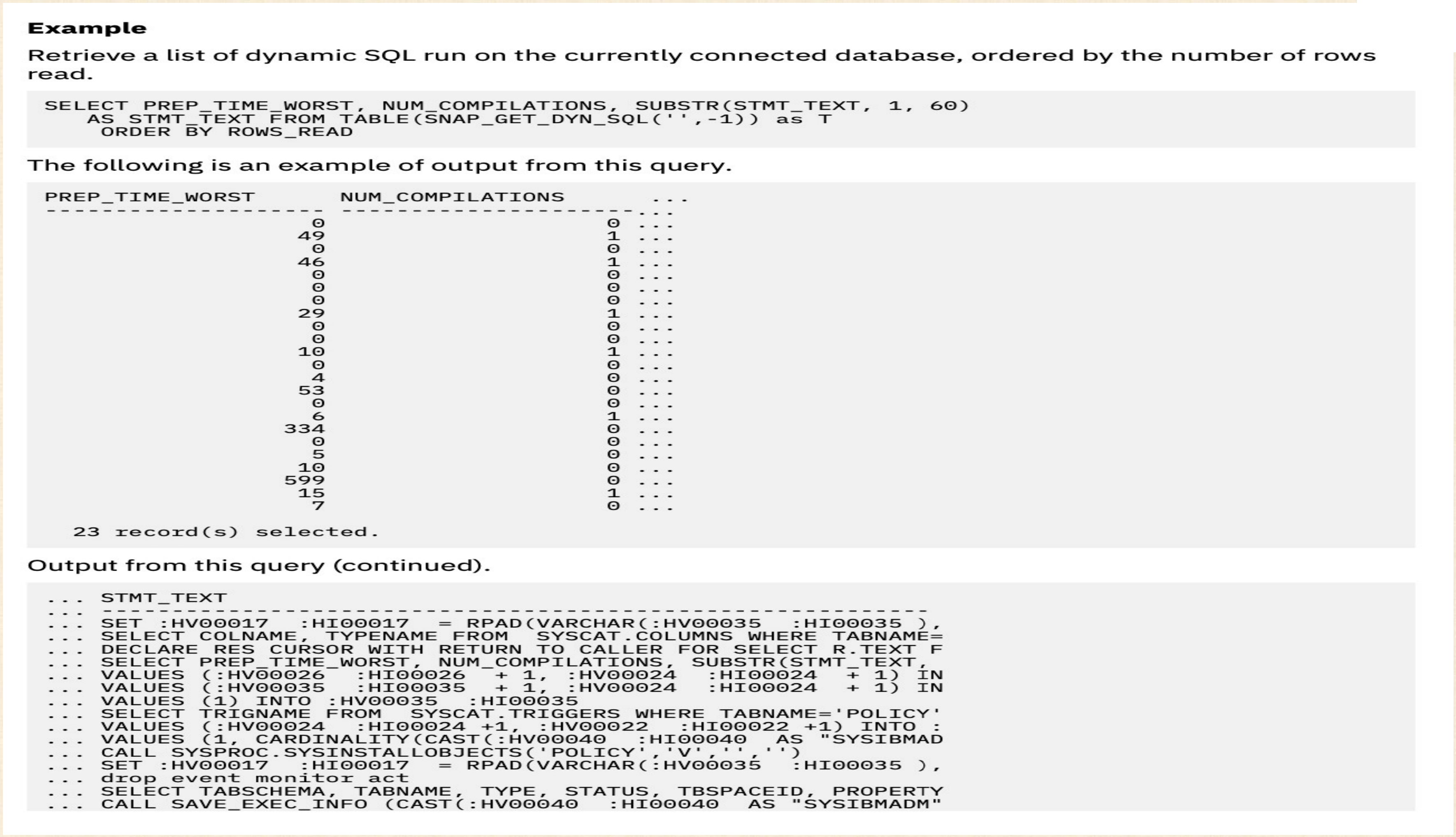
This table function returns information equivalent to the **GET SNAPSHOT FOR DYNAMIC SQL ON database-alias** CLP command.

Authorization
One of the following authorizations is required:

• EXECUTE privilege on the SNAP_GET_DYN_SQL table function
• DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required: • SYSMON
• SYSCTRL
• SYSMAINT
• SYSADM

DB2 LUW



Example

Retrieve a list of dynamic SQL run on the currently connected database, ordered by the number of rows read.

```
SELECT PREP_TIME_WORST, NUM_COMPILATIONS, SUBSTR(STMT_TEXT, 1, 60)
   AS STMT_TEXT FROM TABLE(SNAP_GET_DYN_SQL('',-1)) as T
     ORDER BY ROWS_READ
```

The following is an example of output from this query.

```
PREP_TIME_WORST       NUM_COMPILATIONS      ...
-------------------- ---------------------...
                   0                     0 ...
                  49                     1 ...
                   0                     0 ...
                  46                     1 ...
                   0                     0 ...
                   0                     0 ...
                   0                     0 ...
                  29                     1 ...
                   0                     0 ...
                   0                     0 ...
                  10                     1 ...
                   0                     0 ...
                   4                     0 ...
                  53                     0 ...
                   0                     0 ...
                   6                     1 ...
                 334                     0 ...
                   0                     0 ...
                   5                     0 ...
                  10                     0 ...
                 599                     0 ...
                  15                     1 ...
                   7                     0 ...

  23 record(s) selected.
```

Output from this query (continued).

```
... STMT_TEXT
... ------------------------------------------------------------
... SET :HV00017  :HI00017  = RPAD(VARCHAR(:HV00035  :HI00035 ),
... SELECT COLNAME, TYPENAME FROM  SYSCAT.COLUMNS WHERE TABNAME=
... DECLARE RES CURSOR WITH RETURN TO CALLER FOR SELECT R.TEXT F
... SELECT PREP_TIME_WORST, NUM_COMPILATIONS, SUBSTR(STMT_TEXT,
... VALUES (:HV00026  :HI00026  + 1, :HV00024  :HI00024  + 1) IN
... VALUES (:HV00035  :HI00035  + 1, :HV00024  :HI00024  + 1) IN
... VALUES (1) INTO :HV00035  :HI00035
... SELECT TRIGNAME FROM  SYSCAT.TRIGGERS WHERE TABNAME='POLICY'
... VALUES (:HV00024  :HI00024 +1, :HV00022  :HI00022 +1) INTO :
... VALUES (1, CARDINALITY(CAST(:HV00040  :HI00040  AS "SYSIBMAD
... CALL SYSPROC.SYSINSTALLOBJECTS('POLICY','V','','')
... SET :HV00017  :HI00017  = RPAD(VARCHAR(:HV00035  :HI00035 ),
... drop event monitor act
... SELECT TABSCHEMA, TABNAME, TYPE, STATUS, TBSPACEID, PROPERTY
... CALL SAVE_EXEC_INFO (CAST(:HV00040  :HI00040  AS "SYSIBMADM"
```

DB2 LUW

SNAPUTIL_PROGRESS administrative view and SNAP_GET_UTIL_PROGRESS table function

SNAPUTIL_PROGRESS administrative view

Used in conjunction with the SNAPUTIL administrative view, the SNAPUTIL_PROGRESS administrative view provides the same information as the **LIST UTILITIES SHOW DETAIL** CLP command.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required to use the view:

- SELECT privilege on the SNAPUTIL_PROGRESS administrative view
- CONTROL privilege on the SNAPUTIL_PROGRESS administrative view
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- ACCESSCTRL authority
- SECADM authority

```
SELECT UTILITY_ID, PROGRESS_TOTAL_UNITS, PROGRESS_COMPLETED_UNITS,
   DBPARTITIONNUM FROM SYSIBMADM.SNAPUTIL_PROGRESS
```

The following is an example of output from this query.

```
UTILITY_ID PROGRESS_TOTAL_UNITS PROGRESS_COMPLETED_UNITS DBPARTITIONNU
---------- -------------------- ------------------------ ------------
         7                   10                        5            0
         9                   10                        5            1

  1 record(s) selected.
```

SNAP_GET_UTIL_PROGRESS table function

The SNAP_GET_UTIL_PROGRESS table function returns the same information as the SNAPUTIL_PROGRESS administrative view, but allows you to retrieve the information for a specific database on a specific database member, aggregate of all database members or all database members.

Used in conjunction with the SNAP_GET_UTIL table function, the SNAP_GET_UTIL_PROGRESS table function provides the same information as the **LIST UTILITIES SHOW DETAIL** CLP command.

Authorization
One of the following authorizations is required:

• EXECUTE privilege on the SNAP_GET_UTIL_PROGRESS table function • DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:
• SYSMON
• SYSCTRL
• SYSMAINT • SYSADM

DB2 LUW

Example

Retrieve details on the progress of utilities on the currently connect member.

```
SELECT UTILITY_ID, PROGRESS_TOTAL_UNITS, PROGRESS_COMPLETED_UNITS,
   DBPARTITIONNUM FROM TABLE(SNAP_GET_UTIL_PROGRESS(-1)) as T
```

The following is an example of output from this query.

```
UTILITY_ID PROGRESS_TOTAL_UNITS PROGRESS_COMPLETED_UNITS DBPARTITIONNUM
---------- -------------------- ------------------------ --------------
         7                   10                        5              0

  1 record(s) selected.
```

DB2 LUW

SNAPTAB_REORG administrative view and SNAP_GET_TAB_REORG table function

SNAPTAB_REORG administrative view

This administrative view allows you to retrieve table reorganization snapshot information for the currently connected database.

Used with the SNAPTAB administrative view, the SNAPTAB_REORG administrative view provides the data equivalent to the **GET SNAPSHOT FOR TABLES ON database-alias** CLP command.

The schema is SYSIBMADM.
Authorization

One of the following authorizations is required to use the view:

• SELECT privilege on the SNAPTAB_REORG administrative view
• CONTROL privilege on the SNAPTAB_REORG administrative view • DATAACCESS authority
• DBADM authority
• SQLADM authority

Example

Select details on reorganization operations for all database members on the currently connected database.

```
SELECT SUBSTR(TABNAME, 1, 15) AS TAB_NAME, SUBSTR(TABSCHEMA, 1, 15)
   AS TAB_SCHEMA, REORG_PHASE, SUBSTR(REORG_TYPE, 1, 20) AS REORG_TYPE,
   REORG_STATUS, REORG_COMPLETION, DBPARTITIONNUM
   FROM SYSIBMADM.SNAPTAB_REORG ORDER BY DBPARTITIONNUM
```

The following is an example of output from this query.

```
TAB_NAME     TAB_SCHEMA     REORG_PHASE      ...
--------...- ----------...- ---------------- ...
EMPLOYEE     DBUSER         REPLACE          ...
EMPLOYEE     DBUSER         REPLACE          ...
EMPLOYEE     DBUSER         REPLACE          ...
                                             ...
3 record(s) selected.
```

Output from this query (continued).

```
... REORG_TYPE           REORG_STATUS REORG_COMPLETION DBPARTITIONNUM
... -------------------- ------------ ---------------- --------------
... RECLAIM+OFFLINE+ALLO COMPLETED    SUCCESS                       0
... RECLAIM+OFFLINE+ALLO COMPLETED    SUCCESS                       1
... RECLAIM+OFFLINE+ALLO COMPLETED    SUCCESS                       2
```

SNAP_GET_TAB_REORG table function

The SNAP_GET_TAB_REORG table function returns the same information as the SNAPTAB_REORG administrative view, but allows you to retrieve the information for a specific database on a specific database member, aggregate of all database members or all database members.

Used with the SNAP_GET_TAB table function, the SNAP_GET_TAB_REORG table function provides the data equivalent to the **GET SNAPSHOT FOR TABLES ON database-alias** CLP command.

Authorization

One of the following authorizations is required:
• EXECUTE privilege on the SNAP_GET_TAB_REORG table function • DATAACCESS authority

In addition, to access snapshot monitor data, one of the following authorities is also required:
• SYSMON
• SYSCTRL
• SYSMAINT • SYSADM

Example

Select details on reorganization operations for database member 1 on the currently connected database.

```
SELECT SUBSTR(TABNAME, 1, 15) AS TAB_NAME, SUBSTR(TABSCHEMA, 1, 15)
   AS TAB_SCHEMA, REORG_PHASE, SUBSTR(REORG_TYPE, 1, 20) AS REORG_TYPE,
   REORG_STATUS, REORG_COMPLETION, DBPARTITIONNUM
   FROM TABLE( SNAP_GET_TAB_REORG('', 1)) AS T
```

The following is an example of output from this query.

```
TAB_NAME      TAB_SCHEMA     REORG_PHASE     REORG_TYPE           ...
--------...- ----------...- -----------...- -------------------- ...
EMPLOYEE      DBUSER         REPLACE         RECLAIM+OFFLINE+ALLO ...
                                                                  ...
  1 record(s) selected.                                           ...
```

Output from this query (continued).

```
... REORG_STATUS REORG_COMPLETION DBPARTITIONNUM
... ------------ ---------------- --------------
... COMPLETED    SUCCESS                       1
...
```

Select all information about a reorganization operation to reclaim extents from a multidimensional clustering (MDC) or insert time clustering (ITC) table using the SNAP_GET_TAB_REORG table function.

```
db2 -v "select * from table(snap_get_tab_reorg(''))"

TABNAME  REORG_PHASE       REORG_MAX_PHASE   REORG_TYPE
-------- ----------------- ----------------- --------------------------
T1       RELEASE           3                 RECLAIM_EXTENTS+ALLOW_WRITE

REORG_STATUS REORG_COMPLETION REORG_START                REORG_END
------------ ---------------- -------------------------- --------------------------
COMPLETED    SUCCESS          2008-09-24-14.35.30.734741 2008-09-24-14.35.31.460674
```

IBM

DB2 LUW

TOP_DYNAMIC_SQL administrative view

The TOP_DYNAMIC_SQL administrative view returns the top dynamic SQL statements sortable by number of executions, average execution time, number of sorts, or sorts per statement.

The queries returned by TOP_DYNAMIC_SQL administrative view are the queries that should get focus to ensure they are well tuned.

The schema is SYSIBMADM.

Authorization

One of the following authorizations is required:

- SELECT privilege on the TOP_DYNAMIC_SQL administrative view
- CONTROL privilege on the TOP_DYNAMIC_SQL administrative view
- DATAACCESS authority
- DBADM authority
- SQLADM authority
- ACCESSCTRL authority
- SECADM authority

In addition, to access snapshot monitor data, one of the following authorities is also required: • SYSMON

- SYSCTRL
- SYSMAINT
- SYSADM

DB2 LUW

Example

Identify the top 5 most frequently run SQL.

```
SELECT NUM_EXECUTIONS, AVERAGE_EXECUTION_TIME_S, STMT_SORTS,
   SORTS_PER_EXECUTION, SUBSTR(STMT_TEXT,1,60) AS STMT_TEXT
   FROM SYSIBMADM.TOP_DYNAMIC_SQL
   ORDER BY NUM_EXECUTIONS DESC FETCH FIRST 5 ROWS ONLY
```

The following is an example of output for this query.

```
NUM_EXECUTIONS       AVERAGE_EXECUTION_TIME_S STMT_SORTS           ...
-------------------- ------------------------ -------------------- ...
                 148                        0                    0 ...
                 123                        0                    0 ...
                   2                        0                    0 ...
                   1                        0                    0 ...
                   1                        0                    0 ...

  5 record(s) selected.
```

Output for this query (continued).

```
... SORTS_PER_EXECUTION  ...
... -------------------- ...
...                    0 ...
...                    0 ...
...                    0 ...
...                    0 ...
...                    0 ...
```

Output for this query (continued).

```
... STMT_TEXT
... ------------------------------------------------------------
... SELECT A.ID, B.EMPNO, B.FIRSTNME, B.LASTNAME, A.DEPT FROM E
... SELECT A.EMPNO, A.FIRSTNME, A.LASTNAME, B.LOCATION, B.MGRNO
... SELECT A.EMPNO, A.FIRSTNME, A.LASTNAME, B.DEPTNAME FROM EMP
... SELECT ATM.SCHEMA, ATM.NAME, ATM.CREATE_TIME, ATM.LAST_WAIT,
... SELECT * FROM JESSICAE.EMP_RESUME
```

MONREPORT module

The MONREPORT module provides a set of procedures for retrieving a variety of monitoring data and generating text reports.

The schema for this module is SYSIBMADM.
The MONREPORT module includes the following built-in routines.

CONNECTION procedure : The connection report provides the monitor report for each connection

CURRRENT APPS procedure : The Current Applications report presents the current instantaneous state of processing of units of work, agents, and activities for each connection. The report starts with state information summed across connections, followed by a section for details for each connection.

CURRENTSQL procedure : The Current SQL report lists the top activities currently running, as measured by various metrics.

DBSUMMARY procedure : The Summary report contains in-depth monitor data for the entire database, as well as key performance indicators for each connection, workload, service class, and database member.

LOCKWAIT procedure : The Lock Waits report contains information about each lock wait currently in progress. Details include lock holder and requestor details, plus characteristics of the lock held and the lock requested.

PKGCACHE procedure : The Package Cache report lists the top statements accumulated in the package cache as measured by various metrics.

IBM

DB2 LUW

Examples

The following examples demonstrate various ways to call the CONNECTION procedure.
This example produces a report for all connections, with data displayed corresponding to an interval of 30 seconds:

```
call monreport.connection(30);
```

This example produces a report for a connection with an application handle of 34. Data is displayed based on absolute totals accumulated in the source table functions (rather than based on the current interval):

```
call monreport.connection(0, 34);
```

This next example produces a report for a connection with an application handle of 34. Data is displayed corresponding to an interval of 10 seconds.

```
call monreport.connection(DEFAULT, 34);
```

The final example produces the default report: for all connections, with data displayed corresponding to an interval of 10 seconds:

```
call monreport.connection;
```

IBM

DB2 LUW

The following examples demonstrate various ways to call the DBSUMMARY procedure.

The first example produces a report that displays data corresponding to an interval of 30 seconds.

```
call monreport.dbsummary(30);
```

The next example produces a report that displays data corresponding to an interval of 10 seconds (the default value):

```
call monreport.dbsummary;
```

The following examples demonstrate ways to call the CURRENTAPPS procedure:

```
call monreport.currentapps;

call monreport.currentapps();
```

The following examples demonstrate various ways to call the CURRENTSQL procedure.

The first example produces a report that shows activity metrics aggregated across all members:

```
call monreport.currentsql;
```

The next example produces a report that shows activity metrics specific to the activity performance on member number 4

```
call monreport.currentsql(4);
```

DB2 LUW

The following examples demonstrate various ways to call the LOCKWAIT procedure:

```
call monreport.lockwait;

call monreport.lockwait();
```

The following examples demonstrate various ways to call the PKGCACHE procedure.

The first example produces a report based on all statements in the package cache, with data aggregated across all members:

```
call monreport.pkgcache;
```

The next example produces a report based on both dynamic and static statements in the package cache for which metrics have been updated within the last 30 minutes, with data aggregated across all members:

```
call monreport.pkgcache(30);
```

The next example produces a report based on all dynamic statements in the package cache, with data aggregated across all members:

```
call monreport.pkgcache(DEFAULT, 'd');
```

The next example produces a report based on both dynamic and static statements in the package cache for which metrics have been updated within the last 30 minutes, with data specific to a member number 4:

```
call db2monreport.pkgcache(30, DEFAULT, 4);
```

MON_GET Family of Functions

When I started working with DB2 (on versions 5 and 7), the only option for seeing what was going on in the database was snapshots. I spent a fair amount of time parsing the data in snapshots with Perl scripts to filter out only the data I wanted to see. Slowly, methods of using SQL to access snapshot data were introduced. First table functions for accessing snapshot data, then SYSIBMADM views, and finally we have the MON_GET family of table functions.

Compared to Using Snapshot Data

One of the things I loved about snapshot data is that it covered or could easily cover a discrete time period. By default, that time period was since database activation, but I could reset monitor switches and only see data since that reset. The reset was specific to the session I was in, so other sessions would have other reset points. In fact, I ran scripts that would reset the monitor switches, sleep for an hour, and then grab a set of snapshots. Keeping this data for a week or a month, I then had a basis for comparison for any numbers I was looking at. Granted, this meant that I had a bunch of text files out there to parse through, but grep can work wonders, and when it's not enough there's always Perl's powerful text-parsing capabilities.

There is no reset with data in SYSIBMADM views or monitoring table functions. The data is always since the last database restart. One advantage of using the mon_get monitoring functions is that the collection of data is "in-memory" and considered more lightweight than snapshots. It is also the strategic direction for IBM going forward, and new elements and even table functions are constantly being added.

IBM

DB2 LUW

How to Find what Table Functions are Available?

If I simply want a list of the available monitoring functions, I use this SQL

```
select substr(r.ROUTINENAME,1,48) as ROUTINENAME, substr(r.SPECIFICNAME,1,48) as SPECIFICNAME from
sysibm.sysroutines r where r.function_type='T' and substr(r.ROUTINENAME,1,4) in ('SNAP','MON_','ENV_','COMP')
and substrb(r.SPECIFICNAME,-3,3) not in ('V91', 'V95', 'V97', '_AP') order by
r.ROUTINESCHEMA,r.ROUTINENAME,r.SPECIFICNAME;
```

```
select substr(P.ROUTINENAME,1,48) as ROUTINENAME, substr(P.SPECIFICNAME,1,48) as SPECIFICNAME, case when
P.ROWTYPE in ('B','O','P') then CHAR('IN',3) else CHAR('OUT',3) end as IN_OUT, cast(p.ORDINAL as char(3)) as ORD,
substr(P.PARMNAME,1,40) as PARMNAME, substr(P.TYPENAME,1,16) as TYPE from sysibm.sysroutines r,
sysibm.sysroutineparms p where p.routineschema=r.routineschema and p.routinename=r.routinename and
p.specificname=r.specificname and r.function_type='T' and r.ROUTINENAME='MON_GET_TABLE' order by
P.ROUTINESCHEMA,P.ROUTINENAME,P.SPECIFICNAME,IN_OUT,P.ORDINAL;
```

DB2 LUW

Critical Table Functions that can be used in DB2 LUW

1. MON_GET_BUFFERPOOL :
2. MON_GET_DATABASE:
3. MON_GET_TABLE :
4. MON_GET_INDEX:
5. MON_GET_TABLESPACE:
6. MON_GET_TRANSACTION_LOG
7. MON_GET_CONNECTION
8. MON_GET_INSTANCE
9. MON_GET_LATCH
10. MON_GET_GROUP_BUFFERPOOL
11. MON_GET_MEMORY_POOL
12. MON_GET_MEMORY_SET
13. MON_GET_DATABASE_DETAILS
14. MON_GET_AGENT
15. MON_GET_APPLICATION_HANDLE
16. MON_GET_APPL_LOCKWAIT
17. MON_GET_CONTAINER
18. MON_GET_HADR
19. MON_GET_PAGE_ACCESS_INFO
20. MON_GET_PKG_CACHE_STMT
21. MON_GET_PKG_CACHE_STMT_DETAILS

IBM

DB2 LUW

MON_GET_BUFFERPOOL

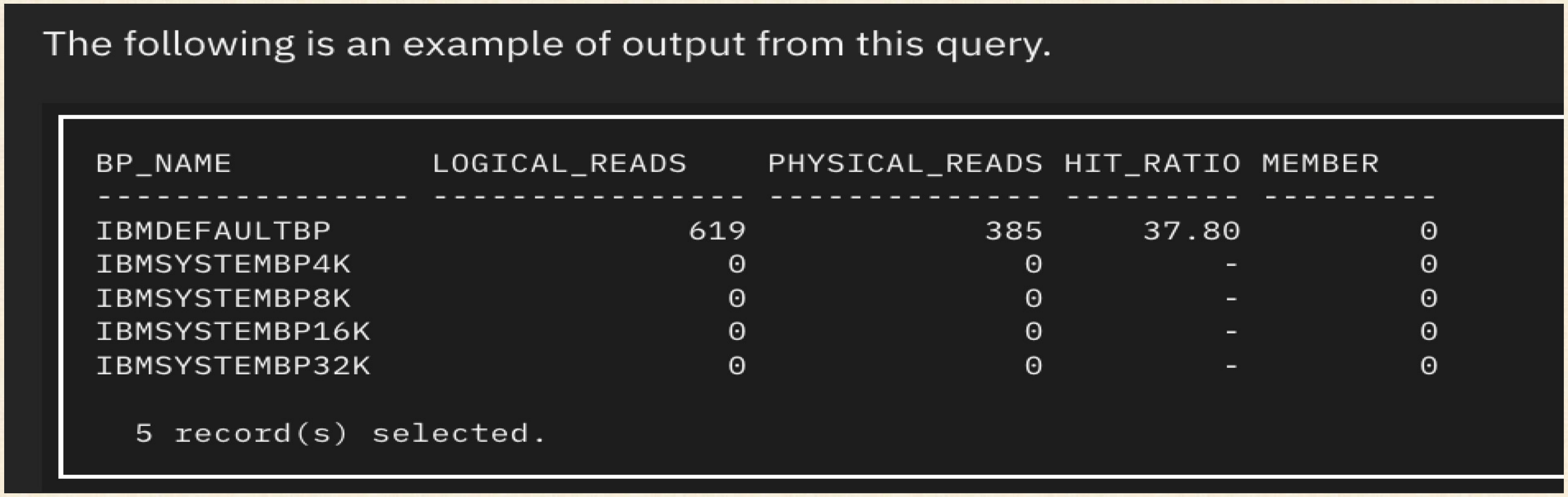
The above table function can be used to get the bufferpool metrics and returns monitor metrics for one or more buffer pools.

Example :

Computing the Bufferpool Hit Ratio

```
WITH BPMETRICS AS
( SELECT bp_name, pool_data_l_reads + pool_temp_data_l_reads +
pool_index_l_reads + pool_temp_index_l_reads + pool_xda_l_reads +
pool_temp_xda_l_reads as logical_reads, pool_data_p_reads +
pool_temp_data_p_reads + pool_index_p_reads + pool_temp_index_p_reads +
pool_xda_p_reads + pool_temp_xda_p_reads as physical_reads, member FROM
TABLE(MON_GET_BUFFERPOOL('',-2)) AS METRICS)
SELECT VARCHAR(bp_name,20) AS bp_name, logical_reads, physical_reads, CASE
WHEN logical_reads > 0 THEN DEC((1 - (FLOAT(physical_reads) /
FLOAT(logical_reads))) * 100,5,2) ELSE NULL END AS HIT_RATIO, member FROM
BPMETRICS;
```

DB2 LUW

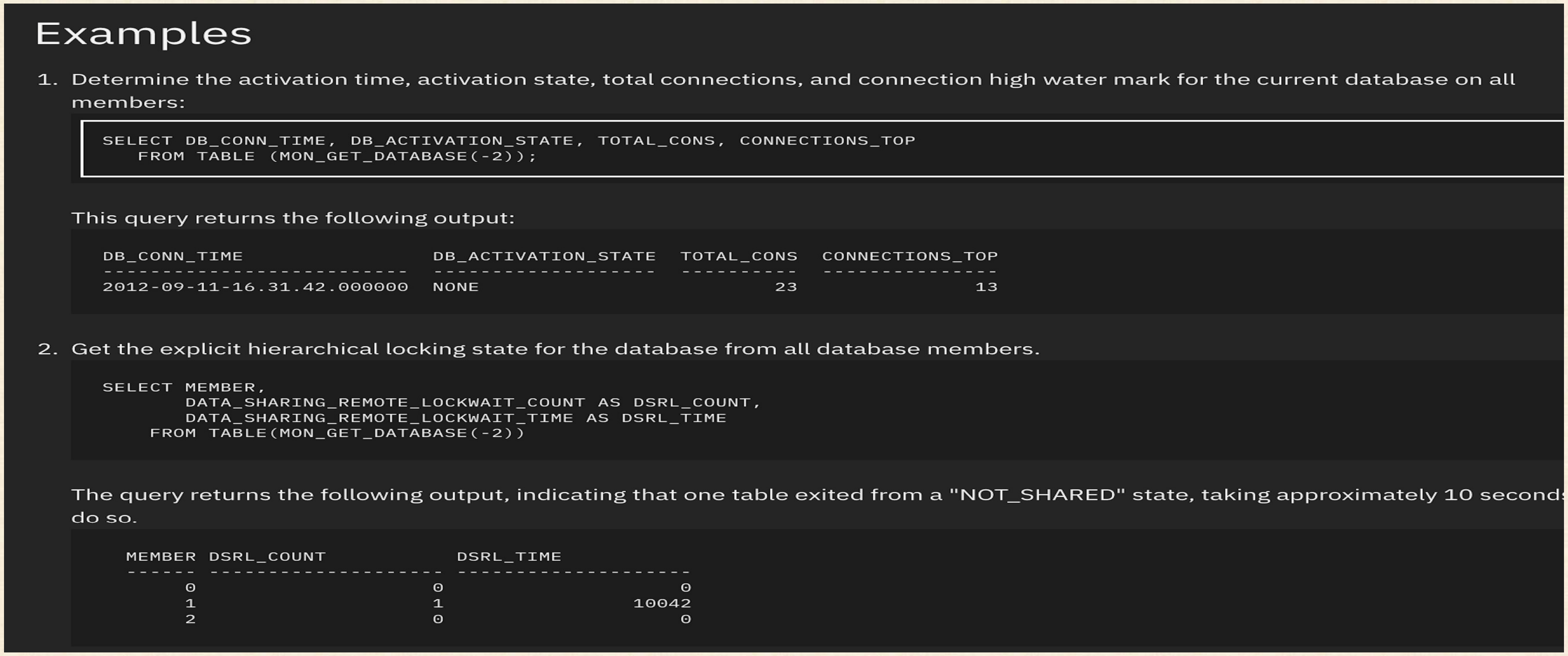

The following is an example of output from this query.

```
BP_NAME          LOGICAL_READS    PHYSICAL_READS HIT_RATIO MEMBER
---------------- ---------------- -------------- --------- ---------
IBMDEFAULTBP                  619            385     37.80         0
IBMSYSTEMBP4K                   0              0         -         0
IBMSYSTEMBP8K                   0              0         -         0
IBMSYSTEMBP16K                  0              0         -         0
IBMSYSTEMBP32K                  0              0         -         0

  5 record(s) selected.
```

MON_GET_DATABASE

The MON_GET_DATABASE table function can be used to get the database metrics, and it returns the database information within the monitor infrastructure

DB2 LUW



Examples

1. Determine the activation time, activation state, total connections, and connection high water mark for the current database on all members:

```
SELECT DB_CONN_TIME, DB_ACTIVATION_STATE, TOTAL_CONS, CONNECTIONS_TOP
   FROM TABLE (MON_GET_DATABASE(-2));
```

   This query returns the following output:

```
DB_CONN_TIME                 DB_ACTIVATION_STATE  TOTAL_CONS  CONNECTIONS_TOP
--------------------------   -------------------  ----------  ---------------
2012-09-11-16.31.42.000000   NONE                         23               13
```

2. Get the explicit hierarchical locking state for the database from all database members.

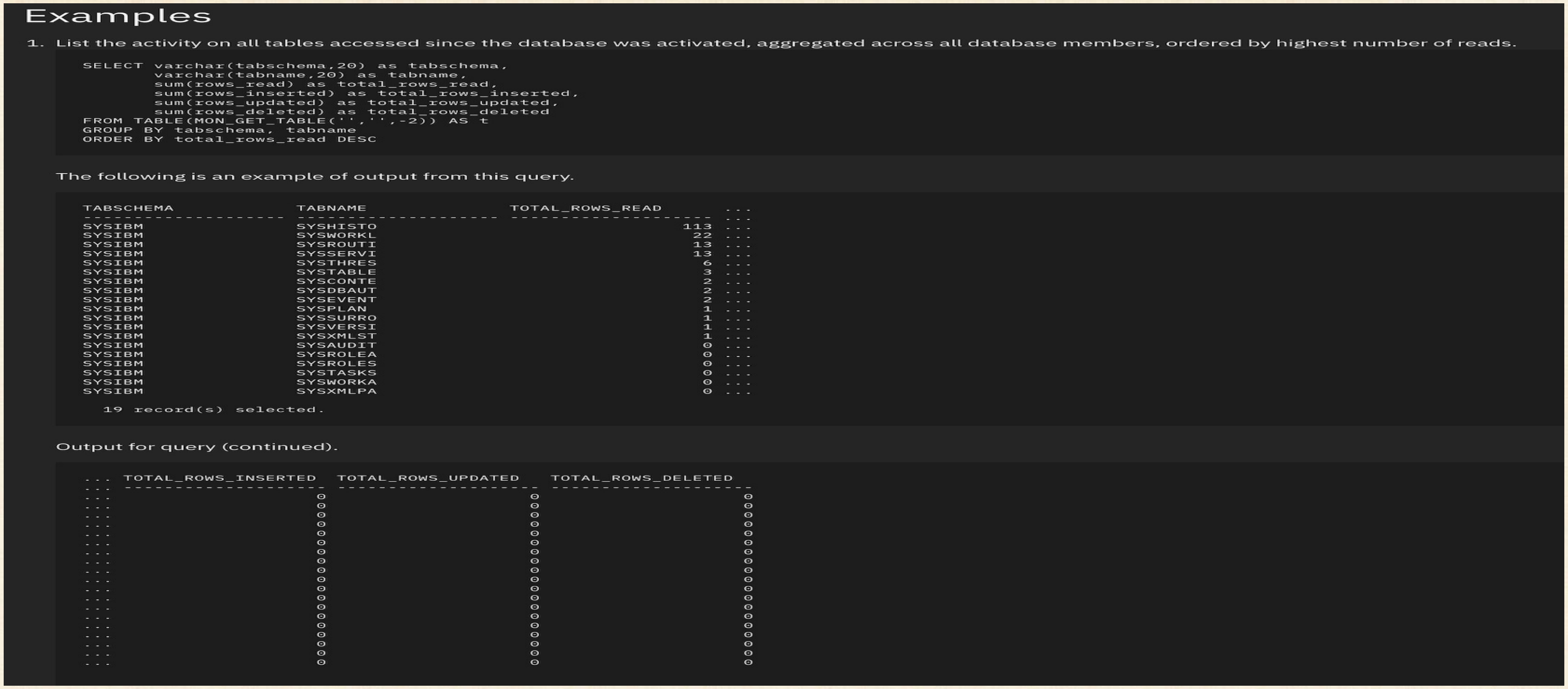
```
SELECT MEMBER,
       DATA_SHARING_REMOTE_LOCKWAIT_COUNT AS DSRL_COUNT,
       DATA_SHARING_REMOTE_LOCKWAIT_TIME AS DSRL_TIME
    FROM TABLE(MON_GET_DATABASE(-2))
```

   The query returns the following output, indicating that one table exited from a "NOT_SHARED" state, taking approximately 10 second do so.

```
  MEMBER DSRL_COUNT           DSRL_TIME
  ------ -------------------- --------------------
       0                    0                    0
       1                    1                10042
       2                    0                    0
```

MON_GET_TABLE :

This table function can be used to get the Table metrics for one or more tables in the database



Examples

1. List the activity on all tables accessed since the database was activated, aggregated across all database members, ordered by highest number of reads.

```
SELECT varchar(tabschema,20) as tabschema,
       varchar(tabname,20) as tabname,
       sum(rows_read) as total_rows_read,
       sum(rows_inserted) as total_rows_inserted,
       sum(rows_updated) as total_rows_updated,
       sum(rows_deleted) as total_rows_deleted
FROM TABLE(MON_GET_TABLE('','',-2)) AS t
GROUP BY tabschema, tabname
ORDER BY total_rows_read DESC
```

The following is an example of output from this query.

```
TABSCHEMA            TABNAME              TOTAL_ROWS_READ      ...
-------------------- -------------------- -------------------- ...
SYSIBM               SYSHISTO                              113 ...
SYSIBM               SYSWORKL                               22 ...
SYSIBM               SYSROUTI                               13 ...
SYSIBM               SYSSERVI                               13 ...
SYSIBM               SYSTHRES                                6 ...
SYSIBM               SYSTABLE                                3 ...
SYSIBM               SYSCONTE                                2 ...
SYSIBM               SYSDBAUT                                2 ...
SYSIBM               SYSEVENT                                2 ...
SYSIBM               SYSPLAN                                 1 ...
SYSIBM               SYSSURRO                                1 ...
SYSIBM               SYSVERSI                                1 ...
SYSIBM               SYSXMLST                                1 ...
SYSIBM               SYSAUDIT                                0 ...
SYSIBM               SYSROLEA                                0 ...
SYSIBM               SYSROLES                                0 ...
SYSIBM               SYSTASKS                                0 ...
SYSIBM               SYSWORKA                                0 ...
SYSIBM               SYSXMLPA                                0 ...

  19 record(s) selected.
```

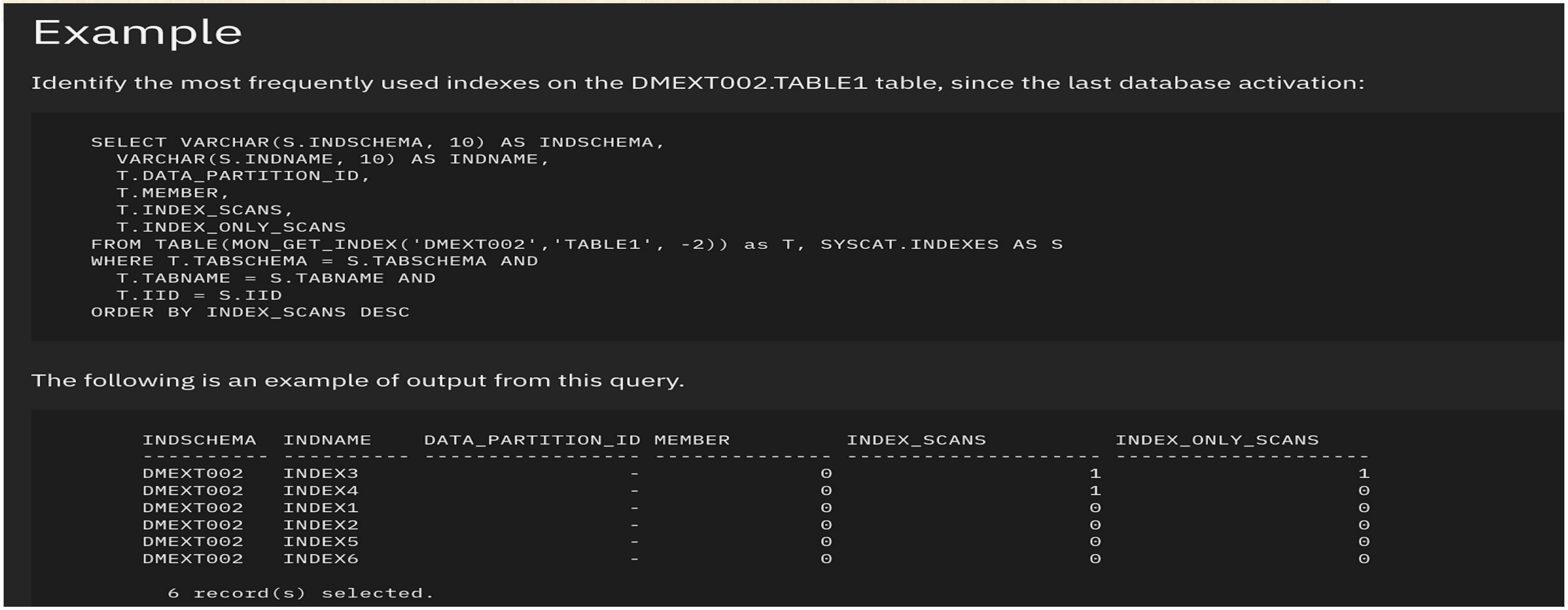
Output for query (continued).

```
... TOTAL_ROWS_INSERTED  TOTAL_ROWS_UPDATED   TOTAL_ROWS_DELETED
... -------------------- -------------------- --------------------
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
...                    0                    0                    0
```

DB2 LUW

MON_GET_INDEX :

This table function can be used to get the index metrics for the table in a database



Example

Identify the most frequently used indexes on the DMEXT002.TABLE1 table, since the last database activation:

```
SELECT VARCHAR(S.INDSCHEMA, 10) AS INDSCHEMA,
  VARCHAR(S.INDNAME, 10) AS INDNAME,
  T.DATA_PARTITION_ID,
  T.MEMBER,
  T.INDEX_SCANS,
  T.INDEX_ONLY_SCANS
FROM TABLE(MON_GET_INDEX('DMEXT002','TABLE1', -2)) as T, SYSCAT.INDEXES AS S
WHERE T.TABSCHEMA = S.TABSCHEMA AND
  T.TABNAME = S.TABNAME AND
  T.IID = S.IID
ORDER BY INDEX_SCANS DESC
```

The following is an example of output from this query.

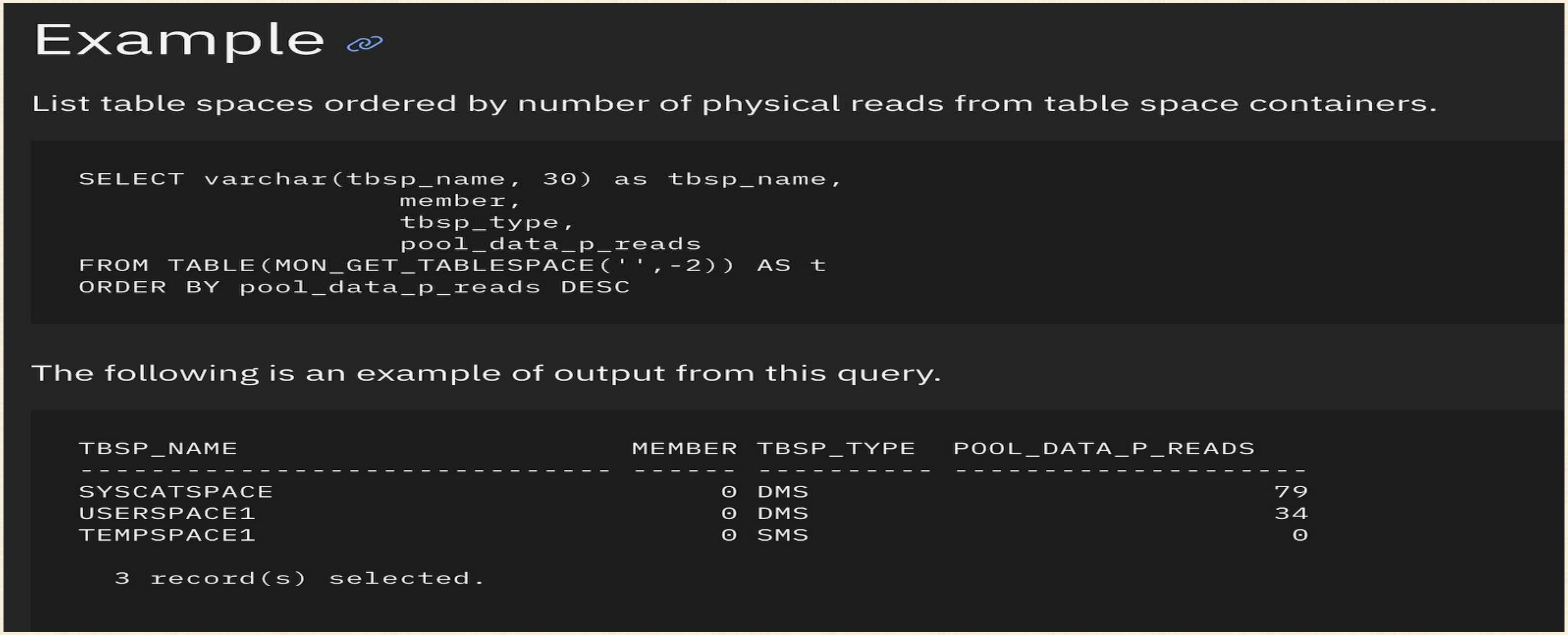
```
INDSCHEMA  INDNAME    DATA_PARTITION_ID MEMBER         INDEX_SCANS          INDEX_ONLY_SCANS
---------- ---------- ----------------- -------------- -------------------- --------------------
DMEXT002   INDEX3                     -              0                    1                    1
DMEXT002   INDEX4                     -              0                    1                    0
DMEXT002   INDEX1                     -              0                    0                    0
DMEXT002   INDEX2                     -              0                    0                    0
DMEXT002   INDEX5                     -              0                    0                    0
DMEXT002   INDEX6                     -              0                    0                    0

  6 record(s) selected.
```

IBM

DB2 LUW

MON_GET_TABLESPACE :

This table function can be used to get the tablespace metrics in the database



Example

List table spaces ordered by number of physical reads from table space containers.

```
SELECT varchar(tbsp_name, 30) as tbsp_name,
                  member,
                  tbsp_type,
                  pool_data_p_reads
FROM TABLE(MON_GET_TABLESPACE('',-2)) AS t
ORDER BY pool_data_p_reads DESC
```

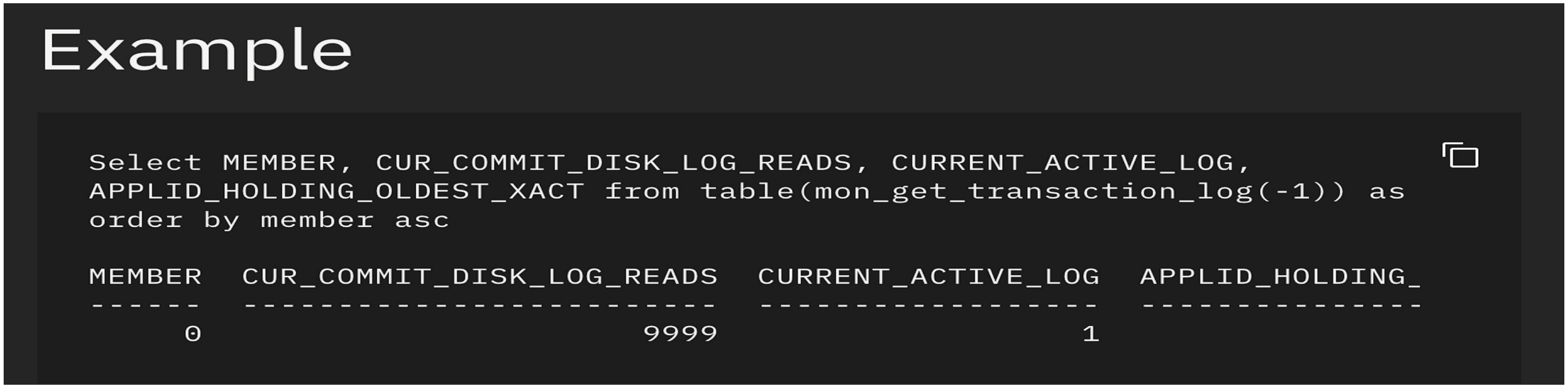
The following is an example of output from this query.

```
TBSP_NAME                      MEMBER TBSP_TYPE  POOL_DATA_P_READS
------------------------------ ------ ---------- --------------------
SYSCATSPACE                         0 DMS                          79
USERSPACE1                          0 DMS                          34
TEMPSPACE1                          0 SMS                           0

  3 record(s) selected.
```

DB2 LUW

MON_GET_TRANSACTION_LOG :

The MON_GET_TRANSACTION_LOG table function returns information about the transaction logging subsystem for the currently connected database.



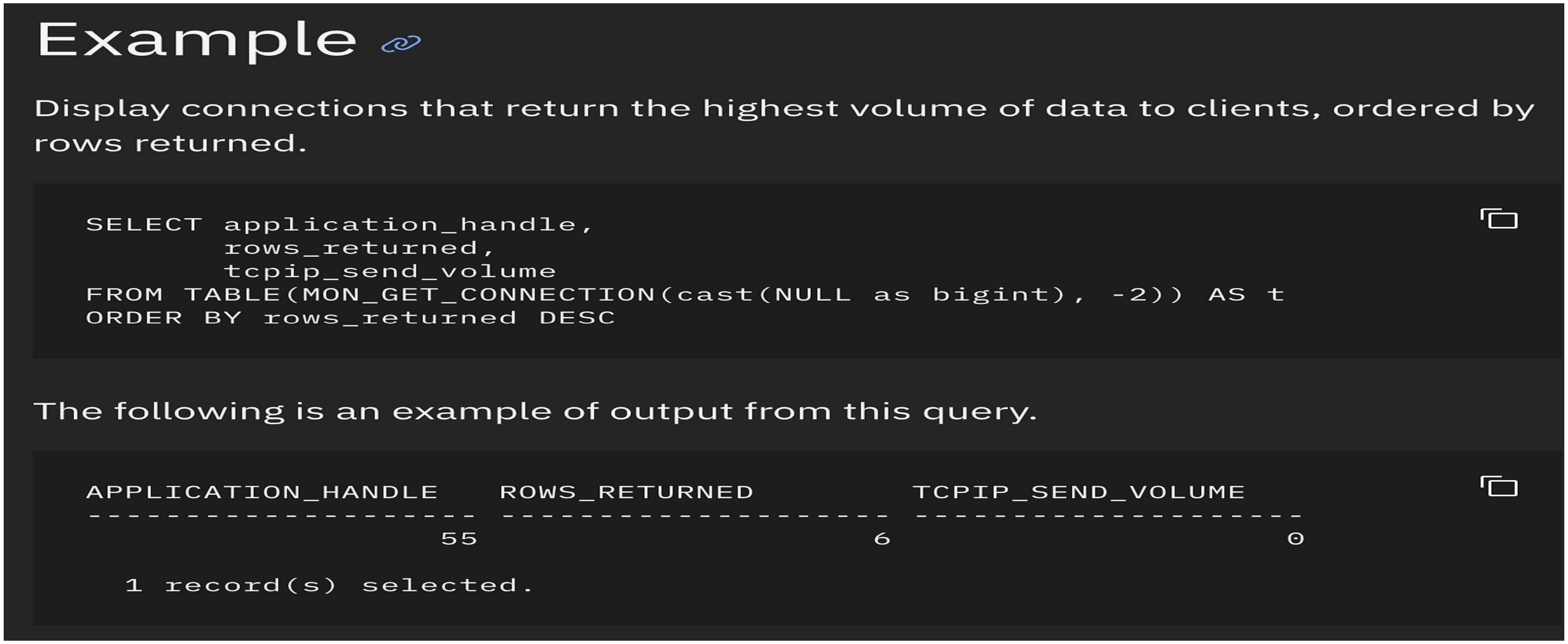
Example

```
Select MEMBER, CUR_COMMIT_DISK_LOG_READS, CURRENT_ACTIVE_LOG,
APPLID_HOLDING_OLDEST_XACT from table(mon_get_transaction_log(-1)) as
order by member asc

MEMBER  CUR_COMMIT_DISK_LOG_READS  CURRENT_ACTIVE_LOG  APPLID_HOLDING_
------  -------------------------  ------------------  ---------------
     0                       9999                   1
```

DB2 LUW

MON_GET_CONNECTION

The MON_GET_CONNECTION table function returns metrics for one or more connections.



Example

Display connections that return the highest volume of data to clients, ordered by rows returned.

```
SELECT application_handle,
       rows_returned,
       tcpip_send_volume
FROM TABLE(MON_GET_CONNECTION(cast(NULL as bigint), -2)) AS t
ORDER BY rows_returned DESC
```

The following is an example of output from this query.

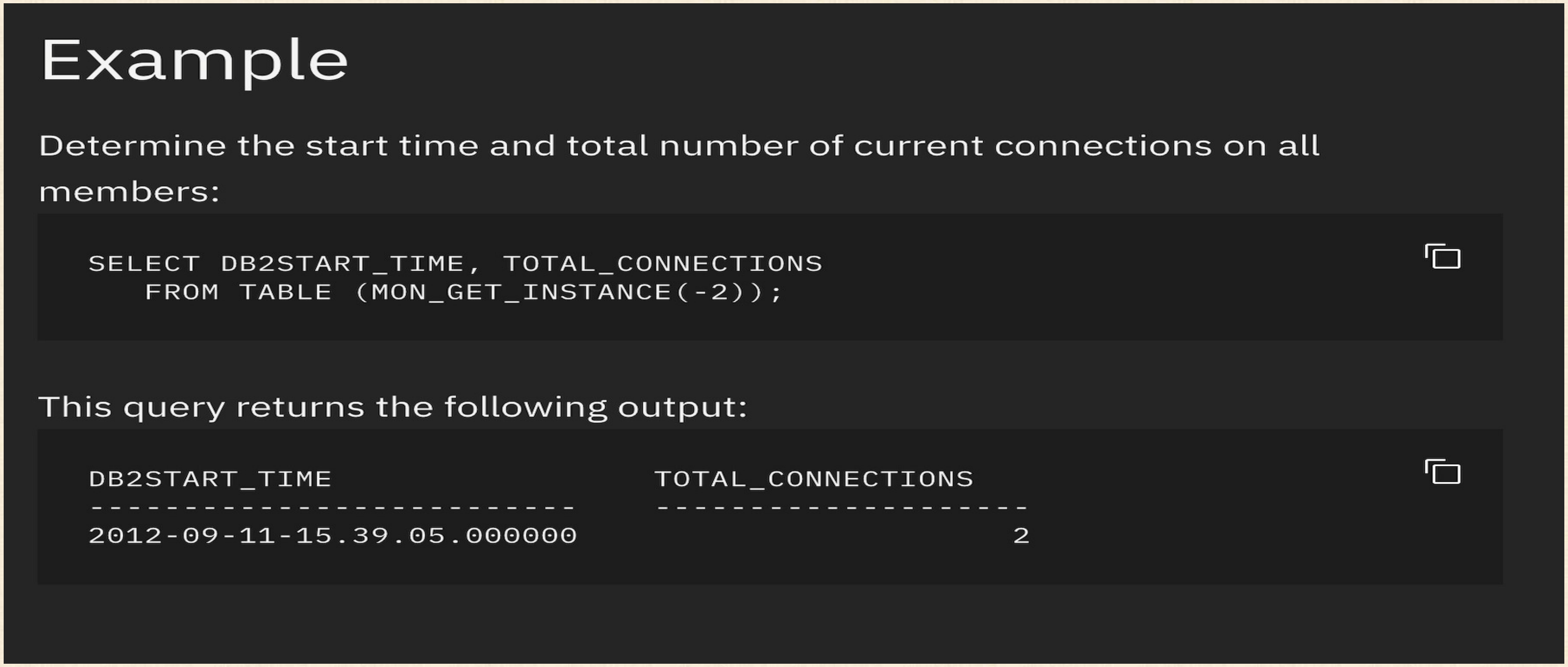
```
APPLICATION_HANDLE   ROWS_RETURNED        TCPIP_SEND_VOLUME
-------------------- -------------------- --------------------
                  55                    6                    0

  1 record(s) selected.
```

IBM

DB2 LUW

MON_GET_INSTANCE

This table function can be used to get the metrics of the db2 instance related



Example

Determine the start time and total number of current connections on all members:

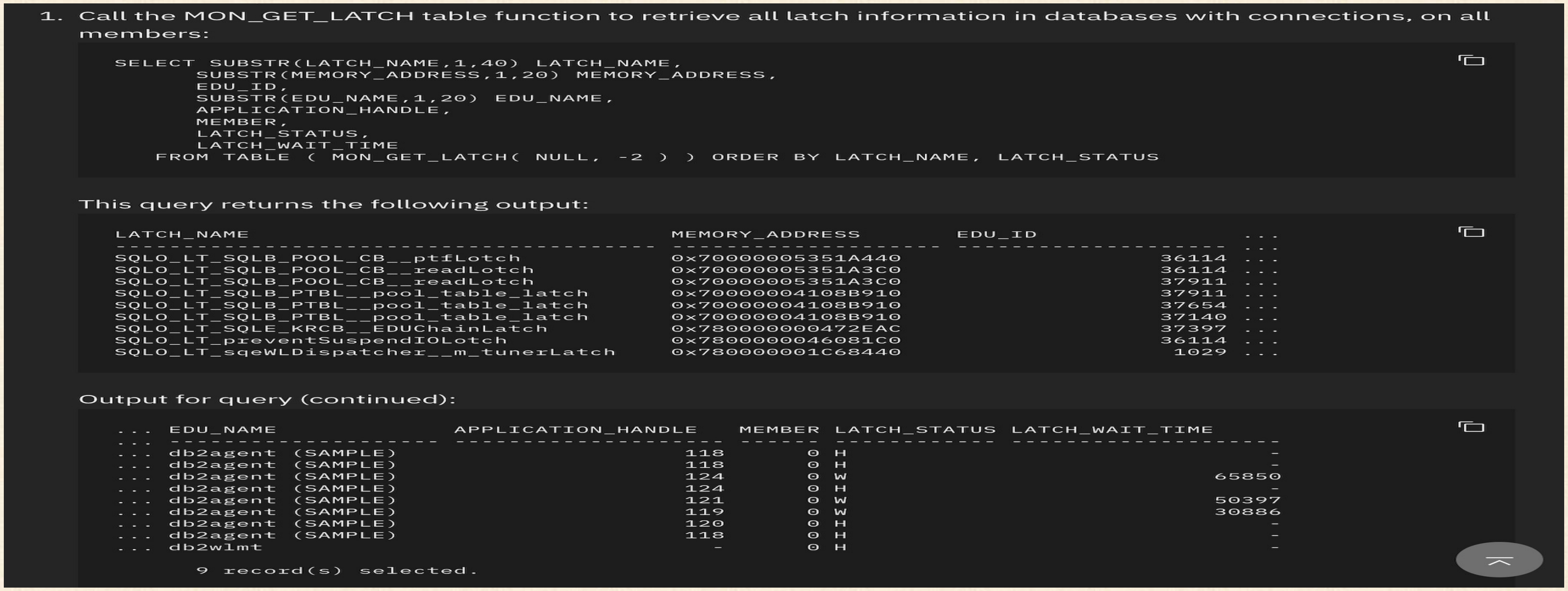
```
SELECT DB2START_TIME, TOTAL_CONNECTIONS
   FROM TABLE (MON_GET_INSTANCE(-2));
```

This query returns the following output:

```
DB2START_TIME                 TOTAL_CONNECTIONS
--------------------------    --------------------
2012-09-11-15.39.05.000000                       2
```

MON_GET_LATCH

The MON_GET_LATCH table function returns a list of all latches in the current member.



1. Call the MON_GET_LATCH table function to retrieve all latch information in databases with connections, on all members:

```
SELECT SUBSTR(LATCH_NAME,1,40) LATCH_NAME,
       SUBSTR(MEMORY_ADDRESS,1,20) MEMORY_ADDRESS,
       EDU_ID,
       SUBSTR(EDU_NAME,1,20) EDU_NAME,
       APPLICATION_HANDLE,
       MEMBER,
       LATCH_STATUS,
       LATCH_WAIT_TIME
   FROM TABLE ( MON_GET_LATCH( NULL, -2 ) ) ORDER BY LATCH_NAME, LATCH_STATUS
```

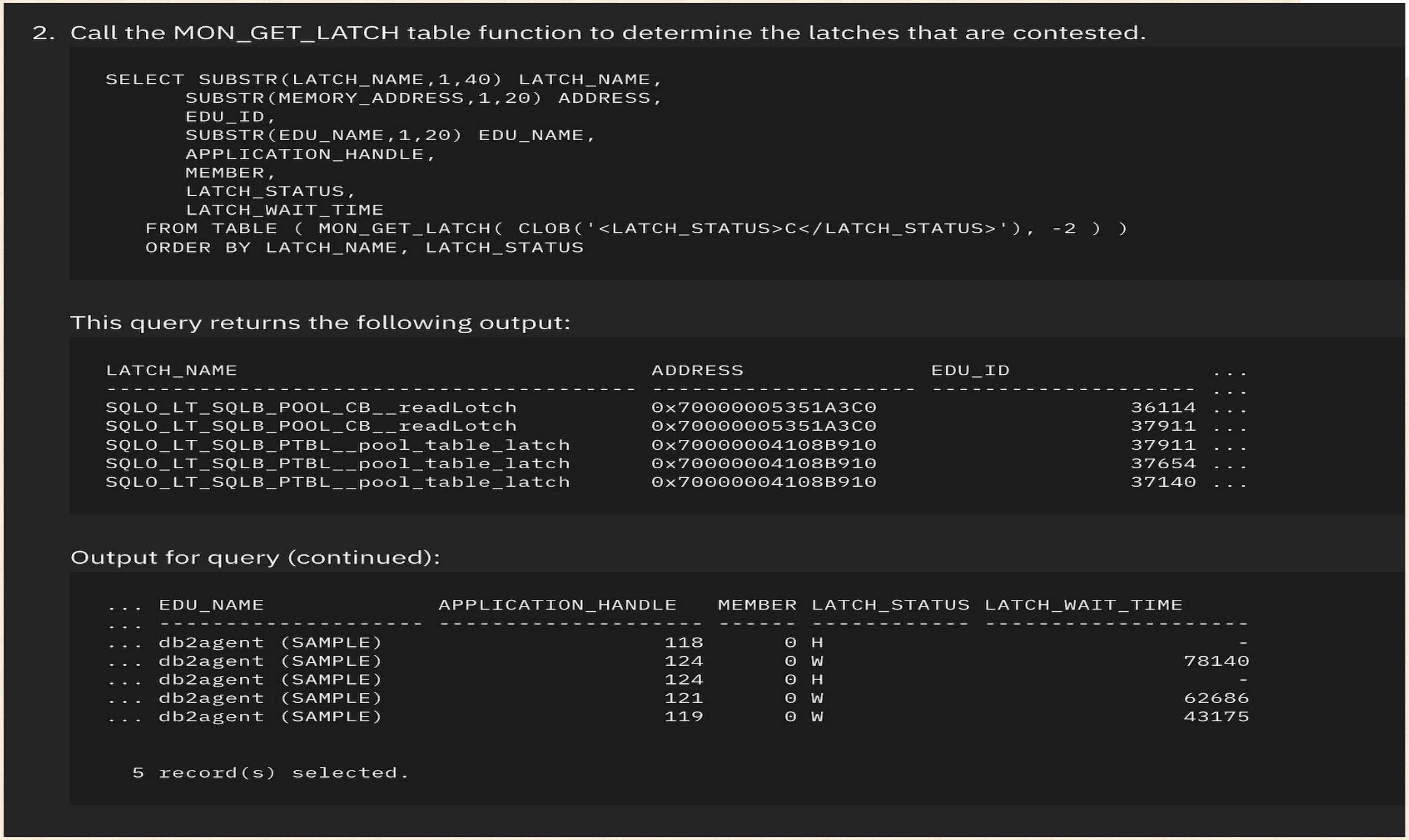
This query returns the following output:

```
LATCH_NAME                                 MEMORY_ADDRESS        EDU_ID                ...
------------------------------------------ --------------------- --------------------- ...
SQLO_LT_SQLB_POOL_CB__ptfLotch             0x70000005351A440                     36114 ...
SQLO_LT_SQLB_POOL_CB__readLotch            0x70000005351A3C0                     36114 ...
SQLO_LT_SQLB_POOL_CB__readLotch            0x70000005351A3C0                     37911 ...
SQLO_LT_SQLB_PTBL__pool_table_latch        0x700000004108B910                    37911 ...
SQLO_LT_SQLB_PTBL__pool_table_latch        0x700000004108B910                    37654 ...
SQLO_LT_SQLB_PTBL__pool_table_latch        0x700000004108B910                    37140 ...
SQLO_LT_SQLE_KRCB__EDUChainLatch           0x780000000472EAC                     37397 ...
SQLO_LT_preventSuspendIOLotch              0x7800000046081C0                     36114 ...
SQLO_LT_sqeWLDispatcher__m_tunerLatch      0x780000001C68440                      1029 ...
```

Output for query (continued):

```
... EDU_NAME             APPLICATION_HANDLE    MEMBER LATCH_STATUS LATCH_WAIT_TIME
... -------------------- --------------------- ------ ------------ --------------------
... db2agent (SAMPLE)                      118      0 H                               -
... db2agent (SAMPLE)                      118      0 H                               -
... db2agent (SAMPLE)                      124      0 W                           65850
... db2agent (SAMPLE)                      124      0 H                               -
... db2agent (SAMPLE)                      121      0 W                           50397
... db2agent (SAMPLE)                      119      0 W                           30886
... db2agent (SAMPLE)                      120      0 H                               -
... db2agent (SAMPLE)                      118      0 H                               -
... db2wlmt                                  -      0 H                               -

       9 record(s) selected.
```

DB2 LUW



2. Call the MON_GET_LATCH table function to determine the latches that are contested.

```
SELECT SUBSTR(LATCH_NAME,1,40) LATCH_NAME,
       SUBSTR(MEMORY_ADDRESS,1,20) ADDRESS,
       EDU_ID,
       SUBSTR(EDU_NAME,1,20) EDU_NAME,
       APPLICATION_HANDLE,
       MEMBER,
       LATCH_STATUS,
       LATCH_WAIT_TIME
   FROM TABLE ( MON_GET_LATCH( CLOB('<LATCH_STATUS>C</LATCH_STATUS>'), -2 ) )
   ORDER BY LATCH_NAME, LATCH_STATUS
```

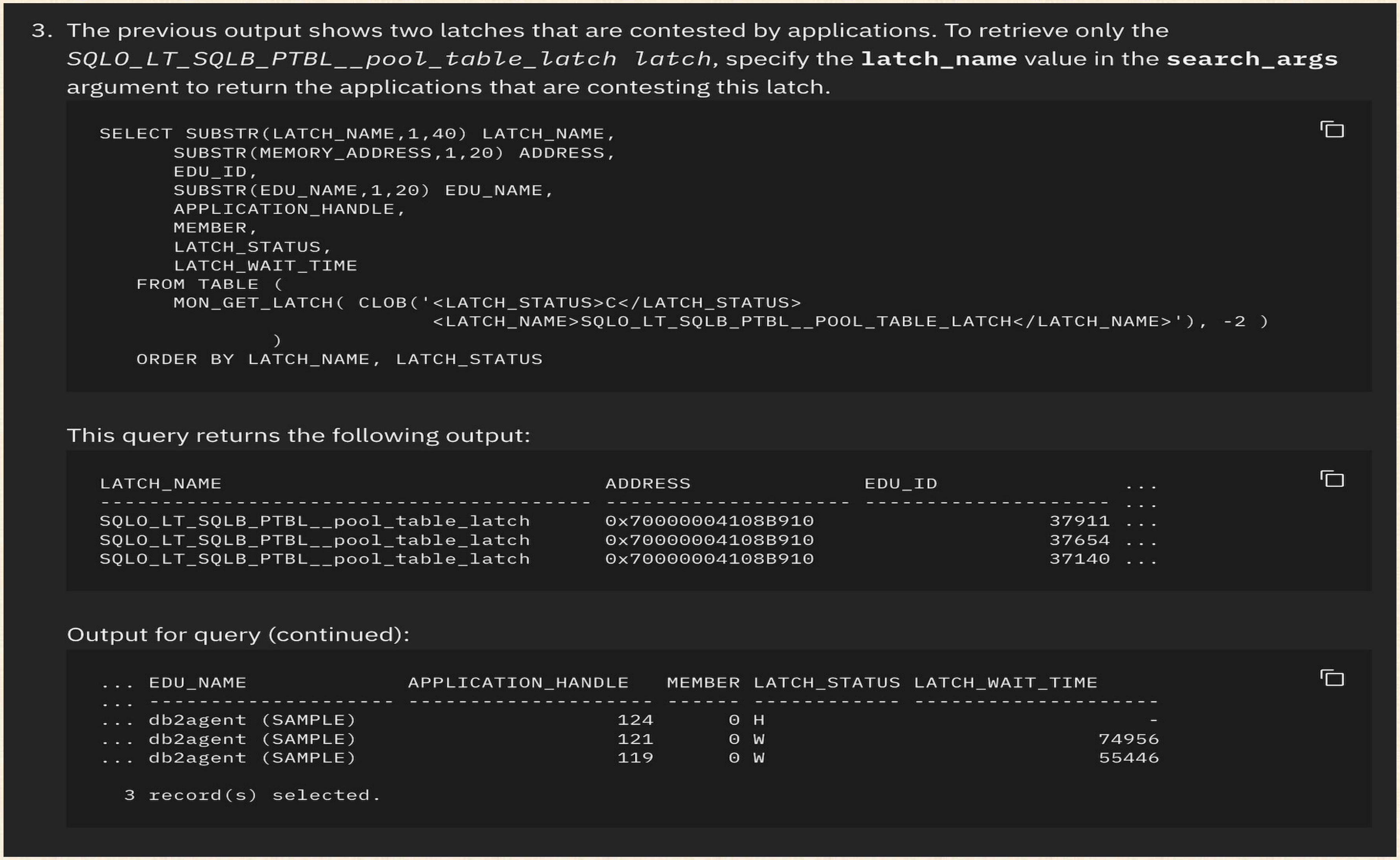
This query returns the following output:

```
LATCH_NAME                               ADDRESS              EDU_ID               ...
---------------------------------------- -------------------- -------------------- ...
SQLO_LT_SQLB_POOL_CB__readLotch          0x70000005351A3C0                   36114 ...
SQLO_LT_SQLB_POOL_CB__readLotch          0x70000005351A3C0                   37911 ...
SQLO_LT_SQLB_PTBL__pool_table_latch      0x70000004108B910                   37911 ...
SQLO_LT_SQLB_PTBL__pool_table_latch      0x70000004108B910                   37654 ...
SQLO_LT_SQLB_PTBL__pool_table_latch      0x70000004108B910                   37140 ...
```

Output for query (continued):

```
... EDU_NAME             APPLICATION_HANDLE   MEMBER LATCH_STATUS LATCH_WAIT_TIME
... -------------------- -------------------- ------ ------------ --------------------
... db2agent (SAMPLE)                     118      0 H                               -
... db2agent (SAMPLE)                     124      0 W                           78140
... db2agent (SAMPLE)                     124      0 H                               -
... db2agent (SAMPLE)                     121      0 W                           62686
... db2agent (SAMPLE)                     119      0 W                           43175


  5 record(s) selected.
```

DB2 LUW



3. The previous output shows two latches that are contested by applications. To retrieve only the *SQLO_LT_SQLB_PTBL__pool_table_latch latch*, specify the **latch_name** value in the **search_args** argument to return the applications that are contesting this latch.

```
SELECT SUBSTR(LATCH_NAME,1,40) LATCH_NAME,
      SUBSTR(MEMORY_ADDRESS,1,20) ADDRESS,
      EDU_ID,
      SUBSTR(EDU_NAME,1,20) EDU_NAME,
      APPLICATION_HANDLE,
      MEMBER,
      LATCH_STATUS,
      LATCH_WAIT_TIME
   FROM TABLE (
      MON_GET_LATCH( CLOB('<LATCH_STATUS>C</LATCH_STATUS>
                         <LATCH_NAME>SQLO_LT_SQLB_PTBL__POOL_TABLE_LATCH</LATCH_NAME>'), -2 )
              )
   ORDER BY LATCH_NAME, LATCH_STATUS
```

This query returns the following output:

```
LATCH_NAME                               ADDRESS              EDU_ID               ...
---------------------------------------- -------------------- -------------------- ...
SQLO_LT_SQLB_PTBL__pool_table_latch      0x70000004108B910                   37911 ...
SQLO_LT_SQLB_PTBL__pool_table_latch      0x70000004108B910                   37654 ...
SQLO_LT_SQLB_PTBL__pool_table_latch      0x70000004108B910                   37140 ...
```

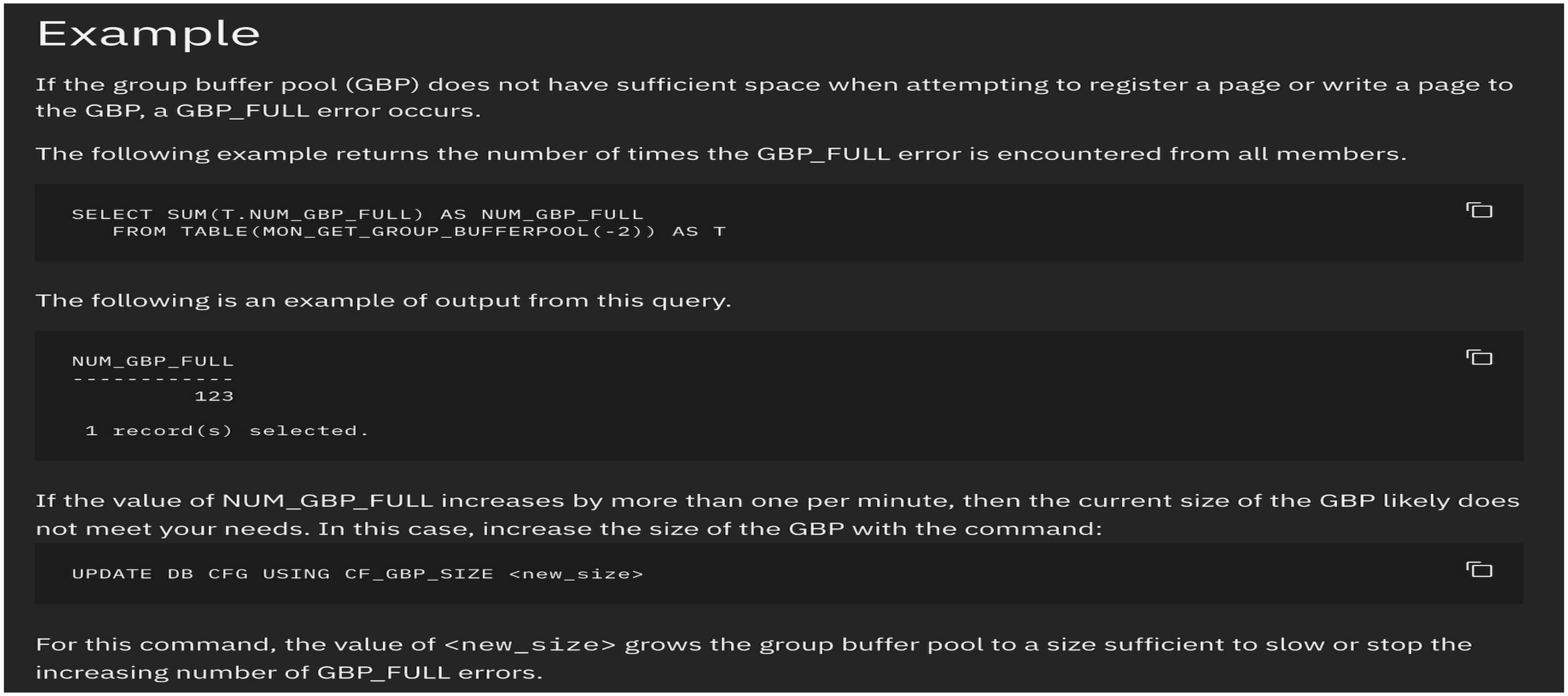
Output for query (continued):

```
... EDU_NAME             APPLICATION_HANDLE   MEMBER LATCH_STATUS LATCH_WAIT_TIME
... -------------------- -------------------- ------ ------------ --------------------
... db2agent (SAMPLE)                     124      0 H                               -
... db2agent (SAMPLE)                     121      0 W                           74956
... db2agent (SAMPLE)                     119      0 W                           55446

  3 record(s) selected.
```

DB2 LUW

MON_GET_GROUP_BUFFERPOOL

The MON_GET_GROUP_BUFFERPOOL table function returns statistics about the group buffer pool (GBP)



Example

If the group buffer pool (GBP) does not have sufficient space when attempting to register a page or write a page to the GBP, a GBP_FULL error occurs.

The following example returns the number of times the GBP_FULL error is encountered from all members.

```
SELECT SUM(T.NUM_GBP_FULL) AS NUM_GBP_FULL
   FROM TABLE(MON_GET_GROUP_BUFFERPOOL(-2)) AS T
```

The following is an example of output from this query.

```
NUM_GBP_FULL
------------
         123

 1 record(s) selected.
```

If the value of NUM_GBP_FULL increases by more than one per minute, then the current size of the GBP likely does not meet your needs. In this case, increase the size of the GBP with the command:

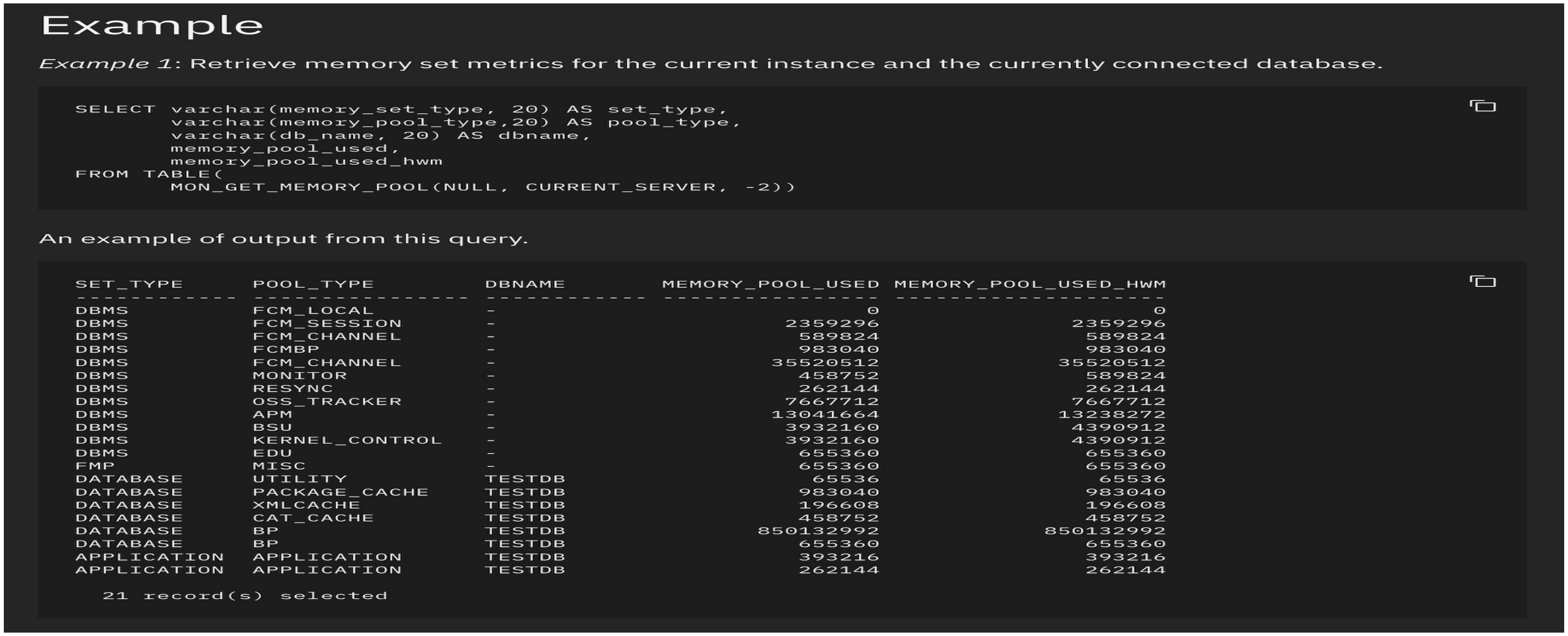
```
UPDATE DB CFG USING CF_GBP_SIZE <new_size>
```

For this command, the value of <new_size> grows the group buffer pool to a size sufficient to slow or stop the increasing number of GBP_FULL errors.

DB2 LUW

MON_GET_MEMORY_POOL :

The MON_GET_MEMORY_POOL table function retrieves metrics from the memory pools contained within a memory set.



Example

Example 1: Retrieve memory set metrics for the current instance and the currently connected database.

```
SELECT varchar(memory_set_type, 20) AS set_type,
       varchar(memory_pool_type,20) AS pool_type,
       varchar(db_name, 20) AS dbname,
       memory_pool_used,
       memory_pool_used_hwm
FROM TABLE(
       MON_GET_MEMORY_POOL(NULL, CURRENT_SERVER, -2))
```

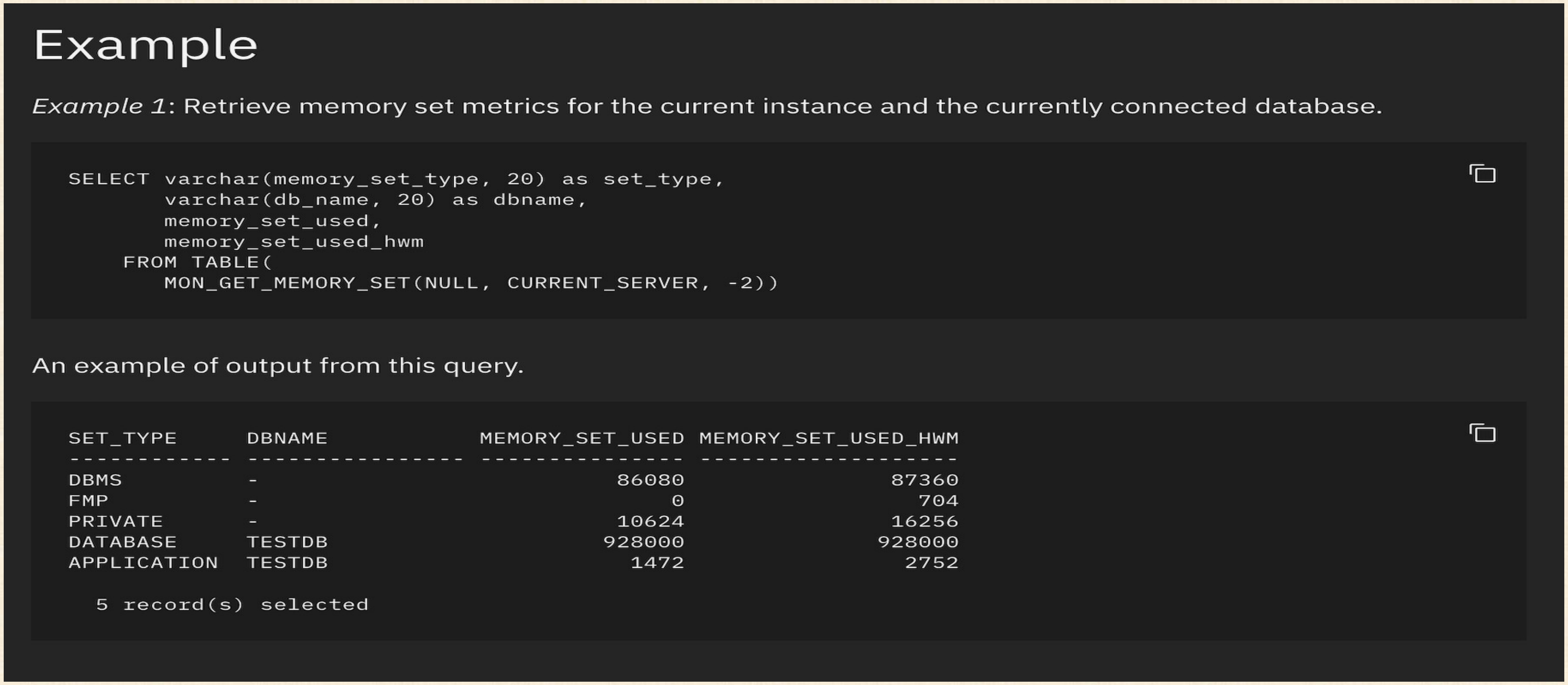
An example of output from this query.

```
SET_TYPE     POOL_TYPE        DBNAME       MEMORY_POOL_USED MEMORY_POOL_USED_HWM
------------ ---------------- ------------ ---------------- --------------------
DBMS         FCM_LOCAL        -                           0                    0
DBMS         FCM_SESSION      -                     2359296              2359296
DBMS         FCM_CHANNEL      -                      589824               589824
DBMS         FCMBP            -                      983040               983040
DBMS         FCM_CHANNEL      -                    35520512             35520512
DBMS         MONITOR          -                      458752               589824
DBMS         RESYNC           -                      262144               262144
DBMS         OSS_TRACKER      -                     7667712              7667712
DBMS         APM              -                    13041664             13238272
DBMS         BSU              -                     3932160              4390912
DBMS         KERNEL_CONTROL   -                     3932160              4390912
DBMS         EDU              -                      655360               655360
FMP          MISC             -                      655360               655360
DATABASE     UTILITY          TESTDB                  65536                65536
DATABASE     PACKAGE_CACHE    TESTDB                 983040               983040
DATABASE     XMLCACHE         TESTDB                 196608               196608
DATABASE     CAT_CACHE        TESTDB                 458752               458752
DATABASE     BP               TESTDB              850132992            850132992
DATABASE     BP               TESTDB                 655360               655360
APPLICATION  APPLICATION      TESTDB                 393216               393216
APPLICATION  APPLICATION      TESTDB                 262144               262144

  21 record(s) selected
```

DB2 LUW

MON_GET_MEMORY_SET :

The MON_GET_MEMORY_SET table function retrieves metrics from the allocated memory sets, both at the instance level, and for all active databases within the instance.



Example

Example 1: Retrieve memory set metrics for the current instance and the currently connected database.

```
SELECT varchar(memory_set_type, 20) as set_type,
       varchar(db_name, 20) as dbname,
       memory_set_used,
       memory_set_used_hwm
    FROM TABLE(
       MON_GET_MEMORY_SET(NULL, CURRENT_SERVER, -2))
```

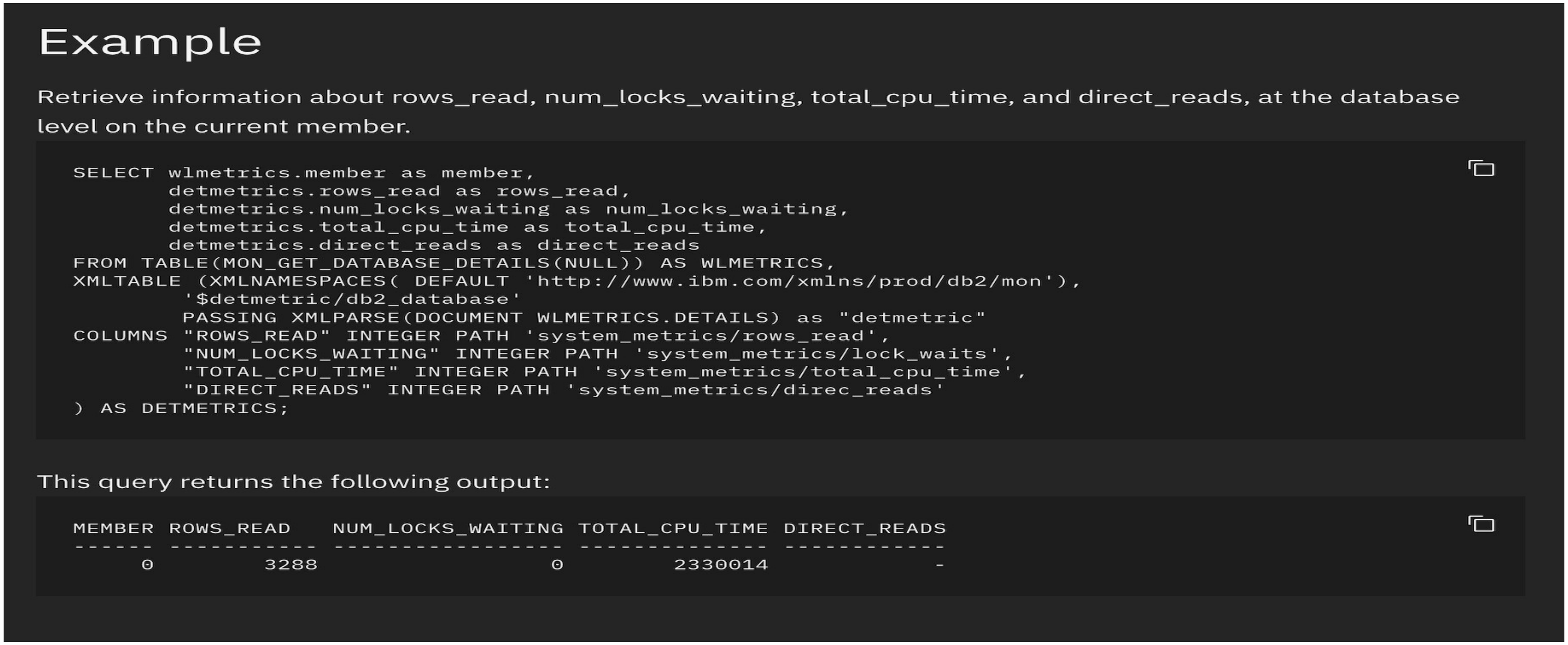
An example of output from this query.

```
SET_TYPE     DBNAME           MEMORY_SET_USED MEMORY_SET_USED_HWM
------------ ---------------- --------------- -------------------
DBMS         -                          86080               87360
FMP          -                              0                 704
PRIVATE      -                          10624               16256
DATABASE     TESTDB                    928000              928000
APPLICATION  TESTDB                      1472                2752

  5 record(s) selected
```

DB2 LUW

MON_GET_DATABASE_DETAILS

The MON_GET_DATABASE_DETAILS table function retrieves database metrics and returns the information in an XML document.



Example

Retrieve information about rows_read, num_locks_waiting, total_cpu_time, and direct_reads, at the database level on the current member.

```
SELECT wlmetrics.member as member,
       detmetrics.rows_read as rows_read,
       detmetrics.num_locks_waiting as num_locks_waiting,
       detmetrics.total_cpu_time as total_cpu_time,
       detmetrics.direct_reads as direct_reads
FROM TABLE(MON_GET_DATABASE_DETAILS(NULL)) AS WLMETRICS,
XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
        '$detmetric/db2_database'
        PASSING XMLPARSE(DOCUMENT WLMETRICS.DETAILS) as "detmetric"
COLUMNS "ROWS_READ" INTEGER PATH 'system_metrics/rows_read',
        "NUM_LOCKS_WAITING" INTEGER PATH 'system_metrics/lock_waits',
        "TOTAL_CPU_TIME" INTEGER PATH 'system_metrics/total_cpu_time',
        "DIRECT_READS" INTEGER PATH 'system_metrics/direc_reads'
) AS DETMETRICS;
```

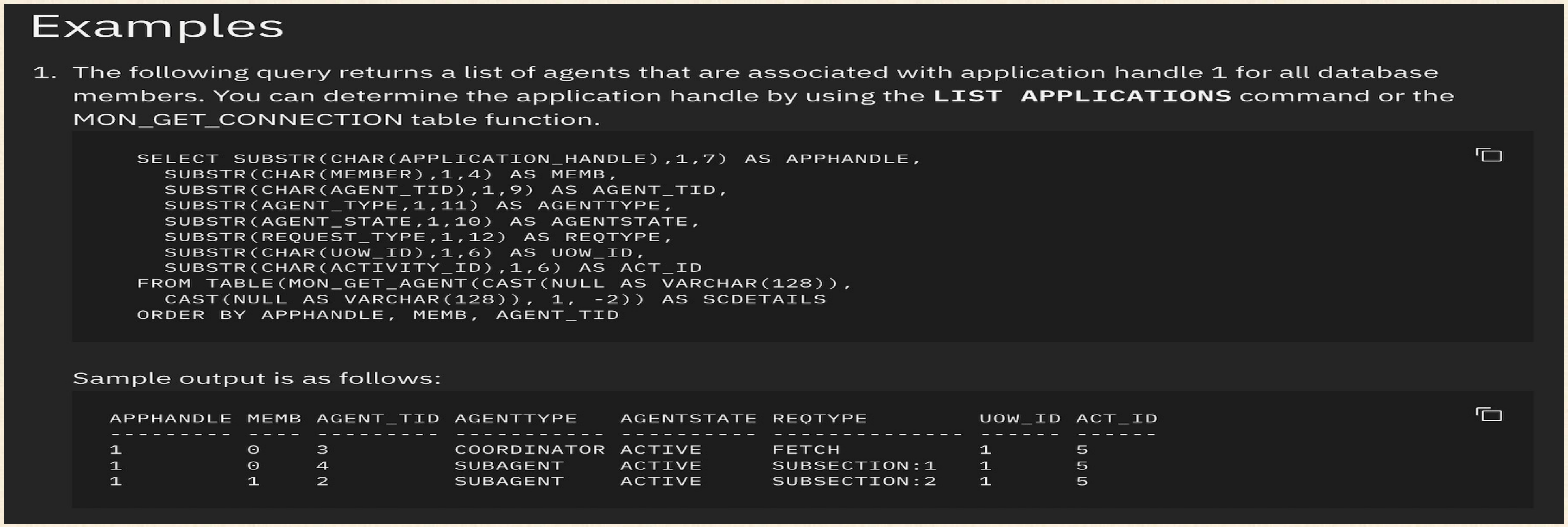
This query returns the following output:

```
MEMBER ROWS_READ   NUM_LOCKS_WAITING TOTAL_CPU_TIME DIRECT_READS
------ ----------- ----------------- -------------- ------------
     0        3288                 0        2330014            -
```

DB2 LUW

MON_GET_AGENT :

The MON_GET_AGENT function returns a list of all agents, fenced mode processes (db2fmp processes), and system entities for the database. The list can be filtered to show information for a specified member, service class, or application.



Examples

1. The following query returns a list of agents that are associated with application handle 1 for all database members. You can determine the application handle by using the **LIST APPLICATIONS** command or the MON_GET_CONNECTION table function.

```
SELECT SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APPHANDLE,
  SUBSTR(CHAR(MEMBER),1,4) AS MEMB,
  SUBSTR(CHAR(AGENT_TID),1,9) AS AGENT_TID,
  SUBSTR(AGENT_TYPE,1,11) AS AGENTTYPE,
  SUBSTR(AGENT_STATE,1,10) AS AGENTSTATE,
  SUBSTR(REQUEST_TYPE,1,12) AS REQTYPE,
  SUBSTR(CHAR(UOW_ID),1,6) AS UOW_ID,
  SUBSTR(CHAR(ACTIVITY_ID),1,6) AS ACT_ID
FROM TABLE(MON_GET_AGENT(CAST(NULL AS VARCHAR(128)),
  CAST(NULL AS VARCHAR(128)), 1, -2)) AS SCDETAILS
ORDER BY APPHANDLE, MEMB, AGENT_TID
```

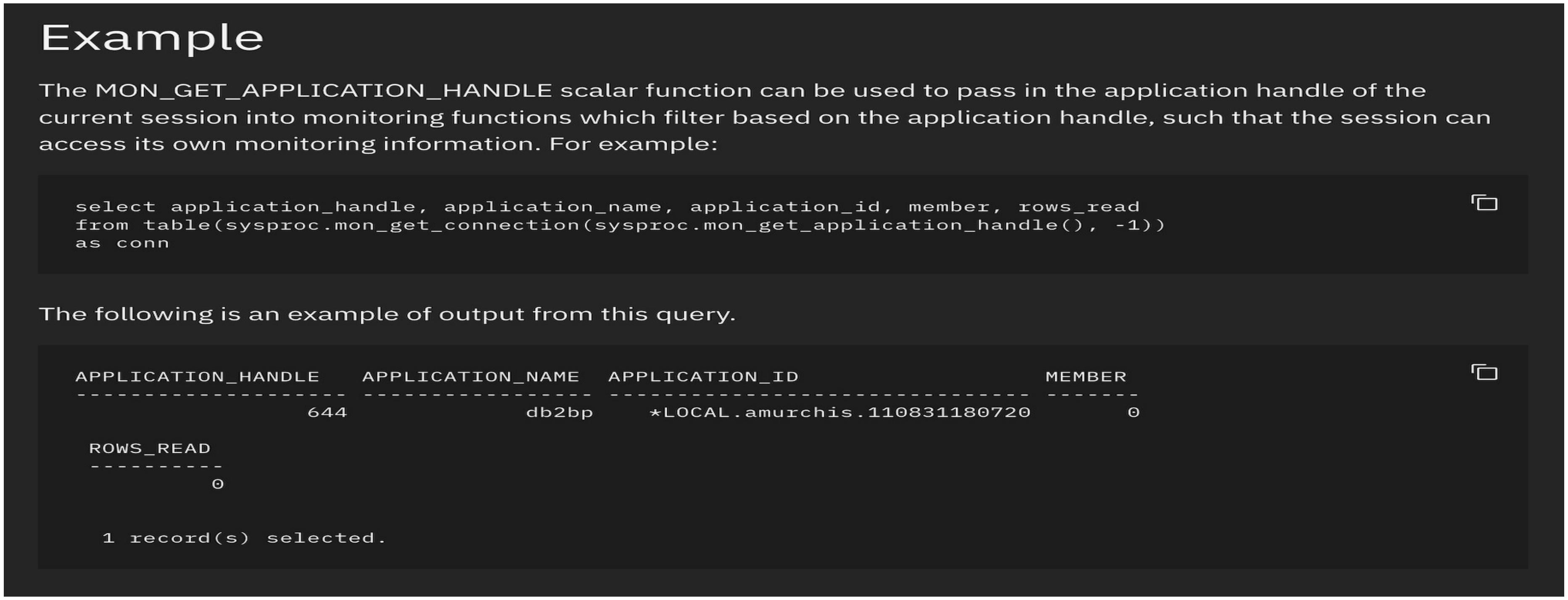
Sample output is as follows:

```
APPHANDLE MEMB AGENT_TID AGENTTYPE   AGENTSTATE REQTYPE        UOW_ID ACT_ID
--------- ---- --------- ----------- ---------- -------------- ------ ------
1         0    3         COORDINATOR ACTIVE     FETCH          1      5
1         0    4         SUBAGENT    ACTIVE     SUBSECTION:1   1      5
1         1    2         SUBAGENT    ACTIVE     SUBSECTION:2   1      5
```

DB2 LUW

MON_GET_APPLICATION_HANDLE

The MON_GET_APPLICATION_HANDLE scalar function returns the application handle of the invoking application.



Example

The MON_GET_APPLICATION_HANDLE scalar function can be used to pass in the application handle of the current session into monitoring functions which filter based on the application handle, such that the session can access its own monitoring information. For example:

```
select application_handle, application_name, application_id, member, rows_read
from table(sysproc.mon_get_connection(sysproc.mon_get_application_handle(), -1))
as conn
```

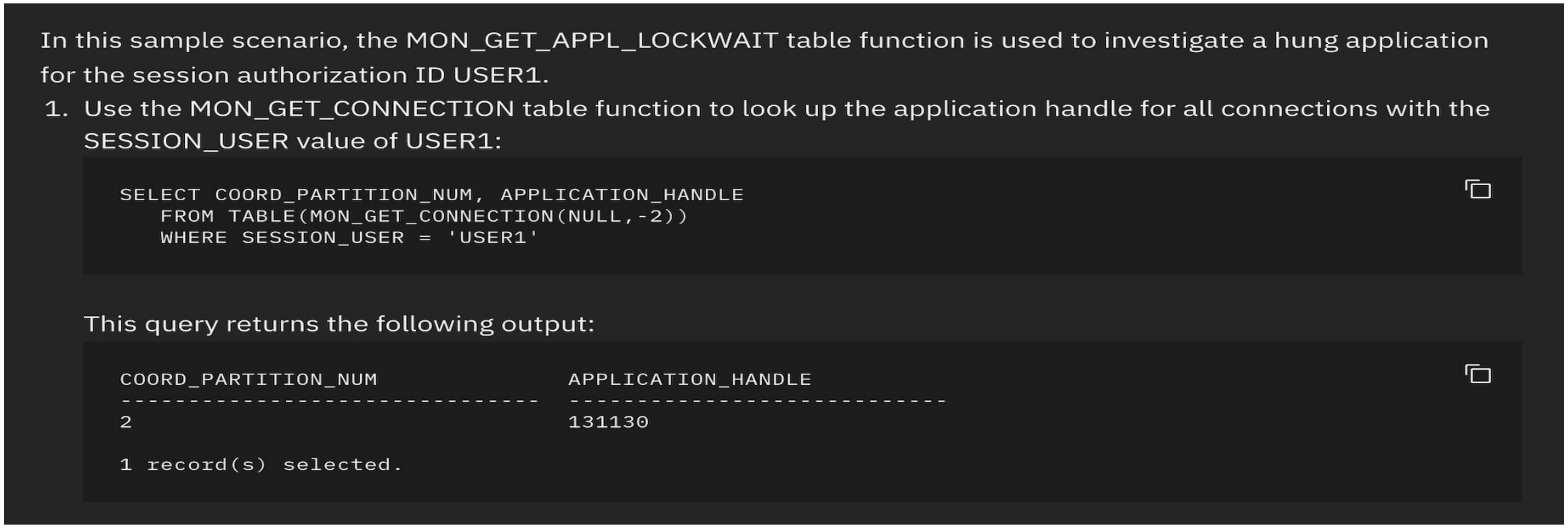
The following is an example of output from this query.

```
APPLICATION_HANDLE   APPLICATION_NAME  APPLICATION_ID                  MEMBER
-------------------- ---------------- ------------------------------- -------
                 644             db2bp    *LOCAL.amurchis.110831180720       0

 ROWS_READ
 ----------
          0


  1 record(s) selected.
```

DB2 LUW

MON_GET_APPL_LOCKWAIT :

The MON_GET_APPL_LOCKWAIT table function returns information about all locks that each application's agents (that are connected to the current database) are waiting to acquire.



In this sample scenario, the MON_GET_APPL_LOCKWAIT table function is used to investigate a hung application for the session authorization ID USER1.

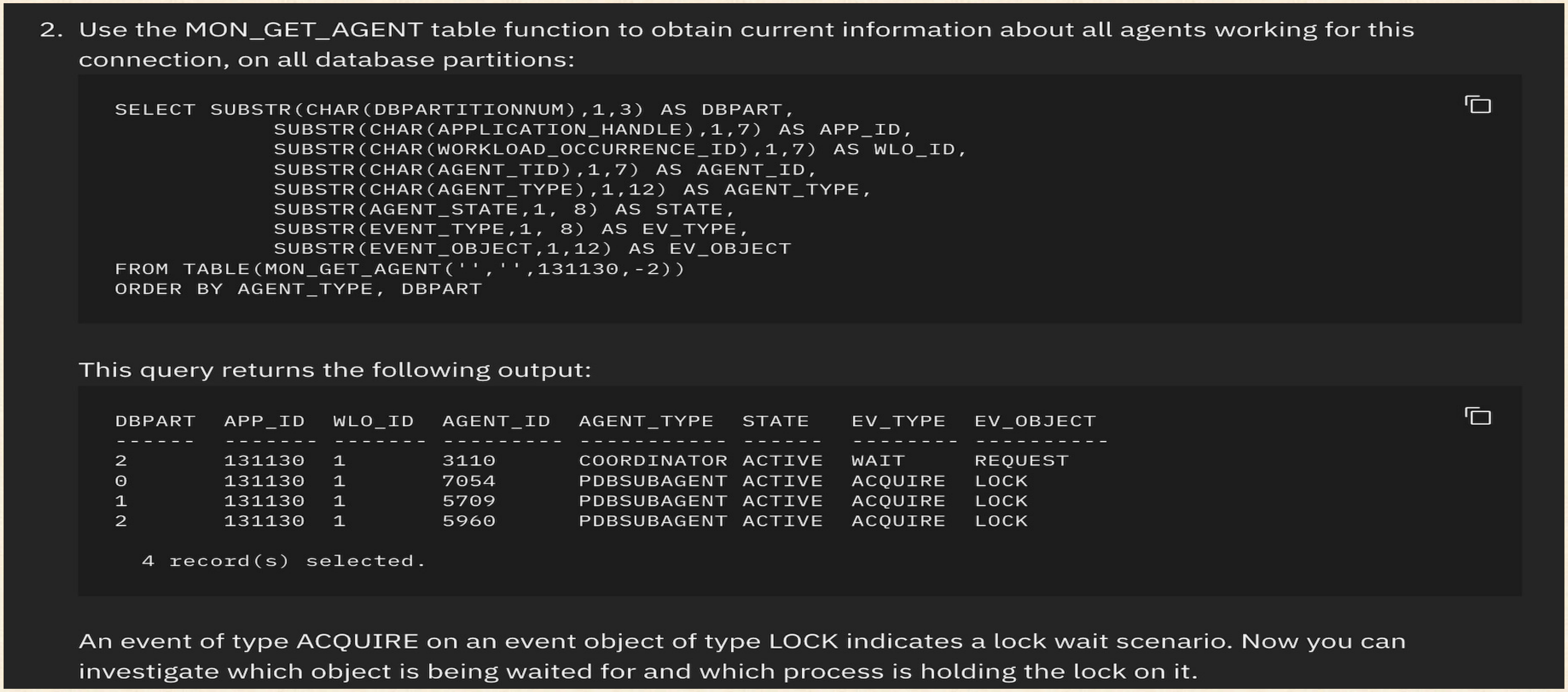
1. Use the MON_GET_CONNECTION table function to look up the application handle for all connections with the SESSION_USER value of USER1:

```
SELECT COORD_PARTITION_NUM, APPLICATION_HANDLE
   FROM TABLE(MON_GET_CONNECTION(NULL,-2))
   WHERE SESSION_USER = 'USER1'
```

   This query returns the following output:

```
COORD_PARTITION_NUM                APPLICATION_HANDLE
-------------------------------    ----------------------------
2                                  131130

1 record(s) selected.
```

DB2 LUW



2. Use the MON_GET_AGENT table function to obtain current information about all agents working for this connection, on all database partitions:

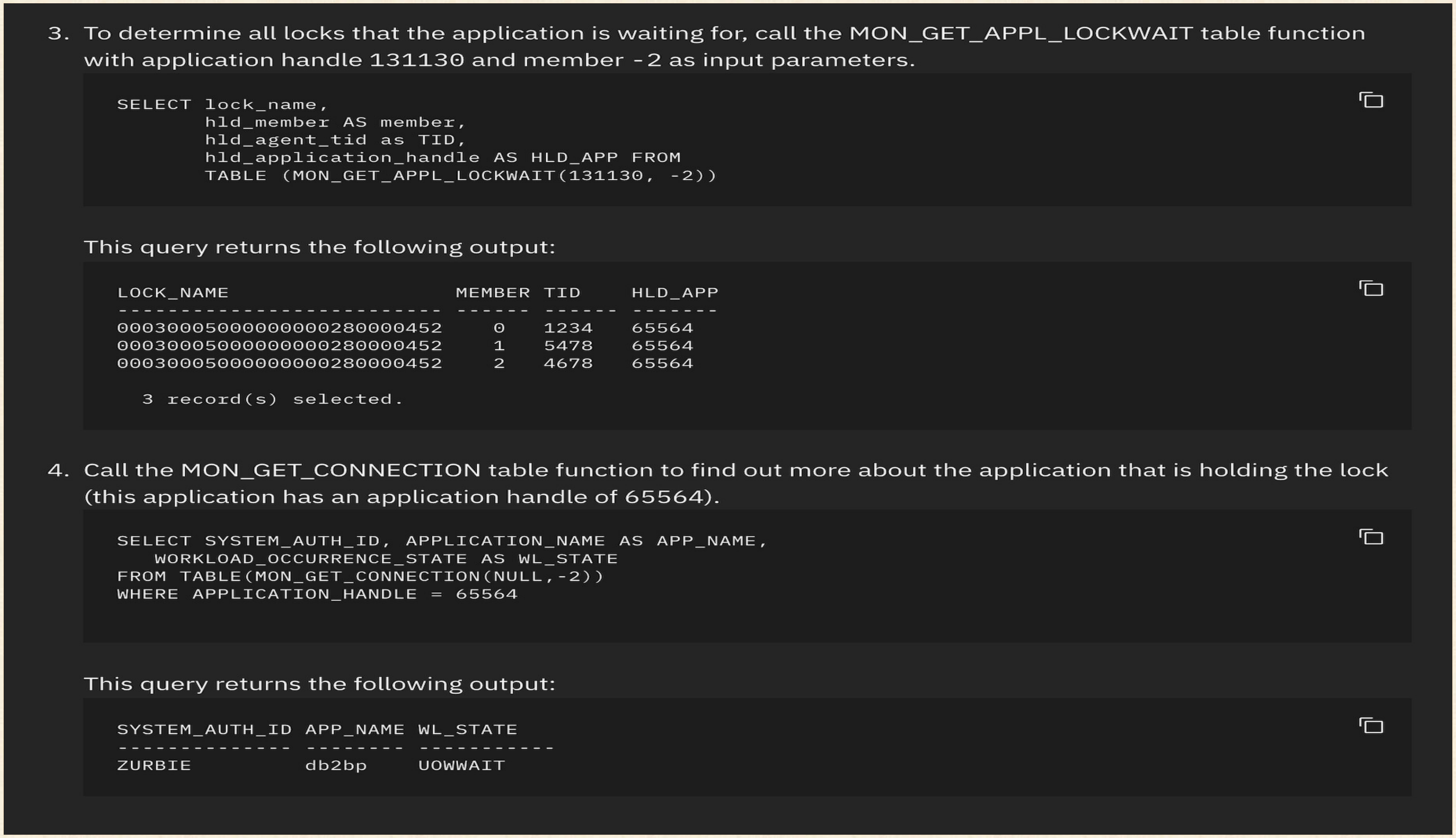
```
SELECT SUBSTR(CHAR(DBPARTITIONNUM),1,3) AS DBPART,
           SUBSTR(CHAR(APPLICATION_HANDLE),1,7) AS APP_ID,
           SUBSTR(CHAR(WORKLOAD_OCCURRENCE_ID),1,7) AS WLO_ID,
           SUBSTR(CHAR(AGENT_TID),1,7) AS AGENT_ID,
           SUBSTR(CHAR(AGENT_TYPE),1,12) AS AGENT_TYPE,
           SUBSTR(AGENT_STATE,1, 8) AS STATE,
           SUBSTR(EVENT_TYPE,1, 8) AS EV_TYPE,
           SUBSTR(EVENT_OBJECT,1,12) AS EV_OBJECT
FROM TABLE(MON_GET_AGENT('','',131130,-2))
ORDER BY AGENT_TYPE, DBPART
```

This query returns the following output:

```
DBPART  APP_ID  WLO_ID  AGENT_ID  AGENT_TYPE  STATE   EV_TYPE  EV_OBJECT
------  ------- ------- --------- ----------- ------  -------- ----------
2       131130  1       3110      COORDINATOR ACTIVE  WAIT     REQUEST
0       131130  1       7054      PDBSUBAGENT ACTIVE  ACQUIRE  LOCK
1       131130  1       5709      PDBSUBAGENT ACTIVE  ACQUIRE  LOCK
2       131130  1       5960      PDBSUBAGENT ACTIVE  ACQUIRE  LOCK

  4 record(s) selected.
```

An event of type ACQUIRE on an event object of type LOCK indicates a lock wait scenario. Now you can investigate which object is being waited for and which process is holding the lock on it.

DB2 LUW



3. To determine all locks that the application is waiting for, call the MON_GET_APPL_LOCKWAIT table function with application handle 131130 and member -2 as input parameters.

```
SELECT lock_name,
       hld_member AS member,
       hld_agent_tid as TID,
       hld_application_handle AS HLD_APP FROM
       TABLE (MON_GET_APPL_LOCKWAIT(131130, -2))
```

This query returns the following output:

```
LOCK_NAME                  MEMBER TID    HLD_APP
-------------------------- ------ ------ -------
00030005000000000280000452    0   1234   65564
00030005000000000280000452    1   5478   65564
00030005000000000280000452    2   4678   65564

  3 record(s) selected.
```

4. Call the MON_GET_CONNECTION table function to find out more about the application that is holding the lock (this application has an application handle of 65564).

```
SELECT SYSTEM_AUTH_ID, APPLICATION_NAME AS APP_NAME,
   WORKLOAD_OCCURRENCE_STATE AS WL_STATE
FROM TABLE(MON_GET_CONNECTION(NULL,-2))
WHERE APPLICATION_HANDLE = 65564
```

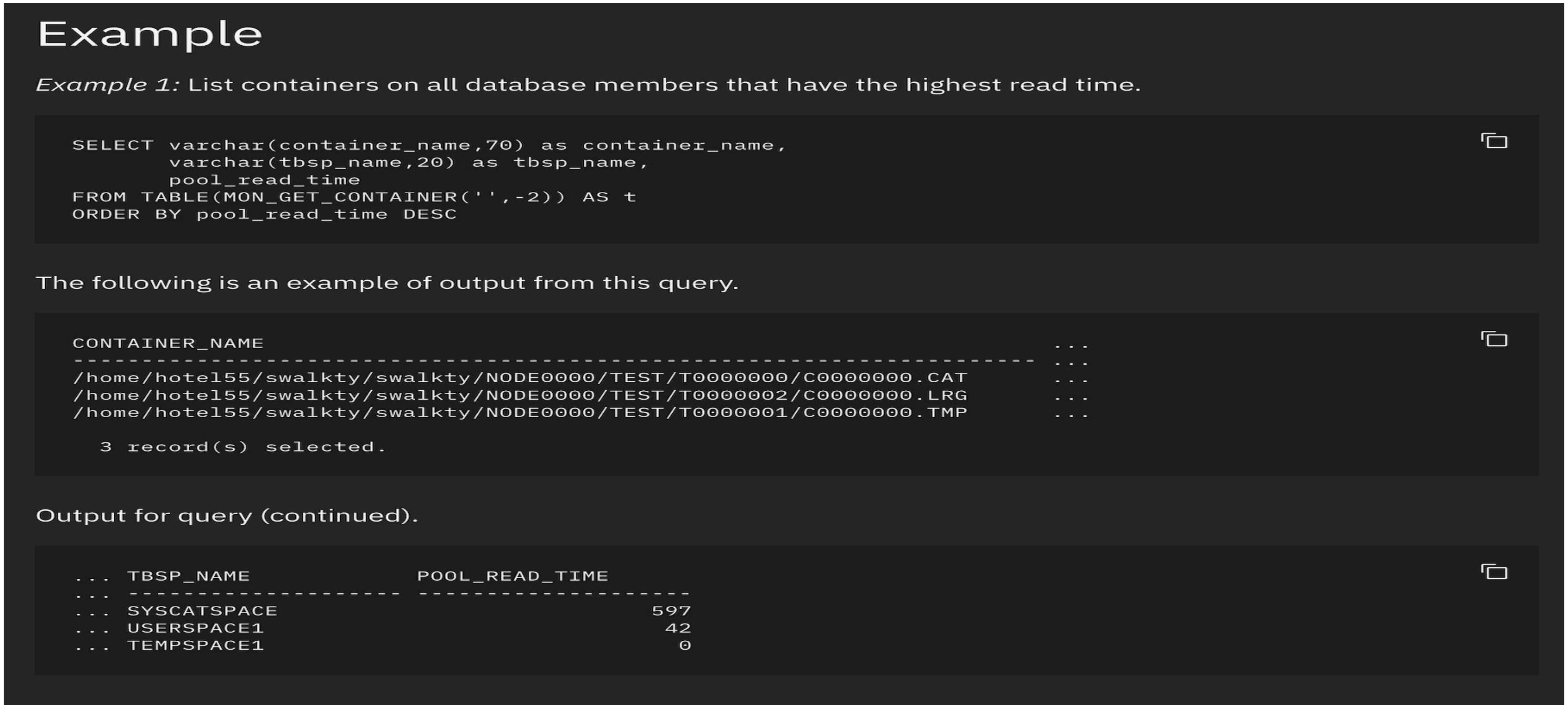
This query returns the following output:

```
SYSTEM_AUTH_ID APP_NAME WL_STATE
-------------- -------- -----------
ZURBIE         db2bp    UOWWAIT
```

DB2 LUW

MON_GET_CONTAINER

The MON_GET_CONTAINER table function returns monitor metrics for one or more table space containers.



Example

Example 1: List containers on all database members that have the highest read time.

```
SELECT varchar(container_name,70) as container_name,
       varchar(tbsp_name,20) as tbsp_name,
       pool_read_time
FROM TABLE(MON_GET_CONTAINER('',-2)) AS t
ORDER BY pool_read_time DESC
```

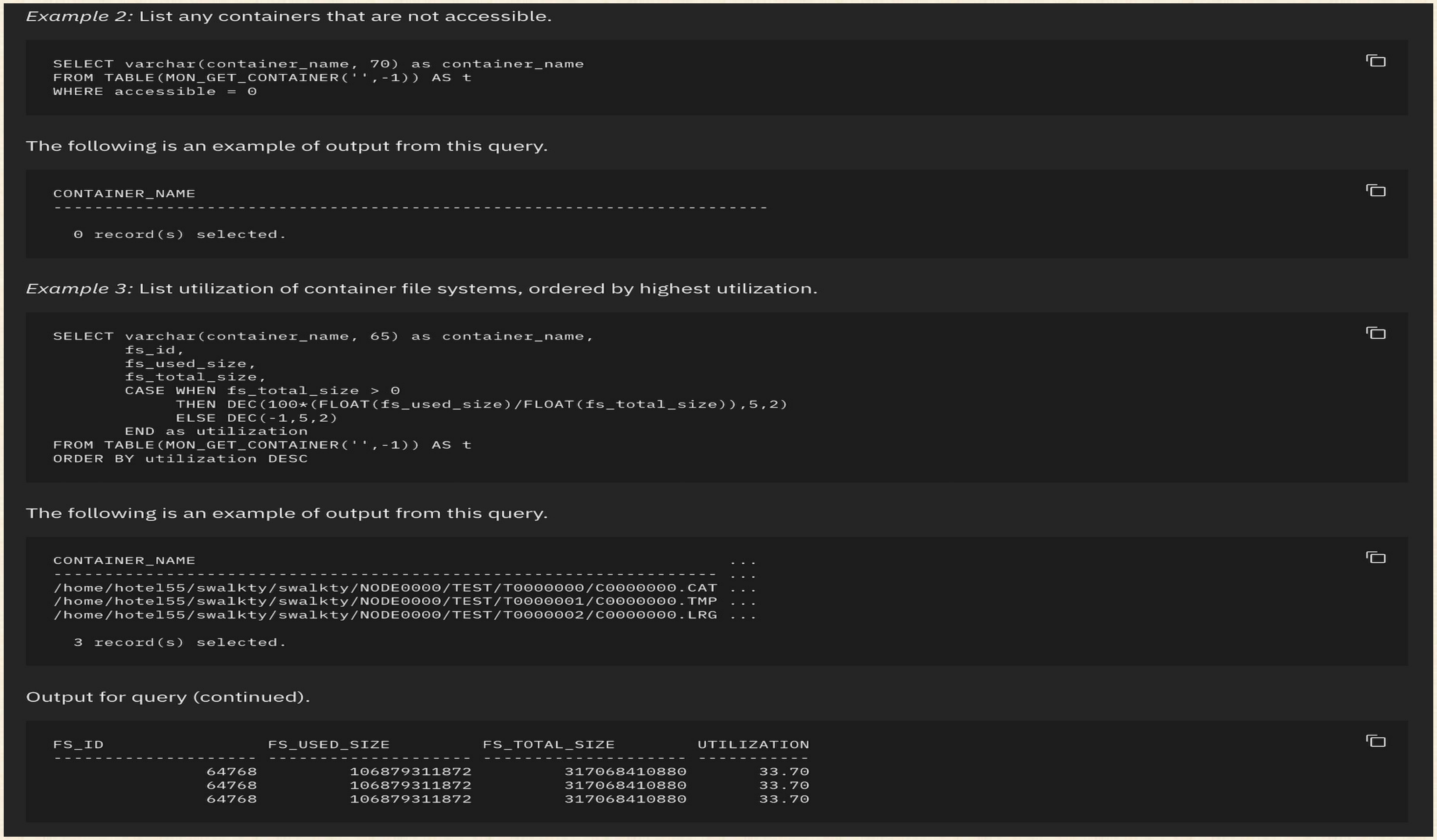
The following is an example of output from this query.

```
CONTAINER_NAME                                                          ...
----------------------------------------------------------------------  ...
/home/hotel55/swalkty/swalkty/NODE0000/TEST/T0000000/C0000000.CAT       ...
/home/hotel55/swalkty/swalkty/NODE0000/TEST/T0000002/C0000000.LRG       ...
/home/hotel55/swalkty/swalkty/NODE0000/TEST/T0000001/C0000000.TMP       ...

  3 record(s) selected.
```

Output for query (continued).

```
... TBSP_NAME            POOL_READ_TIME
... -------------------- --------------------
... SYSCATSPACE                           597
... USERSPACE1                             42
... TEMPSPACE1                              0
```

DB2 LUW



Example 2: List any containers that are not accessible.

```
SELECT varchar(container_name, 70) as container_name
FROM TABLE(MON_GET_CONTAINER('',-1)) AS t
WHERE accessible = 0
```

The following is an example of output from this query.

```
CONTAINER_NAME
----------------------------------------------------------------------

  0 record(s) selected.
```

Example 3: List utilization of container file systems, ordered by highest utilization.

```
SELECT varchar(container_name, 65) as container_name,
       fs_id,
       fs_used_size,
       fs_total_size,
       CASE WHEN fs_total_size > 0
            THEN DEC(100*(FLOAT(fs_used_size)/FLOAT(fs_total_size)),5,2)
            ELSE DEC(-1,5,2)
       END as utilization
FROM TABLE(MON_GET_CONTAINER('',-1)) AS t
ORDER BY utilization DESC
```

The following is an example of output from this query.

```
CONTAINER_NAME                                                    ...
----------------------------------------------------------------- ...
/home/hotel55/swalkty/swalkty/NODE0000/TEST/T0000000/C0000000.CAT ...
/home/hotel55/swalkty/swalkty/NODE0000/TEST/T0000001/C0000000.TMP ...
/home/hotel55/swalkty/swalkty/NODE0000/TEST/T0000002/C0000000.LRG ...

  3 record(s) selected.
```

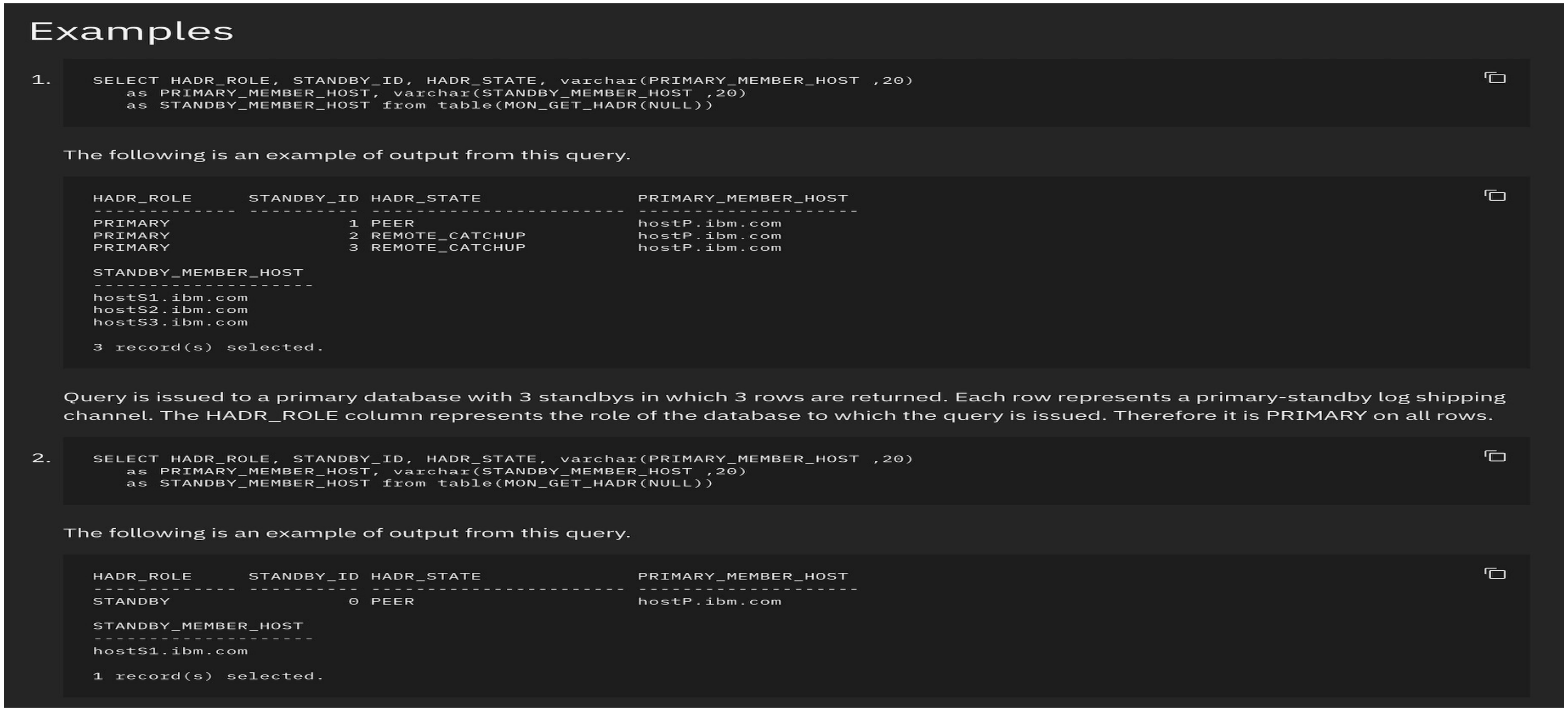
Output for query (continued).

```
FS_ID                FS_USED_SIZE         FS_TOTAL_SIZE        UTILIZATION
-------------------- -------------------- -------------------- -----------
               64768         106879311872         317068410880       33.70
               64768         106879311872         317068410880       33.70
               64768         106879311872         317068410880       33.70
```

DB2 LUW

MON_GET_HADR

This table function returns the metrics related to High availability Disaster Recovery information



Examples

1.
```
SELECT HADR_ROLE, STANDBY_ID, HADR_STATE, varchar(PRIMARY_MEMBER_HOST ,20)
   as PRIMARY_MEMBER_HOST, varchar(STANDBY_MEMBER_HOST ,20)
   as STANDBY_MEMBER_HOST from table(MON_GET_HADR(NULL))
```

The following is an example of output from this query.

```
HADR_ROLE     STANDBY_ID HADR_STATE              PRIMARY_MEMBER_HOST
------------- ---------- ----------------------- --------------------
PRIMARY                1 PEER                    hostP.ibm.com
PRIMARY                2 REMOTE_CATCHUP          hostP.ibm.com
PRIMARY                3 REMOTE_CATCHUP          hostP.ibm.com

STANDBY_MEMBER_HOST
--------------------
hostS1.ibm.com
hostS2.ibm.com
hostS3.ibm.com

3 record(s) selected.
```

Query is issued to a primary database with 3 standbys in which 3 rows are returned. Each row represents a primary-standby log shipping channel. The HADR_ROLE column represents the role of the database to which the query is issued. Therefore it is PRIMARY on all rows.

2.
```
SELECT HADR_ROLE, STANDBY_ID, HADR_STATE, varchar(PRIMARY_MEMBER_HOST ,20)
   as PRIMARY_MEMBER_HOST, varchar(STANDBY_MEMBER_HOST ,20)
   as STANDBY_MEMBER_HOST from table(MON_GET_HADR(NULL))
```

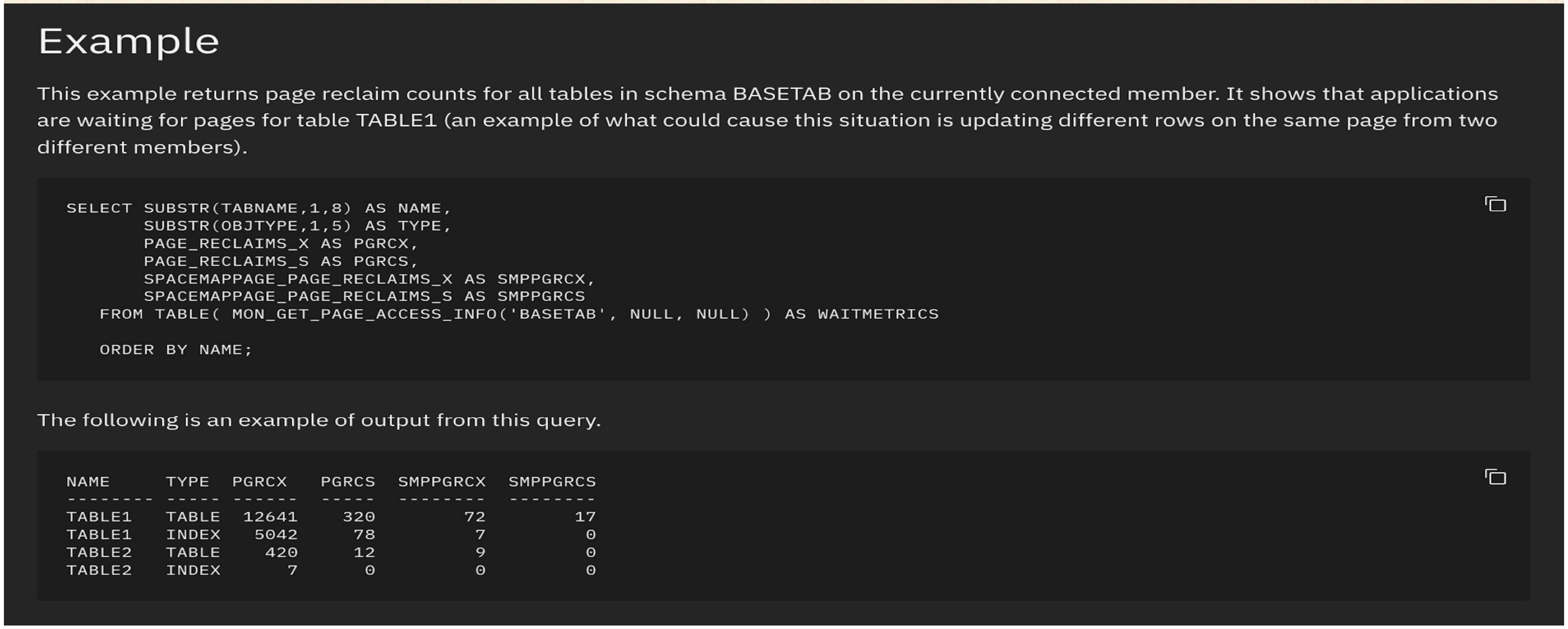
The following is an example of output from this query.

```
HADR_ROLE     STANDBY_ID HADR_STATE              PRIMARY_MEMBER_HOST
------------- ---------- ----------------------- --------------------
STANDBY                0 PEER                    hostP.ibm.com

STANDBY_MEMBER_HOST
--------------------
hostS1.ibm.com

1 record(s) selected.
```

DB2 LUW

MON_GET_PAGE_ACCESS_INFO

The MON_GET_PAGE_ACCESS_INFO table function returns information about bufferpool pages that are being waited on for a specified table. This is only applicable to Db2® pureScale® instances.



Example

This example returns page reclaim counts for all tables in schema BASETAB on the currently connected member. It shows that applications are waiting for pages for table TABLE1 (an example of what could cause this situation is updating different rows on the same page from two different members).

```
SELECT SUBSTR(TABNAME,1,8) AS NAME,
       SUBSTR(OBJTYPE,1,5) AS TYPE,
       PAGE_RECLAIMS_X AS PGRCX,
       PAGE_RECLAIMS_S AS PGRCS,
       SPACEMAPPAGE_PAGE_RECLAIMS_X AS SMPPGRCX,
       SPACEMAPPAGE_PAGE_RECLAIMS_S AS SMPPGRCS
   FROM TABLE( MON_GET_PAGE_ACCESS_INFO('BASETAB', NULL, NULL) ) AS WAITMETRICS

   ORDER BY NAME;
```

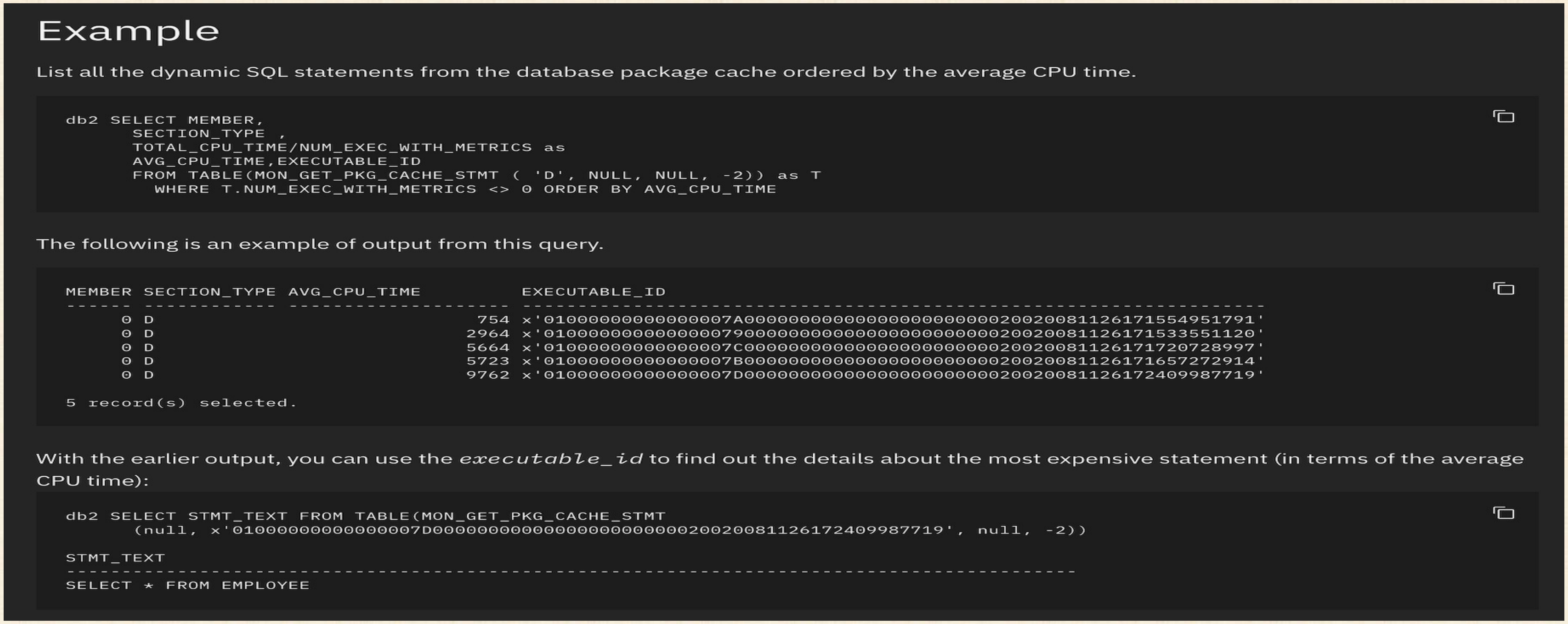
The following is an example of output from this query.

```
NAME     TYPE  PGRCX   PGRCS  SMPPGRCX  SMPPGRCS
-------- ----- ------  -----  --------  --------
TABLE1   TABLE  12641    320        72        17
TABLE1   INDEX   5042     78         7         0
TABLE2   TABLE    420     12         9         0
TABLE2   INDEX      7      0         0         0
```

DB2 LUW

MON_GET_PKG_CACHE_STMT :

The MON_GET_PKG_CACHE_STMT table function returns a point-in-time view of both static and dynamic SQL statements in the database package cache.



Example

List all the dynamic SQL statements from the database package cache ordered by the average CPU time.

```
db2 SELECT MEMBER,
      SECTION_TYPE ,
      TOTAL_CPU_TIME/NUM_EXEC_WITH_METRICS as
      AVG_CPU_TIME,EXECUTABLE_ID
      FROM TABLE(MON_GET_PKG_CACHE_STMT ( 'D', NULL, NULL, -2)) as T
        WHERE T.NUM_EXEC_WITH_METRICS <> 0 ORDER BY AVG_CPU_TIME
```

The following is an example of output from this query.

```
MEMBER SECTION_TYPE AVG_CPU_TIME         EXECUTABLE_ID
------ ------------ -------------------- -------------------------------------------------------------------
     0 D                             754 x'01000000000000007A0000000000000000000000020020081126171554951791'
     0 D                            2964 x'0100000000000000790000000000000000000000020020081126171533551120'
     0 D                            5664 x'01000000000000007C0000000000000000000000020020081126171720728997'
     0 D                            5723 x'01000000000000007B0000000000000000000000020020081126171657272914'
     0 D                            9762 x'01000000000000007D0000000000000000000000020020081126172409987719'

5 record(s) selected.
```

With the earlier output, you can use the *executable_id* to find out the details about the most expensive statement (in terms of the average CPU time):

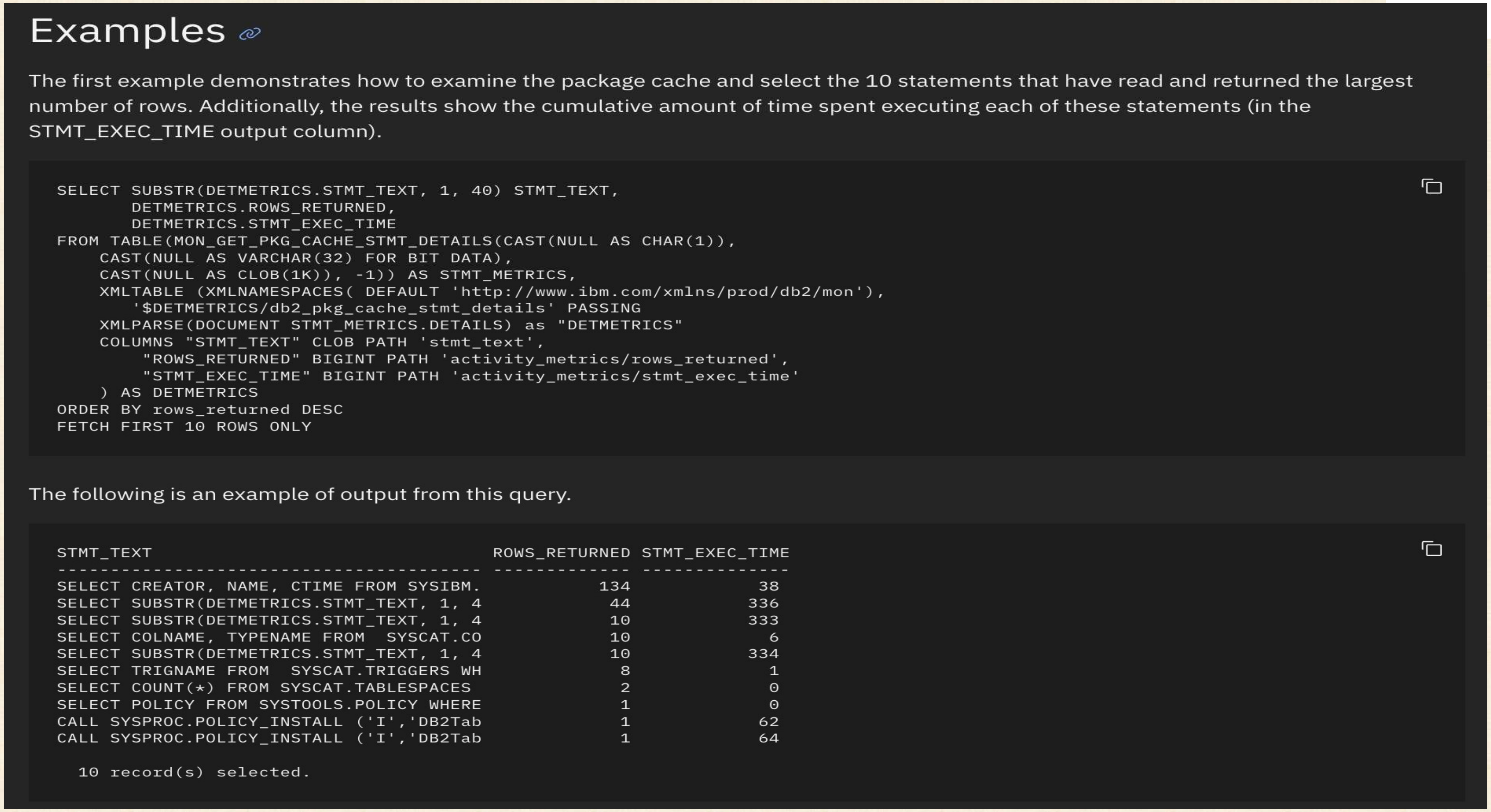
```
db2 SELECT STMT_TEXT FROM TABLE(MON_GET_PKG_CACHE_STMT
      (null, x'01000000000000007D0000000000000000000000020020081126172409987719', null, -2))

STMT_TEXT
-------------------------------------------------------------------------------------------
SELECT * FROM EMPLOYEE
```

MON_GET_PKG_CACHE_STMT_DETAILS :

The MON_GET_PKG_CACHE_STMT_DETAILS table function returns metrics for one or more package cache entries.

The metrics returned by the MON_GET_PKG_CACHE_STMT_DETAILS table function represent the accumulation of all metrics for statements in the package cache. Statement metrics are rolled up to the package cache upon activity completion.

DB2 LUW



Examples

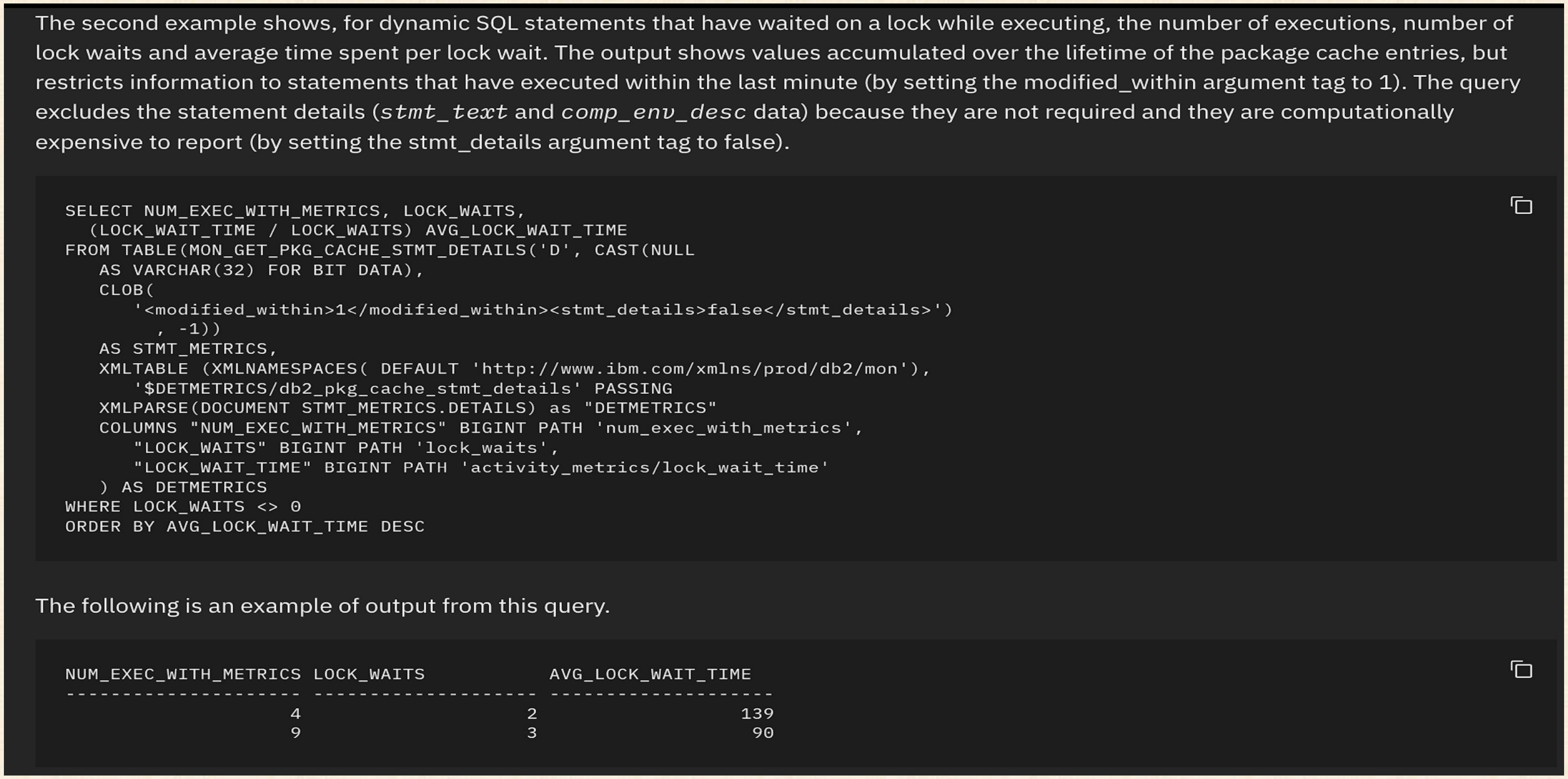
The first example demonstrates how to examine the package cache and select the 10 statements that have read and returned the largest number of rows. Additionally, the results show the cumulative amount of time spent executing each of these statements (in the STMT_EXEC_TIME output column).

```
SELECT SUBSTR(DETMETRICS.STMT_TEXT, 1, 40) STMT_TEXT,
       DETMETRICS.ROWS_RETURNED,
       DETMETRICS.STMT_EXEC_TIME
FROM TABLE(MON_GET_PKG_CACHE_STMT_DETAILS(CAST(NULL AS CHAR(1)),
    CAST(NULL AS VARCHAR(32) FOR BIT DATA),
    CAST(NULL AS CLOB(1K)), -1)) AS STMT_METRICS,
    XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
       '$DETMETRICS/db2_pkg_cache_stmt_details' PASSING
    XMLPARSE(DOCUMENT STMT_METRICS.DETAILS) as "DETMETRICS"
    COLUMNS "STMT_TEXT" CLOB PATH 'stmt_text',
        "ROWS_RETURNED" BIGINT PATH 'activity_metrics/rows_returned',
        "STMT_EXEC_TIME" BIGINT PATH 'activity_metrics/stmt_exec_time'
    ) AS DETMETRICS
ORDER BY rows_returned DESC
FETCH FIRST 10 ROWS ONLY
```

The following is an example of output from this query.

```
STMT_TEXT                                ROWS_RETURNED STMT_EXEC_TIME
---------------------------------------- ------------- --------------
SELECT CREATOR, NAME, CTIME FROM SYSIBM.           134             38
SELECT SUBSTR(DETMETRICS.STMT_TEXT, 1, 4            44            336
SELECT SUBSTR(DETMETRICS.STMT_TEXT, 1, 4            10            333
SELECT COLNAME, TYPENAME FROM  SYSCAT.CO            10              6
SELECT SUBSTR(DETMETRICS.STMT_TEXT, 1, 4            10            334
SELECT TRIGNAME FROM  SYSCAT.TRIGGERS WH             8              1
SELECT COUNT(*) FROM SYSCAT.TABLESPACES              2              0
SELECT POLICY FROM SYSTOOLS.POLICY WHERE             1              0
CALL SYSPROC.POLICY_INSTALL ('I','DB2Tab             1             62
CALL SYSPROC.POLICY_INSTALL ('I','DB2Tab             1             64

  10 record(s) selected.
```

DB2 LUW



The second example shows, for dynamic SQL statements that have waited on a lock while executing, the number of executions, number of lock waits and average time spent per lock wait. The output shows values accumulated over the lifetime of the package cache entries, but restricts information to statements that have executed within the last minute (by setting the modified_within argument tag to 1). The query excludes the statement details (*stmt_text* and *comp_env_desc* data) because they are not required and they are computationally expensive to report (by setting the stmt_details argument tag to false).

```
SELECT NUM_EXEC_WITH_METRICS, LOCK_WAITS,
  (LOCK_WAIT_TIME / LOCK_WAITS) AVG_LOCK_WAIT_TIME
FROM TABLE(MON_GET_PKG_CACHE_STMT_DETAILS('D', CAST(NULL
   AS VARCHAR(32) FOR BIT DATA),
   CLOB(
      '<modified_within>1</modified_within><stmt_details>false</stmt_details>')
        , -1))
   AS STMT_METRICS,
   XMLTABLE (XMLNAMESPACES( DEFAULT 'http://www.ibm.com/xmlns/prod/db2/mon'),
      '$DETMETRICS/db2_pkg_cache_stmt_details' PASSING
   XMLPARSE(DOCUMENT STMT_METRICS.DETAILS) as "DETMETRICS"
   COLUMNS "NUM_EXEC_WITH_METRICS" BIGINT PATH 'num_exec_with_metrics',
      "LOCK_WAITS" BIGINT PATH 'lock_waits',
      "LOCK_WAIT_TIME" BIGINT PATH 'activity_metrics/lock_wait_time'
   ) AS DETMETRICS
WHERE LOCK_WAITS <> 0
ORDER BY AVG_LOCK_WAIT_TIME DESC
```

The following is an example of output from this query.

```
NUM_EXEC_WITH_METRICS LOCK_WAITS           AVG_LOCK_WAIT_TIME
--------------------- -------------------- --------------------
                    4                    2                  139
                    9                    3                   90
```

IBM

DB2 LUW

THANK YOU