Effective use of SQL built-in global variables

Jing Li
Scott Hanson

December 20, 2016

This article explains SQL built-in global variables supported on IBM i and easy access to useful environmental information for users.

SQL built-in global variables are read-only, maintained by IBM® DB2® for i, and are a trusted and easily consumed resource. Some global variables exist for DB2 family compatibility and are contained within the `SYSIBM` schema. Other global variables provide IBM i specific value and are contained within the `QSYS2` schema. Global variables provide applications and users with easy access to useful environmental information that can be used for auditing and debugging. They allow improved application logging as well as more advanced exit point programs, trigger programs, and row and column access control (RCAC) rule texts.

Table 1 contains general information about SQL built-in global variables. The built-in global variables are not available in releases before IBM i 7.2.

Table 1. SQL built-in global variables general information

Variable name	Schema	Data type	Size	SF99702 Level	SF99703 Level
PROCESS_ID	QSYS2	INTEGER	-	Level 11	7.3 base
THREAD_ID	QSYS2	BIGINT	-	Level 11	7.3 base
JOB_NAME	QSYS2	VARCHAR	28	Level 3	7.3 base
SERVER_MODE_JOB_NAME	QSYS2	VARCHAR	28	Level 3	7.3 base
CLIENT_IPADDR	SYSIBM	VARCHAR	128	7.2 base	7.3 base
CLIENT_HOST	SYSIBM	VARCHAR	255	7.2 base	7.3 base
CLIENT_PORT	SYSIBM	INTEGER	-	7.2 base	7.3 base
ROUTINE_SCHEMA	SYSIBM	VARCHAR	128	7.2 base	7.3 base
ROUTINE_SPECIFIC_NAME	SYSIBM	VARCHAR	128	7.2 base	7.3 base
ROUTINE_TYPE	SYSIBM	CHAR	1	7.2 base	7.3 base
PACKAGE_NAME	SYSIBM	VARCHAR	128	7.2 base	7.3 base
PACKAGE_SCHEMA	SYSIBM	VARCHAR	128	7.2 base	7.3 base
PACKAGE_VERSION	SYSIBM	VARCHAR	64	7.2 base	7.3 base

You can find the built-in global variables in the QSYS2.SYSVARIABLES catalog. A query to show all available SQL built-in global variables is:

```
SELECT VARIABLE_SCHEMA, VARIABLE_NAME
    FROM QSYS2.SYSVARIABLES
    WHERE VARIABLE_SCHEMA = 'QSYS2' OR
          VARIABLE_SCHEMA = 'SYSIBM';
```

Before the introduction of these built-in global variables, the only way to obtain this information within SQL would have been to create an external procedure or function to obtain the information from a system API and return that information. Having a built-in global variable eliminates this effort and provides additional efficiency.

QSYS2.THREAD_ID

The `QSYS2.THREAD_ID` built-in global variable contains the thread identifier (ID) of the current thread. The data is `BIGINT`, but if you prefer to see the value in the hexadecimal form, cast the global variable using the `HEX()` built-in function.

The thread ID is used to uniquely identify a thread within a job. While no two threads initiated within the same job will have the same thread ID, it is possible that threads in different jobs may have the same value for the thread ID. When a thread ends, its thread ID is never reused within the job. Thread ID information is becoming more important as thread information surfaces in more ways. This includes functions running in different threads from the query that is using them.

Here is an example to capture the thread scoped record locks over the `SALES` table which are held by the current thread:

```
SELECT * FROM QSYS2.RECORD_LOCK_INFO
  WHERE TABLE_NAME = 'SALES' AND
  TABLE_SCHEMA = 'COMPANY'   AND
  JOB_NAME = QSYS2.JOB_NAME  AND
  THREAD_ID = QSYS2.THREAD_ID;
```

QSYS2.PROCESS_ID

The `QSYS2.PROCESS_ID` built-in global variable contains the process (ID) of the currently running job. The data type is `INTEGER`. The process ID is used to uniquely identify an active job on the system. No two active jobs will have the same process ID.

Here is an example to capture the details for the current job:

```
SELECT USER, CURRENT SERVER, QSYS2.JOB_NAME, QSYS2.THREAD_ID, QSYS2.PROCESS_ID
  FROM SYSIBM.SYSDUMMY1;
```

QSYS2.JOB_NAME

The `QSYS2.JOB_NAME` built-in global variable contains the name of the current job. It is of type, `VARCHAR(28)`.

The job name is used to uniquely identify an active job. No two active jobs will have the same job name.

Here is an example using a view to *hide* the bulky usage of table functions:

```
CREATE OR REPLACE VIEW QGPL.MYJOBINFO AS
(SELECT QSYS2.JOB_NAME, A.* FROM TABLE(QSYS2.GET_JOB_INFO(QSYS2.JOB_NAME)) A);

SELECT * FROM QGPL.MYJOBINFO;
```

You can obtain the job information regardless of how the connection is made. This can ease the frustration of figuring out jobs when using interfaces [such as command-line interface (CLI), Java Database Connectivity (JDBC), or Distributed Relational Database Architecture (DRDA)] where finding the job can be a challenge. You can also use three-part naming to determine the job that is being used on the remote system:

```
SELECT * FROM REMOTESYS.QGPL.MYJOBINFO;
```

Table 2. Places where thread ID, process ID, and job information is referenced

Places where information is referenced	THREAD_ID	PROCESS_ID	JOB_NAME
OBJECT_LOCK_INFO view	✓		✓
RECORD_LOCK_INFO view	✓		✓
AUTHORITY_COLLECTION view	✓		✓
JVM_INFO view		✓	
SYSLIMITS view			✓
OUTPUT_QUEUE_ENTRIES view			✓
OUTPUT_QUEUE_ENTRIES table function			✓
ACTIVE_JOB_INFO table function			✓
Database Monitor (DBMON)	✓		✓
Journal entries	✓		
System APIs	pthread_equal, pthread_getunique_np, pthread_kill, pthread_setname_np, pthread_getname_np, pthread_join_np	getpid, getppid, QlgSpawn, Qp0wChkChld, Qp0wChkPid, Qp0wGetJobID, Qp0wGetPid, Qp0wGetPPid, spawn, wait, waitpid	
System commands	WRKJOB, STRAUTCOL, STRWCH		
IBM PASE for i commands		kill, ps, wait	

QSYS2.SERVER_MODE_JOB_NAME

The `QSYS2.SERVER_MODE_JOB_NAME` built-in global variable contains the name of the job that established the SQL server mode connection. It is of type, `VARCHAR(28)`. If there is no server mode connection, the value is `NULL`.

Here is an example to capture the details for the server mode job:

```
SELECT * FROM TABLE(QSYS2.GET_JOB_INFO(QSYS2.SERVER_MODE_JOB_NAME)) A;
```

SYSIBM.CLIENT_IPADDR

The `SYSIBM.CLIENT_IPADDR` built-in global variable contains the IP address of the current client, as returned by the system. It is of type, `VARCHAR(128)`. If the client did not connect by using the TCP/IP or Secure Sockets Layer (SSL) protocol, the value is `NULL`.

SYSIBM.CLIENT_PORT

The `SYSIBM.CLIENT_PORT` built-in global variable contains the port number used by the current client to communicate with the server. It is of type `INTEGER`. If the client did not connect by using the TCP/IP protocol, the value is `NULL`.

SYSIBM.CLIENT_HOST

The `SYSIBM.CLIENT_HOST` built-in global variable contains the host name of the current client, as returned by the system. It is of type `VARCHAR(255)`.

If the client connection originated from an application running on the local system, the value of the global variable is `NULL`. The server obtains the client IP address from the network when the connection is accepted. If the process did not originate from a remote system using TCP/IP, the value is `NULL`.

Here is an example to define an RCAC permission to allow a specific client IP address, port number, or host name used by the client to access the `CUSTOMER_TABLE` table.

```
CREATE OR REPLACE PERMISSION HOST_ACCESS ON CUSTOMER_TABLE
   FOR ROWS WHERE
       (SYSIBM.CLIENT_IPADDR = '9.181.88.248' OR
        SYSIBM.CLIENT_PORT = 51074 OR
        SYSIBM.CLIENT_HOST = 'IBMSYSTEM.IBM.COM')
   ENFORCED FOR ALL ACCESS
   ENABLE;
```

SYSIBM.ROUTINE_SCHEMA

The `SYSIBM.ROUTINE_SCHEMA` built-in global variable contains the schema name of the currently running routine. It is of type, `VARCHAR(128)`. If there is no routine currently running, the value is `NULL`.

The `ROUTINE_SCHEMA` variable is set only for procedures and functions. The variable is not set for triggers.

SYSIBM.ROUTINE_SPECIFIC_NAME

The `SYSIBM.ROUTINE_SPECIFIC_NAME` built-in global variable contains the name of the currently running routine. It is of type, `VARCHAR(128)`. If there is no routine currently running, the value is `NULL`.

The `ROUTINE_SPECIFIC_NAME` variable is set only for procedures and functions. The variable is not set for triggers.

SYSIBM.ROUTINE_TYPE

The `SYSIBM.ROUTINE_TYPE` built-in global variable contains the type of the currently running routine. It is of type, `CHAR(1)`. The value of the global variable is '`P`' for procedure or '`F`' for function. If there is no routine currently running, the value is `NULL`.

Table 3. Places where routine information is referenced

Places where information is referenced	ROUTINE_SCHEMA	ROUTINE_SPECIFIC_NAME	ROUTINE_TYPE
ROUTINE_PRIVILEGES view	✓	SPECIFIC_NAME	
SYSROUTINEAUTH view	✓	SPECIFIC_NAME	
SYSFUNCS view	✓	SPECIFIC_NAME	
SYSPROCS view	✓	SPECIFIC_NAME	
ROUTINES view	✓	SPECIFIC_NAME	✓
SYSPROGRAMSTAT view		SPECIFIC_NAME	✓
SYSROUTINES table	✓	SPECIFIC_NAME	✓
QSQPRCED API	✓		

Here is an example to define an RCAC mask to allow actual salary data in the `EMPLOYEE` table to be seen only for a particular routine schema, routine name, and routine type:

```
CREATE MASK EMP_ACCESS ON EMPLOYEE
  FOR COLUMN SALARY
  RETURN
  CASE
    WHEN (SYSIBM.ROUTINE_SCHEMA = 'HUMAN_RESOURCE_DEPT' AND
          SYSIBM.ROUTINE_SPECIFIC_NAME = 'HRDEPT' AND
          SYSIBM.ROUTINE_TYPE = 'P')
      THEN SALARY ELSE 00000
    END ENABLE;
```

SYSIBM.PACKAGE_NAME

The `SYSIBM.PACKAGE_NAME` built-in global variable contains the name of the package currently being used for a DRDA connection. It is of type, `VARCHAR(128)`.

If there is no package currently running, the value is `NULL`.

SYSIBM.PACKAGE_SCHEMA

The `SYSIBM.PACKAGE_SCHEMA` built-in global variable contains the schema name of the package currently being used for a DRDA connection. It is of type, `VARCHAR(128)`. If there is no package currently running, the value is `NULL`.

SYSIBM.PACKAGE_VERSION

The `SYSIBM.PACKAGE_VERSION` built-in global variable contains the version identifier of the package currently being used for a DRDA connection. It is of type, `VARCHAR(64)`.

If there is no package currently running or the current running package does not have a version identifier, the value is `NULL`. A package will have a version identifier only when it is created from a server other than DB2 for i.

Table 4. Places where package information is referenced

Places where information is referenced	PACKAGE_NAME	PACKAGE_SCHEMA	PACKAGE_VERSION
SYSPACKAGE view	✓	✓	
SYSPACKAGESTAT view	✓	✓	
SYSPACKAGESTMTSTAT view	✓	✓	
SYSPACKAGEAUTH view	✓	✓	
Database Monitor (DBMON)	✓		

Here is an example to collect package information (name, schema, and version) before each insert on the `DATATAB` table:

```
CREATE TRIGGER RECORD BEFORE INSERT ON DATATAB
REFERENCING NEW ROW AS N
FOR EACH ROW MODE DB2ROW ENABLE SECURED
BEGIN
  INSERT INTO INFOCOLLECTION(SYSIBM.PACKAGE_NAME,
                             SYSIBM.PACKAGE_SCHEMA,
                             SYSIBM.PACKAGE_VERSION);
END;
```

Summary

SQL built-in global variables are intended to provide easy access to useful environmental information for users. As described in this article, these SQL built-in global variables can be easily used for auditing and debugging. Go ahead and give them a try then!

Resources

The following references provide additional information on built-in global variables and DB2 for i:

- DB2 for IBM i Group PTF Schedule
- DB2 for i SQL reference