DbManager - Tuning and Debugging

Debug information and error information of process D2000 DBManager are shown in the process text console and written to the log file in the subdirectory **TRACE** of the application directory as well. If the process D2000 DBManager is started as *SELF.DBM*, it creates the **DBManager.log** and **DBManager_ERR.log** files. If this process is started as a *name.DBM*, it creates the **DBManager_name.log** and **DBManager_meno_ERR.log**.
If the size of log file reaches the predefined value of 10 MB, it will be renamed to **DBManager_name_prev.log** (**DBManager_meno_ERR_prev.log**) and a new log file will be created. Previous **DBManager_name_prev.log** (**DBManager_meno_ERR_prev.log**) will be deleted.

If the parameter Size of the log file is set to a non-zero value in the object of *Database* type, both the debug and error information, generated by action performed in this database, will be written to a separate log file of the database. The name of this log file is **name_of_database.log**, while dots in the name of the *Database* object will be replaced by "_" (e.g. if the object of the *Database* type is named *DB.Test*, its log file will be *DB_Test.log*). For error information, the file name is **name_of_database_ERR.log**.
If the size of the log file reaches a size equal to the value of the parameter Size of the log file, it will be renamed to **name_of_database_prev.log** and a new log file will be created. Previous **name_of_database_name_prev.log** will be deleted. It is the same for the file **name_of_database_ERR.log**.

Process D2000 DBManager provides several debugging levels:

- Error logs
- Action listing
- Listing the actions, the duration of which is longer than the entered number of seconds
- Logs with DBG.DBMANAGER debug category enabled
- Logs with DBG.DBMANAGER.DATA debug category enabled
- Logs with DBG.DBMANAGER.DBCTX debug category enabled
- Logs with DBG.DBMANAGER.DBCTX.CSV debug category enabled
- Logs with DBG.DBMANAGER.SQL_CONNECT debug category enabled
- Logs with startup parameter /DBD<number_of_requests>
- Tell command SHOW_HANDLE
- Tell command SHOW_CONNECT
- Tell command SET_WATCHDOG
- Tell command SET_WATCHDOG_QUEUE
- Tell command TIME_STATISTICS
- Optimization and debugging

Note: Starting with D2000 version 23, a DbManager Diagnostic Pack is available that can visualize the information provided by tell commands in the D2000 Cnf or D2000 GrEditor environment.

Error logs

Errors are always written to the log file which ends with **_ERR.log**. Error log usually contains the SQL command that caused the error.

Example:

*12:40:08.760 25.02 *** Error in con 24:*
12:40:08.766 25.02 con 24:SELF.EVH;E.Test_JS_Column_Types(1928) 2448;: 36

12:40:08.771 25.02 con 24:[InsertRecord] Execute: INSERT INTO "ROVE_OD"."TEST_COLUMN_TYPES" ("ID","ColumnName","ColumnDate") VALUES (:1,:2,:3) ,row 1,Exception name:
OCI.THICK.LIB_ERROR
Message: ORA-00001: unique constraint violated (ROVE_OD.SYS_C00112480)

Call stack traceback locations:
0x453d18 0x4542d1 0x4509dc 0x991db4 0x999458 0x906822 0x8ef428 0x9d187c 0x76b6652b

Note 1: If a script that executes an action, during which an error occurred, is in debug mode, the information about this error is sent to the script editor - *De bug* tab.

Note 2: Each ESL action for work with the database (through D2000 DBManager) saves the information about the position in the ESL code from which it was called. If some error occurs, both the information about the found error and its position are written up (see the beginning of the example).

Action listing

Detailed debug information is written to the log file if the option Debug is checked off in the configuration of the object of Database type. The log includes the information on individual ESL actions (action name, begin and end), database, and connection that the action uses.

Example:

14:18:59.677 Db TestDb con 2:PG_INSERT END
14:19:01.099 Db TestDb con 2:PG_INSERT BEG

The log contains the begin (BEGIN) and end (END) statuses of the PG_INSERT action in the database (*Db*) named *TestDb*. The action was executed through the connection (*con*) no. 2 and took 422 milliseconds.

Note: If the separate log file for the database is used instead of a common log file of DBManager (see the parameter Size of the log file), the logged information on every line does not contain the database name.

Watchdog message (WD) and information database logs are written to the log file with a period of 60 seconds:
09:00:27.786 Db TestDb WD: 5 (4/1/0) cons: avail- 3,normal- 2, handles- 4

The log informs the user that the database *TestDb* has 5 connections - 4 non-transactional, 1 transactional, and 0 reserved for browsers. 3 connections are available (*avail*) and 2 are used (*normal*).

All possible states are listed in the table below:

Status	Description
Init	Connection is in the initial phase (it is being created). Transient status.
Avail	Connection is free (available).
Normal	Connection is used.
Live	Connection is passing from *Avail* status to *Normal* status. Transient status.
Die	The connection is being closed. Transient status.
Zombie	The connection is closed, and the service thread is finished. Transient status.

Handles indicates the number of handles that are opened by **D2000 DBManager**. There are 4 types of handles:

- **table handle** can be opened from process D2000 HI through the list of database tables, in the Browser, or from a script by using the actions DB_CONNECT, PG_CONNECT, SQL_CONNECT, SQL_PREPARE as well as for a short period of time (during an action) by calling DBS_*.
- **transaction handle** can be opened by the action DB_TRANS_OPEN.
- **connection handle** is opened by the SQL_CONNECT, if the first parameter of the action is *connectString*.
- **database handle** is opened by the action SQL_CONNECT, if the first parameter of the action (*dbObjIdent*) is of Database type.

If the database table has the parameter Time depth defined in its configuration and process D2000 DBManager periodically deletes old data from it, the log file will also include the information about the beginning and end of the periodic deleting:

10:29:32.068 11.08 Db TestDB table Time_Test periodic delete BEG
10:29:32.162 11.08 Db TestDB table Time_Test periodic delete END

Listing the actions, the duration of which is longer than the entered number of seconds

Debug information about operations, the duration of which is longer than the entered number of seconds, is written to the log file after setting the parameter Log operations longer than (sec) to a nonzero value in the configuration *Database* object. The log contains the information about connection, on which the action was executed, localization of the action in ESL script, the calling message from DbManager, and the duration of the action in milliseconds.

Example:

19:27:37.286 01.07 con 1:DBS_READ BEG
19:27:37.297 01.07 con 1:DBS_READ END
19:27:37.298 01.07 con 1:DBS_READ
SELF.EVH;E.Test(1934) 1230;: 16
- M.DB.CDB_CTRLMSG - 122[ms]

Logs with DBG.DBMANAGER debug category enabled

Detailed debug information is written to the log file after enabling the *DBG.DBMANAGER* debug category in the process D2000 System Console - Debug info window. Each log of action also contains information on procedures called in process D2000 DBManager (that may be useful for developers) and texts of executed SQL commands.

Example shows the log of execution of the SQL_PREPARE action:

12:05:55.301 22.08 Db TestDB con 1:SQL_PREPARE BEG
12:05:55.302 22.08 PrepareSqlStmt: SELECT ID_MATERIAL,Name FROM material
12:06:30.028 22.08 PrepareSqlStmt end
12:06:30.029 22.08 Db TestDB con 1:SQL_PREPARE END

The examples of logs with differences between ODBC and OCI versions of **D2000 DBManager** are in the chapter Example - logs for process D2000 DBManager.

Logs with DBG.DBMANAGER.DATA debug category enabled

Detailed debug information is written in the log file after enabling the *DBG.DBMANAGER.DATA* debug category through the process D2000 System Console - Debug info window. Each log of action contains also a list of values that were written to or read from the database.

Example shows the action for displaying the first page of the table *MAT_GROUP* in the displayer of Browser type opened in process D2000 HI:

10:35:22.601 25.08 Db MyDB con 5:GetDatabaseData START
10:35:22.602 25.08 Open database DSN: TENTO
10:35:23.727 25.08 Open database DSN: TENTO end
10:35:23.728 25.08 PrepareSelectStmt: SELECT "ID_GROUP","NAME" FROM "SKVI"."MAT_GROUP"
10:35:23.736 25.08 PrepareSelectStmt end
10:35:23.737 25.08 ReadPage (Direction: 2): SELECT "ID_GROUP","NAME" FROM "SKVI"."MAT_GROUP"
10:35:23.741 25.08 ReadPage data row 1: 7; 'wood';
10:35:23.742 25.08 ReadPage data row 2: 1; 'plastic';
10:35:23.742 25.08 ReadPage data row 3: 2; 'metal';
10:35:23.743 25.08 ReadPage data row 4: 3; 'paper';
10:35:23.743 25.08 ReadPage data row 5: 6; 'water';
10:35:23.744 25.08 ReadPage data row 6: 4; 'other';
10:35:23.745 25.08 ReadPage end
10:35:23.746 25.08 Db MyDB con 5:GetDatabaseData END

The examples of logs with differences between ODBC and OCI versions of **D2000 DBManager** are in the chapter Example - logs for process D2000 DBManager.

Logs with DBG.DBMANAGER.DBCTX debug category enabled

Detailed debug information is written into the log file after enabling the DBG.DBMANAGER.DBCTX debug category through the process D2000 System Console - Debug info window.

After modifying the context of the process that is bound in D2000 DBManager (by ESL action DB_SET_PROCESS_PARAMS), the information is written to the main log of the given D2000 DBManager process, which is useful for developers. The following logs are independent of the parameter Debug in the configuration of the object of *Database* type (they are not recorded in the *Trace* directory).

This example shows a log after executing the action DB_SET_PROCESS_PARAMS when setting the parameter:

```
[01-08-2013 13:14:13]  DBCTX: Call Chain of DB_SET_PROCESS_PARAMS - SELF.EVH;E.SetParams( 1347485) 8735;: 12
[01-08-2013 13:14:13] DBCTX: CTX created for idOwner = 55[[01-08-2013 13:14:13] DBCTX: INSERT parameter
"NAME=SELF.DBM" of idOwner = 55
```

Note: The log about creating a context ("CTX created ...") appears only when the action is called for the first time since starting the process.

This example shows a log after executing the action DB_SET_PROCESS_PARAMS when deleting the parameter:

```
[01-08-2013 13:16:53] DBCTX: Call Chain of DB_SET_PROCESS_PARAMS - SELF.EVH;E.SetParams( 1347485) 8936;: 12
[01-08-2013 13:16:53] DBCTX: DELETE parameter "NAME" of idOwner = 55
```

To write the detailed information to the log file, enable the parameter Debug in the configuration of the *Database* object. The log contains the information about the parameters that are set in the context of a given client process, which starts DB actions through D2000 DBManager.

This example shows a log after executing the action SQL_SELECT, if the parameters of process context have been already set by action DB_SET_PROCESS_PARAMS:

```
15:01:14.114 01.08 con 1:SQL_SELECT BEG
15:01:14.121 01.08 con 1: D2000_PROCESS_PARAMS - SET PARAMS "NAME=VERSION27" of dbCtxId = 55
15:01:14.125 01.08 con 1:SQL_SELECT END
```

Logs with DBG.DBMANAGER.DBCTX.CSV debug category enabled

Detailed CSV information is written into the log file after enabling the DBG.DBMANAGER.DBCTX.CSV debug category by the process D2000 System Console - Debug info window. This information is written after enabling the parameter Debug in the configuration of the *Database* object. The log contains the structured CVS information about single parameters that are set in the context of a given client which starts DB actions through D2000 DBManager.

CSV header:

```
TIMESTAMP;D2000_PROCESS_PARAMS;CID;COMMAND;TID;CURRENT_TASK;DB_CTX_ID;DB_CTX_ON;MC1;MC2;OPERATION;SUCCESS
set ;={TRUE/FALSE};
del;={TRUE/FALSE}
```

CSV columns: time stamp, constant "D2000_PROCESS_PARAMS", identifier of connection, DB command, identifier of internal transaction, task name, client ID, existence of context (TRUE/FALSE); modifying counter before and after (MC1 and MC2), operation and its success. The operation is either modification of parameters (set) or deleting the context on the given connection (del).

Example 1:

```
15:28:12.899 01.08 con 1:SQL_SELECT BEG
15:28:12.905 01.08 ;D2000_PROCESS_PARAMS;1;SQL_SELECT;1005;tdb_task_000000000905BB40;55;TRUE;0;1;set
"NAME=VERSION27";TRUE
15:28:12.908 01.08 con 1:SQL_SELECT END
```

Example 2:

```
15:28:22.051 01.08 con 1:SQL_SELECT BEG
15:28:22.054 01.08 ;D2000_PROCESS_PARAMS;1;SQL_SELECT;1006;tdb_task_000000000905BB40;55;TRUE;1;2;set;TRUE
15:28:22.056 01.08 con 1:SQL_SELECT END
```

Logs with DBG.DBMANAGER.SQL_CONNECT debug category enabled

Enabling the debug category has the same effect as inserting "DEBUG" into the *connectString* parameter in all calls of the SQL_CONNECT action, i.e. the writing debug information of the **SQL_CONNECT** action and all other actions using this connection into the log file of **D2000 DBManager**.

Note: The DBG.DBMANAGER.SQL_CONNECT debug category has no effect on the SQL_CONNECT actions which contain the reference to an object of Database or Database Table types, because these actions are influenced by the parameter Debug in the configuration dialog box of the respective *Databas* type object. The category affects only the **SQL_CONNECT** actions using the parameter *connectString* for their connection, i.e. the text connect string, debugging of which cannot be enabled in process **D2000 CNF** and it needs to be enabled by inserting the DEBUG parameter into the *connectString* parameter.

Logs with startup parameter /DBD<number_of_requests>

Starting from version 7.01.023 a performance tuning feature for reporting performance problems in automatic (non-transactional) connections is available. If DBManager is run with parameter /DBD<number_of_requests>, it will write the following information to its log file during startup:
Performance logging is ON.

If a number of requests queued for any of the automatic connections exceeds the value *number_of_requests*, this information will be recorded in the log file (i.e. the log file of the database if the value Size of the log file is configured, otherwise the log file of DBManager). The log will contain the performance warning, connection number, and number of queued requests:
10:32:12.983 14.02 con 1:performance warning: 2 requests

This warning means that incoming non-transactional request will have to wait till the queued requests are processed. If these warnings are more frequent, you might consider increasing the value of automatic connections to increase request throughput. The recommended value of <number_of_requests> is 1, i.e. running DBManager with parameter /DBD1.

Note 1: If the parameter Debug is checked off, the processing of requests is slower due to writing the debug information to the log file. We recommend increasing the number of automatic connections or the value of <number_of_requests> to /DBD2 to avoid performance warnings.

Note 2: Transactional connection does not need this kind of performance tuning, because it is currently being used only sequentially from one event.

Tell command SHOW_HANDLE

An interactive query of open handles with the help of the SHOW_HANDLE Tell command. The syntax of the command is:

SHOW_HANDLE [HOBJ] or [mask [INFO]]

The optional parameters may be:

- HOBJ of the object Database, the open handles of which are to be displayed.
- HOBJ of the object Database table, the open handles of which are to be displayed.
- HOBJ of the object Structure definition - the open handles of database tables whose structure definition matches the given HOBJ will be displayed.
- mask_of_the_name, the open handles of which are to be displayed.
- mask_of_information from where the handle has been opened, if the *INFO* keyword is used.

If the **SHOW_HANDLE** command is called with no parameters, information on all open handles will be written to the log file.
For each handle, the log contains the database name, connection number, descriptor type, handle name, and information from where the handle was open.
The displayed type and name of the handle varies depending on handle types as the following table states:

Descriptor type	Displayed type	Descriptor name
table handle	*table*	Name of object of *Database table* type.
transaction handle	*trans*	Name of object of *Database* type, in which the transaction is open.
connection handle	*connect*	The parameter *connectString* of the SQL_CONNECT action.
database handle	*dbase*	Name of object of *Database* type, that represented the parameter *dbObjIdent* in the action SQL_CONNECT.

Example:

CONO connection established (IPC_TCPIP)
Receiv TELL Command : SHOW_HANDLE
==========
->Db TestDB con 1:table MAT_GROUP: <HI mycomp,S.Test_DBmanager>
->Db TestDB con 2:trans TestDB: <S.Test_DBmanager: 220>
==========

D2000 DBManager has two open descriptors. The first one is on the connection nr. 1. It is a table descriptor, which is opened in the Browser *MAT_GROU Popen*. The descriptor is open from the picture *S.Test_DBmanager* in process D2000 HI running on the computer with the name of *mycomp*.
The second descriptor is on the connection nr. 2, a transaction descriptor that is open from the 220th row of the script in the picture *S.Test_DBmanager*.

Tell command SHOW_CONNECT

Interactive query for open connection by using the SHOW_CONNECT Tell command. The syntax of the command is:

SHOW_CONNECT [HOBJ [ID]] or [mask [ID]] [DETAIL]

The optional parameters are:

- HOBJ of the Database type object, its open connections are to be displayed
- mask (pattern) matched against the *Database* object's name, the open connections of which are to be displayed
- ID - connection number or transaction number (returned by the DB_TRANS_OPEN action in the parameter *handleIdent_Int*) is to be displayed
- if parameter "DETAIL" is specified, also all descriptors, opened on a given connection, will be displayed (in the same format as SHOW_HANDLE)

Log includes:

- name and number of open connections for every database matching the defined filter
- for every connection:
  - database name
  - connection number
  - connection status
  - number of open descriptors
  - time from the last operation performed on this connection (or the word *busy* and information about the internal status of the connection if the operation is currently in progress. Example: *busy (U_EXECDIRECT1/D_EXECDIRECT2)*)
  - information whether the connection is transactional, non-transactional or non-transactional reserved for browsers
  - a transaction number (the parameter *handleIdent_Int* in the DB_TRANS_OPEN action) for transactional connection
- log in the format of SHOW_HANDLE for each descriptor (if parameter "DETAIL" is specified)

Example of log (connection in all databases the names of which start with *Te* along with descriptors log):

```
Receive TELL Command : SHOW_CONNECT Te* DETAIL
=================================================================
->Db TestDB 2 cons
->Db TestDB con 1:normal, 1 handles, idle 04:24:682, non-transact
->Db TestDB con 1:table   MAT_GROUP: <HI mycomp,S.Test_DBmanager>
->Db TestDB con 2:normal, 2 handles, idle 20:037, transact 1053
->Db TestDB con 2:trans TestDB: <S.Test_DBmanager: 220>
->Db TestDB con 2:table   MAT_GROUP: <S.Test_DBmanager: 109>
```

Note: Information on whether the connection is transactional or non-transactional along with the transaction number gives outdated value (transaction has already been finished) for connections in the *Avail* status. Such connections will be recycled in transactional or non-transactional mode if needed.

Tell command SET_WATCHDOG

SQL operations execution which took a longer time can be monitored by a tell command SET_WATCHDOG. The syntax of the command is:

SET_WATCHDOG database_mask seconds

Required parameters:

- database_mask - mask of name of *Database* object which connections are to be monitored
- seconds - the number of seconds to elapse before the database operation will be logged, if it is still executing.
  A value of 0 disables monitoring.

A tracing task is activated at the first run of the Tell command SET_WATCHDOG with nonzero parameter *seconds* (at least one database must match the specified mask). This task checks in every second if the connection to the monitored database did not exceed the set time of performing (parameter *seconds*). If this time is exceeded, this information is written into the log.

The log contains:

- database name (if a dedicated log is not configured for this Database),
- number of connections,
- duration of the operation,
- information on the internal status of the connection (for developers of D2000).

Example of log (from database log, so the database name is not written):

```
16:45:28.763 24.11 Performance WD: con 3: operation lasts 27 sec in U_EXECDIRECT1/D_EXECDIRECT6
16:45:29.767 24.11 Performance WD: con 3: operation lasts 28 sec in U_EXECDIRECT1/D_EXECDIRECT6
16:45:30.782 24.11 Performance WD: con 3: operation lasts 29 sec in U_EXECDIRECT1/D_EXECDIRECT6
16:45:31.798 24.11 Performance WD: con 3: operation lasts 30 sec in U_EXECDIRECT1/D_EXECDIRECT6
16:45:34.829 24.11 Performance WD: con 3: operation lasts 2 sec in U_EXECDIRECT1/D_EXECDIRECT6
```

After the SQL operation is finished, detailed info is written into the error log file. Examples:

```
17:54:00.116 16.02 con 1:Query execution duration 00:00:01.103 SQL_CONNECT TransactId 10319, dbTransId 0, Handle
1104492764, connectString {}, DbTableId 1473 {DB.MATERIAL}, Comment {NS1PHUM3_HI.HIS;S.sql_TEST( 9473) 10864;
BT_connect_OnClick: 248}
17:54:18.926 16.02 con 1:Query execution duration 00:00:01.105 SQL_PREPARE TransactId 10320, dbTransId-1, Handle
10319, Statment {SELECT ID_MATERIAL FROM material }, bBindIn FALSE, FetchSize 1, colNr 17, Comment {NS1PHUM3_HI.
HIS;S.sql_TEST( 9473) 10864;BT_prepare_OnClick: 279}
17:55:59.113 16.02 con 2:Query execution duration 00:00:01.110 DB_TRANS_OPEN TransactId 10325, Comment
{NS1PHUM3_HI.HIS;S.sql_TEST( 9473) 10864;BT_db_trans_open_OnClick: 543}
```

Meaning of individual items:

- SQL_CONNECT / SQL_PREPARE / DB_TRANS_OPEN .. - performed database action
- TransactId - ID of D2000 transaction
- dbTransId - ID of a database transaction. If it is greater than 0, the database action is transactional (executed inside DB_TRANS_OPEN). If it is zero or less, the database action is non-transactional.
- Handle - ID of the handle in the context of DBManager
- DbTableId - HOBJ and name of D2000 object *Table*
- Comment - sequence of procedure calls in ESL scripts (call chain)
- connectString, Statement, bBindIn, FetchSize, colNr, MaxRows .. parameters specific for individual database actions

Note: The start-up of SQL operation monitoring is written in the DBManager log. The log contains the list of databases that match the specified mask *database_mask*. The log also contains the information on the start-up of the watchdog at first calling the tell command SET_WATCHDOG.

```
16:45:27.735 24.11 ======================================================================
16:45:27.736 24.11 ->Db DBC_ROVE_OD 2 cons
16:45:27.737 24.11 ->Db DBC_KOMP_OD 3 cons
16:45:27.742 24.11 Starting performance watchdog task for database operations
16:45:27.743 24.11 ======================================================================
```

If value of parameter *seconds* is 0 in all databases, the watchdog stops database monitoring and the following information appears in the log:

```
17:37:41.588 24.11 Performance watchdog: going to sleep (no more databases to monitor)
```

The watchdog is running so it is not necessary to start it up at the next activating of the monitoring and only information about activation appears in the log:

```
18:45:28.749 24.11 Performance watchdog: starting monitoring
```

Tell command SET_WATCHDOG_QUEUE

Database actions execution of which - **including time spent in queues of DBManager** - took a longer time can be monitored by a tell command SET_WATCHDOG_QUEUE. The syntax of the command is:

SET_WATCHDOG_QUEUE database_mask seconds

Required parameters:

- database_mask - mask of name of *Database* object which connections are to be monitored
- seconds - minimum length of database actions to be monitored.
  A value of 0 disables monitoring.

The difference between SET_WATCHDOG and SET_WATCHDOG_QUEUE is this:
command SET_WATCHDOG monitors operations, the execution of which in a database lasts longer than a specified time. But if several clients share the same automatic (non-transactional) connection, from client's point of view a database action can last long because it is waiting in a queue where it is preceded by other database actions by other clients. For that reason a tell command SET_WATCHDOG_QUEUE was implemented - to monitor the total execution time of database action from od insertion into a queue to the end of its execution.

After the database action is finished, detailed info is written into the error log file. Examples:

```
18:01:21.579 16.02 con 1:Query total duration 00:00:01.130, execution 00:00:00.105 SQL_EXEC_PROC TransactId
10331, dbTransId-1, Statement {{ call TESTF_IN_OUT (?) }}, Comment {NS1PHUM3_HI.HIS;S.sql_TEST( 9473) 11332;
BT_exec_proc_OnClick: 626}
18:02:39.149 16.02 con 1:Query total duration 00:00:01.103, execution 00:00:00.202 SQL_CONNECT TransactId 10335,
dbTransId 0, Handle 1104492765, connectString {}, DbTableId 1473 {DB.MATERIAL}, Comment {NS1PHUM3_HI.HIS;S.
sql_TEST( 9473) 11453;BT_connect_OnClick: 248}
18:02:42.437 16.02 con 1:Query total duration 00:00:01.103, execution 00:00:00.035 SQL_DISCONNECT TransactId
10335, dbTransId-1, Comment {NS1PHUM3_HI.HIS;S.sql_TEST( 9473) 11453;BT_connect_OnClick: 248}
18:02:51.163 16.02 con 1:Query total duration 00:00:01.105, execution 00:00:00.654 SQL_PREPARE TransactId 10337,
dbTransId-1, Handle 10336, Statement {SELECT ID_MATERIAL FROM material }, bBindIn FALSE, FetchSize 1, colNr 17,
Comment {NS1PHUM3_HI.HIS;S.sql_TEST( 9473) 11453;BT_prepare_OnClick: 279}
18:02:54.275 16.02 con 1:Query total duration 00:00:01.110, execution 00:00:00.239 SQL_FETCH TransactId 10338,
dbTransId-1, Handle 10336, MaxRows 100, Comment {NS1PHUM3_HI.HIS;S.sql_TEST( 9473) 11453;BT_fetch_OnClick: 299}
18:02:57.009 16.02 con 1:Query total duration 00:00:01.104, execution 00:00:00.109 SQL_FREE Handle 10336, Comment
{NS1PHUM3_HI.HIS;S.sql_TEST( 9473) 11453;BT_free_OnClick: 325}
```

Meaning of individual items: see a description in section SET_WATCHDOG.

Tell command TIME_STATISTICS

Tell command displays statistics of execution of individual database action types per-database or per-table (if a parameter DETAIL is specified). For every type of database action (browser open, transaction start, DB_* and DBS_* actions, etc) printout contains the total number of executed actions, total and average duration, and duration of the longest action together with a comment (ESL call chain).
Statistics with zero execution counts are discarded.
If a parameter DETAIL is specified, after the printout of database statistics there follows a printout of per-table statistics of all child tables.
Example of a printout:

```
09:36:28.023 21.02 ================================================================
09:36:28.026 21.02 ->Db MesDB total time 000 00:00:40.572
09:36:28.029 21.02  Operation BROWSER_OPEN executions 3 total duration 000 00:00:03.686, average 000 00:00:
01.229, maximum 000 00:00:01.468 by
09:36:28.032 21.02  Operation DB_CONNECT executions 2 total duration 000 00:00:02.207, average 000 00:00:01.104,
maximum 000 00:00:01.104 by NS1PHUM3_HI.HIS;S.Test_DBmanagera( 9424) 10510;DB_CONNECT_OnClick: 109

09:36:28.034 21.02  Operation DB_CONTROL executions 15 total duration 000 00:00:16.589, average 000 00:00:01.106,
maximum 000 00:00:01.129 by con 1:DBS_READ
NS1PHUM3_HI.HIS;S.TestDbRefresh( 10105) 10429;StressTest_OnClick: 20

09:36:28.037 21.02  Operation DB_REFRESH_TABLE executions 2 total duration 000 00:00:02.526, average 000 00:00:
01.263, maximum 000 00:00:01.266 by Triggered by commit of NS1PHUM3_HI.HIS;S.TestDbRefresh( 10105) 10429;
StressTest_OnClick: 10

09:36:28.040 21.02  Operation DB_TRANS_OPEN executions 4 total duration 000 00:00:04.426, average 000 00:00:
01.106, maximum 000 00:00:01.107 by NS1PHUM3_HI.HIS;S.Test_DBmanagera( 9424) 10479;TRANS_OPEN_OnClick: 227

09:36:28.043 21.02  Operation DB_TRANS_CONTROL executions 2 total duration 000 00:00:02.301, average 000 00:00:
01.150, maximum 000 00:00:01.197 by NS1PHUM3_HI.HIS;S.TestDbRefresh( 10105) 10429;StressTest_OnClick: 35

09:36:28.046 21.02  Operation DB_TRANSACT_ABORT executions 8 total duration 000 00:00:08.837, average 000 00:00:
01.105, maximum 000 00:00:01.107 by
09:36:28.049 21.02 ->Table DB.MAT_GROUP total time 000 00:00:04.420
09:36:28.052 21.02  Operation BROWSER_OPEN executions 1 total duration 000 00:00:01.107, average 000 00:00:
01.107, maximum 000 00:00:01.107 by
09:36:28.054 21.02  Operation DB_CONTROL executions 3 total duration 000 00:00:03.313, average 000 00:00:01.104,
maximum 000 00:00:01.105 by NS1PHUM3_HI.HIS;S.Test_DBmanagera( 9424) 10479;DB_CONNECT_OnClick: 109

09:36:28.057 21.02 ->Table DB.MATERIAL total time 000 00:00:16.913
09:36:28.060 21.02  Operation BROWSER_OPEN executions 1 total duration 000 00:00:01.111, average 000 00:00:
01.111, maximum 000 00:00:01.111 by
09:36:28.063 21.02  Operation DB_CONTROL executions 12 total duration 000 00:00:13.276, average 000 00:00:01.106,
maximum 000 00:00:01.129 by con 1:DBS_READ
NS1PHUM3_HI.HIS;S.TestDbRefresh( 10105) 10429;StressTest_OnClick: 20

09:36:28.065 21.02  Operation DB_REFRESH_TABLE executions 2 total duration 000 00:00:02.526, average 000 00:00:
01.263, maximum 000 00:00:01.266 by Triggered by commit of NS1PHUM3_HI.HIS;S.TestDbRefresh( 10105) 10429;
StressTest_OnClick: 10

09:36:28.069 21.02 ================================================================
```

Optimization and debugging

The database parameters can be set in process D2000 CNF. A connection created by the action SQL_CONNECT contains these parameters in its connect string.

Parameter name in CNF	Parameter name in connect string
Precreated connections	PRE=...
Maximum connections	MAX=...
Maximum automatic connections	NTC=...
Reserved browser connections	BRC=...
Close unused connections after (sec)	CLOSE=...
Close DBManager after timeout (min)	WDC=...
Debug	DEBUG
Off	OFF
Maximum returned rows	MR=...
Empty operations after an inactivity period	EO=...

Example: settings of database additional parameters in its connect string:

DEBUG;PRE=10;MAX=120;CLOSE=300;NTC=20;WDC=7;MR=1000

For the database, saving the debug information into the file *DBManager.log* is enabled. After starting the process D2000 DBManager, 10 connections to the database are to be created, maximum number of connections is 120. A connection, that is not used, will be closed after 300 seconds, the maximal number of automatic connections is 20, and the internal watchdog terminates the process D2000 DBManager if some connections to the database are not functioning (is frozen) for more than 7 minutes.

- For reasons of speed optimization and mutual client non-blocking, process D2000 DBManager creates more than one automatic connection to the database. Automatic connections can be created immediately after starting the process D2000 DBManager, so they are prepared to be used (e.g. creating a connection in case of the database of Oracle type may take 1 second or more and it causes delays in ESL scripts). The number of the pre-created connections can be adjusted in process D2000 CNF the object of Database type using the parameter **Precreated connections**. Maximal number of connections can be limited by parameter **Maximum connections**, while the value 0 disables the limitation. If process D2000 DBManager uses all connections, then the next call to create a connection (DB_TRANS_OPEN) will return the error DBM_MAX_CONNECTIONS.
- Using the parameter **Maximum automatic connections** allows to set a maximum number of connections created automatically. When the parameter is not set, then:
    - at most 1/4 of Max connections will be used for automatic connections
    - if Max<4, only 1 automatic connection will be created
    - if Max=0, at most 5 automatic connections will be created (default value)

  If process D2000 DBManager uses the maximum number of automatic connections, then a new connection will not be created as a result of the next request (e.g. non-transactional SQL_CONNECT, PG_CONNECT, or DB_CONNECT), but one of the existing will be used. The parameter **Maximum automatic connections** is important for the database to not exceed the defined number of connections.
- For time optimisation reasons the process D2000 DBManager "recycles" existing connections. If a connection is closed (e.g. DB_TRANS_CLOSE) or is not used (it is the result of SQL_CONNECT and SQL_DISCONNECT performed), then it is signed as available. Available connections can be repeatedly used; it saves time to connect to the database and accelerates the execution of scripts. The available connections, that do not belong to prepared ones and are not used, will be terminated after the expiration of the time defined in the parameter **Close unused connections** in seconds (pre-created connections are not being terminated). Adjusting the parameter to a negative value will tell the process D2000 DBManager not to terminate available connections - in this case, their number will increase and it will reflect the status of D2000 DBManager during the maximum system load.
- To speed up the operations of the Browser displayer it is possible to reserve a pool of connections for these operations (browser opening, changing pages, and the action DB_REFRESH_TABLE) using the parameter **BRC** - Reserved browser connections. This feature is supported starting with D2000 version 8.00.009.
  The advantage is that no other non-transactional operations (which can last several seconds or more) will be executed by these reserved connections and the action DB_REFRESH_TABLE as well as changing pages can be faster than in default configuration when the browser operations are executed by automatic connections.
  Note: if the value of parameter **BRC** - Reserved browser connections is zero, the browser operations are executed by automatic connections, which was default before this parameter was implemented.
- Debug and information logs: process D2000 DBManager enables to display several levels of debugging logs which are in closer detail described in this document.
- Termination of process D2000 DBManager in consequence of "frozen " connections: D2000 DBManager contains the internal watchdog, which detects every minute, whether some connection to the database is not "frozen" (i.e. service thread did not return from calling of ODBC interface). In this case, process D2000 DBManager will write into a log file. If the watchdog detects such a situation n-times in a consecutive sequence (while n can be adjusted in the parameter **Close DBManager after timeout**), it will terminate process D2000 DBManager. A default value of 0 means that D2000 DBManager will not be terminated.
- Parameters for the connections created by means of the command SQL_CONNECT using so-called connect string: as these connections are not related to objects of Database type, but DSN is directly defined in the connect string, setting of the parameters can be performed directly in the connect string.
  Example:
  UID=name;PWD=user's password;DSN=DSN name;ACD;Debug;CLOSE=300
  Debug information for the created connection will be written to the file. An unused connection will be closed up after 300 seconds.

  The available connection can be "recycled" if its connect string is the same as the connect string of the connection being created.

  Debug and information logs of these connections that are "independent" on the object of Database type:
  08:59:45.710 Indep# 3 DSN=TestX:SQL_FETCH BEG
  08:59:45.710 Indep# 3 DSN=TestX:SQL_FETCH END

  the log informs us that "independent" connection no. 3, DSN of which is "TestX", performed the action SQL_FETCH.

  If parameter **Debug** is set on at least one independent connection, then the log also contains periodical information about the independent connections (written to log once a minute):
  09:14:27.752 Db Independent connects WD: 1 (1/0/0) cons:normal- 1

  the log informs us that only one independent connection exists and it is currently in use.
  Numbers in brackets give numbers of non-transactional, transactional, and browser connections.

SV._System_DBMDbPerf

From the above mentioned, it results that three groups of connections exist (database connections along with the service tasks) for each object of *Database* type at any time, but their quantity within a group is limited in some way (or it is not according to settings).

Type of connections:

- **transaction** (they are created by calling DB_TRANS_OPEN in ESL),
- **non-transaction automatic** (NTC),
- **non-transaction browser connections** (BTC) in case of reservation.

When increased load, there occurs such a state that there are more requests than connections, which is why the requests are enqueued by their type. The structured variable "SV._System_DBMDbPerf" is specially used for monitoring of queue status.

Each row of a structured variable describes all three fronts by the three values that are updated each 10[s] (see SV._System_DBMDbPerf).

- **ConNr** - number of connections within a group,
- **CurrRq** - current number of requests for a group,
- **MaxRq** - maximum number of requests for a group during the elapsed period.

The monitoring of requests in the queue enables one to find out whether the number of connections within the group is sufficient, how fast the group is able to service the requests, and whether it is necessary to increase the number of connections within the group (however, increasing of connections need not increase the number of requests processed per unit of time, for example, if the database server is overloaded,...).

The value in the column "Database" in object SV._System_DBMDbPerf identifies:

- name of object *Database* or
- name of object *Process* with a suffix DBM (DbManager). In this case, the values in a row represents the summary number of requests and connections for all databases that are serviced by the appropriate process.