javautil-dblogging Documentation

Release 19.6.0

Jim Schmidt

October 27, 2019

CONTENTS

Application instrumentation is essential to performance monitoring, unfortunately this is often performed by throwing in some logging statements using a java logging framework or wrapper, such as slf,

This however fails to capture any information essential to end-to-end monitoring as it omits what is generally the biggest source of latency, the relational database. What statements are being executed, how long do they take, which statements take up the bulk of the resources in aggreggate (an umder-performing statement invoked thousands of times an hour is not uncommon).

Oracle provides, at great expense, the ASH subystem and even that does not associated the sql statements to the application code.

This utility provides a simple Application Program Interface to allow you to record performance information in a simple, low overhead fashion from any java program or any program that allows pl/sql calls.

Thus a few judicious additions to an Oracle Form or a batch job can provide the foundation of information necessary to establish where database resources are being consumed.

Resolution of these matters is a different matter and may involve gathering statistics, altering execution plans, creating or eliminating indexes or rewriing SQL.

SGA paramaters may have to be changed.

CHAPTER

ONE

DOCUMENT ORGANIZATION

We will introduce the types of logging, show examples of how to instrument your code and what type of output is generated.

Then we will guide you through the installation and options.

CHAPTER

TWO

OVERVIEW OF LOGGING TYPES

2.1 Flat file logging

Writes log messages to a text file on the database server.

Includes:

- log_seq_nbr
- log_level
- job_log_id
- log_msg_id
- line_number
- current_timestamp, 'YYYY-MM-DDTHH24:MI:SSXFF'
- log_msg
- caller_name optional
- call_stack optional

This will allow you to easily write log messages from pl/sql in stored procedures, functions, packages and with most of the flexibility and ease of using a java logging framework.

2.1.1 Logging to flat files

Overview

Logs messages using utl_file to a directory on the database server specified.

First the database directory is created and oracle is granted permission to read and write it, then the ddl "create directory...." and "grant read, write on directory..."

Examples

log_to_file_only.proc.sr.sql

Input

```
set serveroutput on
set echo on
create or replace procedure log_to_file_only is
       long_msg clob := 'this is an absurdly long message, ' ||
                ' interesting stuff to say so I will just write meaningless ' ||
                ' stuff for a little while. ' ||
                ' The quick brown fox jumped over the lazy dog. ';

    my_log_file_name varchar(4096);
begin
    my_log_file_name := pllogger.open_log_file('log_to_file_only.text');
    pllogger.set_filter_level(9); -- all messages should go to log file
    pllogger.info('anonymous',$$PLSQL_LINE,'begin loop');
    pllogger.info($$PLSQL_UNIT,$$PLSQL_LINE,long_msg);
    for i in 1..3
    loop
        pllogger.fine($$PLSQL_UNIT,$$PLSQL_LINE,'i is ' || to_char(i));
    end loop;
    pllogger.close_log_file();
exception when others then
        -- a severe condition is not necessarily fatal
    pllogger.severe($$PLSQL_UINIT,$$PLSQL_LINE,sqlerrm);
    pllogger.close_log_file();
    raise;
end;
/
show errors

exec log_to_file_only();
```

Output

```
"log_level","job_log_id","job_msg_id","line_number","timestamp","log_msg","caller_name","call_stack"
4,,,17,"2019-10-26T17:19:52.885607","begin loop","anonymous",""
4,,,18,"2019-10-26T17:19:52.886020","this is an absurdly long message,  exceeding the length of the l
7,,,22,"2019-10-26T17:19:52.886197","i is 1","LOG_TO_FILE_ONLY",""
7,,,22,"2019-10-26T17:19:52.886357","i is 2","LOG_TO_FILE_ONLY",""
7,,,22,"2019-10-26T17:19:52.886502","i is 3","LOG_TO_FILE_ONLY",""
```

2.2 Database Logging

Record jobs and their steps, how long each step took to execute and optionally extremely detailed information about every database operation as an oracle trace file may be parsed and stored in the log repository.

The log repository may be on the same oracle database server, even the same schema using the same connection as it uses autonomous transactions, or in postgresql or h2.

2.2.1 Database Logging

included file

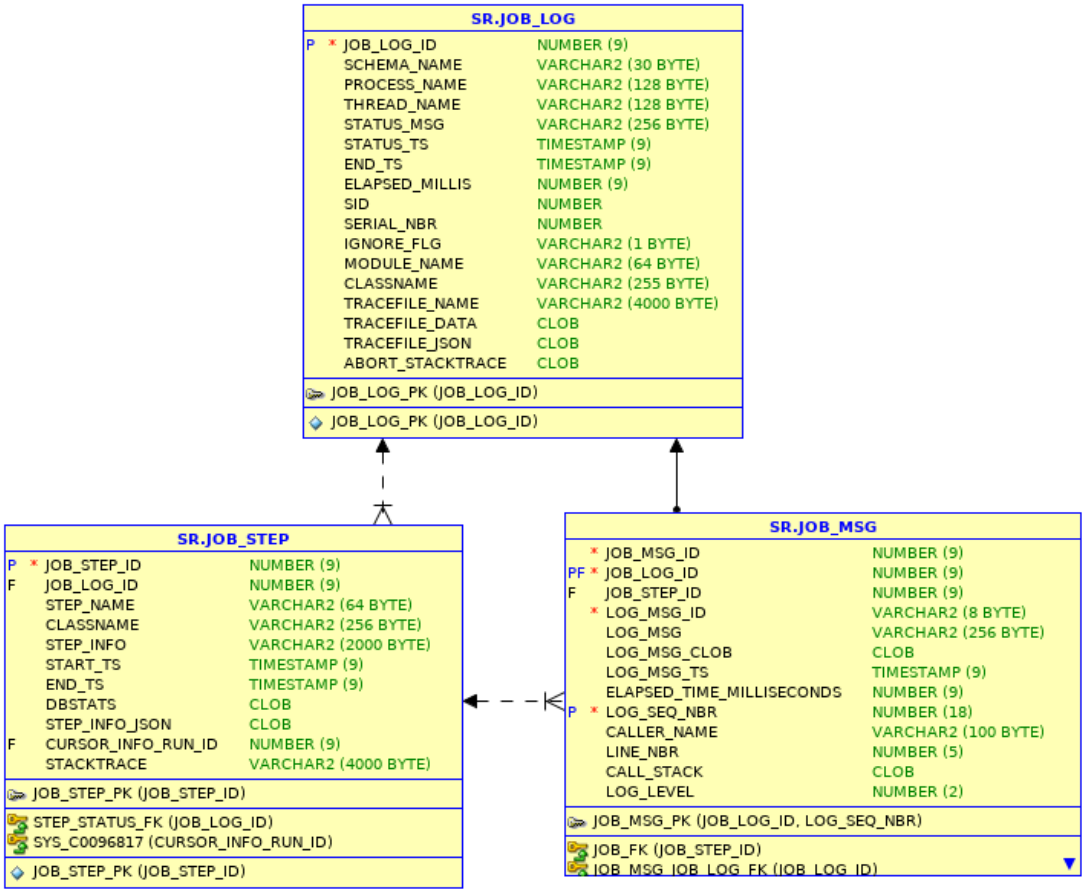
job_tables

In the interest of expediency we have a quick listing of the job tables.

```
SQL> describe job_log
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 JOB_LOG_ID                                NOT NULL NUMBER(9)
 SCHEMA_NAME                                        VARCHAR2(30)
 PROCESS_NAME                                       VARCHAR2(128)
 THREAD_NAME                                        VARCHAR2(128)
 STATUS_MSG                                         VARCHAR2(256)
 STATUS_TS                                          TIMESTAMP(9)
 END_TS                                             TIMESTAMP(9)
 ELAPSED_MILLIS                                     NUMBER(9)
 SID                                                NUMBER
 SERIAL_NBR                                         NUMBER
 IGNORE_FLG                                         VARCHAR2(1)
 MODULE_NAME                                        VARCHAR2(64)
 CLASSNAME                                          VARCHAR2(255)
 TRACEFILE_NAME                                     VARCHAR2(4000)
 TRACEFILE_DATA                                     CLOB
 TRACEFILE_JSON                                     CLOB
 ABORT_STACKTRACE                                   CLOB

SQL> describe job_step
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 JOB_STEP_ID                               NOT NULL NUMBER(9)
 JOB_LOG_ID                                         NUMBER(9)
 STEP_NAME                                          VARCHAR2(64)
 CLASSNAME                                          VARCHAR2(256)
 STEP_INFO                                          VARCHAR2(2000)
 START_TS                                           TIMESTAMP(9)
 END_TS                                             TIMESTAMP(9)
 DBSTATS                                            CLOB
 STEP_INFO_JSON                                     CLOB
 CURSOR_INFO_RUN_ID                                 NUMBER(9)
 STACKTRACE                                         VARCHAR2(4000)

SQL> describe job_msg;
 Name                                      Null?    Type
 ----------------------------------------- -------- ----------------------------
 JOB_MSG_ID                                         NUMBER(9)
 JOB_LOG_ID                                NOT NULL NUMBER(9)
 LOG_MSG_ID                                         VARCHAR2(8)
 LOG_MSG                                            VARCHAR2(256)
 LOG_MSG_CLOB                                       CLOB
 LOG_MSG_TS                                         TIMESTAMP(9)
 ELAPSED_TIME_MILLISECONDS                          NUMBER(9)
 LOG_SEQ_NBR                               NOT NULL NUMBER(18)
 CALLER_NAME                                        VARCHAR2(100)
 LINE_NBR                                           NUMBER(5)
 CALL_STACK                                         CLOB
 LOG_LEVEL                                          NUMBER(2)
```


SR.JOB_LOG
P * JOB_LOG_ID NUMBER (9)
SCHEMA_NAME VARCHAR2 (30 BYTE)
PROCESS_NAME VARCHAR2 (128 BYTE)
THREAD_NAME VARCHAR2 (128 BYTE)
STATUS_MSG VARCHAR2 (256 BYTE)
STATUS_TS TIMESTAMP (9)
END_TS TIMESTAMP (9)
ELAPSED_MILLIS NUMBER (9)
SID NUMBER
SERIAL_NBR NUMBER
IGNORE_FLG VARCHAR2 (1 BYTE)
MODULE_NAME VARCHAR2 (64 BYTE)
CLASSNAME VARCHAR2 (255 BYTE)
TRACEFILE_NAME VARCHAR2 (4000 BYTE)
TRACEFILE_DATA CLOB
TRACEFILE_JSON CLOB
ABORT_STACKTRACE CLOB
JOB_LOG_PK (JOB_LOG_ID)
JOB_LOG_PK (JOB_LOG_ID)
SR.JOB_STEP
P * JOB_STEP_ID NUMBER (9)
F JOB_LOG_ID NUMBER (9)
STEP_NAME VARCHAR2 (64 BYTE)
CLASSNAME VARCHAR2 (256 BYTE)
STEP_INFO VARCHAR2 (2000 BYTE)
START_TS TIMESTAMP (9)
END_TS TIMESTAMP (9)
DBSTATS CLOB
STEP_INFO_JSON CLOB
F CURSOR_INFO_RUN_ID NUMBER (9)
STACKTRACE VARCHAR2 (4000 BYTE)
JOB_STEP_PK (JOB_STEP_ID)
STEP_STATUS_FK (JOB_LOG_ID)
SYS_C0096817 (CURSOR_INFO_RUN_ID)
JOB_STEP_PK (JOB_STEP_ID)
SR.JOB_MSG
* JOB_MSG_ID NUMBER (9)
PF * JOB_LOG_ID NUMBER (9)
F JOB_STEP_ID NUMBER (9)
* LOG_MSG_ID VARCHAR2 (8 BYTE)
LOG_MSG VARCHAR2 (256 BYTE)
LOG_MSG_CLOB CLOB
LOG_MSG_TS TIMESTAMP (9)
ELAPSED_TIME_MILLISECONDS NUMBER (9)
P * LOG_SEQ_NBR NUMBER (18)
CALLER_NAME VARCHAR2 (100 BYTE)
LINE_NBR NUMBER (5)
CALL_STACK CLOB
LOG_LEVEL NUMBER (2)
JOB_MSG_PK (JOB_LOG_ID, LOG_SEQ_NBR)
JOB_FK (JOB_STEP_ID)
JOB_MSG_JOB_LOG_FK (JOB_LOG_ID)

Entity Relationship Diagram

TODO run the python with the comments

Each job has one job_log entry and one or more job_steps.

Job steps may have associated log messages.

dblogger_install_tables

Creates the job and job step tables and views

sequences cursor_info_run_id_seq; cursor_info_id_seq; job_log_id_seq; job_msg_id_seq; job_step_id_seq;

tables

- cursor_explain_plan
- cursor_sql_text
- cursor_info_run
- cursor_info
- cursor_stat
- job_log
- job_msg
- job_step

views

- cursor_info_vw
- job_step_vw
- job_log_vw

Job Logging

Logging information may be written to a text file, stored in a database and written to the oracle trace file.

Steps start job logging.

```
public long sampleUsage(Dblogger dblogger, Connection appConnection) throws SqlSplitterException, Exc
    dblogger.prepareConnection();
    final String processName = "Process Name";
    // Start the job

    final long logJobId = dblogger.startJobLogging(processName,getClass().getName(), null, null,  4);
    dblogger.setModule("SplitLoggerTest", "simple example");
    dblogger.setAction("Some work");
    dblogger.insertStep("Full join", "Meaningless busy work", getClass().getName());
    ConnectionUtil.exhaustQuery(appConnection, "select * from user_tab_columns, user_tables where row
```

```
    dblogger.setAction("Another set of work");
    ConnectionUtil.exhaustQuery(appConnection, "select count(*) from all_tab_columns");
    // End the job
    dblogger.endJob();
    return logJobId;
}
```

job logging persistence has a bit of indirection

Installation

Repositories

RDBMS persistence support is provided for Oracle, H2 and postgresql

H2 is a lightweight database and may be used to eliminate the need for support of another Oracle Database.

Postgresql is a high end database that requires minimimal installation and administration.

You should probable not compound your problem with yet another Oracle install, but if your DBA will allow you a schema in your database for logging, you don't have to learn anything else.

The Oracle database could be the same instance as the application being monitored, but this may raise some objections to the application DBA.

Oracle logging repository

If the logging data is to be persisted in Oracle, the tables must be created and some packages created.

1. job_log
2. job_msg
3. job step

The granularity of job step is left to the invoker.
As the overhead is very low, there is no reason to be parsimonious

with identification, it's a simple one line call in the user app.

These records can be reviewed for job sucess or failure and form a historical basis of time elapsed by job and step.

This may be used as a starting pointing in locating "what processes are using the time?"

Additionally they constitute a base performance metric from which runtime degradation or periodic anomalous runs may be identified.

Data is committed by calls from java to the package logger, provided here.

The package utilizes autonomous commits and hence may be safely called using the same connection as the application.

2.3 Oracle trace information

The third type of logging is an extension of database logging and stores oracle trace information a relational database.

Oracle tracing is turned on and the trace files parsed and aggregated and stored in tables associated with the various job steps.

- oracle
- h2
- postgresql

CHAPTER

THREE

DATABASE LOGS

We will start with an example program and show what is logged.

3.1 Java Example

```
package org.javautil.dblogging;

import java.sql.Connection;
import java.sql.SQLException;

import org.javautil.core.sql.Binds;
import org.javautil.core.sql.ConnectionUtil;
import org.javautil.core.sql.SqlStatement;
import org.javautil.dblogging.logger.Dblogger;
import org.javautil.util.NameValue;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class DbloggerForOracleExample {

    private Dblogger dblogger;
    private Connection connection;
    private String processName;
    private boolean testAbort = false;
    private int traceLevel;
    private final Logger logger = LoggerFactory.getLogger(getClass());

    public DbloggerForOracleExample(Connection connection, Dblogger dblogger, String processName, boo
            int traceLevel) {
        this.connection = connection;
        this.dblogger = dblogger;
        this.processName = processName;
        this.testAbort = testAbort;
        this.traceLevel = traceLevel;
    }

    public long process() throws SQLException {
        dblogger.prepareConnection();
        long id = 0;

        try {
```

```
            id = dblogger.startJobLogging(processName, getClass().getName(), "ExampleLogging", null,
            logger.debug("============================jobId: {}", id);
            limitedFullJoin();
            fullJoin();
            userTablesCount();
            if (testAbort) {
                int x = 1 / 0;
            }
            logger.debug("calling endJob");
            dblogger.endJob();
        } catch (Exception e) {
            logger.error(e.getMessage());
            e.printStackTrace();
            dblogger.abortJob(e);
            throw e;
        }
        return id;
    }

    /**
     * This will set the v$session.action
     */
    private void limitedFullJoin() throws SQLException {
        logger.debug("limitedFullJoin =============");
        dblogger.setAction("actionNoStep");
        ConnectionUtil.exhaustQuery(connection, "select * from user_tab_columns, user_tables where ro
        dblogger.setAction(null);  // no longer performing that action, so clear
        logger.debug("limitedFullJoin complete =============");
    }

    private void fullJoin() throws SQLException {
        logger.debug("fullJoinBegins =============");
        // TODO insertStep should set the action
        dblogger.insertStep("fullJoin", "fullJoin", getClass().getName());
        ConnectionUtil.exhaustQuery(connection, "select * from user_tab_columns, user_tables");
        dblogger.finishStep();
        logger.debug("fullJoin complete =============");
    }

    private void userTablesCount() throws SQLException {
        dblogger.insertStep("count full", "userTablesCount", getClass().getName());
        ConnectionUtil.exhaustQuery(connection, "select count(*) dracula from user_tables");
        dblogger.finishStep();
        // TODO support implicit finish step
    }

    NameValue getJobLog(Connection connection, long id) throws SQLException {
        final String sql = "select * from job_log " + "where job_log_id = :job_stat_id";
        final SqlStatement ss = new SqlStatement(connection, sql);
        Binds binds = new Binds();
        binds.put("job_stat_id", id);
        final NameValue retval = ss.getNameValue();
        ss.close();
        return retval;
    }
}
```

3.2 Analyzing the logs

Separate utilities are used to analyzed the logs. A very useful tool is javautil-condition-identification.

Did any job abort?

What is the trend on elapsed times?

How do elapsed times vary based on time of day?

Getting deeper, with trace information one can drill down to the details, we will cover that later.

CHAPTER

FOUR

TRACEFILE GENERATION AND PERSISTENCE

This utility provides the information to the Oracle Performance specialist to identify the root cause of the problem, how to repair is another speciality.

CHAPTER

FIVE

INSTALLATION OF DATABASE ARTIFACTS FOR ORACLE

These files may be found under *src/main/resources/ddl/oracle*

The script that runs them all is *install.sql*

```
set echo on
@prepare_connection.sql
@my_session_info.sql
@dblogger_uninstall.sr.sql
@logger_message_formatter.plsql.sr.sql
@dblogger_install_tables.sr.sql
@dblogger_install.pks.sr.sql
@dblogger_install.pkb.sr.sql
@logger_persistence.pks.sr.sql
@logger_persistence.pkb.sr.sql
```

5.1 prepare_connnection

prepare-connection provides one procedure.
This will call dbms_session.clear_context for each context variable,

restoring the context for a connection returned from a connection_pool to the state the of an initially opened connection.

Connection pools do not generally clear this information out as it is Oracle specific.

5.2 my_session_info.sql

creates the view *my_session_info* to allow the connected user to obtain the v$session record for the current connection.

5.3 logger_message_formatter

Provides the *logger_message_formatter* function, which creates a single string from all of the logging parameters and makes a call to dbms_output.put_line and then returns the formatted message.

5.4 dblogger_intall_tables

Creates the job and job step tables and views

cursor_info_run_id_seq; cursor_info_id_seq; job_log_id_seq; job_msg_id_seq; job_step_id_seq;

- cursor_explain_plan
- cursor_sql_text
- cursor_info_run
- cursor_info
- cursor_stat
- job_log
- job_msg
- job_step
- cursor_info_vw
- job_step_vw
- job_log_vw

5.5 Job Logging

Logging information may be written to a text file, stored in a database and written to the oracle trace file.

5.5.1 Steps start job logging.

```
public long sampleUsage(Dblogger dblogger, Connection appConnection) throws SqlSplitterException, Exc
    dblogger.prepareConnection();
    final String processName = "Process Name";
    // Start the job

    final long logJobId = dblogger.startJobLogging(processName,getClass().getName(), null, null,  4);
    dblogger.setModule("SplitLoggerTest", "simple example");
    dblogger.setAction("Some work");
    dblogger.insertStep("Full join", "Meaningless busy work", getClass().getName());
    ConnectionUtil.exhaustQuery(appConnection, "select * from user_tab_columns, user_tables where row

    dblogger.setAction("Another set of work");
    ConnectionUtil.exhaustQuery(appConnection, "select count(*) from all_tab_columns");
    // End the job
    dblogger.endJob();
    return logJobId;
}
```

job logging persistence has a bit of indirection

5.6 Installation

5.6.1 Repositories

RDBMS persistence support is provided for Oracle, H2 and postgresql

H2 is a lightweight database and may be used to eliminate the need for support of another Oracle Database.

Postgresql is a high end database that requires minimimal installation and administration.

You should probable not compound your problem with yet another Oracle install, but if your DBA will allow you a schema in your database for logging, you don't have to learn anything else.

The Oracle database could be the same instance as the application being monitored, but this may raise some objections to the application DBA.

5.6.2 Oracle logging repository

If the logging data is to be persisted in Oracle, the tables must be created and some packages created.

Job log tables

1. job_log
2. job_msg
3. job step

The granularity of job step is left to the invoker.
As the overhead is very low, there is no reason to be parsimonious

with identification, it's a simple one line call in the user app.

These records can be reviewed for job sucess or failure and form a historical basis of time elapsed.

This may be used as a starting pointing in locating "what processes are using the time?"

Additionally they constitute a base performance metric from which runtime degradation or periodic anomalous runs may be identified.

Data is committed by calls from java to the package logger, provided here.

The package creates autonomous commits and hence may be safely called using the same connection as the application.

5.6.3 logging package

The logger package provides the following:

```
These primarily set information in the SGA and enable oracle session tracing.
```

begin_java_java

change v$session information

procedure prepare_connection;

set_module set action

5.7 Trace Repository

- cursor_explain_plan
- cursor_sql_text
- cursor_info_run
- cursor_info
- cursor_stat

5.8 logger_persistence package

The logger persistence package provides an API for writing to various tables using autonomous transactions.

```
procedure save_job_log (
    p_job_log_id   in number,
    p_schema_name  in varchar2,
    p_process_name in varchar2,
    p_classname    in varchar2,
    p_module_name  in varchar2,
    p_status_msg   in varchar2,
    p_thread_name  in varchar2,
    p_trace_level  in pls_integer default logger.G_INFO,
    p_tracefile_name in varchar2,
    p_sid          in pls_integer
);
```

The source of work is indentifable down to the java thread.

```
function save_job_step (
    p_job_log_id  in pls_integer,
    p_step_name   in varchar,
    p_step_info   in varchar,
    p_classname   in varchar,
    p_start_ts    in timestamp,
    p_stacktrace  in varchar
```

) return number;

```
procedure finish_step

procedure end_job(p_elapsed_milliseconds in pls_integer)

procedure abort_job(p_elapsed_milliseconds in pls_integer,p_stacktrace in varchar);
```

5.8.1 Install Oracle JDBC

See this post to use Oracle JDBC properly. Or, you could download the JAR file, and then execute this command:

TODO the script to locatge `mvn install:install-file -DgroupId=com.oracle -DartifactId=oracle-jdbc8 -Dversion=12c -Dpackaging=jar -Dfile=<THE_JDBC_JAR_LOCATION>`

Notations in job .sql script used by sqlrunner.

CHAPTER

SIX

SECURITY IN PRODUCTION

CHAPTER

SEVEN

USER PRIVILIGES

CHAPTER

EIGHT

PERFORMANCE

CHAPTER

NINE

TOOLS AND CONCEPTS

User should be familiar with v$ssession view, tkprof command line utility

CHAPTER

TEN

CONNECTION POOLS

10.1 After Getting a connection

10.1.1 Contexts

If a session is being used as part of a connection pool and the state of its contexts are not reinitialized, this can lead to unexpected behavior.

10.1.2 Packages

Sessions have the ability to alter package state by amending the values of package variables. If a session is being used as part of a connection pool and the state of its packages are not reinitialized, this can lead to unexpected behavior. To solve this, Oracle provides the dbms_session.reset_package procedure.

The dbloggging provided procedure clears all context variables and resets package state.

Connections must be reset immediately after being obtained from a connection pool

In src/main/resources/ddl/oracle/prepare_connection

10.1.3 Convenience Procedure

```
CREATE OR REPLACE PROCEDURE prepare_connection
AS
    context_info DBMS_SESSION.AppCtxTabTyp;
    info_count PLS_INTEGER;
    indx PLS_INTEGER;
BEGIN
    DBMS_SESSION.LIST_CONTEXT ( context_info, info_count);
    indx := context_info.FIRST;
    LOOP
        EXIT WHEN indx IS NULL;
        DBMS_SESSION.CLEAR_CONTEXT(
            context_info(indx).namespace,
            context_info(indx).attribute,
            null
        );
        indx := context_info.NEXT (indx);
    END LOOP;
    DBMS_SESSION.RESET_PACKAGE;
END;
```

create public synonym prepare_connection for prepare_connection; grant execute on prepare_connection to public; ‘‘‘

10.1.4 Zaxxer

TODO how to call this procedure in the connection pool

10.2 ## DBMS_SESSION

10.3 Identifier

SET_IDENTIFIER and CLEAR_IDENTIFIER procedures to allow the real user to be associated with a session, regardless of what database user was being used for the connection.

10.4 Metrics

try { String e2eMetrics[] = new String[OracleConnection.END_TO_END_STATE_INDEX_MAX]; e2eMetrics[OracleConnection.END_TO_END_ACTION_INDEX] = null; e2eMetrics[OracleConnection.END_TO_END_MODU = null; e2eMetrics[OracleConnection.END_TO_END_CLIENTID_INDEX] = null; ((OracleConnection) conn).setEndToEndMetrics(e2eMetrics, Short.MIN_VALUE);

} catch (SQLException sqle) { // Do something...

}

0 - No trace. Like switching sql_trace off. 2 - The equivalent of regular sql_trace. 4 - The same as 2, but with the addition of bind variable values. 8 - The same as 2, but with the addition of wait events. 12 - The same as 2, but with both bind variable values and wait events.

Monitoring long running https://oracle-base.com/articles/11g/real-time-sql-monitoring-11gr1

Database Objects

Entity Relationship Diagram logger_tables.png

Table DDL

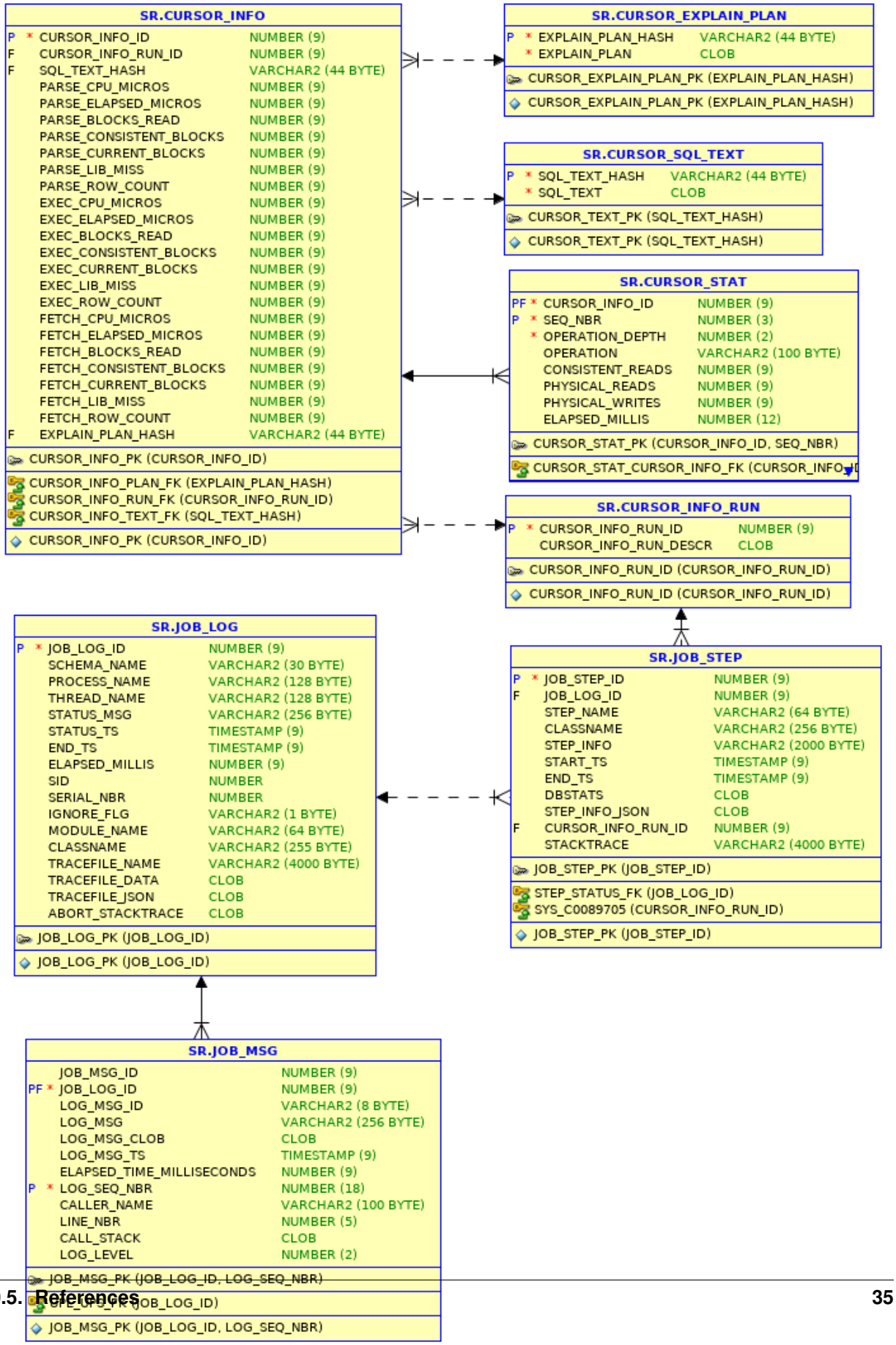
Oracle

10.5 References

!![javadoc] (‘../target/site/apidocs/index.html’)

https://oracle-base.com/articles/misc/dbms_session

https://oracle-base.com/articles/misc/sql-trace-10046-trcsess-and-tkprof


SR.CURSOR_INFO
P * CURSOR_INFO_ID NUMBER (9)
F CURSOR_INFO_RUN_ID NUMBER (9)
F SQL_TEXT_HASH VARCHAR2 (44 BYTE)
PARSE_CPU_MICROS NUMBER (9)
PARSE_ELAPSED_MICROS NUMBER (9)
PARSE_BLOCKS_READ NUMBER (9)
PARSE_CONSISTENT_BLOCKS NUMBER (9)
PARSE_CURRENT_BLOCKS NUMBER (9)
PARSE_LIB_MISS NUMBER (9)
PARSE_ROW_COUNT NUMBER (9)
EXEC_CPU_MICROS NUMBER (9)
EXEC_ELAPSED_MICROS NUMBER (9)
EXEC_BLOCKS_READ NUMBER (9)
EXEC_CONSISTENT_BLOCKS NUMBER (9)
EXEC_CURRENT_BLOCKS NUMBER (9)
EXEC_LIB_MISS NUMBER (9)
EXEC_ROW_COUNT NUMBER (9)
FETCH_CPU_MICROS NUMBER (9)
FETCH_ELAPSED_MICROS NUMBER (9)
FETCH_BLOCKS_READ NUMBER (9)
FETCH_CONSISTENT_BLOCKS NUMBER (9)
FETCH_CURRENT_BLOCKS NUMBER (9)
FETCH_LIB_MISS NUMBER (9)
FETCH_ROW_COUNT NUMBER (9)
F EXPLAIN_PLAN_HASH VARCHAR2 (44 BYTE)
CURSOR_INFO_PK (CURSOR_INFO_ID)
CURSOR_INFO_PLAN_FK (EXPLAIN_PLAN_HASH)
CURSOR_INFO_RUN_FK (CURSOR_INFO_RUN_ID)
CURSOR_INFO_TEXT_FK (SQL_TEXT_HASH)
CURSOR_INFO_PK (CURSOR_INFO_ID)
SR.CURSOR_EXPLAIN_PLAN
P * EXPLAIN_PLAN_HASH VARCHAR2 (44 BYTE)
* EXPLAIN_PLAN CLOB
CURSOR_EXPLAIN_PLAN_PK (EXPLAIN_PLAN_HASH)
CURSOR_EXPLAIN_PLAN_PK (EXPLAIN_PLAN_HASH)
SR.CURSOR_SQL_TEXT
P * SQL_TEXT_HASH VARCHAR2 (44 BYTE)
* SQL_TEXT CLOB
CURSOR_TEXT_PK (SQL_TEXT_HASH)
CURSOR_TEXT_PK (SQL_TEXT_HASH)
SR.CURSOR_STAT
PF * CURSOR_INFO_ID NUMBER (9)
P * SEQ_NBR NUMBER (3)
* OPERATION_DEPTH NUMBER (2)
OPERATION VARCHAR2 (100 BYTE)
CONSISTENT_READS NUMBER (9)
PHYSICAL_READS NUMBER (9)
PHYSICAL_WRITES NUMBER (9)
ELAPSED_MILLIS NUMBER (12)
CURSOR_STAT_PK (CURSOR_INFO_ID, SEQ_NBR)
CURSOR_STAT_CURSOR_INFO_FK (CURSOR_INFO_
SR.CURSOR_INFO_RUN
P * CURSOR_INFO_RUN_ID NUMBER (9)
CURSOR_INFO_RUN_DESCR CLOB
CURSOR_INFO_RUN_ID (CURSOR_INFO_RUN_ID)
CURSOR_INFO_RUN_ID (CURSOR_INFO_RUN_ID)
SR.JOB_LOG
P * JOB_LOG_ID NUMBER (9)
SCHEMA_NAME VARCHAR2 (30 BYTE)
PROCESS_NAME VARCHAR2 (128 BYTE)
THREAD_NAME VARCHAR2 (128 BYTE)
STATUS_MSG VARCHAR2 (256 BYTE)
STATUS_TS TIMESTAMP (9)
END_TS TIMESTAMP (9)
ELAPSED_MILLIS NUMBER (9)
SID NUMBER
SERIAL_NBR NUMBER
IGNORE_FLG VARCHAR2 (1 BYTE)
MODULE_NAME VARCHAR2 (64 BYTE)
CLASSNAME VARCHAR2 (255 BYTE)
TRACEFILE_NAME VARCHAR2 (4000 BYTE)
TRACEFILE_DATA CLOB
TRACEFILE_JSON CLOB
ABORT_STACKTRACE CLOB
JOB_LOG_PK (JOB_LOG_ID)
JOB_LOG_PK (JOB_LOG_ID)
SR.JOB_STEP
P * JOB_STEP_ID NUMBER (9)
F JOB_LOG_ID NUMBER (9)
STEP_NAME VARCHAR2 (64 BYTE)
CLASSNAME VARCHAR2 (256 BYTE)
STEP_INFO VARCHAR2 (2000 BYTE)
START_TS TIMESTAMP (9)
END_TS TIMESTAMP (9)
DBSTATS CLOB
STEP_INFO_JSON CLOB
F CURSOR_INFO_RUN_ID NUMBER (9)
STACKTRACE VARCHAR2 (4000 BYTE)
JOB_STEP_PK (JOB_STEP_ID)
STEP_STATUS_FK (JOB_LOG_ID)
SYS_C0089705 (CURSOR_INFO_RUN_ID)
JOB_STEP_PK (JOB_STEP_ID)
SR.JOB_MSG
JOB_MSG_ID NUMBER (9)
PF * JOB_LOG_ID NUMBER (9)
LOG_MSG_ID VARCHAR2 (8 BYTE)
LOG_MSG VARCHAR2 (256 BYTE)
LOG_MSG_CLOB CLOB
LOG_MSG_TS TIMESTAMP (9)
ELAPSED_TIME_MILLISECONDS NUMBER (9)
P * LOG_SEQ_NBR NUMBER (18)
CALLER_NAME VARCHAR2 (100 BYTE)
LINE_NBR NUMBER (5)
CALL_STACK CLOB
LOG_LEVEL NUMBER (2)
JOB_MSG_PK (JOB_LOG_ID, LOG_SEQ_NBR)
JOB_MSG_PK (JOB_LOG_ID, LOG_SEQ_NBR)

CHAPTER

ELEVEN

SPRING DEVELOPERS

Oracle tracing is a powerful tool that logs detailed information about all calls to the Oracle database.

In order to use this :

- one must turn on tracing for the current connection
- set the log file
- stop tracing
- call a service to store the trace
- store the raw trace file
- analyze the trace file
- store the analyzed trace file

Logs messages using utl_file to a directory on the database server specified.

First the database directory is created and oracle is granted permission to read and write it, then the ddl “create directory....” and “grant read, write on directory...”

```
set serveroutput on
set echo on
create or replace procedure log_to_file_only is
       long_msg clob := 'this is an absurdly long message, ' ||
                ' interesting stuff to say so I will just write meaningless ' ||
                ' stuff for a little while. ' ||
                ' The quick brown fox jumped over the lazy dog. ';

    my_log_file_name varchar(4096);
begin
    my_log_file_name := pllogger.open_log_file('log_to_file_only.text');
    pllogger.set_filter_level(9); -- all messages should go to log file
    pllogger.info('anonymous',$$PLSQL_LINE,'begin loop');
    pllogger.info($$PLSQL_UNIT,$$PLSQL_LINE,long_msg);
    for i in 1..3
    loop
        pllogger.fine($$PLSQL_UNIT,$$PLSQL_LINE,'i is ' || to_char(i));
    end loop;
    pllogger.close_log_file();
exception when others then
        -- a severe condition is not necessarily fatal
    pllogger.severe($$PLSQL_UINIT,$$PLSQL_LINE,sqlerrm);
    pllogger.close_log_file();
    raise;
```

```
end;
/
show errors

exec log_to_file_only();

"log_level","job_log_id","job_msg_id","line_number","timestamp","log_msg","caller_name","call_stack"
4,,,17,"2019-10-26T17:19:52.885607","begin loop","anonymous",""
4,,,18,"2019-10-26T17:19:52.886020","this is an absurdly long message,  exceeding the length of the l
7,,,22,"2019-10-26T17:19:52.886197","i is 1","LOG_TO_FILE_ONLY",""
7,,,22,"2019-10-26T17:19:52.886357","i is 2","LOG_TO_FILE_ONLY",""
7,,,22,"2019-10-26T17:19:52.886502","i is 3","LOG_TO_FILE_ONLY",""
```

CHAPTER

TWELVE

TODO

Tracing should do the following

- Begin with any transaction as annotated by @Transactional

CHAPTER

THIRTEEN

INSTALL

```
cd src/main/resources/ddl/oracle

sqlplus $ORACLE_UID @ pllogger.pkgs.sr.sql
sqlplus $ORACLE_UID @ pllogger.pkgb.sr.sql
```

create directory job_msg_dir as '/common/scratch/ut_process_log_dir'; grant write on directory to sr;

should be granted by user, not by role.

- Configuring to use your database
- Example schema

CHAPTER

FOURTEEN

TRACE FILE FIELDS

14.1 Trace Record Fields

code-block:

		Cursor Operation									
	Type	Parsing	ParseError	Parse	EXec	Fetch	Stat	Lobread	Lobpgsize	Close	
#	cursorNumbe		X		X	X	X				
ad	sgaAddress	X									
bytes	bytes							X	X		
c	CpuMicroSec			X	X	X		X	X	X	
card	cardinality										
cnt							O				
cost	cost (optim										
cr	consistentR			X	X	X		X	X		
cu	currentMode			X	X	X		X	X		
dep	depth	X	X	X	X	X				X	
e	elapsedMicr			X	X	X		X	X	X	
err	oracleError		X								
hv	sqlHashValu	X									
id							X				
len	sqlTeXtLeng	X	X								
lid		X	X								
mis	libraryCach			X	X	X					
obj	objectNumbe						X				
oct	oracleComma	X	X								
og	optimizerGo			X	X	X					
op	operation						X				
p	physicalBlo			X	X	X		X	X		
pid	processId						X				
plh				X	X	X					
pos	position (o						X				
pw	physicalWri						X				
r	rowCount			X	X	X					
size							O				
sqlid	sqlId	X									
str								X			
tim				X	X	X		X	X		X
time											
type											
uid		X	X								

CHAPTER

FIFTEEN

TODO

- security can't specify file name
- need an agent to get the log files for remote users
- TODO escape double quotes in text fields
- check for anomolous run-times using condition identification
- plot runtimes
- TODO describe microservices, multiple connections, tying them all together
- TODO describe using with spring