ORACLE PIPES

NYOUG, DBA SIG, Rachel Carmichael, Chair

session on 5/17/2000

Presenter: Henry Pikner (Pikner747@cs.com)

AGENDA

- Sources of this information
- Pipes: musicians' and engineers' favorite things
- ORACLE pipes: basic properties
- Package DBMS_PIPE, its functions and procedures: CREATE_PIPE, RESET_BUFFER, PACK_MESSAGE, SEND_MESSAGE, RECEIVE_MESSAGE, UNPACK_MESSAGE, PURGE, REMOVE_PIPE
- First example: Disabling the restart of a failed job
- Description of pipes in ORACLE directory
- Second example: Processing of application messages (logging, real-time monitoring from a parallel session)
- Pipe (self-)policing, pipe administration
- ORACLE8 (better?) alternative: DBMS_AQ, DBMS_AQADM

SOURCES

Web site *technet.oracle.com*, search for “session AND pipe”

Feuerstein, Dye, Beresniewicz: Oracle Built-in Packages, published by O’Reilly in April 1998

Feuerstein: “Advanced Queuing with Oracle AQ“, a series of six articles in *Oracle Developer*,May - October 1998

Pipes: musicians' and engineers' favorite things

Origin of the word**:** **Latin** ***pipare***, meaning *to cheep, chirp, peep, of echoic origin* (New Word Dictionary, 2nd College Edition)

That dictionary lists **a dozen meanings**, among them:

1. cylindrical tube, as of reed, straw, wood, or metal, into which air is blown ...
a)panpipe, b)bagpipe, organ pipe

6. a long tube of clay, concrete, metal, wood, etc., for conveying water, gas, oil, etc. or for use in construction

7. a tubular organ or canal of the body; esp., the respiratory organs

10. a) a tube with a small bowl at one end, in which tobacco, etc. is smoked

11. a) a large cask for wine, oil, etc., having a capacity of about two hogsheads, or 126 gallons,...

Close in meaning **to** ***tube***:

5. a) a tunnel through rock, under water, etc. for railroad, subway, etc....

6. *Bot.* the lower, united part of a gamopetallous corolla or a gamosepallous calyx

Without pipes there would be no modern cities:
water pipes, sewage pipes, gas pipes, subway (tube)

Many manufacturing facilities are a tangle of pipes:
oil refineries, pharmaceutical plants, etc.

Car paradigm:

At a gas station: Gas pump hose, air pump hose
Under (or inside) chassis: gas line, exhaust pipe
Under the hood: coolant pipes, power brake, steering

In information technology: a memory structure with FIFO access

ORACLE pipes: basic properties

ORA pipe is a memory structure in SGA (System Global Area)

ORA pipe conveys messages between sessions, on the same instance

The sending and receiving sessions can be active at the same time, OR hours/days apart

For interaction with pipes, a session uses one message buffer, in PGA

Pipes can be PUBLIC or PRIVATE; they have names

Pipes are fast (compared with DB tables)

Pipes are not subject to transaction rules (message sent in a transaction that was rolled back will reach its destination)

Application developers use pipes by invoking procedures and functions of package DBMS_PIPE; they need EXECUTE privilege on that package

DBA’s added worries: memory contention, clogged pipes, illegal pipe tapping

Key to happiness (as always): MODERATION

Some functions and procedures in DBMS_PIPE (in a typical order of appearance in applications)

```
INTEGER Function CREATE_PIPE(pipename
                             ,maxpipesize .. DEFAULT 8192
                             ,private .. DEFAULT TRUE)

Procedure RESET_BUFFER

Procedure PACK_MESSAGE (item)

INTEGER Function SEND_MESSAGE(pipename
                              ,timeout .. DEFAULT MAXWAIT
                              ,maxpipesize .. DEFAULT 8192)

INTEGER Function RECEIVE_MESSAGE(pipename
                              ,timeout .. DEFAULT MAXWAIT )
        Note: 0=received msg, 1=timed out, i.e. no msg, 2 .., 3..

Procedure UNPACK_MESSAGE (item)

Procedure PURGE(pipename)

INTEGER Function REMOVE_PIPE(pipename)

String Function UNIQUE_SESSION_NAME
(useful for SASE requests)
```

First example: Disabling the restart of a failed job

SQL*Plus script: Submitted job will restart on failure

```
variable jn number
execute dbms_job.submit( -
 :jn -
,   'BEGIN ' -
  ||'  aac_main(1234);'-
  ||'END;' -
,sysdate -
,null )
print jn
```

SQL*Plus script: Submitted job, with piped message, will still restart on failure

```
variable jn number
execute dbms_job.submit( -
 :jn -
,   'BEGIN ' -
  ||'  dbms_pipe.reset_buffer; dbms_pipe.pack_message(''X'');' -
  ||'  IF 0<=dbms_pipe.send_message(''QJ''||job,10,10) THEN null; END IF;' -
  ||'  aac_main(1234);'-
  ||' dbms_pipe.purge(''QJ''||job);' -
  ||' IF 0<=dbms_pipe.remove_pipe(''QJ''||job) THEN null; END IF;' -
  ||'END;' -
,sysdate -
,null )
print jn
```

SQL*Plus script: Submitted job, will no longer repeat aac_main on failure

```
variable jn number
execute dbms_job.submit( -
 :jn -
,   'BEGIN ' -
  ||' IF 0<=dbms_pipe.create_pipe(''QJ''||job, 10) THEN null; END IF;'-
   ||' IF 1 =dbms_pipe.receive_message(''QJ''||job,0) THEN ' -
  ||'  dbms_pipe.reset_buffer; dbms_pipe.pack_message(''X'');' -
  ||'  IF 0<=dbms_pipe.send_message(''QJ''||job,10,10) THEN null; END IF;' -
  ||'  aac_main(1234);'-
  ||' END IF;' -
  ||' dbms_pipe.purge(''QJ''||job);' -
  ||' IF 0<=dbms_pipe.remove_pipe(''QJ''||job) THEN null; END IF;' -
  ||'END;' -
,sysdate -
,null )
print jn
```

Description of pipes in ORACLE directory

View V$DB_PIPES

```
OWNERID  NUMBER        Owner ID, for private pipes
                       Null, for public pipes

NAME     VARCHAR2(1000) Name of the pipe, e.g.
                       scott.pn

TYPE     VARCHAR2(7)   PUBLIC or PRIVATE

SIZE     NUMBER        Amount of memory (in bytes)
                       used by the pipe
```

Second example: Application message logging / monitoring

A (long) running application needs to tell
  the business owner
  the DBA
  the supporter
about events during the execution:
  fatal crashes (ERROR exceptions, computer/business)
  near misses (WARNING exceptions, computer/business)
  no news - good news (INFO messages: resources consumed, options used, business rules waived, etc.)

What do application developers do now?
In good shops, they call a stored procedure/function (provided by DBA) like:

```
DBC_MSG(msg_text VARCHAR2 IN
         , severity VARCHAR2 IN DEFAULT 'INFO'
        , job_id  VARCHAR2 IN DEFAULT NULL)
```

How does DBC_MSG put the message out, and where to?

by INSERT into a database table; disadvantages: I/O needed, an eventual ROLLBACK will roll it back; available to be queried only after COMMIT.

OR, by calling DBMS_OUTPUT.PUT_LINE into session's buffer; disadvantages: The message belongs to the session that produced it and sits in its PGA memory. Is such a thing ENABLEd? Is the memory large enough? Why don't I see the messages in real time? What happens to the messages in a batch execution - e.g. when run in a job queue?

OR, by calling UTL_FILE into an OS file; disadvantages: non-SQL tools are needed to read the. messages. OS administrators are apprehensive.

DBF_MSG can do it with a Pipe !

by calling DBMS_PIPE.SEND_MESSAGE (preceded by other calls, among them RESET BUFFER,
  and a few PACK_MESSAGEs)

Yes, the application could also do it with DBMS_AQ, but this is a Pipe presentation. Now, back to Pipes.

Application message logging / monitoring (cont)

Components of the solution:

DB table (named here DBA.DBT_MSG_LOG)

Stored procedure (named here DBA.DBC_MSG) -- called from applications

Stored procedure (named here DBA.DBC_MSG_RCV) -- called from message monitoring queries,
and/or run as a background daemon

A query that reads DBT_MSG_LOG

Layout of DBT_MSG_LOG

send_time	DATE
send_clock_100th	INTEGER
send_user	VARCHAR2(18)
send_stack	VARCHAR2(200)
send_sid	INTEGER
send_serial#	INTEGER
send_osuser	VARCHAR2(18)
msg_text	VARCHAR2(2000) NOT NULL
severity	VARCHAR2(7) NOT NULL
job_id	INTEGER DEFAULT NULL

SQL*Plus script that reads the log table:

```
execute DBA.DBC_MSG_RCV(0)

SELECT send_time, msg_text, severity
FROM dba.dbt_msg_log
WHERE ...
ORDER BY send_time desc
```

Application message logging / monitoring (cont.)

Creation of DBA.DBC_MSG
Caution: this is an untested sketch!

```
CREATE OR REPLACE PROCEDURE  dba.dbc_msg(msg_text VARCHAR2 IN
                        , severity VARCHAR2 IN DEFAULT ‘INFO’
                        , job_id  VARCHAR2 IN DEFAULT NULL)
RETURN INTEGER IS
v_pipename VARCHAR2(32) := ‘DBI_APPL_MSG’;
v_timeout INTEGER:=10;  -- seconds, not 1000 days
v_maxpipesize INTEGER:=8192;  --bytes

BEGIN
  dbms_pipe.reset_buffer;
  dbms_pipe.pack_message(TO_CHAR(sysdate,’SYYYYMMDDHH24MISS’));
  dbms_pipe.pack_message(TO_CHAR(dbms_utility.get_time));
  dbms_pipe.pack_message(USER);
  dbms_pipe.pack_message(
                      dba.dbf_stack_compress(dbms_utility.format_call_stack));
  FOR sc IN (SELECT * FROM v$session WHERE audsid=USERENV(‘SESSIONID’)) LOOP
    dbms_pipe.pack_message(TO_CHAR(sc.sid));
    dbms_pipe.pack_message(TO_CHAR(sc.serial#));
    dbms_pipe.pack_message(sc.osuser);
  END LOOP; -- sc, just one pass
  dbms_pipe.pack_message(msg_text);
  dbms_pipe.pack_message(severity);
  dbms_pipe.pack_message(job_id);
  v_rslt:= dbms_pipe.send_message(v_pipename, v_timeout, v_maxpipesize)
  IF 0 <> v_rslt THEN
      -- see why, perhaps retry once or twice with larger pipe size
  END IF;
EXCEPTION
WHEN OTHERS THEN
  raise_application_error (-20998, SQLERRM);
END;
```

Calls from a PL/SQL application:

```
BEGIN dba.dbc_msg(‘Start’); ...
      dba.dbc_msg(‘Withdrawing $2000 in cash’,’WARNING’); ...
      dba.dbc_msg(‘Insert on AAT_NM failed: ‘||SQLERRM, ‘ERROR’); ...
      dba.dbc_msg(‘End’, ‘INFO’); ...
END;
```

Creation of DBA.DBC_MSG_RCV
Caution: This is an untested sketch

```
CREATE OR REPLACE PROCEDURE  dba.dbc_msg_rcv( p_timeout IN INTEGER)
IS
m_send_time             VARCHAR2(17);
m_send_clock_100th      VARCHAR2(38)      ;
m_send_user             VARCHAR2(18);
m_send_stack            VARCHAR2(200);
m_send_sid              VARCHAR2(38);
m_send_serial#          VARCHAR2(38);
m_send_osuser           VARCHAR2(18);
m_msg_text              VARCHAR2(2000) ;
m_severity              VARCHAR2(7);
m_job_id                VARCHAR2(38);
v_pipename              VARCHAR2(32) := ‘DBI_APPL_MSG’;
v_rslt      INTEGER ;
v_ins_count INTEGER := 0;
v_waiting   INTEGER := -1; --accumulated wait time after last received message
v_waiting_limit INTEGER := LARGEST(0, p_timeout);
v_timeout INTEGER:=LEAST(p_timeout, 20);  -- seconds, not 1000 days
v_time_cutoff DATE := sysdate;
v_send_time DATE := sysdate - 1000; -- later: last seen send time
BEGIN
  IF v_waiting_limit > 0 THEN
    v_time_cutoff := sysdate + 1000; -- practically no cutoff on send_time
  END IF;
  WHILE v_waiting < v_waiting_limit AND v_send_time <= v_time_cutoff LOOP
    v_rslt:= dbms_pipe.receive_message(v_pipename, v_timeout);
    IF 0 = v_rslt THEN -- we got a message, let’s unpack it and store it
      dbms_pipe.unpack_message(m_send_time);
      dbms_pipe.unpack_message(m_send_clock_100th);
      dbms_pipe.unpack_message(m_send_user);
      dbms_pipe.unpack_message(m_send_stack);
      dbms_pipe.unpack_message(m_send_sid);
      dbms_pipe.unpack_message(m_send_serial#));
      dbms_pipe.unpack_message(m_send_osuser);
      dbms_pipe.unpack_message(m_msg_text);
      dbms_pipe.unpack_message(m_severity);
      dbms_pipe.unpack_message(m_job_id);
      INSERT INTO DBA.DBT_MSG_LOG
      VALUES(v_send_time, TO_NUMBER(m_send_clock_100th), m_send_user,
            m_send_stack, TO_NUMBER(m_send_sid), TO_NUMBER(m_send_serial#,
            m_send_osuser, m_msg_text, m_severity, TO_NUMBER(m_jobid)
        );
      v_ins_count:=v_ins_count + 1;

      v_send_time:= TO_DATE(m-send_time,'SYYYYMMDDHH24MISS');
      IF v_ins_count >= 100 THEN
        COMMIT;
        v_ins_count:= 0;
      END IF;
      v_waiting:=0;  -- start the accumulation from scratch
      v_timeout:=0;  -- next attempted receive without waiting
     ELSE -- other than 0: 1=timeout, 3=interrupted, 2=buffer too small?
      IF v_ins_count > 0 THEN
        COMMIT;
        v_ins_count:= 0;
      END IF; -- any inserts done
      v_waiting:=v_waiting + v_timeout;
      v_timeout:=LEAST(p_timeout, 20);
    END IF;  -- values of v_rslt
  END LOOP; -- WHILE
  IF v_ins_count > 0 THEN
      COMMIT;
  END IF; -- any inserts done

EXCEPTION
WHEN OTHERS THEN
      raise_application_error (-20999, SQLERRM);

END;
```

Pipe (self-)policing, pipe administration

Make sure that message sender, message receiver, and the DBA use pipes by following some good protocols (perhaps company-wide standards?). Write the rules down. Topics: pipe naming, message volumes, frequencies, sizing of the pipes. Make sure that pipe names are valid across restarts

Pay attention to timeout values (how about waiting 1000 days for a message?)

Pay attention to exceptions

Collect garbage regularly: remove empty pipes, investigate constipated pipes - full but with no movement

Make sure that a listener daemon is alive for pipes that require it.
[1]
Create company-wide library of procedures, e.g. for (1)message packing and sending, (2)message receiving, unpacking and storing

Itching to ban the use of pipes? Remember the Prohibition!

Key to success: KISS (Keep It Simple, Sister)

Key to happiness: MODERATION
[2]

DBMS_AQ: sometimes a better alternative

AQ stands for Advanced (Message) Queuing

Available first in 8.0.3 (some features in 8.0.4)

For more, see the F&D&B book, or F's articles in ORACLE Developer (now ... Professional)

To use it fully, you need Enterprise Edition with Objects Option.
No Enterprise Edition? No AQ. No Objects Option? No Full AQ.

The Pipe Example 2 (application message logging) was developed under 7.3.4. The presenter would now probably use the Full AQ instead (if available).

Is AQ too complex? Life often is.

Would you prefer to KISS? Do it with a Pipe.

The Pipe Example 1 (preventing queued job restart): do it with a Pipe.