TECHNICAL NOTE

PEOPLESOFT RUN CONTROL PURGE UTILITY

Prepared By David Kurtz, Go-Faster Consultancy Ltd.

Technical Note

Version 1.0

Thursday 26 August 2010

(E-mail: david.kurtz@go-faster.co.uk, telephone +44-7771-760660)

File: Run Control Purge.doc, 26 August 2010

Contents

Introduction

Run Control records are used to pass parameters into processes scheduled processes. These tables tend to grow, and are rarely purged. When operator accounts are deleted, the Run Controls remain, but are no longer accessible to anyone else.

I have worked on systems where new Run Controls, whose IDs contain either a date or sequence number, are generated for each process. The result is that the Run Control tables, especially child tables, grow quickly and if not regularly managed will become very large. On one system, I found 18 million rows on one table!

```
RECNAME         FIELDNAME             NUM_ROWS      BLOCKS
--------------- ------------------ ---------- ----------
TL_RUN_CTRL_GRP RUN_CNTL_ID          18424536     126377
AEREQUESTPARM   RUN_CNTL_ID           1742676      19280
AEREQUESTTBL    RUN_CNTL_ID            333579       3271
XPQRYRUNPARM    RUN_CNTL_ID            121337       1630
TL_TA_RUNCTL    RUN_CNTL_ID            112920        622
…
```

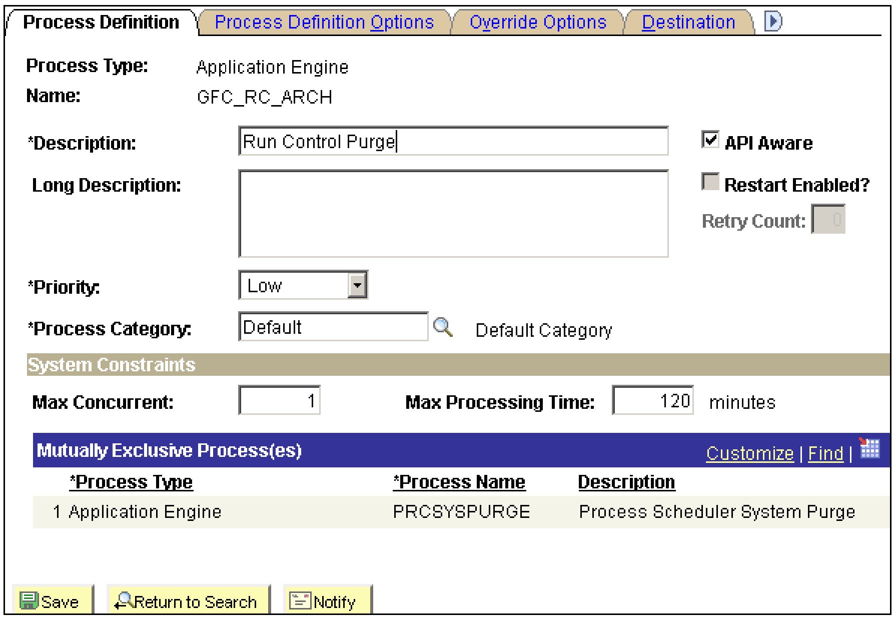
I have written a simple Application Engine process, GFC_RC_ARCH, that purges old Run Controls from these tables.

Run Control records are easily identified. They are characterised by:

- the first column of these tables is always OPRID, and the second is either RUNCNTLID or RUN_CNTL_ID,
- these two columns are also the first two columns of the unique key,
- the Run Control tables appear on pages of components that are declared as the process security component for that process.

I have decided that if the combination of OPRID and RUN_CNTL_ID does not appear in the process scheduler request table, PSPRCSRQST, then the Run Control record should be deleted. Thus, as the delivered Process Scheduler Purge process, PRCSPURGE, deletes rows from the Process Scheduler tables, so my purge process will delete rows from the Run Control tables.

I have chosen to make these two Application Engine processes mutually exclusive, so the Process Scheduler will not run both at the same time, but that configuration cannot be delivered in an Application Designer project.



Process Type: Application Engine
Name: GFC_RC_ARCH
*Description: Run Control Purge
☑ API Aware
☐ Restart Enabled?
Retry Count: 0
Long Description:
*Priority: Low
*Process Category: Default Default Category

System Constraints

Max Concurrent: 1 Max Processing Time: 120 minutes

Mutually Exclusive Process(es)

	*Process Type	*Process Name	Description
1	Application Engine	PRCSYSPURGE	Process Scheduler System Purge

How does it work?

This query identifies the run control records that need to be purged.

```
%Select(RECNAME, FIELDNAME, TABLE_NAME, NUMROWS)
 SELECT DISTINCT r.recname
 , f2.fieldname
 , t.table_name
 , t.num_rows
  FROM psrecdefn r
  , psrecfielddb f1
  , psrecfielddb f2
  , pspnlfield f
  , pspnlgroup g
  , ps_prcsdefnpnl p
  , user_tables t
  WHERE r.rectype = 0[1]
  AND f1.recname = r.recname
  AND f1.fieldnum = 1
  AND f1.fieldname = 'OPRID'[2]
  AND MOD(f1.useedit,2) = 1[3]
  AND f2.recname = r.recname
  AND f2.fieldnum = 2
```

[1] The record to be purged should be a table

[2] The first field should be OPRID

[3] The first field should be a part of the unique key

```
  AND f2.fieldname IN('RUNCNTLID','RUN_CNTL_ID')[4]
  AND MOD(f2.useedit,2) = 1[5]
  AND f.recname = r.recname[6]
  AND f.pnlname = g.pnlname
  AND g.pnlgrpname = p.pnlgrpname[7]
  and t.table_name = DECODE(r.sqltablename,'
','PS_'||r.recname,r.sqltablename)
  and t.num_rows > 0[8]
  ORDER BY t.num_rows DESC
```

Then, for each record that the above query identifies, the following statement is run.

```
DELETE
  FROM %Bind(TABLE_NAME,NOQUOTES)
 WHERE NOT (OPRID, %Bind(FIELDNAME,NOQUOTES)) IN (
 SELECT DISTINCT OPRID
 , RUNCNTLID
  FROM PSPRCSRQST
 WHERE runstatus != '2' )
   AND ROWNUM <= 200000
```

Application Engine dynamically expands the statement for each table, in the case of TL_RUN_CNTL_GRP it becomes

```
DELETE
  FROM PS_TL_RUN_CNTL_GRP
 WHERE NOT (OPRID, RUN_CNTL_GRP) IN (
 SELECT DISTINCT OPRID
 , RUNCNTLID
  FROM PSPRCSRQST
 WHERE runstatus != '2' )
   AND ROWNUM <= 200000
```

Note, there are two Oracle specific constructions in use that must be changed if this process is going to work on other platforms.

- ROWNUM is used to restrict the number of rows that can be deleted in one execution. I arbitarily chose a value of 200,000 rows, to restrict the amount of data copied into the rollback segment.

[4] The second field should be either RUNCNTLID or RUN_CNTL_ID

[5] The second field should be a part of the unique key

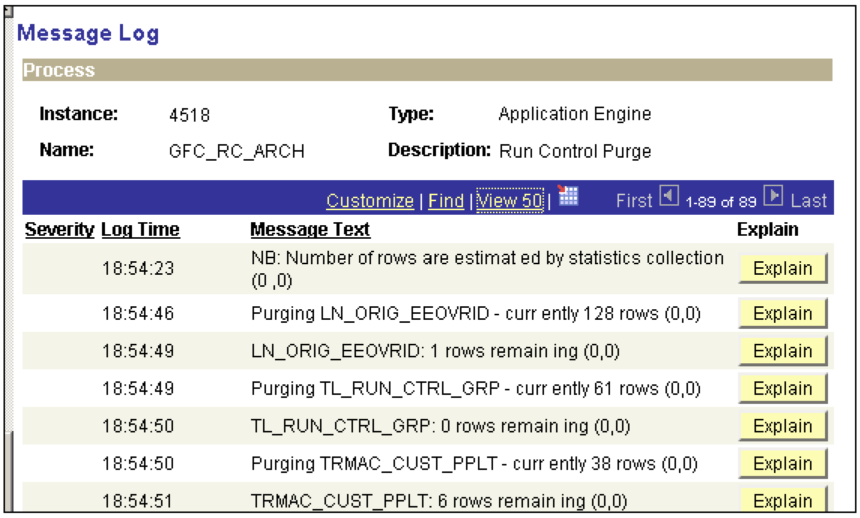
[6] The record should appear on a page

[7] The component (in which the page exists) should be a process security component

[8] The table contain rows as determined when the cost-based optimizer statistics were refreshed. NB: This is Oracle specific.

- I have used a multi-column IN-list to avoid using WHERE NOT EXISTS.

The purge process reports to the message log on the number of rows in each run control table before and after purge.



Message Log

Process

Instance:	4518	**Type:**	Application Engine
Name:	GFC_RC_ARCH	**Description:**	Run Control Purge

Customize | Find | View 50 | First 1-89 of 89 Last

Severity	Log Time	Message Text	Explain
	18:54:23	NB: Number of rows are estimat ed by statistics collection (0 ,0)	Explain
	18:54:46	Purging LN_ORIG_EEOVRID - curr ently 128 rows (0,0)	Explain
	18:54:49	LN_ORIG_EEOVRID: 1 rows remain ing (0,0)	Explain
	18:54:49	Purging TL_RUN_CTRL_GRP - curr ently 61 rows (0,0)	Explain
	18:54:50	TL_RUN_CTRL_GRP: 0 rows remain ing (0,0)	Explain
	18:54:50	Purging TRMAC_CUST_PPLT - curr ently 38 rows (0,0)	Explain
	18:54:51	TRMAC_CUST_PPLT: 6 rows remain ing (0,0)	Explain

Implementation Considerations

This purge process is designed to run in and maintain a relatively steady volume of data. It only deletes data, so space freed up by the deleted rows will remain within the table and index segments and will not be released back to the database.

I recommend that before this Application Engine is implemented, the largest Run Control tables are purged by being rebuilt with only the data to be retained. Otherwise, the purge process will run for extended periods.

I have done this for each record by using Application Designer to generate an alter script (even if there are no changes), and then adding a criteria to the INSERT … SELECT … statement. I also refresh the statistics on the object. Here is an (edited) example

```
-- Start the Transaction

-- Create temporary table

CREATE TABLE PSYTL_RUN_CTRL_GRP
…
/

-- Copy from source to temp table

INSERT INTO PSYTL_RUN_CTRL_GRP (
       OPRID,    RUN_CNTL_ID,    EMPLID,    EMPL_RCD,    GROUP_ID,
INCLUD_EXCLUDE_IND)
  SELECT
       OPRID,    RUN_CNTL_ID,    EMPLID,    EMPL_RCD,    GROUP_ID,
INCLUD_EXCLUDE_IND
  FROM PS_TL_RUN_CTRL_GRP
 WHERE (OPRID, RUN_CNTL_ID) IN (SELECT DISTINCT OPRID, RUNCNTLID
FROM PSPRCSRQST)
/

-- CAUTION: Drop Original Table
DROP TABLE PS_TL_RUN_CTRL_GRP
/

-- Rename Table
RENAME PSYTL_RUN_CTRL_GRP TO PS_TL_RUN_CTRL_GRP
/

-- Done
CREATE UNIQUE  iNDEX PS_TL_RUN_CTRL_GRP ON PS_TL_RUN_CTRL_GRP
(OPRID,   RUN_CNTL_ID,   EMPLID,   EMPL_RCD,
   GROUP_ID)
…
/
ALTER INDEX PS_TL_RUN_CTRL_GRP NOPARALLEL LOGGING
/

begin
 sys.dbms_stats.gather_table_stats(ownname=>'SYSADM'
     ,tabname=>'PS_TL_RUN_CTRL_GRP'
     ,estimate_percent => DBMS_STATS.AUTO_SAMPLE_SIZE
     ,cascade => TRUE
 );
end;
/
```