DBMS_PROFILER

PACKAGE

Author	이상신
Creation Date	2019-06-01
Last Updated	
Version	1.0

Version	변경일자	변경자(작성자)	주요내용
1	2019-06-01	이상신	최초작성

Contents

1. DBMS_PROFILER PACKAGE

1.1. 개요

Oracle 8i 부터 Profiler AP 을 제공하여 PL/SQL 애플리케이션을 프로파일링 하고 성능 병목 현상 식별할 수 있게 되었습니다.

수집된 프로파일러(성능) 데이터를 이용하여 PL/SQL 부하 코드 지점을 쉽게 확인하여 개발자는 코드 수정에 집중할 수 있습니다.

이 인터페이스를 사용하면 세션에서 실행되는 모든 라이브러리 단위에 대한 성능정보를 생성할 수 있습니다.

이정보에는 각 행이 실행된 총 횟수, 총 시간, 최대/최소 시간등을 수집합니다.

1.2. 설치

-SYSDBA 권한으로 접속(SQLPLUS)

```
-- DBMS_PROFILER PACKAGE 설치
@?/rdbms/admin/profload.sql

-- SCOTT 계정에 DBA 권한 주기
GRANT DBA TO SCOTT;

-- SCOTT 계정으로 접속
CONN SCOTT/TIGER

-- PROFLER 성능 TABLE 생성
@?/rdbms/admin/proftab.sql
```

1.3. 생성 OBJECT 확인

SCOTT 계정으로 proftab.sql 수행시 생성되는 OBJECT

오프젝트 유형	오브젝트 명
테이블	PLSQL_PROFILER_RUNS
테이블	PLSQL_PROFILER_UNITS
테이블	PLSQL_PROFILER_DATA
시퀀스	PLSQL_PROFILER_RUNNUMBER

PLSQL_PROFILER 로 시작하는 테이블 3 개 와 시퀀스 1 개가 생성되었습니다.

1.4. 생성 데이블 설명

PLSQL_PROFILER_RUNS

컬럼	타입	설명
runid	NUMBER	고유 실행 식별자
related_run	NUMBER	Runid (클라이언트/ 서버 상관 관계)
run_owner	VARCHAR2(32)	실행한 사용자
run_date	DATE	실행 시작 시간
run_comment	VARCHAR2(2047)	실행한 사용자가 생성한 설명
run_total_time	NUMBER	이 실행의 경가 시간 (**나노초 단위**)

PLSQL_PROFILER_UNITS

컬럼	타입	설명
runid	NUMBER	고유 실행 식별자
unit_number	NUMBER	내부적으로 생성된 라이브러리 유닛
unit_type	VARCHAR2(32)	라이브러리 단위 유형
unit_owner	VARCHAR2(32)	소유자 이름
unit_name	VARCHAR2(32)	라이브러리 유닛 이름
unit_timestamp	DATE	실행 사이의 단위 변경을 감지하는 데 사용
total_time	NUMBER	이 단위에서 보낸 총 시간 (**나노초 단위**).

PLSQL_PROFILER_UNITS

컬럼	타입	설명
runid	NUMBER	고유 실행 식별자
unit_number	NUMBER	내부적으로 생성된 라이브러리 유닛
line#	NUMBER	줄 번호 (단위)
total_occur	NUMBER	라인이 실행된 횟수
total_time	NUMBER	실행 시간 (**나노초 단위**)
min_time	NUMBER	이 라인의 최소 실행 시간 (**나노초 단위**)
max_time	NUMBER	이 라인의 최대 실행 시간 (**나노초 단위**)

1.5. 사용 방법

프로파일러 서브 프로그램

서브 프로그램	설명
START_PROFILER()	사용자 세션에서 프로파일 데이터 **수집 시작**
STOP_PROFILER()	사용자 세션에서 프로파일 데이터 **수집 중지**
FLUSH_DATA()	사용자 세션에서 수집된 프로파일 데이터 플러시
PAUSE_PROFILER()	데이터 수집 일시 중지
RESUME_PRFILER()	데이터 수집 다시 시작

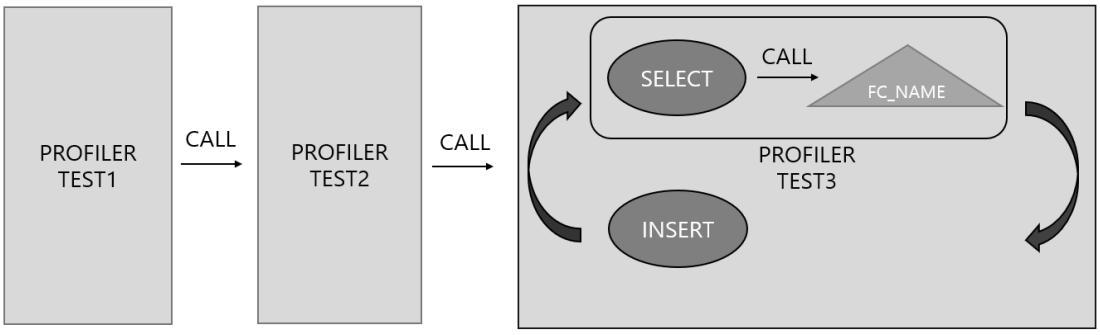
PL/SQL PROFILER는 START_PROFILER 와 STOP_PROFILER 함수를 호출하여 실행의 시작과 끝을 제어합니다.

- **주의점 -**

수집된 데이터는 메모리에 저장되어 있어 사용자가 연결을 끊어지면 자동으로 저장되지 않습니다.
반드시 세션이 끝날 때 FLUSH_DATA 또는 STOP_PROFILER 함수를 명시적으로 호출하여 데이터를 저장하여야 합니다.

2. 예제 생성

2.1. 호출 흐름도



성능 문제가 있는 프로시저을 찾았을 때 프로시저에서 프로시저을 호출하고 호출 받은 프로시저가 또 다른 프로시저나 사용자 정의함수를 또 다시 호출하는 경우가 많다.
그 경우을 가정하여 예제를 생성하였다.

예제는 프로시저가 프로시저 호출하고 또다시 프로시저을 호출하여 호출 받은 프로시저가 SELECT 및 사용자 정의함수 또 호출하는 흐름으로 구성하였다.

2.2. 검증 TABLE 생성

CHECK_LOOP_TIME

```
CREATE TABLE CHECK_LOOP_TIME
(
  RUNID NUMBER
, CNT NUMBER
, CURRENT_TIME DATE DEFAULT SYSDATE
);
```

2.3. FUNCTION 생성

F_GET_NAME

```
CREATE OR REPLACE FUNCTION F_GET_NAME( V_EMPNO IN EMP.EMPNO%TYPE)
    RETURN VARCHAR2 IS V_ENAME EMP.ENAME%TYPE;
 BEGIN

 SELECT ENAME
   INTO V_ENAME
   FROM EMP
  WHERE EMPNO = V_EMPNO;
```

2.4. PROCEDURE 생성

PROFILER_TEST_1

```
CREATE OR REPLACE PROCEDURE PROFILER_TEST_1(PROC_CNT IN  NUMBER) AS
    V_GD_CUSTOMER_CNT NUMBER(15);
BEGIN

  FOR I IN 1 .. PROC_CNT LOOP
    PROFILER_TEST_2(PROC_CNT => PROC_CNT);
  END LOOP;
END;
/
;
```

PROFILER_TEST_2

```
CREATE OR REPLACE PROCEDURE  PROFILER_TEST_2(PROC_CNT  IN  NUMBER) AS
BEGIN
  FOR I IN 1 .. PROC_CNT LOOP
    PROFILER_TEST_3(PROC_CNT => PROC_CNT);
  END LOOP;
END;
/
;
```

PROFILER_TEST_3

```
CREATE OR REPLACE PROCEDURE  PROFILER_TEST_3(PROC_CNT IN NUMBER) AS
  V_EMPNO NUMBER(8);
  V_NAME  VARCHAR2(50);
  V_JOB   VARCHAR2(50);
  V_MGR   NUMBER(8);
  V_SAL   NUMBER(8);
BEGIN
  FOR I IN 1 .. PROC_CNT LOOP
    SELECT E.EMPNO
         , F_GET_NAME(E.EMPNO) AS NAME
         , E.JOB
         , E.MGR
         , E.SAL
      INTO V_EMPNO
         , V_NAME
         , V_JOB
         , V_MGR
         , V_SAL
      FROM EMP E
     WHERE E.DEPTNO IN (SELECT DEPTNO FROM EMP)
       AND ROWNUM <= 1;

    INSERT
    INTO CHECK_LOOP_TIME
    VALUES ( (SELECT TO_NUMBER(MAX(RUNID))
                FROM PLSQL_PROFILER_RUNS)
           , (SELECT NVL(MAX(CNT), 0) + I
                FROM CHECK_LOOP_TIME
                WHERE RUNID IN (SELECT TO_NUMBER(MAX(RUNID))
                                  FROM PLSQL_PROFILER_RUNS))
           , SYSDATE );
    COMMIT;
  END LOOP;
END;
/
```

3. 성능데이터 수집 및 결과

3.1. PROFILER 수행

성능 데이터 수집(기본)

```
-- PROFILER 시작(성능데이터 수집 시작)
EXEC DBMS_PROFILER.START_PROFILER
(TO_CHAR(SYSDATE, 'YYYY/MM/DD/HH24:MI:SS')||'_PROFILER_TEST_1');

-- 프로시져 호출
EXEC PROFILER_TEST_1(30);

-- PROFILER 종료 (메모리에 있는 성능데이터 Table로 쓰기 및 PROFILER 종료)
EXEC DBMS_PROFILER.STOP_PROFILER();
```

성능 데이터 수집

```
-- PROFILER 시작(성능데이터 수집 시작)
EXEC DBMS_PROFILER.START_PROFILER
(TO_CHAR(SYSDATE, 'YYYY/MM/DD/HH24:MI:SS')||'_PROFILER_TEST_1');

-- 프로시져 호출
EXEC PROFILER_TEST_1(30);

-- PROFILER 종료 (메모리에 있는 성능데이터 Table로 쓰기 및 PROFILER 종료)
EXEC DBMS_PROFILER.STOP_PROFILER();
```

.

3.2. 성능데이터 수집 결과

PLSQL_PROFILER_RUNS

```
SELECT RUNID
     , RUN_OWNER
     , RUN_DATE
     , RUN_COMMENT
     , RUN_TOTAL_TIME
FROM PLSQL_PROFILER_RUNS;
```

RUNID	RUN_OWNER	RUN_DATE	RUN_COMMENT	RUN_TOTAL_TIME
1	SCOTT	2019/06/01 21:45:54	2019/06/01/21:45:54_PROFILER_TEST_1	19,372,000,000

RUNID 1로 START_PROFILER() 수행시 자동으로 1씩 증가한다.

PLSQL_PROFILER_UNITS

```
SELECT RUNID
     , UNIT_NUMBER
     , UNIT_TYPE
     , UNIT_OWNER
     , UNIT_NAME
     , UNIT_TIMESTAMP
  FROM PLSQL_PROFILER_UNITS;
```

RUNID	UNIT_NUMBER	UNIT_TYPE	UNIT_OWNER	UNIT_NAME	UNIT_TIMESTAMP
1	1	ANONYMOUS	<anonymous>	<anonymous>	-10100/00/00
1	2	ANONYMOUS	<anonymous>	<anonymous>	-10100/00/00
1	3	PROCEDURE	SCOTT	PROFILER_TEST_1	2019/06/01 21:45:29
1	4	PROCEDURE	SCOTT	PROFILER_TEST_2	2019/06/01 21:45:25
1	5	PROCEDURE	SCOTT	PROFILER_TEST_3	2019/06/01 21:45:23
1	6	FUNCTION	SCOTT	F_GET_NAME	2019/06/01 21:45:21
1	7	ANONYMOUS	<anonymous>	<anonymous>	-10100/00/00

- **특이점**

  UNIT_TIMESTAMP 시간이 PROFILER_RUNS TABLE RUN_DATA COLUMN 값보다 작아 버그로 의심된다.

PLSQL_PROFILER_UNITS

```
SELECT RUNID
     , UNIT_NUMBER
     , LINE#
     , TOTAL_OCCUR
     , TOTAL_TIME
     , MIN_TIME
     , MAX_TIME
  FROM PLSQL_PROFILER_DATA;
```

RUNID	UNIT_NUMBER	LINE#	TOTAL_OCCUR	TOTAL_TIME	MIN_TIME	MAX_TIME
1	1	1	1	375	375	375
1	2	1	2	203,212	706	201,314
1	3	1	0	905	905	905
1	3	5	31	7,845	141	1,360
1	3	6	30	90,533	764	61,744
1	3	7	0	0	0	0
1	3	9	1	1,119	1,119	1,119
1	4	1	0	12,903	315	858
1	4	3	930	326,475	27	18,404
1	4	4	900	1,166,942	420	45,124
1	4	5	0	0	0	0
1	4	6	30	38,063	586	18,225
1	5	1	900	960,442	191	27,856
1	5	8	27,900	6,361,707	18	33,759
1	5	9	27,000	1,332,762,433	288	3,240,170
1	5	23	27,000	15,485,918,552	49,348	6,048,032
1	5	32	27,000	1,187,448,093	18,434	226,700
1	5	33	0	0	0	0
1	5	34	900	1,948,900	854	34,866
1	6	1	27,000	13,623,525	21	25,704
1	6	5	27,000	922,818,860	23,785	619,640
1	6	10	27,000	5,994,791	51	31,164
1	6	12	27,000	26,642,931	451	24,320
1	7	1	2	31,056	521	29,192

CHECK_LOOP_TIME

```
SELECT * FROM CHECK_LOOP_TIME ;
```

RUNID	CNT	CURRENT_TIME
1	1	2019/06/01 21:45:54
1	2	2019/06/01 21:45:54
1	3	2019/06/01 21:45:54
1	4	2019/06/01 21:45:54
1	5	2019/06/01 21:45:54
1	6	2019/06/01 21:45:54
1	7	2019/06/01 21:45:54
1	8	2019/06/01 21:45:54
1	9	2019/06/01 21:45:54
1	10	2019/06/01 21:45:54
....	....	....
1	26,991	2019/06/01 21:46:14
1	26,992	2019/06/01 21:46:14
1	26,993	2019/06/01 21:46:14
1	26,994	2019/06/01 21:46:14
1	26,995	2019/06/01 21:46:14
1	26,996	2019/06/01 21:46:14
1	26,997	2019/06/01 21:46:14
1	26,998	2019/06/01 21:46:14
1	26,999	2019/06/01 21:46:14
1	27,000	2019/06/01 21:46:14

4. 성능데이터 분석

4.1. 성능 분석 쿼리

성능 분석 쿼리

```
SELECT
       RUN_OWNER
     , RUN_DATE
     , RUN_COMMENT
     , RUN_TOTAL_TIME/1000000 AS TOTAL_MSEC
     , PU.UNIT_TYPE
     , PU.UNIT_OWNER
     , PU.UNIT_NAME
     , PU.UNIT_TIMESTAMP
     , PD.TOTAL_OCCUR
     , ROUND(DECODE(PD.TOTAL_OCCUR,NULL,0,0,0,
               PD.TOTAL_TIME/PD.TOTAL_OCCUR/1000000), 2) AS AVG_TIME
     , ROUND(NVL(PD.TOTAL_TIME, 0), 2) AS TOTAL_TIME
     , ROUND(SUM(PD.TOTAL_TIME/1000000)
               OVER (PARTITION BY PD.UNIT_NUMBER), 2) AS UNIT_TOTAL_TIME
     , ROUND(SUM(PD.TOTAL_TIME/1000000) OVER (), 2) AS GRAND_TOTAL_TIME
     , RANK() OVER(ORDER BY PD.TOTAL_TIME DESC) AS TOTAL_TIME_RANK
     , PD.LINE# || ' : ' || DS.TEXT AS LINE_TEXT
  FROM PLSQL_PROFILER_RUNS PR
     , PLSQL_PROFILER_UNITS PU
     , PLSQL_PROFILER_DATA PD
     , ALL_SOURCE DS
 WHERE PR.RUNID = 1
   AND PD.RUNID = PR.RUNID
   AND PD.RUNID = PU.RUNID
   AND PD.UNIT_NUMBER = PU.UNIT_NUMBER
   AND PU.UNIT_NAME = DS.NAME
   AND PU.UNIT_OWNER = DS.OWNER
   AND PU.UNIT_TYPE = DS.TYPE
   AND PD.LINE# = DS.LINE
 ORDER BY PD.UNIT_NUMBER, PD.LINE#;
```

위 쿼리는 유닛(프로시저, 사용자 정의 함수)수행 순서 유닛 안에 라인당 수행 횟수, 총수행 시간, 평균시간, 라인 번호, 소스, 그리고 가장 부하 있는 지점까지 한번에 볼 수 있게 작성하였다.

UTYPE	NAME	TOTAL_OC	AVG_	TOTAL	LINE_TEXT
PROCEDURE	PROFILER_TEST_1	0	0	0	1 : PROCEDURE PROFILER_TEST_1(PROC_
PROCEDURE	PROFILER_TEST_1	31	0	0.01	5 : FOR I IN 1 .. PROC_CNT LOOP
PROCEDURE	PROFILER_TEST_1	30	0	0.09	6 : PROFILER_TEST_2(PROC_CNT =>
PROCEDURE	PROFILER_TEST_1	0	0	0	7 : END LOOP;
PROCEDURE	PROFILER_TEST_1	1	0	0	9 : END;
PROCEDURE	PROFILER_TEST_2	0	0	0.01	1 : PROCEDURE PROFILER_TEST_2(PROC
PROCEDURE	PROFILER_TEST_2	930	0	0.33	3 : FOR I IN 1 .. PROC_CNT LOOP
PROCEDURE	PROFILER_TEST_2	900	0	1.17	4 : PROFILER_TEST_3(PROC_CNT =>
PROCEDURE	PROFILER_TEST_2	0	0	0	5 : END LOOP;
PROCEDURE	PROFILER_TEST_2	30	0	0.04	6 : END;
PROCEDURE	PROFILER_TEST_3	900	0	0.96	1 : PROCEDURE PROFILER_TEST_3(PROC
PROCEDURE	PROFILER_TEST_3	27,900	0	6.36	8 : FOR I IN 1 .. PROC_CNT LOOP
PROCEDURE	PROFILER_TEST_3	27,000	0.05	1,332.76	9 : SELECT E.EMPNO
PROCEDURE	PROFILER_TEST_3	27,000	0.57	15,485.92	23 : INSERT
PROCEDURE	PROFILER_TEST_3	27,000	0.04	1,187.45	32 : COMMIT;
PROCEDURE	PROFILER_TEST_3	0	0	0	33 : END LOOP;
PROCEDURE	PROFILER_TEST_3	900	0	1.95	34 : END;
FUNCTION	F_GET_NAME	27,000	0	13.62	1 : FUNCTION F_GET_NAME(V_EMPNO IN
FUNCTION	F_GET_NAME	27,000	0.03	922.82	5 : SELECT ENAME
FUNCTION	F_GET_NAME	27,000	0	5.99	10 : RETURN V_ENAME;
FUNCTION	F_GET_NAME	27,000	0	26.64	12 : END;

```
-- PLSQL_PROFILER_UNITS 테이블 UNIT_TIMESTAMP 컬럼 값 검증 --
SELECT
      MIN(cnt) AS MIN_CNT
    , MAX(cnt) AS MAX_CNT
    , MIN(current_time) AS MIN_TIME
    , MAX(current_time) AS MAX_TIME
    , ((MAX(current_time) - MIN(current_time)) *24*60*60 ) AS RUNING_TIME
  FROM CHECK_LOOP_TIME
 WHERE RUNID = 1;
```

MIN_CNT	MAX_CNT	MIN_TIME	MAX_TIME	RUNING_TIME
1	27,000	2019/06/01 21:45:54	2019/06/01 21:46:14	19.99999

PLSQL_PROFILER_UNITS 테이블 UNIT_TIMESTAMP 컬럼 내용 확인 시 시작 TIME 21:45:23 초로 CHECK_LOOP_TIME 테이블 MIN_TIME 컬럼 21:45:54 초 보다 작아 UNIT_TIMESTAMP 컬럼 버그로 판단된다.

4.2. 분석 결과 설명

위 예제에서는 PROFILER_TEST_3 프로시저 23 번 라인 INSERT 라인에서
27,000 번 호출 / 평균 0.57 초 / 총 15.485 초로 가장 부하가 큰 지점을
어렵지 않게 찾을 수 있다.

위 예제에서 보듯 PL/SQL 성능 데이터 수집하여 어떤 프로시저 또는 함수에서
성능 저하가 발생하였는지 간편하게 알 수 있고 해당 라인까지 성능 관련된 정보를
상세하게 파악할 수 있어 개발자 또는 SQL 튜너는 성능 개선에만 집중할 수 있다.

<끝>