

데이터베이스 프로그래밍-PL/SQL Package 활용(10차시)

10주차: ORACLE 제공 PACKAGE 활용

I. 시작

[학습목표]

ORACLE社에서 제공하는 PACKAGE를 PL/SQL 개발시 활용하는 방법을 학습

[학습목차]

10-1 PACKAGE 개요

10-2 UTL_FILE

10-3 DBMS_JOB

10-4 DBMS_RANDOM

10-5 DBMS_ERRLOG

10-1 PACKAGE 개요

ORACLE社에서 제공하는 PACKAGE의 이름에는 규칙이 있습니다.

DBMS_ 또는 UTL_로 시작합니다.

ORACLE 버전별로 PACKAGE의 개수를 비교를 해봅시다

| 버전 | DBMS_ | UTL_ |
|-----|-------|------|
| 9i | 258 개 | 13 개 |
| 10G | 388 개 | 20 개 |
| 11G | 525 개 | 20 개 |

버전이 증가 할수록 제공되는 PACKAGE도 늘어 납니다.

PL/SQL의 개발시 주로 DBMS내에 저장된 데이터를 처리하는 용도로 개발을 한다.

ORACLE社에서 제공하는 PACKAGE는 개별 프로젝트의 개별 데이터 처리 용도가 아닌 PL/SQL을 마치 일반 개발언어와 같은 기능과 유연성을 제공 한다.

간단한 특징을 보면

(1) PACKAGE의 소유자는 SYS 계정

(2) DBMS내의 각 계정은 PACKAGE에 대한 기본적인 실행 권한을 가지고 있으며
필요시 OBJECT의 EXECUTE 권한을 부여 함으로써 개개의 PACKAGE를 실행
할수 있다 (EX GRANT EXECUTE ON DBMS_FLASHBACK TO SCOTT;)

(3) PACKAGE HEADER는 TEXT SOURCE 형태로 제공되며

PACKAGE BODY는 WRAP에 의해 BINARY SOURCE 형태로 제공.

데이터베이스 프로그래밍-PL/SQL Package활용(10차시)

```
PACKAGE utl_file AUTHID CURRENT_USER AS

/*
** FILE_TYPE - File handle
**/

TYPE file_type IS RECORD (id BINARY_INTEGER, datatype BINARY_INTEGER);

/*
** Exceptions
**/
file_open          EXCEPTION;
charsetmismatch    EXCEPTION;
invalid_path        EXCEPTION;
invalid_mode        EXCEPTION;
invalid_filehandle  EXCEPTION;
invalid_operation   EXCEPTION;
read_error          EXCEPTION;
write_error         EXCEPTION;
internal_error      EXCEPTION;
invalid_maxlinesize EXCEPTION;
invalid_filename    EXCEPTION;
```

(EX UTL_FILE PACKAGE 의 HEADER 중 일부)

```
PACKAGE BODY utl_file wrapped
0
abcd
abcd
abcd
abcd
abcd
a0 105:2 a0
105 a0 105:2 a0 105:2 a0 105 a0
105:2 a0 105:2 a0 105 a0 105:2 a0
105:2 a0 105 a0 105:2 a0 105 a0
105:2 a0 105 a0 105:2 a0 105 a0
b0 3d 8f a0 b0 3d
b4 55 6a a3 a0 51 a5 1c
81 b0:2 a0 a5 b 7e b4 2e
284 288
28c 290 294 298 29c 2a0 2a4 2a8
2ac 2b0 2b4 2b8 2bc 2c0 2c4 2c6
2ca 2ce 2d0 2d1 2d5 2d9 2dd 2e1
2e5 301 2fd 2fc 309 316 312 2f9
31e 311 323 327 32b 32f 333 337
33b 348 34c 350 355 359 30e 35d
361 365 369 36d 371 375 379 37d
212d5 12d9 12dd 12de 12e0 12e3 12e8
12e9 12ee 12f1 12f5 12f9 12fd 1301 1306
130a 130e 1312 1315 1316 1318 131c 1320
1324 1327 132b 132f 1333 1337 133b 133a
51eca 1ecd 1ed1 1ed5 1ed6 1edb 1edd
1ee1 1ee5 1ee8 1eec 1eeF 1ef0 1ef5 1ef8
1efc 1f00 1f01 1f05 1f08 1f0c 1f0f 1f11
1f15 1f19 1f1c 1f20 1f24 1f28 1f2b 1f2f
1f30 1f35 1f37 1f3c 1f3d 1f40 1f43 1f46
c 2ac0 2ac4 2ac5 2ac7 2ac8 2acd 2acf
2c0 2c3 2c8 2c9 2ca 2cb 2cc 2cd 2ce 2cf
```

(EX UTL_FILE PACKAGE 의 BODY 중 일부)

<참고> ORACLE 제공 PACKAGE 의 접두어를 보면 2 가지 유형 입니다.

DBMS_ 와 UTL_ 은 ORACLE 社 내에서 해당 PACKAGE 를 개발하는 부서에 따라
달리 부여된 이름이라고 한다. 사용자 입장에서는 접두어의 구분은 없다.

데이터베이스 프로그래밍-PL/SQL Package활용(10차시)

10-2 UTL_FILE

PL/SQL 을 통해 ORACLE DBMS 가 설치된 SERVER 내에서 TEXT FILE 을 생성/읽기/쓰기/삭제 등 FILE 과 관련된 다양한 핸들링이 가능한 PACKAGE 이다. 활용성 과 위험성 측면에서 주의가 필요한 PACKAGE 입니다.

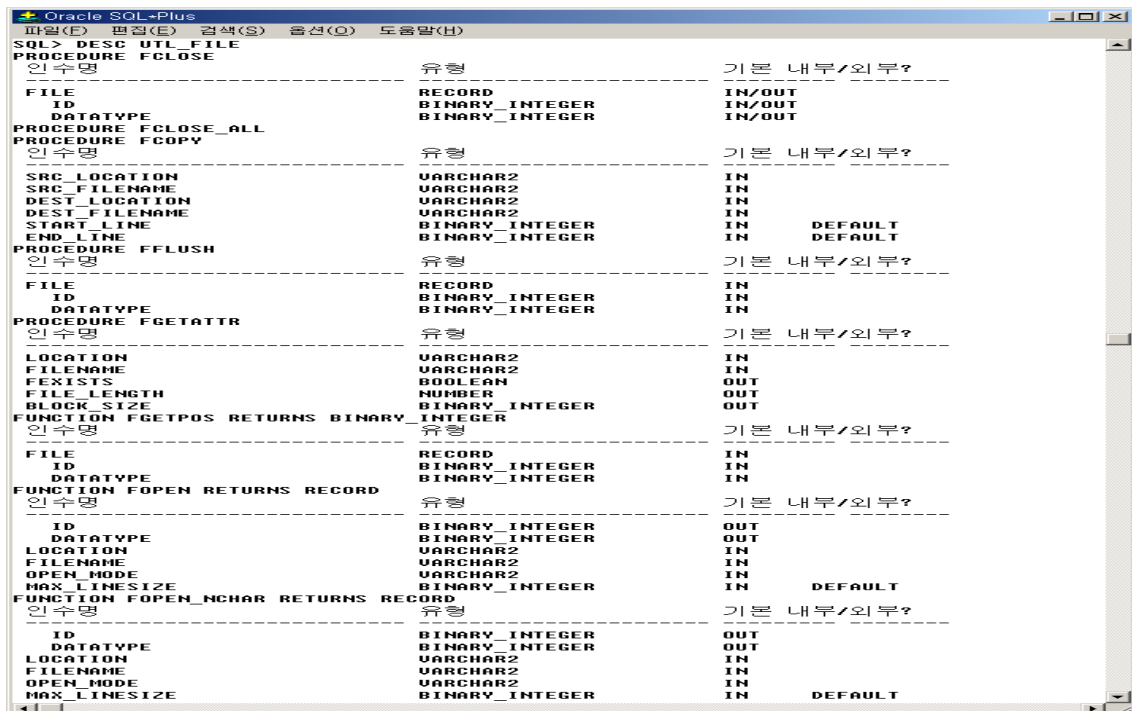
위험성은?

SQL*PLUS 나 CLIENT PROGRAM 내에서 ANONYMOUS PL/SQL BLOCK 을 실행하게 되면 실제적으로는 해당 BLOCK 이 서버에 전달되어 DBMS SERVER 내에서 실행 된다.

CLIENT 에서 DBMS SERVER 내의 FILE 에 접근하여 생성/삭제/변경 할수 있기 때문에 악용하면 문제가 발생할수 있습니다. 서버내의 보안을 요하는 파일을 읽거나 중요한 파일을 훼손할수 있기 때문입니다.

데이터베이스 관리자(DBA) 측면에서 이런 위험성에 대한 대비책이 준비 되어 있다 UTL_FILE PACKAGE 를 개발자 측면에서 사용하려 할 때 데이터베이스 관리자(DBA)의 도움을 얻어야 하는 이유 이다..

DESC UTL_FILE 을 실행해서 UTL_FILE 이 제공하는 기능을 찾아 보시다.



| 인수명 | 유형 | 기본 내부/외부? |
|------------------------------------------------|----------------|------------|
| PROCEDURE FCLOSE | | |
| FILE | RECORD | IN/OUT |
| ID | BINARY_INTEGER | IN/OUT |
| DATATYPE | BINARY_INTEGER | IN/OUT |
| PROCEDURE FCLOSE_ALL | | |
| PROCEDURE FCOPY | | |
| 인수명 | 유형 | 기본 내부/외부? |
| SRC_LOCATION | VARCHAR2 | IN |
| SRC_FILENAME | VARCHAR2 | IN |
| DEST_LOCATION | VARCHAR2 | IN |
| DEST_FILENAME | VARCHAR2 | IN |
| START_LINE | BINARY_INTEGER | IN DEFAULT |
| END_LINE | BINARY_INTEGER | IN DEFAULT |
| PROCEDURE FFLUSH | | |
| 인수명 | 유형 | 기본 내부/외부? |
| FILE | RECORD | IN |
| ID | BINARY_INTEGER | IN |
| DATATYPE | BINARY_INTEGER | IN |
| PROCEDURE FGETATTR | | |
| 인수명 | 유형 | 기본 내부/외부? |
| LOCATION | VARCHAR2 | IN |
| FILENAME | VARCHAR2 | IN |
| EXISTS | BOOLEAN | OUT |
| FILE_LENGTH | NUMBER | OUT |
| BLOCK_SIZE | BINARY_INTEGER | OUT |
| FUNCTION FGETPOS RETURNS BINARY_INTEGER | | |
| 인수명 | 유형 | 기본 내부/외부? |
| FILE | RECORD | IN |
| ID | BINARY_INTEGER | IN |
| DATATYPE | BINARY_INTEGER | IN |
| FUNCTION FOPEN RETURNS RECORD | | |
| 인수명 | 유형 | 기본 내부/외부? |
| ID | BINARY_INTEGER | OUT |
| DATATYPE | BINARY_INTEGER | OUT |
| LOCATION | VARCHAR2 | IN |
| FILENAME | VARCHAR2 | IN |
| OPEN_MODE | VARCHAR2 | IN |
| MAX_LINESIZE | BINARY_INTEGER | IN DEFAULT |
| FUNCTION FOPEN_NCHAR RETURNS RECORD | | |
| 인수명 | 유형 | 기본 내부/외부? |
| ID | BINARY_INTEGER | OUT |
| DATATYPE | BINARY_INTEGER | OUT |
| LOCATION | VARCHAR2 | IN |
| FILENAME | VARCHAR2 | IN |
| OPEN_MODE | VARCHAR2 | IN |
| MAX_LINESIZE | BINARY_INTEGER | IN DEFAULT |

<참고> 다른 일반적인 개발언어 처럼 FILE OPEN/CLOSE/COPY 가 가능한 PROCEDURE 가 PACKAGE 에서 제공한다.

데이터베이스 프로그래밍-PL/SQL Package활용(10차시)

10_UTL_FILE.SQL

```
REM *****
REM **      NAME
REM **      WRITE_LOG
REM **
REM **      NOTES
REM **      -PL/SQL에서 TEXT FILE을 READ/WRITE할수 있는 기능을 이용하여 log file 생성 procedure
REM **      -WRITE_LOG기능을 사용하기 위해서는 DBMS SERVER에서 DBA 계정을 통해 적절한 SETTING 필요
REM **      (1) DIRECTORY 생성
REM **      SQL> CREATE OR REPLACE DIRECTORY APP_LOG_DIR AS 'C:\WDB_LOG';
REM **
REM **      (2) DIRECTORY 에 대한 READ/WRITE 사용권한 부여
REM **      SQL> GRANT READ,WRITE ON DIRECTORY APP_LOG_DIR TO SCOTT;
REM **
REM **      MODIFIED
REM **      김길동   15/04/02   - 신규 작성
REM **      홍길동   15/05/01   - 예외처리부 추가, NULL 명령어 사용
REM *****
```

```
CREATE OR REPLACE PROCEDURE WRITE_LOG(A_PROGRAM_NAME IN VARCHAR2,A_ERROR_MESSAGE IN VARCHAR2,A_DESCRIPTION IN VARCHAR2)
IS
    V_LOG_FILE_NAME      VARCHAR2(40)  := 'DBLOG_'||TO_CHAR(SYSDATE,'YY_MM_DD')||'.LOG' ;      -- LOGFILE명
    F_HANDLER            UTL_FILE.FILE_TYPE;      -- FILE HANDLER

BEGIN
    F_HANDLER :=UTL_FILE.FOPEN('APP_LOG_DIR',V_LOG_FILE_NAME,'A'); -- A:APPEND MODE

    UTL_FILE.PUT_LINE(F_HANDLER,TO_CHAR(SYSDATE,'YYYY-MM-DD:HH24:MI:SS ')||
        A_PROGRAM_NAME||' , '||A_ERROR_MESSAGE||' , '||A_DESCRIPTION);

    UTL_FILE.FCLOSE(F_HANDLER);

EXCEPTION
    WHEN OTHERS THEN
        BEGIN
            UTL_FILE.FCLOSE(F_HANDLER);
        EXCEPTION
            WHEN OTHERS THEN
                NULL;      -- NULL 명령어
        END;

END;
/
-- -----
-- write_log test
-- -----
DESC WRITE_LOG

EXECUTE WRITE_LOG('TEST_APP','ERROR MESSAGE','WRITE_LOG PACKGE TEST');      -- C:\DB_LOG DIRECTORY에서 파일 확인

EXECUTE CHANGE_SALARY(A_EMPNO => 7369 , A_SALARY => 987654321);      -- 자리수 초과 예러 유발
```

① 주석 작성

현업에서 PL/SQL MODULE 을 작성할때는 위의 주석 예제 처럼 자세한 주석을

데이터베이스 프로그래밍-PL/SQL Package활용(10차시)

작성 필요. 주석 자체가 개발 문서 역할 과 변경 이력을 가지고 있도록 하는 것이 개발 및 유지보수시에 생산성을 좋게 한다.

NAME 은 MODULE 의 이름

NOTES 는 관련된 주의 사항 및 기타 설명

MODIFIED 는 변경 이력

위의 예제는 ORACLE社에서 제공하는 PACKAGE 나 SQL SCRIPT 에서 사용되는 주석 FORMAT 입니다. 여러분이 개발시에는 각 해당 PROJECT 의 표준 주석 FORMAT 에 맞추어 개발을 하게 된다.

MODULE 개발후 일정한 시점이 지나서 유지 보수 및 기능 개선을 해야 할 때 가장 시간을 많이 소요하는 단계가 해당 MODULE 에 대한 이해 단계 입니다. 이해 단계의 시간을 줄이기 위해서는 자세하고 정확한 주석이 필요합니다.

② PL/SQL 내에서 DBMS 서버내의 FILE 을 핸들링할수 있는 기능은 유용하면서도 보안적인 위험을 가능하게 기능 이다. 위험성을 줄이기 위해서 FILE 을 핸들링할수 있는 디렉토리 범위를 데이터베이스 관리자(DBA)가 제한 할수 있게 안전 장치를 해두었다.

디렉토리(DIRECTORY)라는 ORACLE Object 를 사용하여 파일 입출력 할수 있는 디렉토리를 사전에 지정 한다. DIRECTORY 는 CREATE DIRECTORY 또는 CREATE ANY DIRECTOTY 권한을 가지는 USER 에서 생성되거나 DBA 계정에서 생성 가능하다. 'C:WDB_LOG' 를 가리키는 APP_LOG_DIR 라는 DIRECTORY 생성 해당 디렉토리에 대한 사용권한(READ/WRITE)을 DB 사용자 에게 부여합니다.

③ 선언부에 2 개의 변수를 선언 합니다.

V_ LOG_FILE_NAME 은 생성되는 LOG FILE 의 이름이며 파일명에

년_월_일 이 포함됩니다. 선언부 와 ④의 두번째 라인을 보면 LOG FILE 은 일자별로 생성되는 것을 알수 있습니다.

F_HANDLER 는 파일 핸들러로 역할은 파일을 제어 하는데 사용합니다.

파일 핸들러는 파일을 OPEN 시에 할당 되며 파일에 기록하거나 파일을 CLOSE 할 때 사용 됩니다.

④ UTL_FILE.FOPEN 을 통해 파일을 OPEN 합니다.

FOPEN 함수의 첫번째 파라미터는 파일이 위치하는 디렉토리 입니다.

직접적인 경로를 사용하거나 이전에 선언한 ORACLE DIRECTORY 이름을 사용합니다.

데이터베이스 프로그래밍-PL/SQL Package활용(10차시)

두번째 파라미터는 파일명 입니다.

세번째 파라미터는 파일 OPEN MODE 입니다.

‘A’ 는 APPEND 모드로 기존에 파일이 있으면 OPEN 하여
내용을 추가(APPEND)하고 파일이 없으면 신규 생성하여
내용을 추가(APPEND)하는 모드 입니다.

‘W’ 나 ‘R’ 가 올수 있습니다. W 는 WRITE 를 의미하며

‘R’은 READ ONLY 를 의미 합니다.

PUT_LINE PROCEDURE 를 사용하여 OPEN 한 FILE 에 기록(WRITE) 합니다.

FCLOSE 를 사용하여 파일을 CLOSE 합니다.

⑤ 예외 (EXCEPTION) 발생시 OPEN 된 파일을 CLOSE 합니다.

예외처리부에서 CLOSE 시에도 또 에러가 발생하면 OTHERS 에서 예외를
잡지만(CATCH) NULL 명령어를 사용하여 실제적인 후속 조치는 없습니다.

⑥ DESC 를 통해 생성된 PROCEDURE 확인

EXECUTE WRITE_LOG 를 직접 호출해 LOG FILE 에 직접 기록

<참고>

(1) CREATE DIRECTORY 로 생성할 때 OS 상에서 해당 DIRECTORY 를 자동으로
만들어주지 않는다. ‘C:WDB_LOG’는 직접 생성 해주어야 한다.

(2) ‘C:WDB_LOG’생성 없이 EXECUTE WRITE_LOG 를 실행하면

```
SQL> EXECUTE WRITE_LOG('TEST_APP','ERROR MESSAGE','WRITE_LOG PACKGE TEST');
```

PL/SQL 처리가 정상적으로 완료되었습니다.

이런 결과가 나오게 됩니다. WRITE_LOG 실행시 에러가 발생하면 예외처리부에서
NULL 명령어를 수행하고 후속 처리를 하지 않기때문 이다. 이종의 안전 장치를
하려면 LOG TABLE 에 INSERT 를 통해 기록하는 방법도 고려해볼수 있다.

(3) 아래 SQL 을 사용하면 사용가능한 모든 DIRECTORY 를 조회할수 있다.

```
SELECT * FROM ALL_DIRECTORIES;
```

10-3 DBMS_JOB

운영체제에서는 주기적으로 특정 JOB 이 실행되도록

UNIX 계열에서는 CRONTAB 에 등록하고

NT 계열에서는 AT 명령어를 통해 OS 스케줄링 기능을 사용한다.

데이터베이스 프로그래밍-PL/SQL Package활용(10차시)

ORACLE DBMS 내에서 DBMS_JOB PACKAGE 를 통해서 특정 JOB 이 주기적으로 실행 될수 있도록 할수 있다.

주기적으로 수행하기 원하는 JOB 을 JOB QUEUE 에 등록하면 COORDINATOR JOB QUEUE PROCESS(CJQ0 백그라운드 프로세스)가 JOB QUEUE SLAVE PROCESS 를 통해 해당 작업을 주기적으로 수행한다.

< 참고>

DBMS_JOB 이 수행이 안되는 경우에는 DBA 의 도움을 얻어야 합니다. JOB 을 직접 수행하는 JOB QUEUE SLAVE PROCESS 가 기동되도록 요청해야 한다.

부하를 주는 무분별한 JOB 의 주기적인 수행은 DBMS 성능에 문제를 초래할수 있기 때문에 현업에서 사용시 DBA 와 상의를 하거나 성능 영향도를 분석한후 사용해야 한다.

이전 CHAPTER 에서 만든 WRITE_LOG PROCEDURE 를 10 초간격으로 호출 하도록 DBMS_JOB 을 통해 등록 해봅시다.

10_DBMS_JOB.SQL

```
REM -----
REM  WRITE_LOG 프로시저를 DBMS_JOB을 통해 JOB QUEUE에 등록한다.
REM -----
VARIABLE H_JOB_ID      NUMBER
BEGIN
    DBMS_JOB.SUBMIT(
        JOB => :H_JOB_ID,
        WHAT => 'WRITE_LOG(''DBMS_JOB'', ''TEST'', ''TEST'');', -- JOB QUEUE에 등록되는 JOB 고유 번호
        NEXT_DATE => SYSDATE + 5/86400, -- 프로시저명
        INTERVAL => 'SYSDATE + 10/86400' -- NEXT_DATE , 5초뒤 시작
        -- INTERVAL , 10초간격으로 실행
    );

    DBMS_OUTPUT.PUT_LINE('등록된 JOB ID => '||TO_CHAR(:H_JOB_ID));
END;
/
COMMIT;

PROMPT JOB QUEUE에 등록된 JOB LIST 조회
SELECT JOB,WHAT FROM USER_JOBS;

PROMPT 일정한 시간 WAIT

BEGIN
    FOR I IN 1..60
    LOOP
        DBMS_LOCK.SLEEP(1);
        NULL;
    END LOOP;
END;
/

PROMPT 등록된 JOB을 JOB QUEUE에서 삭제 한다.
EXECUTE DBMS_JOB.REMOVE(:H_JOB_ID);

PROMPT JOB QUEUE에 등록된 JOB LIST 조회
SELECT JOB,WHAT FROM USER_JOBS;
```

① VARIABLE 명령어를 사용하여 BIND 변수 H_JOB_ID 를 선언

선언된 BIND 변수는 JOB 등록 결과 확인 및 JOB 삭제시 사용

예제 상에서 JOB 등록 결과를 쉽게 확인 하고 삭제하기 위해서 별도로 지정

데이터베이스 프로그래밍-PL/SQL Package활용(10차시)

한것으로 실제 업무 적용시에는 BIND 변수로 정의할 필요가 없다.

DBMS_JOB.SUBMIT 를 사용하여 JOB 을 JOB QUEUE 에 등록 하고

DBMS_JOB.REMOVE 를 사용하여 JOB QUEUE 에서 JOB 을 삭제 한다.

SUBMIT 의 첫번째 파라미터는 OUT MODE 파라미터로 등록된 JOB 의 JOB ID 입니다. JOB 이 정상적으로 등록되면 정수형의 JOB ID 가 리턴 된다.

따라서 변수를 해당 파라미터에 사용해야 하며 예제상에서는 BIND 변수를 사용

두번째 파라미터는 등록되는 JOB 의 이름을 등록 WRITE_LOG PROCEDURE 에 전달되는 데이터를 표현 하기 위해 연속된 ' ' (싱글쿼테이션)을 사용하여 문자열 데이터 표현

세번째 파라미터는 JOB QUEUE 에 등록된후 첫 시작 시간을 표시하며

파라미터의 데이터 TYPE 은 DATE 형으로 SYSDATE + 5/86400 은 등록후 5 초 후에 시작 한다는 의미.

네번째 파라미터는 JOB 이 실행되는 간격을 의미하며 10/86400 은 10 초 간격을 의미 합니다.

H_JOB_ID 는 JOB QUEUE 에 정상적으로 등록되면 정수형 값을 가지게 됩니다.

DBMS_OUTPUT 으로 출력되는 JOB ID 를 실행 시점에 확인 해보시기 바랍니다.

② PROMPT 는 SQL*PLUS 명령어로 PROMPT 이하의 문자열을 실행시 SQL*PLUS 화면에 출력해주는 기능을 합니다. 중요하지는 않지만 SQL SCRIPT 로 작업시 각 단계별 과정을 표시해줄 때 간혹 사용됩니다.

USER_JOBS 는 JOB QUEUE 에 등록된 JOB 리스트 정보를 관리하는 DD.

③ DBMS_LOCK.SLEEP(1); 는 1 초간 WAIT 를 하도록하는 기능입니다.

FOR LOOP 를 사용하여 60 초를 WAIT 한다. WAIT 하는 동안에 JOB QUEUE 에 등록된 JOB 이 실행되어 로그 파일에 기록 한다.

④ JOB QUEUE 에 등록된 JOB 을 삭제할때는 REMOVE 를 사용합니다.

H_JOB_ID 변수에는 이전 ① 단계에서 JOB 등록시 JOB ID 가 저장되어

있습니다. 등록된 JOB 을 삭제 하기 위해서는 해당 JOB 의 ID 를 사용해야

합니다. 실습 예제 특성상 JOB 등록 ➔ JOB 실행 ➔ JOB 삭제의 일련 과정을 나타낼려고 H_JOB_ID 를 변수에 JOB ID 를 등록후 삭제 했지만

실제 사용할때는 어떻게 할까?

데이터베이스 프로그래밍-PL/SQL Package활용(10차시)

당연히 `SELECT * FROM USER_JOBS;` 를 조회하여 `JOB` 컬럼에서 `JOB ID` 값을 확인한후 `DBMS_JOB.REMOVE` 를 사용하여 삭제 한다.

⑤ `JOB QUEUE` 에서 삭제한후 `SELECT` 를 통해 삭제된 결과를 조회 합니다.

<참고>

버전 및 환경에 따라 `DBMS_LOCK` 을 실습 계정에서 사용하지 못할수도 있습니다. 그런 경우에는 `DBA` 에게 `DBMS_LOCK` 에 대한 실행 권한을 요청 해야 합니다. 테스트 DB 환경에서 여러분이 `DBA` 계정을 가지고 있다면

```
SQL> GRANT EXECUTE ON DBMS_LOCK TO SCOTT;
```

<참고>

`DBMS_JOB` 을 `ORACLE 10g` 버전에서는 `DBMS_SCHEDULER` 로 확장 개편 하였다. `DBMS_SCHEDULER` 의 개편된 기능 중 가장 주요한 점은

`DBMS_JOB` 은 `PL/SQL PROCEDURE/PACKAGE` 만 가능했지만

`DBMS_SCHEDULE` ① `PL/SQL PROCEDURE/PACKAGE/ANONYMOUS BLOCK`

② `External Executable program(ex shell,OS command 등)`

데이터베이스 프로그래밍-PL/SQL Package 활용(10차시)

10-4 DBMS_RANDOM

C 언어의 random() 함수나 JAVA 언어의 java.util.Random Class 와 동일한 기능 즉 Random Number 를 생성하는 PACKAGE 입니다.

10_DBMS_RANDOM_1.SQL

```
REM -----
REM   랜덤 숫자 생성
REM -----

SELECT  DBMS_RANDOM.RANDOM,
        DBMS_RANDOM.VALUE,
        DBMS_RANDOM.VALUE(1,14)
FROM    EMP
WHERE   ROWNUM <= 5
/

COL  UPPER_COL          FORMAT A10
COL  LOWER_COL          FORMAT A10
COL  ALPHA_NUM_COL      FORMAT A20
COL  UPPER_ALPHA_NUM_COL FORMAT A20

REM -----
REM   랜덤 문자열 생성
REM -----

SELECT
    DBMS_RANDOM.STRING('U', 10) UPPER_COL,
    DBMS_RANDOM.STRING('L', 10) LOWER_COL,
    DBMS_RANDOM.STRING('A', 10) ALPHA_NUM_COL,
    DBMS_RANDOM.STRING('X', 10) UPPER_ALPHA_NUM_COL
FROM    EMP
WHERE   ROWNUM <= 5
/

REM -----
REM   랜덤 조회
REM -----

SELECT * FROM (
                SELECT  EMPNO,ENAME
                FROM    EMP
                ORDER   BY DBMS_RANDOM.VALUE
            )
WHERE   ROWNUM <=3
/
```

① RANDOM 한 숫자값을 생성하는 예제.

RANDOM 함수는 $-2^{31} \sim 2^{31}$ 사이의 BINARY INTEGER 를 리턴한다.

VALUE 함수는 0 ~ 1 사이의 값을 리턴 한다.

데이터베이스 프로그래밍-PL/SQL Package활용(10차시)

VALUE(low_bound,high_bound) 함수는 low_bound <= ~ <= high_bound 사이의 값을 리턴 한다.

- ② COL 은 SQL*PLUS 상에서 데이터 조회시 출력되는 컬럼의 FORMAT 을 지정하는 SQL*PLUS 명령어 이다. ③ 조회시 각 컬럼의 길이를 20 자로 제한 합니다. 상업용 TOOL 인 ORANGE 나 TOAD 에서 데이터 조회시에는 해당 TOOL 에서 출력 포맷을 정해 읽기해서 쉬운 형태로 보여지지만 SQL*PLUS 에서는 COL 명령어를 통해 읽기 쉬운 형태로 표현 한다. COL 명령어가 없다면 SQL*PLUS 상에서 ③번 SQL 실행시 가독성이 떨어지게 된다.

- ③ RANDOM 한 문자값을 생성하는 예제.
STRING 함수에서 'U'는 대문자 , 'L'는 소문자 ,

'A'는 Alphanumeric, 'X'는 대문자 Alphanumeric

10 은 랜덤 문자열의 길이를 지정 한다.

- ④ 랜덤 데이터 조회하는 예제.

ORDER BY 에 DBMS_RANDOM 을 사용하고 ROWNUM 을 사용하여 실행시 마다 3 개의 랜덤 데이터를 조회 하게 됩니다.

<참고>

위의 예제를 사용시 성능적인 관점에서 주의가 필요 하다. SUBQUERY 에서 전체 데이터를 DBMS_RANDOM.VALUE 값에 따라 정렬한후 MAIN-QUERY 에서 그중에 일부값을 사용하는 방식이기 때문에 SUB-QUERY 에서 접근하는 데이터가 대량인 경우 성능적인 관점에서 주의가 필요 하다

10-5 DBMS_ERRLOG

(DML 연산 실행시) ERROR 가 발생할 때 ERROR 관련된 사항을 LOG 로 기록한다
정리하면 10g R2에 새로 추가된 PACKAGE로 DML ERROR LOGGING 기능입니다.

많은 개발자들이 오랫동안 기다려 왔던 강력한 기능입니다.

OLTP Transaction 에서는 효용가치가 적지만 BATCH Transaction 에서는 매우 유용하게 사용 된다.

예를 들면 데이터베이스 업무 경험이 많으신분들은 BATCH Transaction 관련해서 누구나 이런 경험을 한번쯤은 또는 매우 빈번하게 경험하게 됩니다.

데이터베이스 프로그래밍-PL/SQL Package활용(10차시)

예1) BATCH 작업시

1천만건을 처리 BATCH Transaction 이 999만건 처리하는 도중에
공간 부족이나 데이터 오류,데이터 무결성으로 인해 전체 작업을 중지하고
재실행하는 경우

예2) DW의 STAGING 작업시

DW(데이터 웨어 하우스) 시스템 환경에서 운영계 데이터를 정보계 데이터로
이행하는 도중 일부 에러로 인해 전체 작업을 재시작하는 경우

예3) DB 마이그레이션 작업시

현행 디비(AS-IS) 와 목표 디비(TO-BE)의 설계가 달라져서
1개의 테이블이 N 개의 테이블로 데이터가 나뉘어져서 이관되거나
N개의 테이블 데이터가 1개의 테이블로 이관되기 위해서 SQL 로 이관 작업시
일부 데이터가 오류가 발생하는데 오류가 발생한 데이터를 추적하기 어려워서
이관 데이터 정합성에 많은 시간을 소모하는 경우

위의 3가지 사례에서 처럼 문제가 발생하면 그동안은 어떻게 해결해 왔을까?

여러가지 노가다 방법이 있지만 그나마 CURSOR를 사용하여 1 ROW씩 건건이 조건 처리나
검증 하면서 처리하는 방식이 있으나 대용량 데이터 처리시에는 성능적인 관점에서는 성능
목표를 충족 할 수가 없기에 문제 해결의 방안이 되지 못하였다.

따라서 문제가 되는 데이터를 찾기 위해 상당히 많은 시간이 소모되어 왔습니다.

DBMS_ERRLOG PACKAGE의 등장으로 인해

99%의 데이터를 처리하는 도중에 문제(ERROR)가 발생하는 일부 데이터를
기록(LOG 테이블에 INSERT)해 두고 나머지 작업은 계속 진행 한후
작업 종료후 LOG 테이블에 기록된 오류 데이터를 보정 하여 오류분만
재처리를 수행할수 있게 되었습니다.

매우 강력하고 필요한 새로운 기능(NEW FEATURE)이 PACKAGE 형태로 제공되었습니다.

자! 실습을 해보죠

데이터베이스 프로그래밍-PL/SQL Package 활용(10차시)

10_DBMS_ERRLOG.SQL

```
REM -----
REM      ERROR LOGGING 테이블 생성
REM -----

BEGIN
    DBMS_ERRLOG.CREATE_ERROR_LOG(dml_table_name=>'EMP',
                                err_log_table_name=>'ERR$_EMP',
                                err_log_table_owner=>'HR',
                                err_log_table_space=>'USERS');

END;
/

DESC ERR$_EMP

REM *****
REM      P.K 에 의한 Unique Key 오류 발생
REM *****

INSERT INTO EMP SELECT * FROM EMP WHERE EMPNO IN (7782,7839,7934);

INSERT INTO EMP SELECT * FROM EMP WHERE EMPNO IN (7782,7839,7934)
LOG ERRORS INTO ERR$_EMP ('Insert1_Test_Tag') REJECT LIMIT 2;

SELECT ORA_ERR_TAG$,EMPNO,ENAME,ORA_ERR_MSG$ FROM SCOTT.ERR$_EMP;

REM *****
REM      컬럼길이 초과 오류 발생
REM *****

INSERT INTO EMP
SELECT EMPNO+1000, ENAME,JOB||'X',MGR,HIREDATE,SAL,COMM,DEPTNO FROM EMP WHERE EMPNO IN (7782,7839,7934)
LOG ERRORS INTO ERR$_EMP('Insert2_Test_Tag') REJECT LIMIT UNLIMITED;

SELECT ORA_ERR_TAG$,EMPNO,ENAME,ORA_ERR_MSG$ FROM SCOTT.ERR$_EMP;
```

① Error Log Table 생성

파라미터 dml_table_name 은 DML 처리 대상 테이블

파라미터 err_log_table_name 은 Error Log 테이블 이름, 파라미터 생략시

Default로 'ERR\$_' + 'dml_table_name' 조합하여 Error Log 테이블 이름 지정

실제 사용시 대상 테이블명 만 기입하면 자동으로 ERR\$_ + 테이블명 즉

ERR\$_EMP 를 지정한다.

테이블명이 30자 인경우 1~25의 이름을 추출하여 ERR\$_를 접두어로 붙인다.

파라미터 err_log_table_owner는 Error Log 테이블의 소유자 ,생략시 현재 세션 Schema사용

파라미터 err_log_table_space는 Error Log 테이블이 생성될 위치(테이블스페이스명) 지정

위의 실행 결과 EMP 테이블에 대한 DML Error Logging을 위해 SCOTT 계정 소유의

ERR\$_EMP 테이블을 USERS 테이블스페이스에 생성 한다.

② 생성된 Error Logging 테이블 구조 조회

데이터베이스 프로그래밍-PL/SQL Package활용(10차시)

SQL> DESC ERR\$_EMP

| 이름 | 널? | 유형 |
|------------------|----|----------------|
| ORA_ERR_NUMBER\$ | | NUMBER |
| ORA_ERR_MSG\$ | | VARCHAR2(2000) |
| ORA_ERR_ROWID\$ | | ROWID |
| ORA_ERR_OPTYP\$ | | VARCHAR2(2) |
| ORA_ERR_TAG\$ | | VARCHAR2(2000) |
| EMPNO | | VARCHAR2(4000) |
| ENAME | | VARCHAR2(4000) |
| JOB | | VARCHAR2(4000) |
| MGR | | VARCHAR2(4000) |
| HIREDATE | | VARCHAR2(4000) |
| SAL | | VARCHAR2(4000) |
| COMM | | VARCHAR2(4000) |
| DEPTNO | | VARCHAR2(4000) |

① 영역은 발생한 에러 CODE 및 메시지 ,TAG 정보를 저장하며

② 영역을 제외한 나머지 모든 컬럼은 EMP 테이블내의 모든 컬럼을 그대로 가지고 있습니다. 모든 데이터를 표시 하기 위해 컬럼의 데이터 타입이 VARCHAR2(4000) 으로 일괄 생성 되어 있습니다.

SQL> DESC EMP

| 이름 | 널? | 유형 |
|----------|----------|--------------|
| EMPNO | NOT NULL | NUMBER(4) |
| ENAME | | VARCHAR2(10) |
| JOB | | VARCHAR2(9) |
| MGR | | NUMBER(4) |
| HIREDATE | | DATE |
| SAL | | NUMBER(7,2) |
| COMM | | NUMBER(7,2) |
| DEPTNO | | NUMBER(2) |

자! ERR\$_EMP 와 EMP의 구조를 비교해 보시기 바랍니다.

③ 우리가 그동안 일반적으로 사용하던 첫번째 INSERT~ SELECT는 P.K 중복으로 인해 에러가 발생 한다.

두번째 INSERT ~ SELECT 구문을 자세히 보면

LOG ERRORS INTO ERR\$_EMP ('Insert1_Test_Tag') REJECT LIMIT 2 구문이 추가되

었다. 해석을 해보면 - DML 연산시 에러가 발생하면 ERR\$_EMP 테이블이 기록하

- 에러기록 태그로 'Insert1_Test_Tag' 를 사용하라

- 에러의 개수가 2개의 제한(LIMIT)을 초과하면

진행중인 트랜잭션을 중지하고 ROLLBACK 처리하라

<참고> 태그의 역할은 식별성!

동일하거나 유사한 작업을 여러 번 반복 수행할 때 마다 발생하는

데이터베이스 프로그래밍-PL/SQL Package활용(10차시)

에러 로그에 구분자를 붙여 식별성을 부여 할수 있습니다.>

< 참고> REJECT LIMIT 구문을 자세히 들여다 봅시다.,

```
LOG ERRORS INTO ERR$_EMP ('Test_Tag') REJECT LIMIT 2;
```

➔ ERROR 로그를 기록하고 에러가 2개 이상인 경우 TRANSACTION을
ROLLBACK처리

```
LOG ERRORS INTO ERR$_EMP ('Test_Tag')
```

➔ REJECT LIMIT 생략시 DEFAULT는 REJECT LIMIT 1

```
LOG ERRORS INTO ERR$_EMP ('Test_Tag2') REJECT UNLIMIT;
```

➔ 발생하는 모든 에러를 개수에 제한 없이 기록하라는 의미

REJECT 의 역할은 에러가 일정 개수 이상이 되면 TRANSACTION을 취소처리
(ROLLBACK) 하는 제한 역할

④ 7782,7839,7934 사원의 데이터는 아래와 같다.

| EMPNO | ENAME | JOB |
|-------|--------|-----------|
| 7782 | CLARK | MANAGER |
| 7839 | KING | PRESIDENT |
| 7934 | MILLER | CLERK |

INSERT ~ SELECT 구문에서

- EMPNO+1000 는 P.K 에의한 Unique Key 오류가 발생하지 않도록 EMPNO + 1000 수행

- JOB 컬럼의 정의는 JOB VARCHAR2(9) 되어 있는 상황에서

JOB||'X' 연산은 두번째 ROW에서 자리수 초과 오류 발생

즉 처리 대상 3개의 ROW 중 2번째 ROW에서 자리수 초과 오류 발생

REJECT LIMIT UNLIMITED 구문에 의해

에러가 발생하지 않은 모든 데이터는 정상 처리

에러가 발생한 데이터는 에러 테이블에 기록

Insert2_Test_Tag 태그를 사용하여 이전 ③ 의 작업과 에러 로그 데이터 구분

SELECT ORA_ERR_TAG\$,EMPNO,ENAME,ORA_ERR_MESG\$ FROM SCOTT.ERR\$_EMP;를
통해 에러가 발생한 데이터 조회

SELECT EMPNO,JOB FROM EMP WHERE EMPNO IN (8782,8839,8934); 를 통해
정상 처리된 2개의 데이터 조회

데이터베이스 프로그래밍-PL/SQL Package활용(10차시)

[학습정리]

1. ORACLE 제공 PACKAGE는 SYS 계정 소유이며 DBMS 운영 및 보안성에 영향을 주는 PACKAGE를 제외하고 대부분은 모든 계정에서 사용 가능하다. 일반 계정에서 사용 불가능한 PACKAGE는 필요시 DBA가 해당 PACKAGE에 대한EXECUTE 권한을 부여하면 사용이 가능 해진다.
2. UTL_FILE PACKAGE는 ORACLE DBMS 가 설치된 SERVER 상의 TEXT FILE을 생성/읽기/쓰기/삭제 및 복사/이동등이 가능하다.
3. DBMS_RANDOM은 임의의 RANDOM 숫자나 문자 값을 생성하는 PACKAGE 이다.
4. DBMS_ERRLOG는 DML 연산시 에러가 발생한 원인을 쉽게 찾을수 있도록 하는 PACKAGE,로 특히 배치처리시 매우 유용한 PACKAGE 이다.