

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

7차시: PL/SQL Module(Stored Block 2)

I. 시작

[학습목표]

PL/SQL은 Anonymous Block 과 Stored(Named) Block 2가지 유형의 Block이 있습니다. 7차시에는 Stored Block을 활용하여 사용자 정의Function, Procedure, Package, Trigger 생성하는것을 학습합니다.

[학습목차]

7-1 VARIABLE & PRINT

7-2 PROCEDURE

7-3 PROCEDURE 와 실행권한(EXECUTE PRIVILEGE)

<참고> 바인드 변수 (BIND VARIABLE= HOST VARIABLE = GLOBAL VARIABLE)

다음은 PRO*C의 Sample Source 입니다.

1

```
for (;;) /* Loop infinitely */
{
    printf("\n** Debit which account number? (-1 to end) ");
    gets(buf);
    acctnum = atoi(buf);
    if (acctnum == -1) /* Need to disconnect from ORACLE */
    {
        EXEC SQL COMMIT RELEASE;
        exit(0);
    }
    printf("    What is the debit amount? ");
    gets(buf);
    debit = atof(buf);
    /* ----- Begin the PL/SQL block ----- */
    /* ----- Begin the PL/SQL block ----- */
    EXEC SQL EXECUTE
```

2

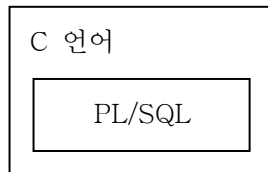
```
DECLARE
    insufficient_funds EXCEPTION;
    old_bal NUMBER;
    min_bal CONSTANT NUMBER := 500;
```

3

```
BEGIN
    SELECT bal INTO old_bal FROM accounts
    WHERE account_id = :acctnum;
    -- If the account doesn't exist, the NO_DATA_FOUND
    -- exception will be automatically raised.
    :new_bal := old_bal - :debit;
    IF :new_bal >= min_bal THEN
        UPDATE accounts SET bal = :new_bal WHERE account_id = :acctnum;
        INSERT INTO journal VALUES (:acctnum, 'Debit', :debit, SYSDATE);
        :status := 'Transaction completed.';
    ELSE
        RAISE insufficient_funds;
    END IF;
    COMMIT;
EXCEPTION
    WHEN NO_DATA_FOUND THEN
        :status := 'Account not found.';
```

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

- ① Pro*C는 C 언어 안에 PL/SQL BLOCK이 포함되어 있는 구조로



C 언어는 주인이 되는 언어라고 해서 Host Language
PL/SQL은 전세사는는 언어 라고해서 Embedded Language

- ② PL/SQL BLOCK 내의 선언부에서 변수를 선언

- ③ :new_bal := old_bal - :debit;

연산에 사용된 변수중 :new_bal 와 :debit 는

㉠ 변수명 앞에 : (콜론)

㉡ PL/SQL Block의 외부인 C 언어에서 선언된 변수.

```
int      acctnum;  
double   debit;  
double   new_bal;
```

:new_bal,:debit는 Host Language 에서 선언한 변수 여서 Host Variable(호스트 변수) 라고 호칭 한다.

Global Variable(이하 글로벌 변수)라고 부르는 이유?

PL/SQL블록에서 선언한 변수는 Local Variable(이하 로컬변수)가 되고 Pro*C에서 선언한 변수는 Global Variable이 되는 것 이다. 글로벌 변수와 로컬변수를 구분 하기 위해서 세미콜론(:)을 사용하여 구분 한다.

<참고>

명칭의 유래 보다는 바인드변수의 용도는 무엇인가?

주인과 세입자간의 의사 소통 용도 인거죠

Host Language(주인) 과 Embedded Language(세입자)간의 데이터 교환 (의사 소통)을 위한 용도.

7-1 VARIABLE & PRINT

SQL*PLUS를 PL/SQL 프로그램 개발 및 테스트 TOOL로 사용

- ① VARIABLE & PRINT

- ② DBMS_OUTPUT

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

VARIABLE

- (1) PL/SQL BLOCK에서 참조 할수 있는 BIND VARIABLE(바인드변수) 선언
- (2) 선언된 바인드 변수 조회

PRINT

BIND VARIABLE(이하 바인드변수)의 값을 SQL*PLUS 화면에 출력 하는 기능

7_BIND_1.SQL

```
REM -----
REM 변수선언 : 바인드변수=호스트변수=글로벌 변수
REM -----

VARIABLE H_SALARY      NUMBER
VARIABLE H_TAX         NUMBER

DECLARE
    C_TAX_RATE          NUMBER(2,3);
BEGIN
    C_TAX_RATE := 0.05;      -- 근로소득세율 (Local Variable)
    :H_SALARY := 1000;      -- 급여 (Host Variable)

    -- 근로소득세 계산, PL/SQL BLOCK내의 SQL 함수 사용
    :H_TAX := ROUND(:H_SALARY * C_TAX_RATE,2);
END;
/

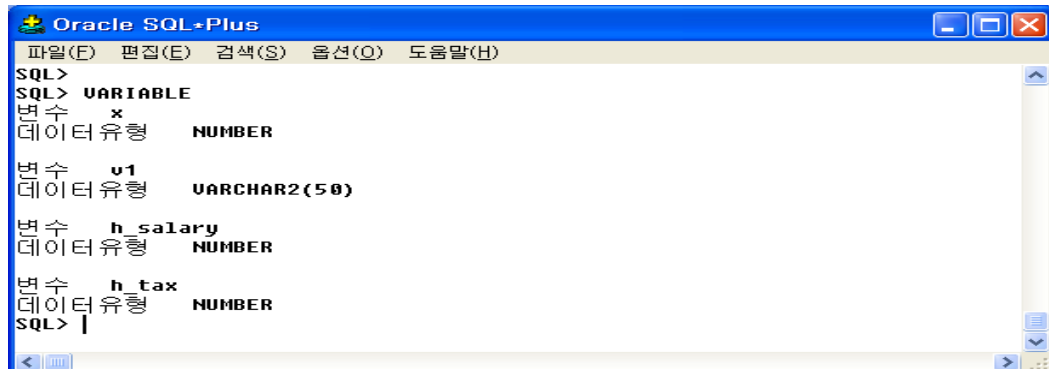
REM -----
REM 변수출력 : 바인드변수=호스트변수=글로벌 변수
REM -----

PRINT H_SALARY
PRINT H_TAX
```

- ① PL/SQL BLOCK 외부에서 NUMBER TYPE의 2개 변수를 선언
PL/SQL BLOCK에서는 호스트변수,글로벌 변수에 해당하는 변수
- ② PRO*C 예제를 참고하면 Embedded PL/SQL에서 Host 언어인 C에서
정의한 변수를 PL/SQL BLOCK 내부에서는 변수명 앞에 : (세미콜론)을 붙여
참조(사용) 했던 방식과 동일하게 사용
- ③ PL/SQL BLOCK내에서 처리되어 호스트 변수에 반영된 결과를 BLOCK 밖에서
PRINT 명령어를 통해서 확인

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

<참고>



```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL>
SQL> VARIABLE
변수      x
데이터 유형   NUMBER

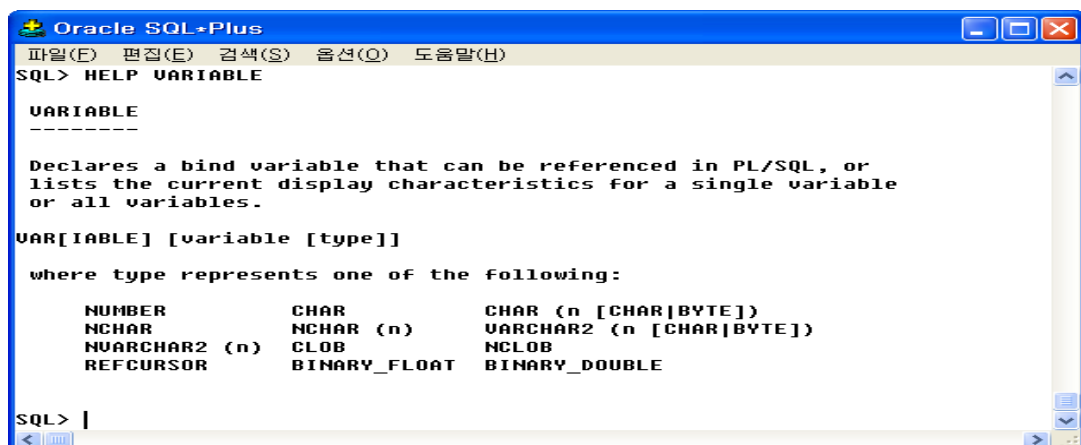
변수      v1
데이터 유형   VARCHAR2(50)

변수      h_salary
데이터 유형   NUMBER

변수      h_tax
데이터 유형   NUMBER
SQL> |
```

<참고> SQL*PLUS 명령어들은 HELP 라는 명령어를 통해서 간단한 설명 조회
VARIABLE,PRINT는 SQL*PLUS 명령어이므로 HELP 사용 가능

1) VARIABLE 명령어의 도움말 조회



```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL> HELP VARIABLE

VARIABLE
-----

Declares a bind variable that can be referenced in PL/SQL, or
lists the current display characteristics for a single variable
or all variables.

VAR[IABLE] [variable [type]]

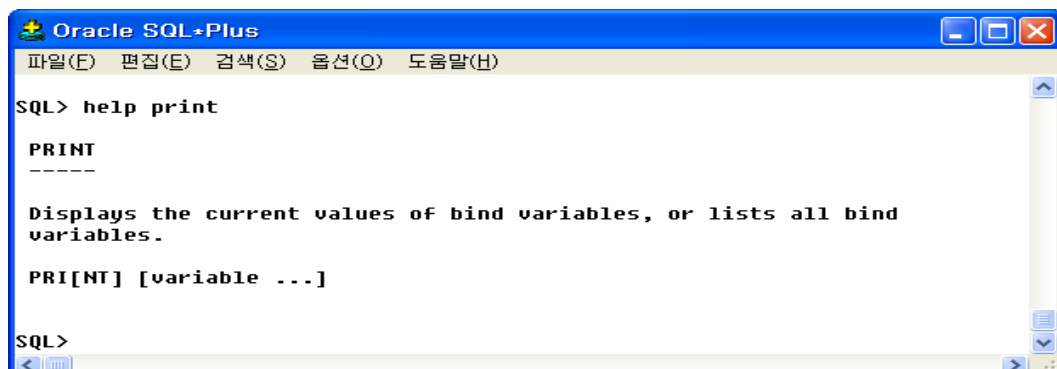
where type represents one of the following:

      NUMBER          CHAR          CHAR (n [CHAR|BYTE])
      NCHAR           NCHAR (n)      VARCHAR2 (n [CHAR|BYTE])
      NVARCHAR2 (n)   CLOB           NCLOB
      REFCURSOR       BINARY_FLOAT  BINARY_DOUBLE

SQL> |
```

문법 및 사용가능한 데이터 타입 조회,NUMBER 유형을 주의 깊게 보면
NUMBER 유형에 데이터의 길이를 표시할수 없습니다.

2) PRINT 명령어의 도움말 조회



```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL> help print

PRINT
-----

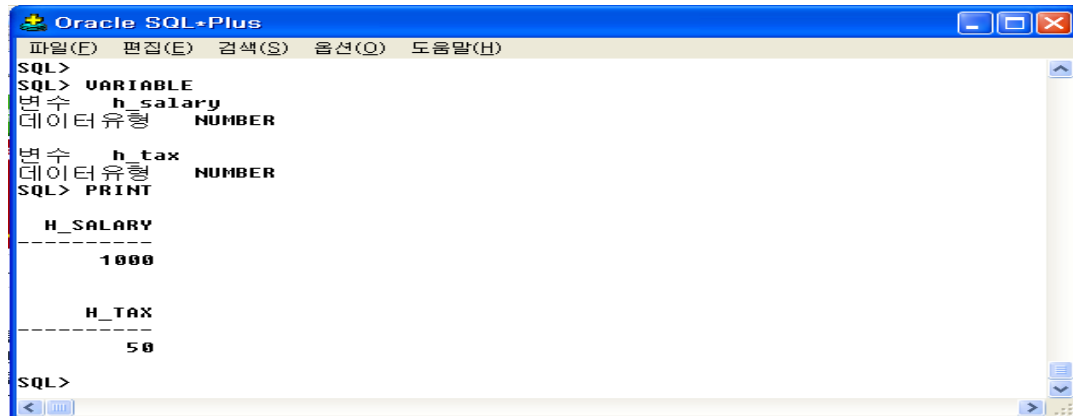
Displays the current values of bind variables, or lists all bind
variables.

PRI[NT] [variable ...]

SQL>
```

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

<참고> VARIABLE,PRINT 명령어뒤에 특정 변수명을 입력하지 않는 경우에는 선언된 모든 변수의 리스트와 값을 출력



```
Oracle SQL*Plus
파일(F) 편집(E) 검색(S) 옵션(O) 도움말(H)
SQL>
SQL> VARIABLE
변수      h_salary
데이터 유형  NUMBER
변수      h_tax
데이터 유형  NUMBER
SQL> PRINT
H_SALARY
-----
      1000

H_TAX
-----
       50

SQL>
```

[정리]

VARIABLE,PRINT는 SQL*PLUS를 개발툴로 만들어주는 중요한 명령어로
PRO*C 나 JAVA에서 사용할 ANONYMOUS BLOCK을 만들거나
Stored FUNCTION/PROCEDURE/PACKAGE를 만들어야 한다고 가정하면

먼저 JAVA 개발툴이나 PRO*C 개발툴을 띄워놓고 개발을 시작하는건 아니겠죠
SQL*PLUS에서 해당 PL/SQL BLOCK을 개발/테스트를 하는것이 효율적 입니다.
다만 문제가 되는 것은 JAVA나 PRO*C 와 주고 받을 데이터가 문제인거죠!!!
Stored FUNCTION을 개발한다면 JAVA나 PRO*C 프로그램이 완성되지 않은 상태에
서는 테스트시에 입력 파라미터/테스트 결과값을 확인하는게 어렵습니다.
위의 이런 문제를 VARIABLE과 PRINT 명령어를 통해 해결할수 있습니다.
SQL*PLUS를 개발툴로 사용할수 있는 익혀두셔야할 명령어들 입니다.
SQL*PLUS를 JAVA나 PRO*C 로 개발중인 Application Program 이라고 생각하고
개발할수 있게 합니다.

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

7-2 PROCEDURE

7-2-1 PROCEDURE 생성

7_PROCEDURE_1.SQL

```
REM -----
REM  PROCEDURE 생성   :  CHANGE_SALARY
REM -----

CREATE OR REPLACE PROCEDURE  CHANGE_SALARY(A_EMPNO IN NUMBER , A_SALARY NUMBER DEFAULT 2000 )
AS
BEGIN
    UPDATE  EMP
    SET      SAL      = A_SALARY
    WHERE    EMPNO    = A_EMPNO;

    COMMIT;
END  CHANGE_SALARY;
/

DESC      CHANGE_SALARY

REM -----
REM  PROCEDURE 테스트 1
REM -----

VARIABLE  P_EMPNO  NUMBER
VARIABLE  P_SALARY NUMBER

BEGIN
    :P_EMPNO  := 7369;
    :P_SALARY := 7369;

    CHANGE_SALARY(:P_EMPNO, :P_SALARY);
END;
/

SELECT  EMPNO, SAL  FROM  EMP WHERE EMPNO = 7369;

REM -----
REM  PROCEDURE 테스트 2
REM -----

EXECUTE  CHANGE_SALARY(:P_EMPNO);

SELECT  EMPNO, SAL  FROM  EMP WHERE EMPNO = 7369;
```

① OR REPLACE는 선택적 구문이지만 PL/SQL개발자들은 편리성으로 습관적 사용.

② PROCEDURE에 정의된 2번째 매개변수(파라미터)

A_SALARY NUMBER DEFAULT 2000

㉠ 파라미터 모드가 생략 되면 ?

㉡ DEFAULT 2000의 의미는?

③ 실행부의 내용은 입력받은 사번 과 변경하려는 급여를 가지고

UPDATE 연산 수행후 COMMIT 하는 간단한 처리.

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

④ DESC 를 사용하여 PROCEDURE의 파라미터 구성 정보를 조회할수 있다.

⑤ EXECUTE의 용도는 무엇일까요 ?

STORED BLOCK 테스트 및 실행을 간편하게 할수 있도록 하는 명령어

EXECUTE의 정체는?

SQL*PLUS 명령어로 SQL*PLUS가 BEGIN ~ END; 구문으로 바꾸어서
ORACLE DBMS에 전송

⑥ 생략된 두번째 파라미터 자리에는 DEFAULT로 정의된 2000이 사용 된다.

<참고> PROCEDURE / FUNCTION / PACKAGE를

① 생성할때는 CREATE 를 사용합니다.

② 삭제할때는 DROP을 사용합니다.

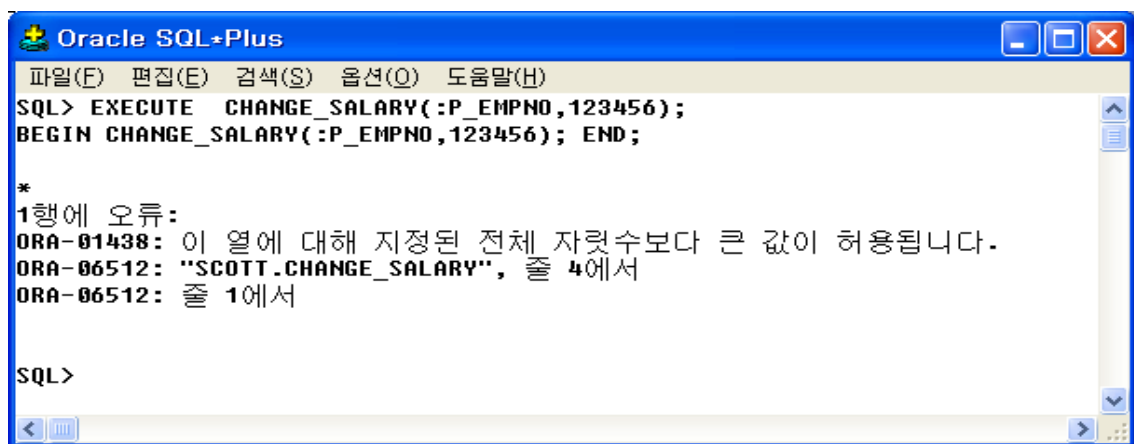
예를들면 DROP PROCEDURE CHANGE_SALARY;

③ 변경할때는 ALTER를 사용하는 것이 아니라 OR REPLACE를 사용.

7-2-2 예외처리(Exception) 구문을 PROCEDURE에 추가

이전 차수에 생성한 PROCEDURE를 실행 시켜 봅시다.

EXECUTE CHANGE_SALARY(:P_EMPNO,123456);



EMP.SAL은 NUMBER(7,2)로 정의 되어 있는 상황에서 실행시점에 정의된 자리수를 초과하는 에러가 발생한다.

EXCEPTION 처리 영역이 선택적 영역이기는 하지만 사용하지 않는 경우는

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

실행시점에 예상치 못한 결과를 초래하여 문제를 발생시킬 수 있다.

7_PROCEDURE_2.SQL

```
REM -----
REM  PROCEDURE 생성 : CHANGE_SALARY + EXCEPTION
REM -----

CREATE OR REPLACE PROCEDURE CHANGE_SALARY(A_EMPNO IN NUMBER , A_SALARY NUMBER DEFAULT 2000 )
AS
BEGIN
    UPDATE EMP
    SET     SAL      = A_SALARY
    WHERE  EMPNO    = A_EMPNO;

    COMMIT;

EXCEPTION
    WHEN VALUE_ERROR THEN
        DBMS_OUTPUT.PUT_LINE('VALUE_ERROR => '||SQLERRM);
        NULL;

    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('NO_DATA_FOUND => '||SQLERRM);
        ROLLBACK;

    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('OTHERS      =>: '||SQLERRM);
        ROLLBACK;
END CHANGE_SALARY;
/

REM -----
REM  PROCEDURE 테스트
REM -----

EXECUTE CHANGE_SALARY(A_EMPNO => 7369 , A_SALARY => 1234567);
```

- ① “ ORA-01438: 지정한 정도를 초과한 값이 열에 지정되었습니다 “
위의 에러는 PRE-DEFINED EXCEPTION이 아닙니다. 즉 EXCEPTION의 이름을 가지고 있지 않습니다.

예외처리부(EXCEPTION SECTION)에서 3개의 예외를 정의합니다.

VALUE_ERROR 나 NO_DATA_FOUND는 1438 에러와는 아무런 관계가 없는 EXCEPTION 입니다. 예제를 위한 예제 입니다.

OTHERS 에서 위의 EXCEPTION이 처리 됩니다.처리되는 내용을 보면

- ㉠ EXCEPTION 발생 사실을 사용자에게 알립니다.
- ㉡ ROLLBACK 을 처리를 합니다.

- ㉢ EXCEPTION 발생 사실을 사용자에게 알리는데

개발시점에서는 DBMS_OUTPUT을 통해 ERROR MESSAGE를 화면에 출력하면서 디버깅을 합니다. 개발이 완료되면 Stored Procedure 내에서 DBMS_OUTPUT 구문을 삭제 해야 합니다.

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

운영시점에서는 어떻게 사용자에게 알릴까요? DBMS_OUTPUT을 운영시점에 사용하기에 적절하지 않기 때문에 아래의 2가지 방법을 사용합니다.

- LOG TABLE에 EXCEPTION의 발생 사항을 INSERT 처리
 - LOG FILE에 EXCEPTION의 발생 사항을 기록
- 하는 2가지 방법을 사용합니다.

㉞ EXCEPTION 발생을 TRANSACTION을 명시적으로 종료 처리를 한다.

② 파라미터 변수에 값을 Mapping 하는 2가지 방법을 보았습니다.

㉠ 위치에 의한 방법 : 파라미터가 정의된 위치에 값을 지정(실습예제)

㉡ 지정에 의한 방법 : 파라미터의 이름 과 값을 Mapping 하는 방법
EMPNO => 7369)

[질문] 데이터 처리가 안되었습니다.

실행시 발생한 에러가 나타나지 않는 이유?

- ➔ 예외처리부(EXCEPTION SECTION)로 예외처리를 했기 때문
- DBMS_OUTPUT의 결과가 화면에 나타나지 않는 이유?
- ➔ SET SERVEROUTPUT ON

7-2-3 PROCEDURE 생성 실습 3

PROCEDURE에 절차적 언어의 여러가지 특징중 변수선언, 조건처리를 추가해봅시다. 실습예제의 의미는 중요 하지 않습니다. 예제를 위한 예제 입니다.

이전 차시에 학습한 변수 및 제어처리 기능을 PROCEDURE에 추가하여 파라미터로 입력된 값에 대한 조건처리 및 연산을 수행하는 것을 보여주는 것이 실습의 목적 입니다.

7_PROCEDURE_3.SQL

입력된 금액이 (1) 평균 이하의 금액이면 2%를 추가 보정

(2) 금액 상한액(90000) 을 초과하면 오류로 -99999 값 입력

(3) 금액에 NULL이 들어오면 0 으로 치환

DEFAULT의 의미는 암시적인 NULL이 지정되면 Default 로 지정된 값이므로 치환하라는 의미 입니다. 암시적으로 NULL을 지정하는 방법은 해당 모듈 호출시 해당 파라미터를 생략하면 됩니다.

명시적으로 NULL이 지정되는 경우에는 DEFAULT가 적용되지 않습니다.

명시적으로 NUL을 지정하는 방법은 ? A_SALARY => NULL 표기법 사용

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

```
CREATE OR REPLACE PROCEDURE CHANGE_SALARY(A_EMPNO IN NUMBER , A_SALARY NUMBER DEFAULT 2000 )
AS
    V_AVG_SALARY      EMP.SAL%TYPE;      -- 평균 급여
    V_SALARY           EMP.SAL%TYPE;      -- 급여
    V_MAX_SALARY       EMP.SAL%TYPE := 90000; -- 급여 상한액
    V_ADD_SALARY_RATE  NUMBER           := 0.02 ; -- 급여 보정 비율
BEGIN
    -- 평균 급여 산출
    SELECT  AVG(SAL) INTO  V_AVG_SALARY FROM EMP;

    IF  A_SALARY < V_AVG_SALARY THEN  -- 평균이하의 급여인 경우  2% 추가 보정
        V_SALARY :=      A_SALARY  + ROUND(A_SALARY * V_ADD_SALARY_RATE,1);

    ELSIF A_SALARY > 99999 THEN      -- 급여상한액을 초과하는 경우  Error로 -99999 입력
        V_SALARY :=  -99999 ;

    ELSIF A_SALARY IS NULL THEN      -- NULL 인경우  0으로 치환
        V_SALARY := 0;
    ELSE
        V_SALARY := A_SALARY;
    END IF;

    UPDATE  EMP
    SET      SAL      = V_SALARY
    WHERE    EMPNO    = A_EMPNO;

    COMMIT;

EXCEPTION
    WHEN  OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('OTHERS      => '||SQLERRM);
        ROLLBACK;
END  CHANGE_SALARY;
/
```

- ① 선언부에서 참조연산자(%TYPE)을 사용하여 간단하게 변수 선언
- ② 실행부에서 SELECT 와 AVG함수를 사용하여 평균 급여 조회
- ③ 실행부에서 IF 조건문을 사용하여 제어 처리
ELSIF 에 IS NULL 이라는 SQL 연산자 사용

<참고> SQL 연산자 와 SQL 함수를 사용할수 있다는 점은 단순하지만 기억을 해둘 사항입니다. PL/SQL로 개발시 논리 표현이 용이 합니다.

예를 들면 A_SALARY BETWEEN 0 AND 99998 등의 표현은
개발시 코드의 단순성 과 명료성을 줍니다.

```
EXECUTE  CHANGE_SALARY(A_EMPNO => 7369 , A_SALARY => 1234567);
```

를 실행하게 되면 자리수를 초과 하게 되지만 PROCEDURE내부의 조건 처리로
인해서 데이터가 보정되어 처리가 되는 것을 확인 할수 있습니다.

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

7-2-4 PROCEDURE 생성 실습 4

PL/SQL을 프로젝트에서 원활히 활용하는 경우에는 많게는 수백개~수천개의 PL/SQL BLOCK들이 사용 됩니다. PL/SQL BLOCK이 또다른 PL/SQL BLOCK을 호출하는 경우가 빈번히 있습니다. PL/SQL을 활용하여 복잡하게 개발된 프로젝트에서 운영중에 EXCEPTION이 발생하면 발생 원인을 찾아 문제를 해결하는데 시간이 많이 소요 되는 경우가 종종 있습니다.

이유는 EXCEPTION 발생시 로그를 남기지 않기 때문입니다.

로그를 남기는 방법은 2가지 입니다.

㉠ 관계형 데이터베이스의 테이블에 기록

㉡ 운영체제의 파일에 기록

현 시점에서는 ㉠에 대해 실습을 합니다.

㉡에 대해서는 UTL_FILE 을 사용하여 프로젝트를 수행합니다.

(Oracle 社 제공 Package) 실습을 하게 됩니다.

예제는 EXCEPTION 로그에 집중하기 위해 이전 실습 스크립트에서 변수선언 및 조건 처리 로직을 삭제했습니다. 7_PROCEDURE_4.SQL

```
REM -----
REM EXCEPTION 발생을 기록하는 LOG 테이블
REM -----

CREATE TABLE EXCEPTION_LOG
(
  LOG_DATE      VARCHAR2(8)  DEFAULT TO_CHAR(SYSDATE, 'YYYYMMDD'), -- 로그 기록 일자 YYYYMMDD
  LOG_TIME      VARCHAR2(6)  DEFAULT TO_CHAR(SYSDATE, 'HH24MISS'), -- 로그 기록 시간 HH24MISS
  PROGRAM_NAME  VARCHAR2(100), -- EXCEPTION 발생 프로그램
  ERROR_MESSAGE VARCHAR2(250), -- EXCEPTION MESSAGE
  DESCRIPTION   VARCHAR2(250) -- 비고 사항
);

CREATE OR REPLACE PROCEDURE CHANGE_SALARY(A_EMPNO IN NUMBER , A_SALARY NUMBER DEFAULT 2000 )
AS
  V_ERROR_MESSAGE EXCEPTION_LOG.ERROR_MESSAGE%TYPE; -- 에러메세지
BEGIN
  UPDATE EMP SET SAL = A_SALARY WHERE EMPNO = A_EMPNO;
  COMMIT;

  EXCEPTION
    WHEN OTHERS THEN
      ROLLBACK;

      BEGIN -- EXCEPTION을 LOG 테이블에 기록
        V_ERROR_MESSAGE := SQLERRM;
        INSERT INTO EXCEPTION_LOG(PROGRAM_NAME, ERROR_MESSAGE, DESCRIPTION)
          VALUES('CHANGE_SALARY', V_ERROR_MESSAGE,
            'VALUES : [1]>'||A_EMPNO||' [2]>'||A_SALARY);

        COMMIT;
      EXCEPTION
        WHEN OTHERS THEN
          NULL;

      END;
END CHANGE_SALARY;
/

REM -----
REM PROCEDURE 테스트
REM -----

EXECUTE CHANGE_SALARY(A_EMPNO => 7369 , A_SALARY => 1234567);

SELECT * FROM EXCEPTION_LOG;
```

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

① 아래의 내용을 기록하는 로그 테이블을 생성 합니다

EXCEPTION이 발생한 일자 와 시간

발생한 프로그램명 OR 프로그램 ID

EXCEPTION 메시지 및 기타 사항 기록

컬럼 정의시 DEFAULT는 INSERT시 해당 컬럼을 생략하면 자동으로 채우는 값을 나타냅니다. 발생 일자 와 발생 시간은 자동으로 기록 됩니다.

잠시 ② 단락의 INSERT 문장을 보시면 INSERT 문에서 LOG_DATE, LOG_TIME 컬럼이 생략 되어 있는 것을 알수 있습니다. INSERT문이 실행되는 날짜 와 시간이 기록 되겠죠~ 즉 LOG에 기록하는 날짜와 시간이 자동으로 저장 됩니다.

② ROLLBACK은 이전 실패한 트랜잭션에 대한 명시적 종료 입니다.

예외처리부안에 NESTED BLOCK을 정의한 것 보이시나요 ?

NESTED BLOCK 안에도 예외처리부를 명시했죠!

EXCEPTION을 기록하는 INSERT 문 실행시에도 예러 즉 EXCEPTION이 발생할수 있죠! 2중으로 안정 장치 입니다.

EXCEPTION이 발생한 프로그램 이름, 메시지, 기타 사항을 INSERT 합니다.

INSERT 후 해당 내용을 기록하기 위해 COMMIT을 수행합니다.

< 참고>

(1) 수백개의 PL/SQL BLOCK을 사용하는 개발 현장에서 EXCEPTION을 기록하는 로직을 매번 아래의 예제처럼 BEGIN ~ END로 만들어야 할까요 ? 어떤 방법이 있을까요 ?

(2) NESTED BLOCK내의 EXCEPTION을 보면 OTHERS 구문에 NULL 명령어가 사용되었습니다. NULL 명령어는 PL/SQL 명령어중 하나 입니다.데이터 처리시의 NULL 과는 동일한 철자 이지만 의미가 다릅니다. 아무런일을 하지 않고 자리만 지키는 명령어 입니다. EXCEPTION을 OTHERS에서 받아서 외부로 전파 하지 않기 위해 EXCEPTION절을 구성해야 하는데 처리해야 할일이 아무것도 없을 때 자리만 지키는 명령어 입니다. 간혹 요긴하게 사용합니다.

(3) 약간 어려운 내용이지만 이해할 필요가 있는 내용입니다.

위의 예제에서는 2개의 트랜잭션이 종료가 됩니다.

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

UPDATE 시 EXCEPTION이 발생 (TRANSACTION 시작 OR 진행중)
ROLLBACK으로 이전 트랜잭션 종료

INSERT를 로그테이블에 수행 (새로운 TRANSACTION 시작)
COMMIT 으로 트랜잭션 종료

만약 UPDATE시 에러가 나면 트랜잭션을 종료 하지 않고 PROCEDURE
를 호출한 상위 모듈로 EXCEPTION을 전파하여 상위 모듈에서
COMMIT 이나 ROLLBACK 여부를 결정해야 하는 상황이라면
즉 해당 PROCEDURE내 에서는 트랜잭션을 제어하지 못하는 경우이므로
INSERT ~ EXCEPTION_LOG 를 수행후 COMMIT을 하지 않은 상태에서
상위 모듈이 ROLLBACK을 한다면 EXCEPTION 기록도 ROLLBACK
되겠죠! 에러에 대한 기록을 하지 못하게 됩니다.
방법 및 문제점을 고민 해보시기 바랍니다.

③ 실행시 발생한 EXCEPTION(RUN TIME ERROR)이 로그 테이블에 기록됩니다.

실행 결과 입니다. 우리는 위의 결과 보다는 EXCEPTION_LOG 테이블의 내용에
더 관심이 갑니다.

EXCEPTION_LOG 테이블을 조회 해봅시다.

SQL*PLUS상에서는 조회시 조회되는 ROW의 길이가 길어서 상용툴에서
SELECT * FROM EXCEPTION_LOG 를 실행한 결과 입니다.

LOG_DATE	LOG_TIME	PROGRAM_NAME	ERROR_MESSAGE	DESCRIPTION
▶ 20090424	130127	CHANGE_SALARY	ORA-01438: 이 열에 대해 지정된 전체 자릿수보다 큰 값이 허용됩니다.	VALUES : [1]=>7369 [2]=>1234567

로그를 정밀하게 남기게 된다면 오류시 문제의 원인에 빠른 접근을 통해 빠른
해결 조치가 가능 하게 됩니다.>

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

7-2-5 PROCEDURE 생성 실습 5

EXCEPTION 로그를 기록하는 기능을 프로젝트내의 모든 PL/SQL BLOCK 에서 공통적으로 사용할수 있도록 공통 MODULE 로 작성하여 호출 해봅시다.

7_PROCEDURE_5.SQL

```
REM -----
REM EXCEPTION을 기록하는 WRITE_LOG PROCEDURE 생성
REM -----

CREATE OR REPLACE PROCEDURE
  WRITE_LOG(A_PROGRAM_NAME IN VARCHAR2,A_ERROR_MESSAGE IN VARCHAR2,A_DESCRIPTION IN VARCHAR2)
AS
BEGIN
  -- EXCEPTION을 LOG 테이블에 기록
  INSERT INTO EXCEPTION_LOG(PROGRAM_NAME,ERROR_MESSAGE,DESCRIPTION)
    VALUES(A_PROGRAM_NAME,A_ERROR_MESSAGE,A_DESCRIPTION);
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    NULL;
END;
/

CREATE OR REPLACE PROCEDURE CHANGE_SALARY(A_EMPNO IN NUMBER , A_SALARY NUMBER DEFAULT 2000 )
AS
  V_ERROR_MESSAGE          EXCEPTION_LOG.ERROR_MESSAGE%TYPE; -- 에러메세지
BEGIN
  UPDATE EMP SET SAL = A_SALARY WHERE EMPNO = A_EMPNO;
  COMMIT;
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
    WRITE_LOG('CHANGE_SALARY',SQLERRM,'VALUES : [1]>'||A_EMPNO||' [2]>'||A_SALARY);
END CHANGE_SALARY;
/

REM -----
REM PROCEDURE 테스트
REM -----

EXECUTE CHANGE_SALARY(A_EMPNO => 7369 , A_SALARY => 987654321);
SELECT EMPNO,SAL FROM EMP WHERE EMPNO = 7369;
```

- ① WRITE_LOG 라는 공통 MODULE 을 생성합니다.
- ② 공통 MODULE 로 생성되어 모든 PL/SQL BLOCK 에서 간편하게 호출하여 사용이 가능 합니다. 사용자가 정의한 PROCEDURE(CHANGE_SALARY)가 사용자 정의 PROCEDURE(WRITE_LOG) 를 호출하는 관계 입니다.

<참고>

호출관계가 약간 복잡해졌습니다.

호출관계에 따르는 트랜잭션 문제를 잠깐 생각해봅시다.

CHANGE_SALARY ➔ WRITE_LOG ➔ EXCEPTION_LOG(테이블)

WRITE_LOG PROCEDURE를 보면 EXCEPTION_LOG 테이블에 INSERT 를 하고 나서 COMMIT을 수행합니다. 트랜잭션이 명시적으로 종료가 되는거죠!

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

CHANGE_SALARY에서 EXCEPTION이 발생하여 WRITE_LOG를 호출(CALL) 합니다.
호출한 Main BLOCK내에서는 트랜잭션의 종료 여부에 상관없이 호출된 Sub Block내에서 트랜잭션의 시작/종료 수행이 필요한 상황입니다.

즉 실행시점에

- (1) 메인 BLOCK에서의 트랜잭션의 종료가 호출된 서브 BLOCK의 DML연산에 영향을 주게 되거나
- (2) 서브 BLOCK의 트랜잭션 종료가 메인BLOCK의 트랜잭션에 영향을 주게 됩니다.

메인 BLOCK과 서브 BLOCK의 트랜잭션이 상호 영향을 주지 않는 방법이 없었지만 Oracle 8i 이후 버전부터 AUTONOMOUS TRANSACTION 이라는 개념을 지원 하면서 상호 독립적인 트랜잭션 처리가 가능하게 되었습니다.

.AUTONOMOUS는 자치의, 독립적인 이라는 사전적의미를 가지고 있습니다.

AUTONOMOUS TRANSACTION 는 호출한 메인 Module 과 독립적인 트랜잭션을 가질수 있는 방법 입니다. (9 차시)

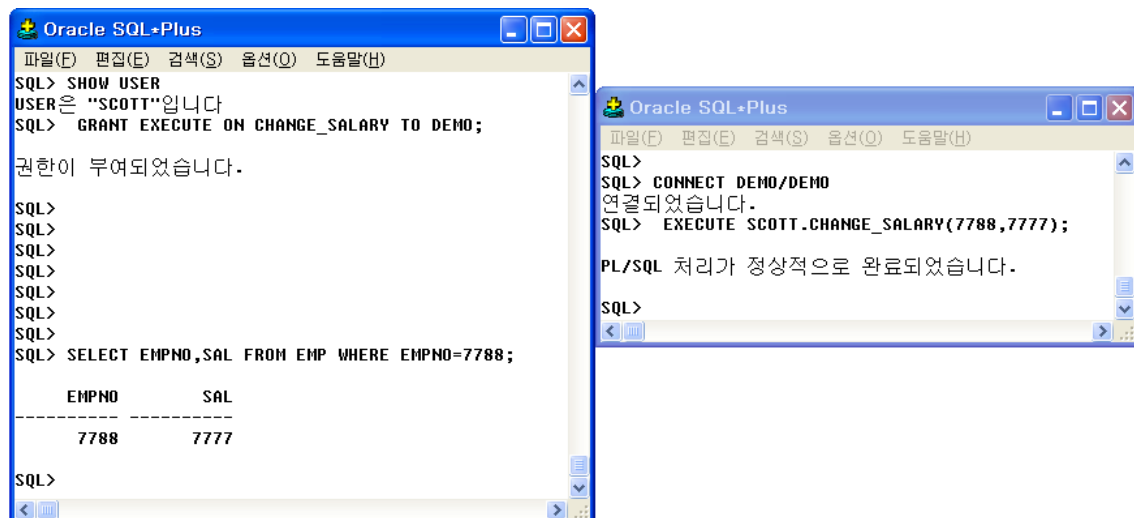
7-3 PROCEDURE 와 실행권한(EXECUTE PRIVILEGE)

사용자 정의로 생성된 FUNCTION/PROCEDURE/PACKAGE는 실행권한(EXECUTE) 이 있어야 실행이 가능합니다.

OBJECT 생성자는 해당 OBJECT에 대한 실행권한을 가지게 됩니다.

OBJECT 생성자가 아닌 계정의 사용자가 해당 OBJECT를 사용하기 위해서는 실행 권한을 가져야 합니다.

이번 CHAPTER에서는 실습 환경적인 문제로 실습이 없습니다.



- ① SCOTT 계정에서 CHANGE_SALARY PROCEDURE에 대한 실행권한을 DEMO 계정에 부여 합니다.

데이터베이스 프로그래밍-PL/SQL Named Block2(7차)

② DEMO 계정에서 SCOTT 소유의 CHANGE_SALARY를 실행할수 있습니다.

<참고> DEMO 계정은 EMP 테이블의 데이터에 대한 접근 권한이 없지만 CHANGE_SALARY 를 통해 해당 데이터에 간접적으로 접근 할수 있다는 사실도 기억해두어야 합니다. 실무에서 응용가능한 부분이 있습니다. 약간 더 깊게 접근하면 생성자 권한과 호출자 권한의 구분이 필요 하지만 PL/SQL에 대한 현장 경험을 가진후에 개념 학습을 개인적으로 하십시오

③ DEMO 계정에서 실행된 결과에 의해 SCOTT 계정의 데이터가 변경되었습니다.

[학습정리]

1. VARIABLE,PRINT는 SQL*PLUS를 PL/SQL BLOCK의 개발도구로 사용할수 있도록 한다. VARIABLE은 바인드 변수를 정의하고 정의된 바인드 변수의 리스트를 조회한다. PRINT는 바인드 변수의 내용을 출력한다.
2. EXECUTE 는 SQL*PLUS의 명령어로 STORED BLOCK을 편하게 실행/테스트 할수 있도록 해준다.
3. 바인드 변수는 HOST VARIABLE 또는 GLOBAL VARIABLE로도 불리운다.
4. 바인드 변수의 구분자는 : (세미콜론이 사용된다)
5. SQLCODE는 가장 최근에 발생한 EXCEPTION CODE 값을 리턴한다.
SQLERRM은 가장 최근에 발생한 EXCEPTION MESSAGE를 리턴한다.
6. FUNCTION/PROCEDURE/PACKAGE를 다른 계정에서 사용하기 위해서는 실행 권한(EXECUTE PRIVILEGE)를 가져야 한다.